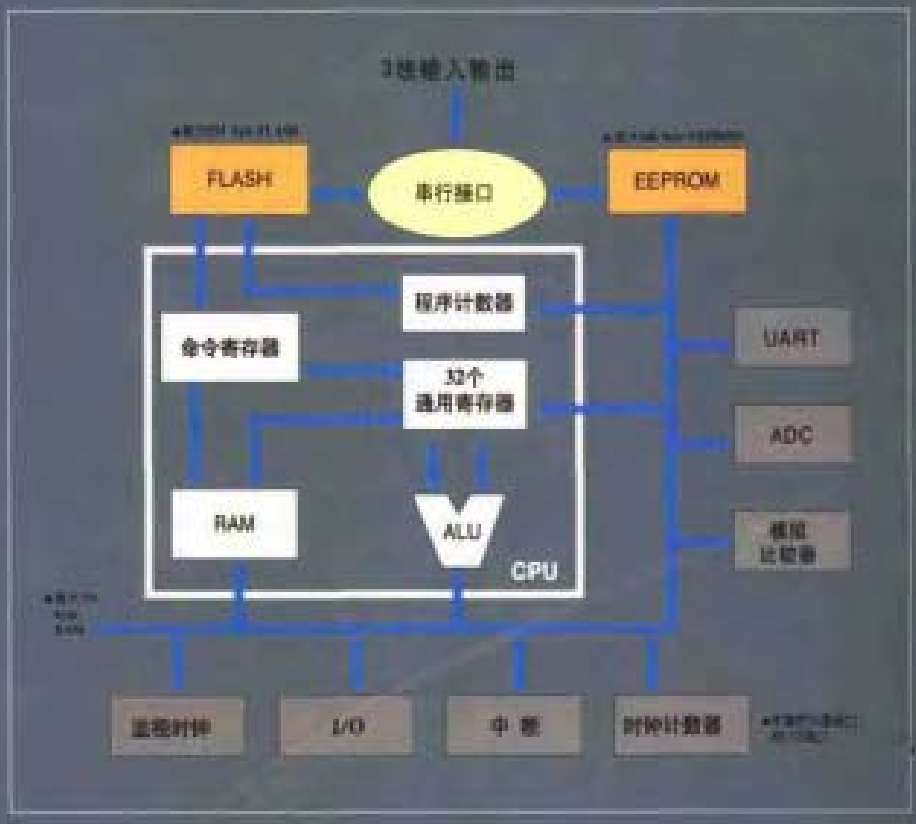
MCU

ATMEL

AVR高速嵌入式单片机原理与应用（修订版）

耿德根 宋建国 马 潮 叶勇建 编著

北京航空航天大学出版社
http://www.buaapress.com.cn



ISBN 7-81077-222-8

9 787810 772228 >

ISBN 7-81077-222-8/TP·121

定价：35.00元

AVR 高速嵌入式单片机原理与应用

（修订版）

耿德根　宋建国　马　潮　叶勇建　编著

北京航空航天大学出版社

内容简介

本书详细介绍 ATMEL 公司开发的 AVR 高速嵌入式单片机的结构；讲述 AVR 单片机的开发工具和集成开发环境(IDE)，包括 Studio 调试工具、AVR 单片机汇编器和单片机串行下载编程；学习指令系统时，每条指令均有实例，边学习边调试，使学习者看得见指令流向及操作结果，真正理解每条指令的功能及使用注意事项；介绍 AVR 系列多种单片机功能特点、实用程序设计及应用实例；作为提高篇，讲述简单易学、适用 AVR 单片机的高级语言 BASCOM－AVR 及 ICC AVR C 编译器。

本书的每个实验应用程序都是在 SL－AVR 开发编程实验器上，由广州天河双龙电子有限公司的科技人员和华东师范大学电子工程系(AVR 实验室)师生实验通过的。源程序清单及硬件接线图、系统工作软件，可上网(http://www.sl.com.cn)下载。广州天河双龙电子有限公司还可提供图文并茂的相关工作软件和实验应用源程序的光盘，作为本书的补充。

本书有较强的系统性和实用性，可作为高等院校自动化、计算机、电子、仪表等专业的教学参考及工程技术人员的实用参考，亦可作为应用技术的培训教材。

图书在版编目(CIP)数据

AVR 高速嵌入式单片机原理与应用/耿德根等编著. —修订版. —北京：北京航空航天大学出版社，2002.10

ISBN 7－81077－222－8

Ⅰ.A… Ⅱ.耿… Ⅲ.单片微型计算机，AVR 高速嵌入式 Ⅳ.TP368.1

中国版本图书馆 CIP 数据核字(2002)第 073760 号

AVR 高速嵌入式单片机原理与应用

(修订版)

耿德根 宋建国 马 潮 叶勇建 编著

责任编辑 王小青 王 瑛

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100083) 发行部电话(010)82317024 †

http://www.buaapress.com.cn

E-mail:pressell@publica.bj.cninfo.net

河北省涿州市新华印刷厂印制 各地书店经销

*

开本：787×1 092 1/16 印张：27 字数：691 千字

2002 年 10 月第 2 版 2002 年 10 月第 4 次印刷 印数：14 000～18 000 册

ISBN 7－81077－222－8/TP·121 定价：35.00 元

致 读 者

美国 ATMEL 公司率先将 Flash 存储技术应用于单片机产品中，推出了 AT89 系列单片机，在全球电子业内引起了巨大的反响，在中国也受到了众多用户的喜爱。在此，特向各位用户表示感谢！

继 AT89 系列之后，ATMEL 公司又向世界发布了本书所介绍的全新配置的精简指令集(RISC)AVR(ADVANCE RISC)AT90 系列单片机。ATMEL 公司的步伐始终在不断向前，现在，更强的 32 位 AT91M 系列单片机也已上市。

ATMEL 公司是一家跨国的专业半导体企业。总部设在美国著名的硅谷圣何塞，在美国、法国、德国、英国、马来西亚都设有大型生产厂，有十几个专门的配套和封装厂，另还在几十个国家和地区设有设计中心和办事处。其中 AVR 产品的设计中心就设在挪威。ATMEL 以其最先进的工艺和技术生产各种存储器、可编程逻辑器件、多种系列的单片机及数十种智能卡。其中以 AVR 为核的几款 CPU 卡的性能非常具有竞争力。最新推出的 AVR + FPGA + SRAM (FPSLIC——Field Programmable System Level ICs)现场可编程系统级电路，更占据业内的领先地位。ATMEL 还为宇航、军事、工业及民用等许多专门用户设计生产了大量的数字、模拟、逻辑混合型专用电路(ASIC)。

ATMEL 在电可擦技术上，拥有世界上最多的专利和最先进的工艺，又有制作混合电路的技术和经验，是少数可将 SiGe(硅锗)技术用于 RF 射频通讯产品的公司之一。我们将这些特点与先进的 AVR 核相结合，为您提供最优配置的系列单片机。AVR 系列单片机的主要特点是：程序区 Flash 可多次电擦写；内部含电可擦数据 EEPROM 存储器，可串行下载；最新、最优、最全的配置；执行速度高，指令高效率；低电压，低功耗，高驱动；程序加密性好；简便易学，开发工具廉价；型号全，适用范围广；可扩展性强，可为用户做专用芯片。最重要的还有：AVR 物美价廉，雅俗共赏，生命力长久。

ATMEL 公司很欢迎各位朋友选用 AVR 产品，愿为您的设计提供服务，也愿为您要求的专用芯片提供设计和生产。

希望这本书能给您了解和使用 AVR 提供方便，也希望得到您的建议和帮助。

向为本书出版作出巨大贡献的广州天河双龙有限公司的耿德根先生、北京航空航天大学的何立民教授、上海华东师范大学的马潮教授、ATMEL 北京办事处的叶勇建先生、北京航空航天大学出版社等朋友，致以深深的谢意；再次向 1998 年为 AVR 出书的宋建国先生表示感谢；在此还要向 ATMEL 在中国的所有代理商及分销商表示感谢。

美国爱特梅尔(ATMEL)公司

2000 年 12 月 1 日

ATMEL 公司代表处及双龙电子有限公司联络方式

美国爱特梅尔股份有限公司
www. atmel. com
香港九龙尖沙咀东部麼地道 77 号华懋广场 1219 室
852－27219778,27221369(F)

爱特梅尔北京代表处
北京海淀区学院南路 70 号 710 室(100081)
010－62186224,62186225,62180478,62186227(F)

爱特梅尔上海代表处
021－62809234,62809241,62808604,62807592(F)

广州天河双龙电子有限公司
www. sl. com. cn
广州双龙:广州天河路 561 号新赛格电子城 331 室(510630)
电话:020－87578852、87505012　传真:分机 620

北京双龙:北京海淀区知春路 132 号中发大厦 616 室(100086)
电话:010－82623551、62653785　传真:010－82623550

上海双龙:上海北京东路 668 号科技京城东楼 12H2 室(200001)
电话:021－53081501、53081502　传真:021－53081502

修订版前言

本书从2001年2月出版以来，已第4次印刷，台湾全华科技有限公司购本书版权出版。根据ATMEL公司AVR单片机新品种、新软件的不断推出及读者的要求，决定对本书进行修订。

对第三章“AVR单片机开发工具”作了重要修订。广州天河双龙电子有限公司的SL－AVR“四合一”AVR单片机开发实验器的硬件作了改进：新版SL－AVR下载插座改为DC10插座，使用编平压线技术，接插可靠；也可使用SL－AVRISP并行高速下载线，对大容量器件ISP下载编程带来方便。该机工作晶振改为可插拔更换，便于用户超频、降频实验。SL－AVR还可配AVR单片机组态互动监控，使学习AVR单片机的过程更加形象直观、有声有色、通俗易懂。这样既充分利用了PC机的资源，又可学习工控组态，与现实社会的应用接轨。开发软件改为ATMEL AVR Studio 3.5X以上集成开发环境(IDE)，包括：① AVR Assembler编译器；② AVR Studio调试功能；③ AVR Prog串行、并行下载功能；④ JTAG ICE仿真；等功能。

第五章5.3节改为“ATmega8/8L”和5.10节改为“ATmega128/128L”新器件资料，去掉停产的“AT90S2323/2343”及“ATmega603/103”资料。

第九章AVR C语言的应用，原来的IAR C软件很好，但价格在万元以上，一般用户承受不了，不易推广；现改为ICC AVR C高级语言编译器。C编译器提供Image Craft Inc.的30天免费试用版ICC AVR demo，30天后转为2KB限止版。广州双龙电子有限公司是ICC AVR的国内独家代理，有技术支持，正版价格仅在千元左右。该软件及其升级版均可从互联网(www.imagecraft.com)上免费获得。国内将出版ICC AVR C编译器的应用专著。

本书对AVR器件的选购指南，提供ATMEL公司的最新器件资料。

对原书中的错误也作了修订。

由于我们水平有限，本书难免有错误和不当之处，望读者指正。

作　者

2002年7月于广州

前　言

随着电子技术的迅猛发展，单片机更广泛地应用于军事、工业、通讯、家用电器、智能玩具、便携式智能仪表等领域，使产品功能、精度和质量大幅度提高，而电路设计更简单、故障率低、可靠性高且成本低廉。应该看到，Flash 技术、在线可编程、低功耗、大规模集成是今后单片机的发展方向。

ATMEL 公司是全球著名的半导体公司之一。90 年代初，ATMEL 率先把 MCS－51 内核与其擅长的 Flash 技术相结合，推出轰动业界的 AT89 系列单片机。至今，ATMEL 在 MCS－51 市场上仍占据主要份额。1997 年，ATMEL 挪威设计中心的 A 先生与 V 先生出于市场需求考虑，充分发挥其 Flash 技术优势，推出全新配置的精简指令集（RISC）单片机，简称 AVR。几年来，AVR 单片机已形成系列产品，其 ATtiny、AT90 与 ATmega 分别对应为低、中、高档产品（高档产品含 JTAG ICE 仿真功能）。为了使广大用户和读者对 AVR 单片机的原理与应用有一个系统、详细的认识，尤其针对 AT90 系列，我们特意编写了这本书。

一、AVR AT90S 系列单片机的优点

（1）价格低廉的、可擦写 1 000 次以上的、16（字）位指令（程序存储器数据为 16 位，即 XXXX×16；也可理解为 8 位，即 2×XXXX×8）。因采用了 Flash 技术，不再有报废品产生。数据存储器为 8 位，AVR 还是属于 8 位单片机。

（2）高速度（50ns）、低功耗（μA）！具有 Sleep（休眠）功能及 CMOS 技术，每一指令执行速度可达 50ns（20MHz），而耗电则在 1～2.5mA 间（典型功耗，WDT 关闭时为 100nA）。AVR 运用 Harward 结构概念（具有预取指令功能），即对程序和数据存储带有不同的存储器和总线。当执行某一指令时，下一指令被预先从程序存储器中取出，这使得指令可以在每一个时钟周期内被执行。

（3）高度保密（LOCK）！保密位在芯片底部，无法用设备看到。可多次烧写的 Flash 且具有多重密码保护锁死（LOCK）功能，因此可低价快速完成产品商品化，并可多次更改程序（产品升级）而不必浪费 IC 或电路板，大大提高产品质量及竞争力。

（4）工业级（WDT）产品！具有大电流（灌电流）10～20mA 或 40mA（单一输出），可直接驱动 SSR 或继电器；有看门狗定时器（WDT），安全保护，防止程序走飞，提高产品的抗干扰能力。

（5）超功能精简指令！具有 32 个通用工作寄存器（相当于 8051 中的 32 个累加器，克服了单一累加器数据处理造成的瓶颈现象）及 128B～4KB 个 SRAM，可灵活使用指令运算，并可用功能很强的 C 语言编程，易学、易写、易移植。

（6）程序写入器件可以并行写入（用万用编程器或并行高速下载），也可串行在线下载（ISP）擦写。也就是说不必将 IC 拆下拿到万用编程器上烧录，而可直接在电路板上进行程序修改、烧录等操作。方便产品升级，尤其是 SMD 封装，更利于产品微型化。

（7）并行 I/O 口输入/输出特性与 PIC 的 HI/LOW 输出及三态高阻抗 HI-Z 输入类同外，

也可设定类同 8051 系列内部拉高电阻作输入端的功能，便于各种应用特性的所需（多功能 I/O 口）。AVR 是真正的 I/O 口，能正确反映 I/O 口的输入/输出真实情况。

(8) 单片机内有模拟比较器，I/O 口可作 A/D 转换用，可组成廉价的 A/D 转换器。

(9) 像 8051 一样，有多个固定中断向量入口地址，可快速响应。

(10) 像 PIC 一样，可重设启动复位。AVR AT90S 系列也有内部电源开关启动计数器，可将低电平复位($\overline{RESET}$)直接接到 V_{CC} 端。当电源开时，由于利用内部的 RC 看门狗定时器，可延迟 MCU 启动执行程序。这种延时使 I/O 口稳定后执行程序，以提高单片机工作可靠性，同时可节省外加复位延时电路。

(11) 具有休眠省电功能(POWER DOWN)及闲置(IDLE)低功耗功能。

(12) AT90S1200 等部分 AVR 器件具有内部 RC 振荡器——1MHz 的工作频率，使该类单片机无需外加元器件即可工作，可谓简单方便。

(13) 计数器/定时器、C/T 有 8 位和 16 位，可作比较器；计数器外部中断和 PWM(也可当 D/A)用于控制输出。

(14) 有串行异步通讯 UART，不占用定时器和 SPI 传输功能；因其高速故可以工作在一般标准整数频率，而波特率可达 576 000。

(15) AT90S4414 及 AT90S8515 具有可扩展外部数据存储器达 64KB。它们的引脚排列及功能与 8051 相似，即可替代 8051 系列单片机(8751/8752)的应用系统。当然，还在硬件、软件上带来很多优点(如 WDT 看门狗，模拟比较器作 A/D，PWM 作 D/A 等)。

(16) 工作电压范围宽(2.7～6.0V)，电源抗干扰性能强。

(17) 有多通道 10 位 A/D 及实时时钟 RTC。具有 8 路 10 位 A/D 器件的有 AT90S4434/8535；具有 6 路 10 位 A/D 器件的有 AT90S2333/4433。ATmega103(L)单片机更有 Flash 128KB，EEPROM 4KB，RAM 4KB，I/O 端口 48 个，中断源 16 个，外部中断 8 个，SPI 1 个，UART 1 个，8 位定时器 2 个，16 位定时器 1 个，PWM(D/A)4 个；还有看门狗定时器、实时时钟 RTC、模拟比较器，8 路 10 位 A/D，可在线编程 ISP，工作电压为 2.7～5.5V。

(18) AVR 由 Flash 存储器构成，并具有较大容量、可擦写 100 000 次的 EEPROM，对掉电后数据的保存带来方便，来电后能记住掉电时的工作状态。

(19) 从高级语言 C 代码，看各种单片机性能比较(见表 0.1)

```
/* 一个小C函数
/* Return the maximum value of a table of 16 integers */

int max(int *array)
{
  char a;
  int maximum=  32768;

  for  (a=0;a<16;a++)
    if  (array[a]>maximum)

      maximum=array[a];
  return  (maximum);
```

}

AVR 汇编输出：字节数——46 B，执行时间——335 周期。

80C51 汇编输出：字节数——112 B，执行时间——9 384 周期。

68HC11 汇编输出：字节数——57 B，执行时间——5 244 周期。

PIC16C74 汇编输出：字节数——87 B，执行时间——2 492 周期。

AT90S8515 8MHz/80C51 24MHz/68HC11A8 12MHz/PIC16C74 20MHz。

表 0.1　从 C 代码比较各种单片机性能

	字节数/B	执行时间/μs	耗电/mA	功耗/s·(mW)$^{-1}$
AT90S8515	46(1)	42(1)	11(1)	434(1)
80C51	112(2.4)	391(9)	16(1.5)	32(0.07)
68HC11A8	57(1.2)	437(10)	27(2.5)	17(0.04)
PIC16C74	87(1.9)	125(3)	13.5(1.2)	119(0.27)

从上述得出的结论为：8MHz AVR 等于 224MHz 80C51。

二、AVR 的 2 种开发工具

1. ICE200

ICE200 采用 AVR 专用仿真 CPU 与监控 CPU 独立设计的方案，充分提供各种调试手段，真实再现被仿 AVR 的各种特性。它可仿真 AVR 的器件有 ATtiny10/11/12(V/L)、AT90S1200/2313、AT90(L)S2333/4433、AT90S4414/8515、AT90(L)S4434/8535。由于仿真器的电源不对外，所以 ICE200 也支持低电压器件。ICE200 的仿真软件最新版为 STUDIO 3.X，在支持以上 11 种 AVR 以外，还可模拟其它 AVR 器件的运行，支持汇编及 C 高级语言。其中汇编编译器免费提供，C 编译器只提供 Image Craft Inc. 的 30 天免费试用版 Icc AVR demo。该软件及其升级版均可从互联网(www.imagecraft.com)上免费获得。

ICE200 包括一个仿真器主板、一个 POD 板(AtadapEM04，有仿真 CPU)、五块适配器板(适合 DIP8/20/28/40 封装 AVR CPU)、一块诊断保护板(ATadap4000)、两根柔性印刷电缆、一根 9 针串行通讯电缆、一个 9V 直流电源(赠送)、一份工作光盘(含 ICE200 中文使用手册)、一份 ICE200 简介。

广州市天河双龙电子有限公司为了使 AVR 单片机在我国迅速得到应用，引进美国原装 AVR 实时在线仿真器 ICE200 及 Icc AVR C 编译器，推广价定位在国人能承受的价位。

2. SL－AVR

经济普及型“四合一”AVR 串行下载开发实验器 SL－AVR，等于 AVR 编程器＋模拟仿真器＋实验器＋科研样机。它的硬件采用模块化设计，便于用户灵活组成科研项目所需的各种硬件结构。硬件有：RS232 通信接口；串行下载监控；DIP8/20/28/40 通用锁紧插座，DIP40 端口用短路块连接作输出，用 LED 发光二极管显示器件引脚高低电平，也可用短路块断开，作输入或其它用途；6 位 LED 数码管作显示；2×16 点阵 LCD 液晶显示器；17 键的键盘；改进版 SL－AVR＋，增加 PC 机键盘接口及对 AT89S 开发实验；模拟比较输入电路；音响电路；复位电路；模拟电压输入电路等；随机附 120mm×170mm 万通实验板及一片 AT90S8515 器件。

SL－AVR 适用于所有具有串行下载编程功能的 AVR 单片机，用户板上的 AVR 器件无需拆下即可编程，同时还可作 AVR 单片机的 I/O 口、A/D、D/A、LED、LCD、键盘输入、步进电机控制、音频输出、模拟比较等开发实验；提供功能强大的 WIN 版汇编级编译器 WAVRASM、模拟仿真调试软件 AVR Studio3. X 及串行下载软件 AVR PROG，同时也提供限制版的 C(IAR、ICC)编译器、限 2KB 的 BASCOM－AVR 编译器；对初学 AVR 单片机的设计者，可暂时节省购买较昂贵的实时仿真器及万用编程器的费用。

SL－AVR 开发实验器提供了几十个实用实验程序，用户也可改变硬件接口、修改程序，实现源程序的其它功能。这对大专院校学生发挥其创造性思维及培养其动手能力特别有用，可改变我国传统教育下的“高分低能”的弊病。该开发实验器也可当作科研样机使用。

SL－AVR 开发实验器是由广州天河双龙电子有限公司开发出的。本书的每个实验应用程序都是在 SL－AVR 开发编程实验器上，由广州天河双龙电子有限公司的科技人员和华东师范大学电子工程系(ATMEL 实验室)师生实验通过的。源程序清单及硬件接线图、系统工作软件，可上网(http://www. sl. com. cn)下载。广州天河双龙电子有限公司还可提供图文并茂的相关工作软件和实验应用源程序的光盘*，作为本书的补充。

SL－AVR 现在还具有单片机组态开发功能，可以生动、形象、直观地了解单片机 I/O 口基本功能、I/O 口的扩展功能及单片机组态开发的复杂应用。

SL－AVR 还可利用宽带网做 ISP 下载编程实验。

SL－MEGA 高档开发实验器，可开发 ATmega103/128 单片机。

本书共分九章。第一章 ATMEL 单片机简介；第二章介绍 AVR 单片机系统结构；第三章介绍 AVR 单片机开发工具；第四章介绍 AVR 单片机指令系统；第五章介绍 AVR 单片机 AT90 系列；第六章介绍实用程序设计；第七章介绍 AVR 单片机的应用；第八章介绍 BASCOM－AVR 的应用；第九章介绍 AVR C 语言的应用。附录简单介绍 AT89、AT94K 系列单片机和指令集综合。书后附有 ATMEL 公司的产品目录、公司代表处及购买 SL－AVR 的优惠证。

本书由耿德根主编。耿德根编写第三、六、七、九章，并对第四章每条指令用程序调试方法验证指令功能；宋建国编写第一、二、四章；叶勇建编写第五章；马潮编写第八章。本书承蒙何立民教授的关心和支持，在审校中得到了王小青副总编等的大力支持，在录入、校稿、实验等方面得到了耿陆卫、沈延红、周惠忠、葛仁春、钟楚洪、覃辉等同志的大力协助；ATMEL 公司提供全部技术资料、广州天河双龙电子有限公司提供实验设备、华东师范大学电子工程系(ATMEL 实验室)等给予多方面的帮助，在此一并致谢！

编　者

2000 年 12 月于广州

* 本书配套光盘的邮购方法

邮购地址：(邮编 510630)广州天河路 561 号新赛格电子城 331 室

邮购费：20 元(平寄)/30 元(特快专递)

联系人：耿德根(电话 020－87578852　E-mail：SLLG@sl. com. cn)

目　录

第一章　ATMEL 单片机简介

1.1　ATMEL 公司产品的特点

ATMEL 公司是世界上著名的高性能、低功耗、非易失性存储器和数字集成电路的一流半导体制造公司。ATMEL 公司最引人注目的是它的 EEPROM 电可擦除技术，闪速存储器技术和高质量、高可靠性的生产技术。在 CMOS 器件生产领域中，ATMEL 的先进设计水平、优秀的生产工艺及封装技术，一直处于世界领先地位。这些技术用于单片机生产，使单片机也具有优秀的品质，在结构、性能等方面都有明显的优势。ATMEL 公司的单片机是目前世界上一种独具特色且性能卓越的单片机。它在计算机外部设备、通讯设备、自动化工业控制设备、宇航设备、仪器仪表和各种消费类产品中都有着广泛的应用前景。ATMEL 公司产品的主要特点表现为如下几点。

一、以 EEPROM 电可擦除及 Flash 技术为主导

ATMEL 公司把其 EEPROM 及 Flash 技术巧妙地结合形成特殊的集成电路，从面使一些芯片的应用领域扩大，其中闪速可编程逻辑器件 Flash PLD、Flash 存储器、AT90 系列 Flash 单片机、AVR 增强型单片机、智能 IC 卡等是最典型的产品。这些产品内部含有 Flash 存储器，从而使它们在无交流电环境下的、便携类的产品中大有作为。含有 EEPROM 及 Flash 存储器是 ATMEL 公司有关产品的明显特色之一。

在 ATMEL 公司的 Flash 产品中，一共有商业 C 档（0～＋70℃）、工业 I 档（－40～＋85℃）、汽车 A 档（－40～＋125℃）和军用 M 档（－55～＋125℃）四种档次的产品。

二、多种封装形式和高的质量

ATMEL 公司有多种封装形式的集成电路，封装形式有：DIP、PGA、PQFP、TQFP、SOIC、CBGA、μBGA 和客户专门定制等多种。

ATMEL 公司的封装是按军工标准进行的，有优秀的商品质量。军工产品封装和测试按军用标准 MIL－STD－883 进行。所有的军工产品在制造和开发过程中均以 MIL－M－38510 标准说明书为依据，并且追踪这个标准的最新版本。同时，ATMEL 公司将统计过程控制 SPC 标准用于军用 IC 的装配和测试中，从而优化质量和产品的稳定性。

三、高标准的质量检测

ATMEL 公司具有高质量、高水准的检测能力，可以对数字集成电路和模拟集成电路、军用集成电路进行质量检测。就 ATMEL 公司的军用集成电路产品而言，其工作性能是完全符合军用标准的，在－55～＋125℃范围内正常工作，甚至在高达＋150℃（特殊档）的条件下集成电路仍然实现正常的输出功能。ATMEL 公司的 Audo、Sentry 和 Teraclyne 测试器件符合统计过程控制 SPC 标准，并且依照美国国家标准局的测试标准执行。

由于 ATMEL 公司的产品具有优秀的品质，故在航空航天仪器、雷达系统、导弹、智能自适应仪器、机器人、各种武器电子系统、抗恶劣环境电子系统等都被广泛加以应用。

1.2 AT90 系列单片机简介

AT90 系列单片机是增强 RISC 结构、内载 Flash 的单片机，通常简称为 AVR 单片机。

AT90 系列单片机是基于新的精简指令 RISC 结构，在 90 年代开发出来的，综合了半导体集成技术和软件性能的新型单片机。这种结构使得在 8 位微处理器市场上，AVR 单片机具有最高 1 MIPS/MHz 能力。

为了缩短进入市场的时间和简化维护的支持，对于单片机来说，用高级语言编程是一种标准编程方法。AVR 单片机的开发目的就是在于能采用 C 语言编程，从而能高效地开发出目标产品。为了对目标代码大小、性能及功耗的优化，AVR 单片机采用了大型快速存取寄存器文件和快速单周期指令。

快速存取 RISC 寄存器文件由 32 个通用工作寄存器组成。传统的基于累加器的结构需要大量的程序代码，以实现在累加器和存储器之间的数据传送。在 AVR 单片机中，用 32 个通用寄存器代替累加器，从而避免了传统的累加器和存储器之间的数据传送造成的瓶颈现象。

在 AVR 单片机中，在前一条指令执行的时候就取出现行的指令，然后以一个周期执行指令。在其它的 CISC 以及类似的 RISC 结构中，外部振荡器的时钟被分频降低到传统的内部执行周期，这种分频最大达 12 倍。AVR 单片机是用一个时钟周期执行一条指令的，在 8 位单片机中它是第一种真正的 RISC 单片机。

AVR 单片机具有良好的性能价格比。这个系列有引脚少的器件，也有含较大容量存储器、引脚较多的器件。由于 AVR 单片机是采用 Harvard 结构的，故它们的程序存储器和数据存储器是分开的。可直接访问 8M 字节程序存储器和 8M 字节数据存储器，寄存器文件被双向映射，并能访问像片内允许快速上下转换的那部分 SRAM 存储器。

AVR 单片机采用低功率、非挥发的 CMOS 工艺制造。通过 SPI 口和一般的编程器，可以对 AVR 单片机的 Flash 存储器进行编程。

AT90 系列单片机目前有 AT90S1200、AT90S2313、AT90S4414、AT90S8515、ATmega8/8L、ATmega128/128L、AT90S4434、AT90S8535 等多种型号。它们在功能和存储器容量等方面有一定的区别，但都是比 AT89 系列要强的单片机。

1.3 AT91M 系列单片机简介

AT91M 系列单片机是基于 ARM7TDMI 嵌入式微处理器的、16/32 位微处理器系列中的一个新成员。该微处理器用高密度的 16 位指令集实现了高效的 32 位 RISC 结构，且功耗很低。此外，内部的工作寄存器很多，使该器件非常适用于实时控制的应用。该器件使用 ATMEL公司的高密度 CMOS 技术，通过在一个单片上集成了 ARM7TDMI 和大量的 ROM 程序区，以及片内 RAM 和广泛的外设功能，使得 AT91M 成为一个强有力的微控制器，为许多需要加强运算的嵌入式微控制器提供了高度的灵活性和高性能价格比。

AT91M 使用了基于先进微控制器总线结构(AMBA)的模块化设计方法，具有综合、快速、高性能价格比的特点。

表 1.1 为 AT91M 系列部分产品的 ROM 大小表。

表 1.1 AT91M 系列部分产品的 ROM 大小表

型 号	速度 /MHz	工作温度	Flash /MB	Mask ROM /KB	SRAM /KB	封 装	电压 V	IEEE 1149.1	省 电	其他特点
M40400	25, 33	C/I	—	—	4	TQFP100	2.7～3.6		空闲模式	3 个定时器 2 个 UARTs 看门狗 外设数据控制器
M40100	12	C/I		—	4	TQFP100	1.8～3.6	—		
M40416	25	C/I	2	—	4	BGA120	2.7～3.6	—		
M40100	33, 40	C/I	—	—	1	TQFP100	2.7～3.6	—	取消 CPU 和外设时钟	
M40800	33, 40	C/I	—	—	8	TQFP100	2.7～3.6	-		
R40807	33	C/I	—	—	8+128	TQFP100	2.7～3.6	-		
M40403	33	C/I	-	32	4	TQFP100	2.7～3.6	—		
M40807	33	C/I	—	128	8	TQFP100	2.7～3.6	—		
M63200	25	C/I	-	-	2	TQFP176	2.7～3.6	y		多处理器接口 6 个定时器，3 个 UARTs SPI 外设数据控制器 看门狗
M55200	33	C/I	—		2	TQFP176	2.7～3.6	y	取消时钟、低速、待命和省电模式	8 通道 ADC，2 通道 DAC 实时时钟，Osc + PLL 6 个定时器，3 个 UARTs SPI 外设数据控制器 看门狗
M55800	33	C/I	—	—	8	TQFP176	2.7～3.6	y		

第二章　AVR单片机系统结构

ATMEL公司的AT90系列嵌入式单片机是一种基于AVR增强性能、RISC结构的、低功耗、CMOS技术、8位微控制器(Enhanced RISC Microcontrollers)，通常简称为AVR单片机。目前有AT90S1200、AT90S2313、AT90S4414、AT90S8515、ATmega8/8L、AT90S8535、ATmega83/163、ATmega128/128L、ATmega161、ATtiny10/11/12/15/22/24/28等多种型号。它们的基本结构都比较相近。这一章以AT90S8515单片机的内部结构为主，叙述AVR单片机系统结构。

2.1　AVR单片机总体结构

AT90系列单片机通过在单一时钟周期内执行功能强大的指令，每MHz可实现1MIPS的处理能力，这使得设计者可以优化功耗与速度之间的矛盾。

AVR核为32个通用工作寄存器与丰富指令集的组合。32个寄存器全部直接地与运算逻辑单元(ALU)连接，这使得可以通过在一个时钟周期内执行一条指令来访问到2个独立的寄存器。这种组合机构具备的代码效率比完成同样处理能力的常规CISC微控制器要快十倍。

AT90系列单片机具有以下特征：1K～128K字节可下载的Flash存储器、64～4K字节EEPROM、128～4K字节RAM、5～53条通用的I/O线、32个通用工作寄存器、带模拟比较器的定时器/计数器、可编程的异步UART串行口、内部及外部中断、带内部晶振的可编程看门狗定时器、一个为下载程序而设计的SPI串行口、10位A/D转换器以及2个可通过软件选择的省电模式。空闲模式停止CPU的工作，而SRAM、定时器/计数器、片内振荡器(RTC)、SPI口及中断系统继续工作。电源检测功能(BOD)、掉电模式保留寄存器的内容，但冻结晶振，终止芯片的其它功能，直至下一次外部中断或硬件复位。

该器件的制造运用了ATMEL公司的高密度、非易失存储器技术。芯片内可下载的Flash存储器可以通过SPI串行接口或通过通用的非易失存储器、编程器对程序存储器进行系统内的重新编程。通过在单一芯片内将一个增强性能的RISC 8位CPU与可下载的Flash结合，使得ATMEL的AT90系列单片机成为一种满足于许多要求、具有高度灵活性和低成本的嵌入式控制应用的高效微控制器。

AT90系列单片机AVR的全套编程和系统开发工具包括：C编译器、BASCOM－AVR、宏汇编器、程序调试器/程序仿真器、系统在线仿真器和SL－AVR开发下载实验器。

图2.1为AT90S8515单片机方框图，说明了AT90系列单片机的内部结构。

一、引脚说明

AT90S4414和AT90S8515引脚相同，仅Flash、SRAM和EEPROM相差一倍。AT90S4414/AT90S8515引脚与MCS－51系列单片机8X51/8X52的引脚兼容，仅复位电平不同，AVR低电平复位，MCS-51高电平复位。这给用AVR单片机替代MCS-51单片机

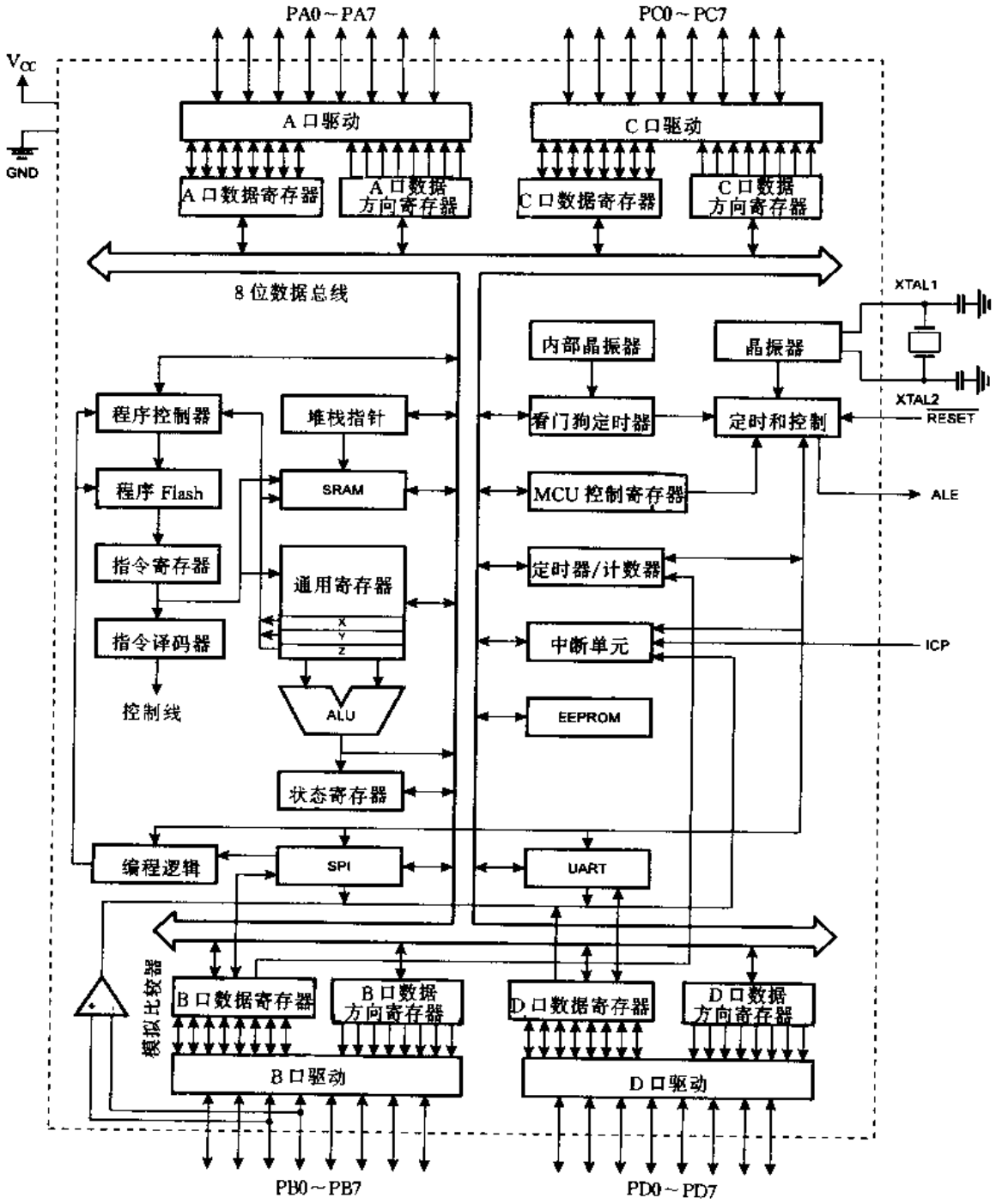


图 2.1　AT90S8515 单片机方框图

硬件电路带来方便。

V_{CC}：V_{CC}为供电引脚，连接到正电源。

GND：GND 为接地引脚，连接到电源地。

A 口(PA7～PA0)：A 口为一个 8 位双向 I/O 口，每一引脚内部都有上拉电阻。A 输出

口的缓冲器可以吸收 20mA 的电流，因而能直接驱动 LED 显示器。当 A 口被用于输入且内部上拉被触发时，如果外部被拉低，则会输出电流。当使用外部 SRAM 时，A 口作为复用的地址/数据和输入/输出口。

B 口(PB7～PB0)：B 口为一个 8 位双向 I/O 口，每一引脚内部都有上拉电阻。B 口的输出缓冲器可以吸收 20mA 的电流。当 B 口被用于输入且内部上拉被触发时，如果外部被拉低，则会输出电流。B 口也提供后面列出的 AT90 系列单片机许多特殊功能。

C 口(PC7～PC0)：C 口为一个 8 位双向 I/O 口，每一引脚内部都有上拉电阻。C 口的输出缓冲器可以吸收 20mA 的电流。当 C 口被用于输入且内部上拉被触发时，如果外部被拉低，则会输出电流。当使用外部 SRAM 时，C 口作为地址输出。

D 口(PD7～PD0)：D 口为带有内部拉高的 8 位双向 I/O 口。D 口的输出缓冲器可以吸收 20mA 的电流。当 D 口被用于输入且内部上拉被触发时，如果外部被拉低，则会输出电流。D 口也提供后面列出的 AT90 系列单片机许多特殊功能。

$\overline{\text{RESET}}$：$\overline{\text{RESET}}$为复位输入。当晶振运行时，引脚上一个两周期的低电平可对器件进行复位。

XTAL1：XTAL1 为晶振反相放大器的输入端和内部时钟操作电路的输入端。

XTAL2：XTAL2 为晶振反相放大器的输出端。

ICP：ICP 是定时器/计数器 1 的输入捕获功能的输入引脚。

OC1B：OC1B 是定时器/计数器 1 的输出比较功能 B 的输出引脚。

ALE：ALE 是使用外部存储器时的地址锁存触发端。ALE 选通门被用于在第一个访问周期中将低位地址锁存到地址锁存器中，而 PD0～PD7 在第二个访问周期中被用作数据。

二、晶振

XTAL1 和 XTAL2 单独地作为反相放大器的输入和输出，该放大器如图 2.2 所示，可被设置为片内的晶振。可使用石英晶振或陶瓷谐振器。为了由外部源驱动器件，当 XTAL1 被驱动时，XTAL2 不能连接，如图 2.3 所示。

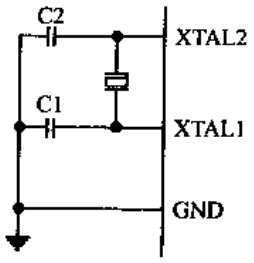


图 2.2 晶振连接

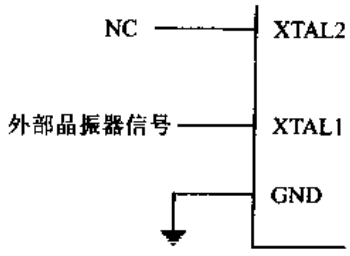


图 2.3 外部时钟驱动设置

2.2 AVR 单片机中央处理器 CPU

AT90 系列单片机 AVR RISC 微控制器向上与 AVR 增强 RISC 结构兼容。为 AT90 系列单片机 MCU 而编写的程序与 AVR 8 位 MCU(AT90SXXXX)全部系列产品的源代码和运行的时钟周期相兼容。

2.2.1 结构概述

快速访问寄存器堆概念包括32个带有单一时钟周期访问时间的8位通用工作寄存器。这意味着在一个单一时钟周期内，执行一个ALU操作(运算逻辑单元)。从寄存器堆中输出2个操作数，且操作数被执行，运行结果被存储回寄存器堆　　这些操作在一个时钟周期内完成。

32个寄存器中的6个寄存器作为3个16位间接地址寄存器指针，被用于数据空间寻址，从而可以进行高效的地址计算。3个寄存器中的一个还被用作地址指示器，完成常量表的查询功能。这些新加的功能寄存器为16位X-寄存器，16位Y-寄存器，16位Z-寄存器。

ALU支持寄存器之间的运算和逻辑功能，以及常数和寄存器之间的运算与逻辑功能。ALU也执行单一的寄存器操作，图2.4所示为AT90S8515单片机AVR的结构。

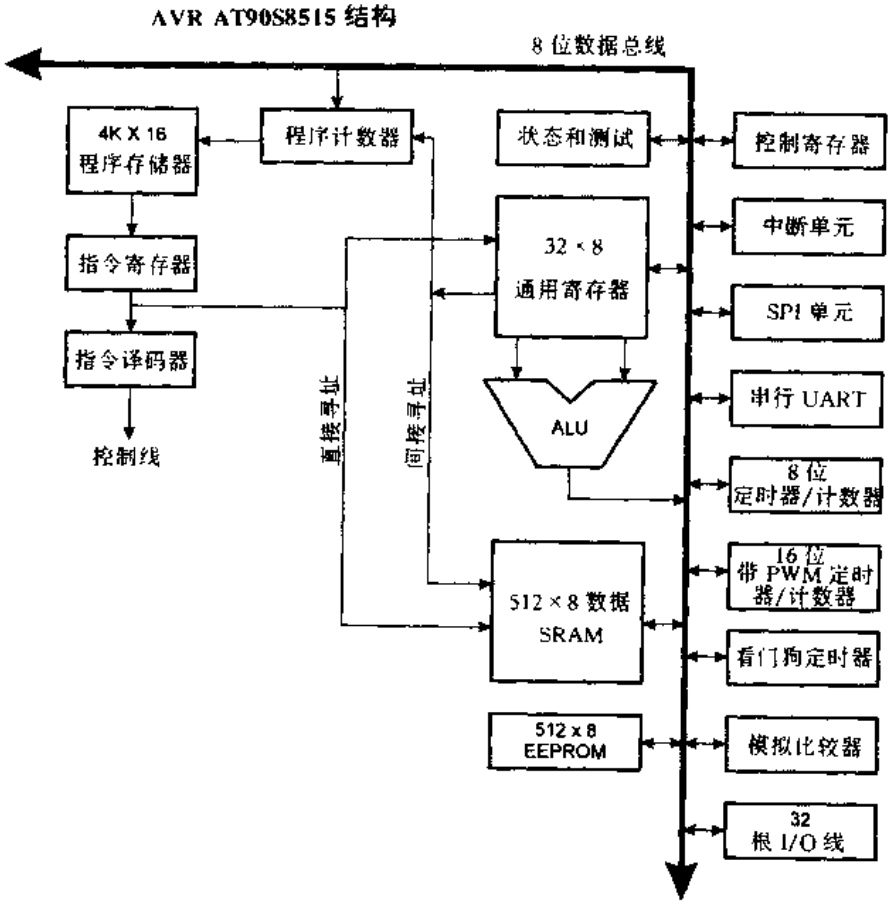


图2.4 AT90S8515单片机AVR增强性RISC结构

除了寄存器操作功能之外，常规存储器寻址模式还可用在寄存器堆上。寄存器堆被分配在最低的32个数据空间地址($00～$1F)，当触发时，即使它们为一般的存储地址，也能访问到。

I/O存储器空间包含64个作为CPU外围功能的地址，为控制寄存器、定时器/计数器、A/D转换器及其它I/O功能。I/O存储器可被直接访问，或作为寄存器堆之后的数据空间，

地址为 $20～$5F。

AVR 运用 Harvard 结构概念，即对程序存储和数据带有不同的存储器和总线。通过单一级的流水线可对程序存储器进行访问。当执行某一指令时，下一指令被预先从程序存储器中取回。这使得指令可以在每一个时钟周期内被执行。芯片内的程序存储器为系统内可下载的 Flash 存储器。

通过相关的转移和调用指令，全部的 4K 地址空间可以被直接访问到。所有 AVR 指令均有一个单一的 16 位字格式，这意味着每个程序存储器地址包含了一个单一的 16 位或 32 位指令。图 2.5 为程序存储器。

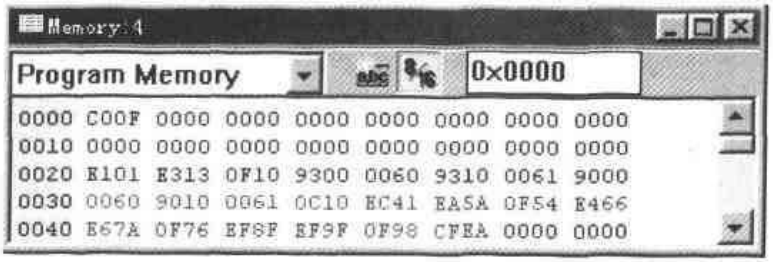


图 2.5 程序存储器

在中断和子程序调用过程中，返回地址程序计数器(PC)被存储于堆栈之中。该堆栈被高效率地放置在通用 SRAM 中，作为结果，堆栈的大小只被 SRAM 的大小及对 SRAM 的使用而限制。所有的程序必须在复位状态初始化 SP(在子程序和中断程序被执行之前，因为复位后，SP 为 0000H)。16 位的堆栈指针 SP 可在 I/O 空间之中被进行读/写访问。

128～4K 字节的数据 SRAM 可以通过 AVR 结构中的不同寻址模式很容易地被访问。

AVR 结构中的存储器空间均为线性和有规律的存储器映射。图 2.6 为存储器映射图。

一个灵活的中断模块，在 I/O 空间中有它自己的控制寄存器，并且在状态寄存器中带有一个附加的全局中断触发位。所有不同的中断，在程序存储器开始位置的中断向量(对应固定中断入口地址及中断申请名称)表中，带有一个独立分开的中断向量。不同的中断优先级与中断向量位置相一致。中断地址向量的位置越低，优先级越高(中断向量控制方式同 MCS－51 单片机)。

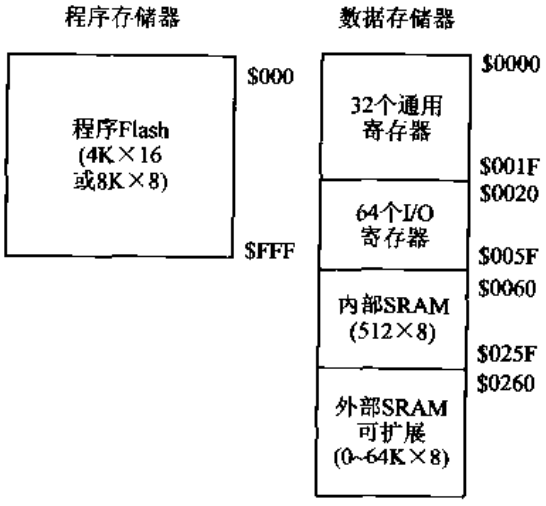


图 2.6 存储器映射图

2.2.2　通用寄存器堆

图 2.7 为 AVR 单片机 AT90S8515 CPU 中 32 个通用寄存器的结构图。

指令集中所有的寄存器操作指令均带有方向，并能在单一周期中访问所有的寄存器。例外的是存在于常量和寄存器之间的 5 个常量运算和逻辑指令：SBCI（带 C 减立即数）、SUBI（减立即数）、CPI（与立即数比较）、ANDI（与立即数）、ORI（或立即数）以及直接装入立即数的 LDI 指令。这些指令适用于寄存器堆中后半部分的寄存器（R16～R31）。通用的 SBC（带进位减）、SUB（减法）、CP（比较）、AND（与）、OR（或）指令，以及 2 个寄存器之间的操作指令，或单一寄存器操作指令，在整个寄存器堆中（R0～R31）都是适用的。

如图 2.7 所示，每个寄存器被分配给一个数据存储器地址，将其直接映射到数据空间的前 32 个地址。虽然寄存器堆并未像 SRAM 定位那样提供物理地址，但寄存器 X、Y、Z 可被设置用来对寄存器堆中的寄存器做索引。这种存储器的组成结构在访问寄存器时具有极大的灵活性。

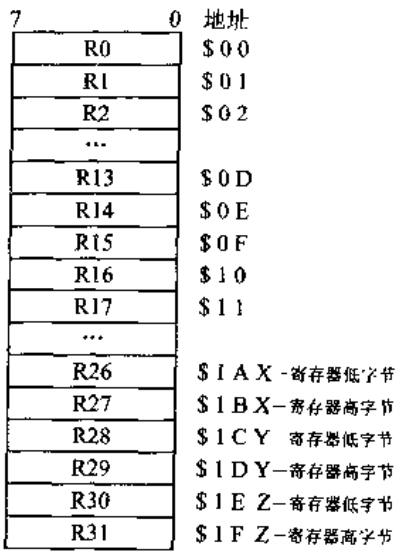


图 2.7　AVR 单片机 CPU 通用寄存器

2.2.3　X、Y、Z 寄存器

为了实现通用目的，寄存器 R26～R31 具有一些新加的功能。这些寄存器作为对数据空间间接寻址的地址指针。这 3 个间接寄存器 X、Y、Z 由图 2.8 定义。在不同的寻址模式下，这些地址寄存器的功能为固定位移、自动增量和减量（请参考不同的指令）。

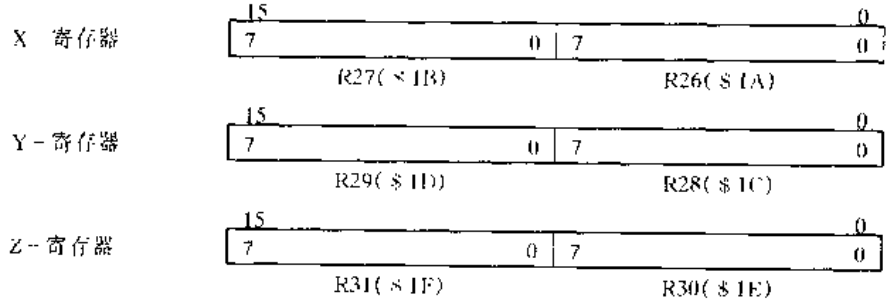


图 2.8　X、Y、Z 寄存器

2.2.4　ALU 运算逻辑单元

高性能的 AVR 运算逻辑单元的操作与所有的 32 个通用寄存器保持直接的连接。在单一时钟周期内，ALU 是在寄存器堆中的寄存器之间执行操作。ALU 操作被分为 3 个主要的目的：算法、逻辑和位功能。AVR 产品家族中 ATmega161/L 等微控制器的 ALU 运算部件中，有硬件乘法器。

2.3 AVR 单片机存储器组织

2.3.1 可下载的 Flash 程序存储器

AT90 系列单片机包括 1K～ 128K 字节的片内可下载 Flash 存储器。由于所有指令为 16 位字或 32 位双字，用于存储程序的 Flash 的结构为(512～64K 字)×16。Flash 存储器的使用寿命最少为 1 000 次写/擦循环。AT90S8515 单片机程序计数器宽为 12 位，以此来对 4K 个程序存储器地址寻址。

常量表必须被设定在 0～ 4K 的地址之间(请参考 LPM· 装载程序存储器指令说明)。

2.3.2 内部和外部的 SRAM 数据存储器

图 2.9 为 AT90S8515 单片机的数据存储器组织。低端 608 个数据存储器地址编址为寄存器堆、I/O 存储器和内部数据 SRAM。前 96 个地址编址为寄存器堆 + I/O 存储器，接下来的 512 个地址是内部数据 SRAM。可选的外部数据 SRAM 可以放置在同一个 SRAM 空间中，该 SRAM 将占据内部 SRAM 之后的地址直到 64K－1，这由 SRAM 的大小来定。

图 2.9 SRAM 组织

当访问超出内部数据 SRAM 地址的数据存储器空间时就访问外部数据 SRAM，使用与访问内部数据 SRAM 同样的指令。当访问内部的 SRAM 时，读写控制信号($\overline{RD}$和$\overline{WR}$)在整个访问周期中是非触发的，外部的 SRAM 的操作通过设置 MCUCR 寄存器的 SRE 位来触发，详见后面 MCU 控制寄存器的说明。

访问外部 SRAM 存储器比访问内部 SRAM 存储器多用 1 个时钟周期。使用命令 LD、ST、LDS、STS、PUSH、POP 能访问外部 SRAM 存储器。如果堆栈被放置在外部 SRAM 中，则中断程序调用和返回指令将多用 2 个时钟周期。当使用带有等待状态的外部 SRAM 时，外部的 SRAM 将额外花费 4 个时钟周期。

对数据存储器的 5 个不同寻址模式为：直接、带位移的间接、间接、带预减量的间接和带后增量的间接寻址。在寄存器堆中，寄存器 R26～R31 具有间接寻址指针寄存器的特性。间接寻址到达数据地址空间的尽头。带位移的间接寻址模式的特性为，它可到达由寄存器 Y 和 Z

给出的基本地址的 63 个地址位。当使用自动预减量和后增量的间接寻址模式时，地址寄存器 X、Y 和 Z 被使用，或被增大、减小。

32 个通用工作寄存器、64 个 I/O 存储器以及 AT90S8515 单片机中的 512 字节数据 SRAM，可通过所有这些地址模式被直接访问到。

2.3.3　EEPROM 数据存储器

AT90 系列单片机包括 64～4K 字节的 EEPROM 存储器。它被组织为一个分开的数据空间，这个数据空间用单字节可被读写。EEPROM 的使用寿命至少为 100 000 次写/擦循环。在 EEPROM 和 CPU 之间的访问详见后面对 EEPROM 的地址寄存器、数据寄存器和控制寄存器的说明。

2.3.4　存储器访问和指令执行时序

本节说明 AT90 系列单片机指令执行和内部存储器访问的时序。

AVR CPU 由系统时钟 Φ 驱动，直接由芯片的外部时钟晶振触发，没有使用内部时钟分频。

图 2.10 所示为 Harward 结构和快速访问寄存器堆概念触发的并行指令存取和指令执行时序。这种基本的流水线概念，目的是为了获得高达每 1 MIPS/MHz 的效率。

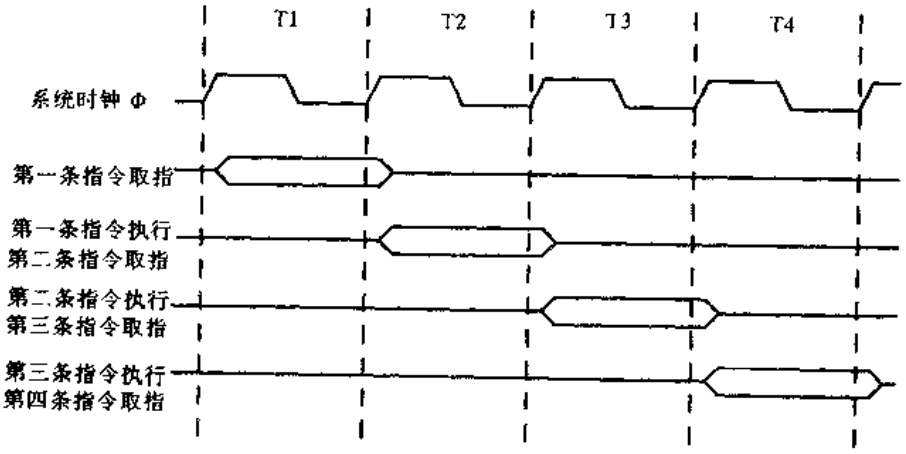


图 2.10　并行指令存取和指令执行时序

图 2.11 所示为寄存器堆的内部定时概念。在单一时钟周期内，使用 2 个寄存器操作数的 1 个 ALU 操作被执行，而其结果被存储回目的寄存器。

图 2.12 所示为在 2 个系统时钟周期内，完成内部数据 SRAM 的访问。

图 2.13 所示为在 2 个系统时钟周期内，完成外部数据 SRAM 的访问。

图 2.14 所示为含有等待状态的(等待状态触发)外部数据 SRAM 的访问。

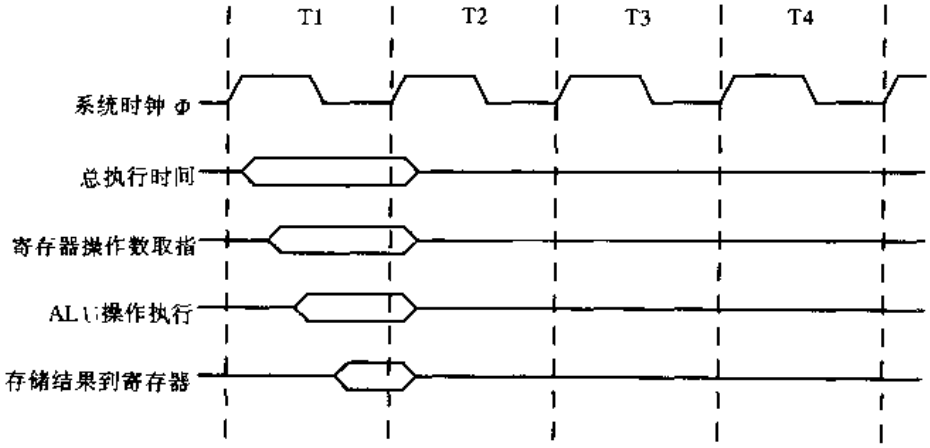


图 2.11 单周期 ALU 操作

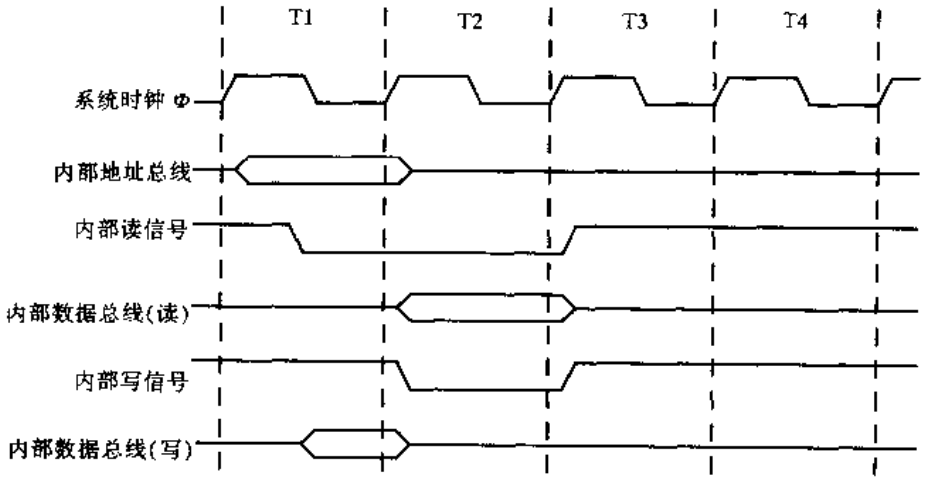


图 2.12 内部数据 SRAM 访问周期

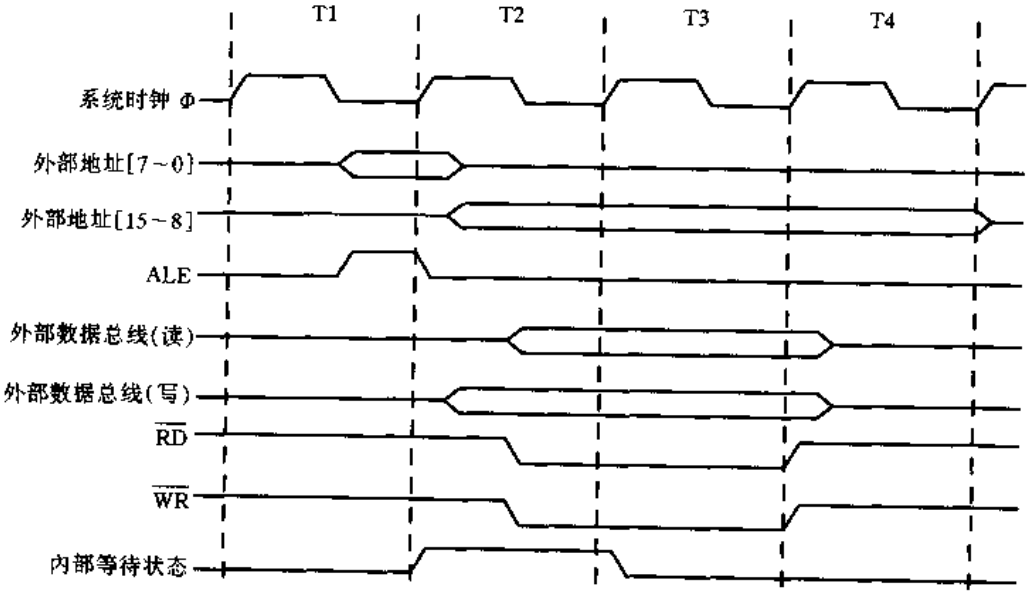


图 2.13 无等待状态的外部数据 SRAM 访问周期

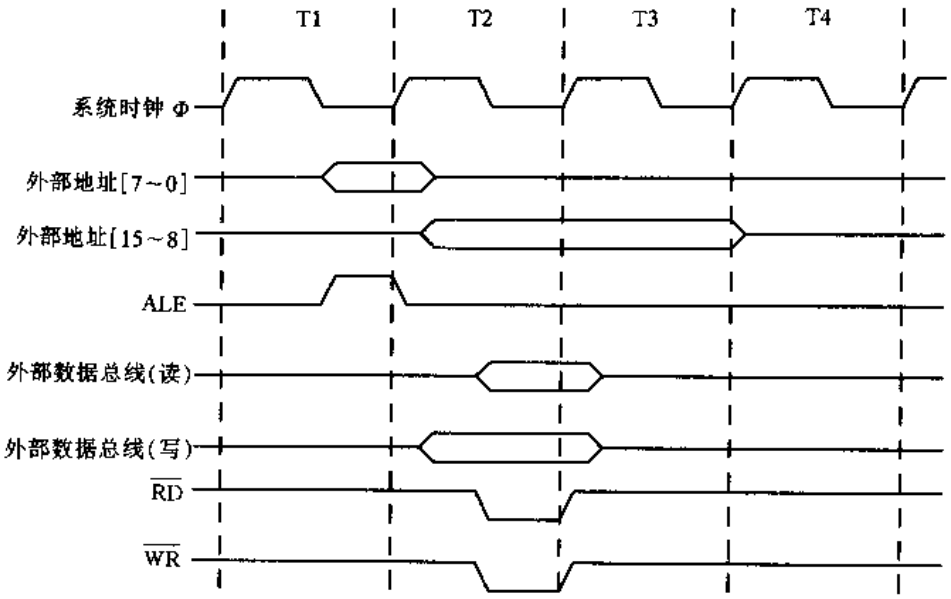


图 2.14　有等待状态的外部数据 SRAM 访问周期

2.3.5　I/O 存储器

在编写源文件时一定要写该器件的配置文件，作为源文件的文件头。如选用 AT90S8515 单片机，源文件的文件头为：

```
.include"8515def.inc"          ;文件头就是该器件的 I/O 存储器及位地址的定义文件，汇编时用到它
```

可查看光盘中"\AVR\asmpack\appnotes*.inc"文件。

表 2.1 所列为 AT90S8515 单片机的 I/O 空间定义。

表 2.1　AT90S8515 I/O 空间

十六进制地址	名　称	功　能
$ 3F($ 5F)	SREG	状态寄存器
$ 3E($ 5E)	SPH	堆栈指针高位
$ 3D($ 5D)	SPL	堆栈指针低位
$ 3B($ 5B)	GIMSK	通用中断屏蔽寄存器
$ 3A($ 5A)	GIFR	通用中断标志寄存器
$ 39($ 59)	TIMSK	定时器/计数器中断屏蔽寄存器
$ 38($ 58)	TIFR	定时器/计数器中断标志寄存器
$ 35($ 55)	MCUCR	MCU 通用控制寄存器
$ 33($ 53)	TCCR0	定时器/计数器 0 控制寄存器
$ 32($ 52)	TCNT0	定时器/计数器 0(8 位)
$ 2F($ 4F)	TCCR1A	定时器/计数器 1 控制寄存器 A
$ 2E($ 4E)	TCCR1B	定时器/计数器 1 控制寄存器 B
$ 2D($ 4D)	TCNT1H	定时器/计数器 1 高字节
$ 2C($ 4C)	TCNT1L	定时器/计数器 1 低字节
$ 2B($ 4B)	OCR1AH	定时器/计数器 1 输出比较寄存器 A 高字节

表 2.1(续)

十六进制地址	名 称	功 能
$ 2A($ 4A)	OCR1AL	定时器/计数器 1 输出比较寄存器 A 低字节
$ 29($ 49)	OCR1BH	定时器/计数器 1 输出比较寄存器 B 高字节
$ 28($ 48)	OCR1BL	定时器/计数器 1 输出比较寄存器 B 低字节
$ 25($ 45)	ICR1H	T/C1 输入捕获寄存器高字节
$ 24($ 44)	ICR1L	T/C1 输入捕获寄存器低字节
$ 21($ 41)	WDTCR	看门狗定时控制寄存器
$ 1F($ 3F)	EEARH	EEPROM 地址寄存器高字节
$ 1E($ 3E)	EEARL	EEPROM 地址寄存器低字节
$ 1D($ 3D)	EEDR	EEPROM 数据寄存器
$ 1C($ 3C)	EECR	EEPROM 控制寄存器
$ 1B($ 3B)	PORTA	A 口数据寄存器
$ 1A($ 3A)	DDRA	A 口数据方向寄存器
$ 19($ 39)	PINA	A 口输入脚
$ 18($ 38)	PORTB	B 口数据寄存器
$ 17($ 37)	DDRB	B 口数据方向寄存器
$ 16($ 36)	PINB	B 口输入脚
$ 15($ 35)	PORTC	C 口数据寄存器
$ 14($ 34)	DDRC	C 口数据方向寄存器
$ 13($ 33)	PINC	C 口输入脚
$ 12($ 32)	PORTD	D 口数据寄存器
$ 11($ 31)	DDRD	D 口数据方向寄存器
$ 10($ 30)	PIND	D 口输入脚
$ 0F($ 2F)	SPDR	SPI I/O 数据寄存器
$ 0E($ 2E)	SPSR	SPI 状态寄存器
$ 0D($ 2D)	SPCR	SPI 控制寄存器
$ 0C($ 2C)	UDR	UART I/O 数据寄存器
$ 0B($ 2B)	USR	UART 状态寄存器
$ 0A($ 2A)	UCR	UART 控制寄存器
$ 09($ 29)	UBRR	UART 波特率寄存器
$ 08($ 28)	ACSR	模拟比较控制和状态寄存器

注意:保留的和未用到的地址未列。

希望用户打开器件配置文件(*.inc)查看一下,防止没有器件配置文件头,汇编时出错。有了器件配置文件头,在编写源程序时就不必重复定义 I/O 口及位地址等。

AT90 系列单片机所有不同的 I/O 口和外围设备均在 I/O 空间中已设置好。不同的 I/O 地址可以通过 IN 和 OUT 指令访问,这些指令在 32 个通用寄存器与 I/O 空间之间传输数据。地址范围 $00～$1F 之间的寄存器可通过 SBI(I/O 寄存器置 $FF)和 CBI(I/O 寄存器清零)指令直接一位一位地访问。使用 SBIS(寄存器位置 1)和 SBIC(寄存器位清 0)指令对这些寄存器中的每一位值进行检验。请参考指令系统一章。

当使用 I/O 特定的命令时,必须使用 IN(I/O 口输入)、OUT(输出到 I/O 口)、SBIS(I/O 位置位跳行)和 I/O 地址的 $00～$3F。当寻址的 I/O 存储器为 SRAM 时,必须向该地址加入 $20。本资料中的全部 I/O 存储器地址均带有列在圆括号中的 SRAM 地址。以下部分说明了不同 I/O 和外围设备的控制寄存器。

一、状态寄存器——SREG

AVR 的状态寄存器 SREG 在 I/O 空间的地址为 $3F($5F),定义如下:

位	7	6	5	4	3	2	1	0	
$3F($5F)	I	T	H	S	V	N	Z	C	SREG
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:	$00								

状态寄存器 SREG 的作用很大,不同位代表不同的含义。对某一位的操作,置 1 或清 0,反映运算、操作结果状态,有置 1、清 0、为 1 转移、为 0 转移等。状态寄存器 SREG 有 36 条指令供使用。

位 7 I:全局中断触发

全局中断触发位必须被设置(1),以便允许中断。在这之后,单独的中断触发控制在中断屏蔽寄存器 GIMSK/TIMSK 中完成。如果全局中断触发寄存器被清空(0),所有中断被禁止,GIMSK/TIMSK 值独立存在。在中断发生后,I 位由硬件清除,并由 RETI(中断返回)指令设置,从而允许子序列的中断。

位 6—T:位复制存储

位复制指令 BLD(SREG 中 T 标志到寄存器某位)和 BST(寄存器某位到 SREG 中 T 标志)使用 T 位作为源操作位和目标操作位。寄存器堆中某一寄存器的某一位可以通过 BST 指令被复制到 T,用 BLD 指令则可将 T 中的位值复制到寄存器堆中的某一寄存器的某一位。

位 5—H:半进位标志位

半进位标志位 H 指示了在一些运算操作过程中的半进位(低四位向高四位进位),请参考指令集说明。

位 4—S:标志位,S=N+V

S 位是负数标志位 N 和 2 的补码溢出标志位 V,两者异或值,请参考指令集说明。

位 3—V:2 补码溢出标志位

2 的补码溢出标志位 V,支持 2 的补码运算,请参考指令集说明。

位 2—N:负数标志位

负数标志位 N 指示在不同的运算和逻辑操作之后的负数结果,请参考指令集说明。

位 1—Z:零值标志位

零值标志位 Z 指示在不同的运算和逻辑操作之后的零值结果,请参考指令集说明。

位 0—C:进位标志位

进位标志位 C 指示在某一运算和逻辑操作中的某一进位,请参考指令集说明。

二、堆栈指针——SP

在 I/O 地址 $3E($5E)和 $3D($5D)的两个 8 位寄存器构成了 AT90 系列单片机的 16 位堆栈指针。由于 AT90S8515 单片机支持 64K 字节的 SRAM,所以所有 16 位都被使用。

位	15	14	13	12	11	10	9	8	
$3E($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:	$00 00								

因为复位后堆栈为 SPH=$00,SPL=$00,AVR 单片机堆栈是-1 或-2 进栈,所以主程

序一开始，堆栈指针必须设在 SRAM 最高处，如 AT90S8515 的堆栈设在 SRAM 的 $025F 处。堆栈有自动进栈、出栈(调用指令、中断指令，子程序返回指令 RET，中断返回指令 RETI)和人工进栈、出栈指令，有压栈(PUSH)、出栈(POP)指令。

堆栈指针指示了数据 SRAM 堆栈区域，子程序和中断堆栈被放置在该区域中。在数据 SRAM 中的该堆栈空间必须在执行任何子程序调用或中断触发之前被程序定义。当执行 PUSH 指令，数据被压入堆栈时，堆栈指针减少 1；当执行子程序 CALL 和中断而将数据压入堆栈时，堆栈指针减少 2；当执行 POP 指令而数据从堆栈弹出时，堆栈指针增大 1；当从子程序 RET 返回或从中断 RETI 返回、数据从堆栈弹出时，堆栈指针增大 2。

2.4 AVR 单片机系统复位

AT90 系列单片机提供多种不同的中断源。这些中断和与之分开的复位向量均在程序存储器空间中各自带有分开的程序向量地址。所有中断均分配给单独的触发位，这些触发位须与状态寄存器中的 I 位一起被设为 1，以便能执行中断。

程序存储器空间中最低的地址被自动地定义为复位和中断向量。如表 2.2 所列，表中还列出了不同中断的优先级。地址越低，优先级越高。$\overline{\text{RESET}}$有最高的优先级，以下依次为 INT0、INT1……。

表 2.2 复位和中断向量

向量号	程序地址	源	中断定义
1	$ 000	$\overline{\text{RESET}}$	硬件脚和看门狗复位
2	$ 001	INT0	外部中断请求 0
3	$ 002	INT1	外部中断请求 1
4	$ 003	TIMER1 CAPT	定时器/计数器 1 捕获事件
5	$ 004	TIMER1 COMPA	定时器/计数器 1 比较匹配 A
6	$ 005	TIMER1 COMPB	定时器/计数器 1 比较匹配 B
7	$ 006	TIMER1 OVF	定时器/计数器 1 溢出
8	$ 007	TIMER0 OVF	定时器/计数器 0 溢出
9	$ 008	SPI,STC	串行传送完成
10	$ 009	UART,RX	UART,RX 完成
11	$ 00A	UART,UDRE	UART 数据寄存器空
12	$ 00B	UART,TX	UART,TX 完成
13	$ 00C	ANA_COMP	模拟比较器

以下为复位和中断向量地址的典型和通用的程序设置：

```
地 址      标 号   代 码          注  释
$ 000      RJMP    RESET          ; 复位处理
$ 001      RJMP    EXT_INT0       ; 转 INT0 处理
$ 002      RJMP    EXT_INT1       ; 转 INT1 处理
$ 003      RJMP    TIM1_CAPT      ; 转定时器 1 捕获处理
```

```
$ 004          RJMP   TIM1_COMPA      ; 转定时器 1 比较 A 处理
$ 005          RJMP   TIM1_COMPB      ; 转定时器 1 比较 B 处理
$ 006          RJMP   TIM1_OVF        ; 转定时器 1 溢出处理
$ 007          RJMP   TIM0_OVF        ; 转定时器 0 溢出处理
$ 008          RJMP   SPI_HANDLE      ; 转 SPI TX 处理
$ 009          RJMP   UART_RXC        ; 转 UART,RX 完成处理
$ 00A          RJMP   UART_DRE        ; 转 UART,UDR 空处理
$ 00B          RJMP   UART_TXC        ; 转 UART,TX 完成处理
$ 00C          RJMP   ANA_COMP        ; 转模拟比较器处理
;不用的中断入口地址写上 RETI,有抗干扰作用
$ 010  MAIN:  <instr>XXX              ;主程序开始,主程序必须跳过中断区
...            ...      ...
```

2.4.1　复位源

AT90 系列单片机有 3 个复位源：

- 加电复位。当供电电平加至 V_{CC} 和 GND 引脚时，MCU 进行复位。
- 外部复位。当一个低电平加到$\overline{RESET}$引脚多于 2 个 XTAL 周期时，MCU 进行复位。
- 看门狗复位。当看门狗定时器超时，且看门狗为触发时，MCU 进行复位。

在复位过程中，所有的 I/O 寄存器被设为初始值，程序从地址 $0000 开始执行。$0000 地址中放置的指令须为某一 RJMP——相对转移，即到达复位处理路径的指令。若程序从没有对中断源触发，则中断向量无法使用，正常的程序代码可以放置在这些地址中。图 2.15 的电路图说明了复位逻辑。表 2.3 定义了复位电路的时序和电参数。

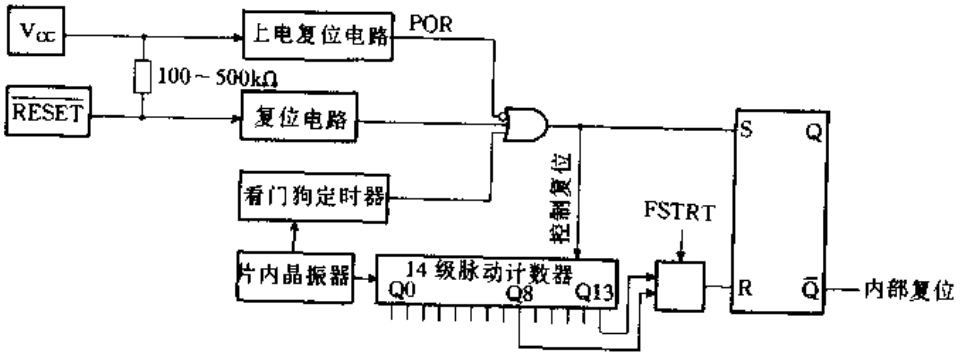


图 2.15　复位逻辑

表 2.3　复位特性(V_{CC}=5.0 V)

符　号	参　数	最小值	典型值	最大值	单　位
V_{POT}	上电复位门限电压	1.8	2	2.2	V
V_{RST}	复位脚门限电压		$V_{CC}/2$		V
t_{POR}	上电复位周期	2	3	4	ms
t_{TOUT}	复位延时定时输出，周期 FSTRT 不编程	11	16	21	ms
t_{TOUT}	复位延时定时输出，周期 FSTRT 编程	1.0	1.1	1.2	ms

2.4.2 加电复位

加电复位(POR)电路确保了只有当 V_{CC} 达到一个安全电平时，器件才开始工作。如图 2.16所示，当看门狗定时器晶振对内部定时器定时时，不启动 MCU，直到 V_{CC} 到达 Power_on 门槛电压 V_{POT} 一定时间之后才启动 MCU(如图 2.17 和 2.18 所示)。全部的复位时间为 Power_on 复位时间 t_{POR} 加上延时时间 t_{TOUT}。

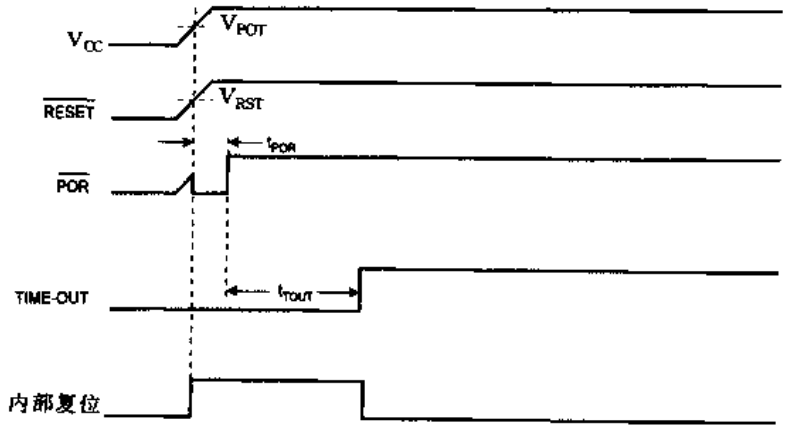


图 2.16 MCU 启动，$\overline{RESET}$ 连或不连到 V_{CC}，V_{CC} 快速上升

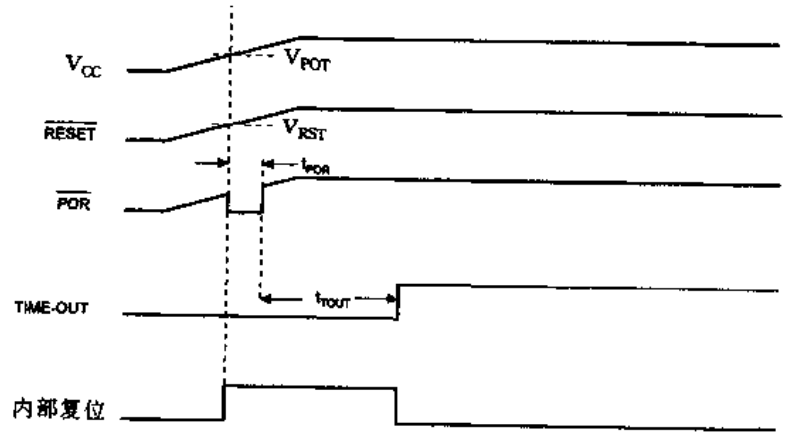


图 2.17 MCU 启动，$\overline{RESET}$ 连或不连到 V_{CC}，V_{CC} 慢速上升

如果使用了陶瓷谐振器或其它快速启动的晶振来为 CPU 定时，Flash 中的 FSTRT 熔丝位可被编程，给出一个更短的启动时间。由于片内电阻将 $\overline{RESET}$ 拉高，在无需外部复位时，引脚不连接。将 $\overline{RESET}$ 与 V_{CC} 连接，则产生同样效果。在对 V_{CC} 加电一段时间后，通过将 $\overline{RESET}$ 引脚拉低，加电复位时间可以被延长。请参考图 2.18 的时序例子。

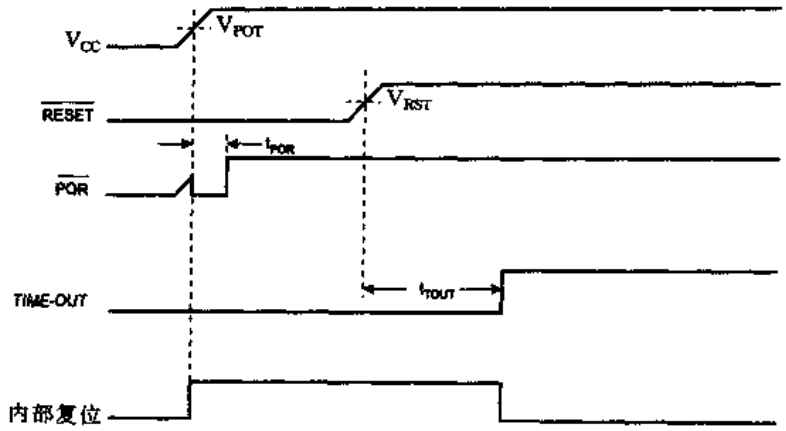


图 2.18 MCU 启动，RESET由外部控制

2.4.3 外部复位

RESET复位引脚上的低电压引发外部复位。该引脚必须被拉低至少 2 个晶振时钟周期，在 t_{TOUT} 周期结束后，当在其正边缘达到复位门槛电压时，延迟定时器启动 MCU。图 2.19 为操作期间由外部复位的复位脉冲。

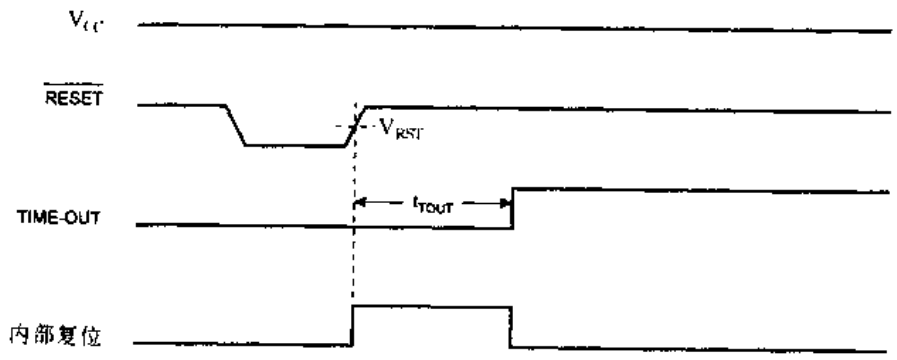


图 2.19 在操作期间由外部复位

2.4.4 看门狗复位

看门狗定时输出时，它将产生 1 个 XTAL 的短暂复位脉冲，在此脉冲的下降沿，延迟定时器开始对超时时间 t_{TOUT} 计数。图 2.20 为在操作期间由看门狗复位的复位脉冲。

使用 AVR 的 EEPROM 注意 3 点：①避开零地址；②程序在做初始化的时候不要读写 EEPROM，在进入读写之前确保有数毫秒的延时，即等机器完全稳定以后才能操作 EEPROM；③最安全方法加复位电压检测器件（附图见 68 页图 2.56 复位检测器件引脚接线图）。

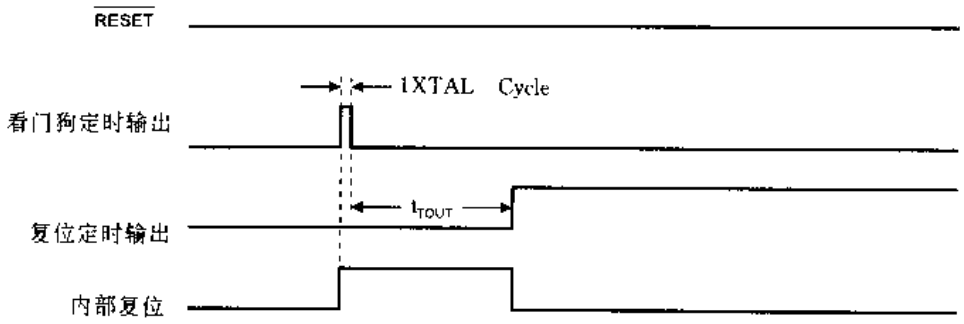


图 2.20 在操作期间由看门狗复位

2.5 AVR 单片机中断系统

2.5.1 中断处理

AT90 系列单片机有 2 个 8 位中断屏蔽控制寄存器，即 GIMSK ——通用中断屏蔽寄存器和 TIMSK-- -定时器/计数器中断屏蔽寄存器。

当某一中断发生时，全局中断触发位 I 位被清空(0)，则全部中断被禁止。用户软件可以将 I 位置 1 对中断触发。当执行 RETI，从中断指令返回时，I 位被设为 1。

由于维持静态的事件引发中断(即输出比较寄存器 1 与定时器/计数器 1 的值相匹配)，当事件发生时，中断标志被设置为 1。若中断位被清空，且中断条件维持原状，标志直到下次事件发生时才被设置为 1。

当程序计数器指向现行中断时，执行中断处理程序，硬件将相应地产生中断标志位清空。

一、通用中断屏蔽寄存器——GIMSK

位	7	6	5	4	3	2	1	0	
$3B($5B)	INT1	INT0	—	—	·	—	—		GIMSK
读/写：	R/W	R/W	R	R	R	R	R	R	
初始化值： $00									

位 7 —INT1：外部中断请求 1 触发

当 INT1 被置为 1，且状态寄存器(SREG)中 I 位被置为 1 时，外部引脚中断被触发。MCU 通用控制寄存器(MCUCR)中的中断检测控制 1 位 1/0(ISC11 和 ISC10)定义的外部中断，在 INT1 引脚上是在电平还是在上升或下降沿被触发。即使 INT1 被定义为输出，引脚上的事件也将引发一个中断请求。外部中断请求 1 的中断从程序存储器地址 $002 中被执行。

位 6—INT0：外部中断请求 0 触发

当 INT0 被设为 1，且状态寄存器(SREG)中 I 位被设为 1 时，外部引脚中断被触发。MCU 通用控制寄存器(MCUCR)中的中断检测控制 0 位 1/0(ISC01 和 ISC00)定义的外部中断，在 INT0 引脚上是在电平还是在上升或下降沿被触发。即使 INT0 被定义为输出，引脚上

的事件也将引发一个中断请求。外部中断请求0的中断从程序存储器地址$001中被执行。

位5～0 -Res：保留位

AT90系列单片机的该位为保留位，总读0。

二、通用中断标志寄存器——GIFR

位	7	6	5	4	3	2	1	0	
$3A($5A)	INTF1	INTF0	—	—	—	—	—	—	GIFR
读/写：	R/W	R/W	R	R	R	R	R	R	
初始化值：	$00								

位7—INTF1：外部中断标志1

当INT1引脚上的某一事件触发了某一中断请求，INTF1置为1。若SREG中的I位以及GIMSK中的INT1位被置1，MCU将在地址$002向中断向量转移。当中断程序执行时，标志被清空。标志位还可以通过向其写入逻辑1来清空。

位6 INTF0：外部中断标志0

当INT0引脚上的某一事件触发了某一中断请求，INTF0变为置1。若SREG中的I以及GIMSK中的INT0位被设为1，MCU将在地址$001向中断向量转移。当中断程序执行时，标志被清空。标志位还可以通过向其写入逻辑1来清空。

位5～0—Res：保留位

AT90系列单片机的该位为保留位，总读0。

三、定时器/计数器中断屏蔽寄存器——TIMSK

位	7	6	5	4	3	2	1	0	
$39($59)	TOIE1	OCIE1A	OCIE1B	—	TICIE1	—	TOIE0	—	TIMSK
读/写：	R/W	R/W	R/W	R	R/W	R	R/W	R	
初始化值：	$00								

位7—TOIE1：定时器/计数器1溢出中断触发

当TOIE1被设为1，且状态寄存器中I位被设为1时，定时器/计数器1溢出中断触发。如定时器/计数器1产生溢出，相应的中断(在向量$006)被执行，在定时器/计数器中断标志寄存器TIFR中的溢出标志(定时器1)被设置为1。当定时器/计数器1处于PWM模式下，计数器在$0000变化计数方向时，定时器溢出标志被设置。

位6—OCIE1A：定时器/计数器1输出比较匹配A中断触发

当OCIE1A被设为1，且状态寄存器中的I位被设为1时，定时器/计数器1的比较匹配A中断触发。若发生了定时器/计数器1中的比较匹配A，则中断(在向量$ 004)被执行，在定时器/计数器中断标志寄存器TIFR中的定时器/计数器1中的比较A标志被设为1。

位5—OCIE1B：定时器/计数器1输出比较匹配B中断触发

当OCIE1B被设为1，且状态寄存器中的I位被设为1时，定时器/计数器1的比较匹配B中断触发。若发生了定时器/计数器1中的比较匹配B，则中断(在向量$005)被执行，在定时器/计数器中断标志寄存器TIFR中的定时器/计数器1中的比较B标志被设为1。

位4,2,0—Res：保留位

AT90系列单片机的该位为保留位，总读0。

位 3—TICIE1：定时器/计数器 1 输入捕获中断触发

当 TICIE1 被设为 1，且状态寄存器中的 I 位被设为 1 时，定时器/计数器 1 输入捕获事件中断触发。若在引脚 31，即 PD6(ICP)上发生捕获触发事件，则中断(在向量 $003)被执行，在定时器/计数器中断标志寄存器 TIFR 中的定时器/计数器 1 输入捕获标志被设置为 1。

位 1—TOIE0：定时器/计数器 0 溢出中断触发

当 TOIE0 被设为 1，且状态寄存器中的 I 位被设为 1 时，定时器/计数器 0 溢出中断触发。若在定时器/计数器 0 上发生溢出，则中断(在向量 $007)被执行，在定时器/计数器的中断标志寄存器 TIFR 中的溢出标志(定时器 0)被设置为 1。

四、定时器/计数器中断标志寄存器——TIFR

位	7	6	5	4	3	2	1	0	
$38($58)	TOV1	OCF1A	OCF1B	—	ICF1	—	TOV0	—	TIFR
读/写：	R/W	R/W	R/W	R	R/W	R	R/W	R	
初始化值：	$00								

位 7—TOV1：定时器/计数器 1 溢出标志

当定时器/计数器 1 中产生溢出时，TOV1 位被设为 1；当执行相应中断处理向量时，TOV1 被硬件复位。TOV1 可由写入一个逻辑 1 到标志位而被清除。当 SREG(状态寄存器)的 I 位和 TOIE1(定时器/计数器 1 溢出中断触发)以及 TOV1 被设为 1 时，定时器/计数器 1 的溢出中断被执行。在 PWM 模式下，当定时器/计数器 1 在 $0000 变化计数方向时，该位被设置为 1。

位 6—OCF1A：输出比较标志 1A

当 OCR1A，即输出比较寄存器 1A 中的数据与定时器/计数器 1 之间发生比较匹配时，OCF1A 被设为 1；当执行相应中断处理向量时，OCF1A 由硬件清除。OCF1A 亦可通过向标志写逻辑 1 清除。当 SREG 中的 I 位、OCIE1A(即定时器/计数器 1 比较匹配 A 中断触发)，以及 OCF1A 被设为 1 时，定时器/计数器 1 比较匹配中断触发被执行。

位 5—OCF1B：输出比较标志 1B

当 OCR1B，即输出比较寄存器 1B 中的数据与定时器/计数器 1 之间发生比较匹配时，OCF1B 被设为 1；当执行相应中断处理向量时，OCF1B 由硬件清除。OCF1B 亦可通过向标志写逻辑 1 来清除。当 SREG 中的 I 位、OCIE1B(即定时器/计数器 1 比较匹配 B 中断触发)，以及 OCF1B 被设为 1 时，定时器/计数器 1 比较匹配中断触发被执行。

位 4，2，0——Res：保留位

AT90 系列单片机的该位为保留位，总读 0。

位 3——ICF1：中断捕获标志 1

当一个输入捕获事件发生时，ICF1 位标志被设为 1。表明定时器/计数器 1 的值已被输送到输入捕获寄存器——ICR1。当执行相应中断处理向量时，ICF1 被硬件清除。ICF1 亦可通过向标志写逻辑 0 清除。

位 1——TOV0：定时器/计数器 0 位溢出标志位

当定时器/计数器 0 中产生溢出时，TOV0 位被设为 1；当执行相应中断处理向量时，TOV0 被硬件清除。TOV0 可写入一个逻辑 0 到标志位而被清除。当 SREG 的 I 位和 TOIE0

(定时器/计数器0溢出中断触发)以及TOV0被设为1时,定时器/计数器0的溢出中断被执行。

2.5.2 外部中断

外部中断由引脚INT1和INT0触发。请注意,在触发时,即使INT0和INT1引脚被设置为输出,亦会触发中断。这一特征可用来进行软件中断。外部中断可通过上升或下降边沿及低电平触发。这在MCU控制寄存器——MCUCR中已说明。当外部中断触发,且被设置为电平触发时,只要引脚保持低电平,中断将被触发。

外部中断的设置已在MCU控制寄存器——MCUCR的规格中被说明。

2.5.3 中断应答时间

AVR所有允许的中断执行应答最少为4个时钟周期。在4个时钟周期之后,用于现行中断控制程序的程序向量寻址被执行。在4个时钟周期中,程序计数器(2个字节)被压入堆栈,且堆栈指针被减量2。该向量为一个向中断程序的相关转移指令,需2个时钟周期。若在一个多周期指令的执行中发生中断,该指令在中断执行前结束。

从中断处理程序(如调用子程序)的返回需要4个时钟周期。在这4个时钟周期之中,程序计数器(2个字节)从堆栈中被弹出,且堆栈指针被增量2。当AVR从某一中断中出来时,它总是返回到主程序,并在任何挂起的中断执行前执行多于一条的指令。

注意:状态寄存器SREG不由AVR硬件处理,也不为中断或子程序工作。由于中断处理程序需要一个SREG的存储,这一切均由用户软件完成。

由事件发生的中断触发能保留状态,即当事件发生时,中断标志被设置。若中断标志已被清除,但中断条件持续,则在下次事件发生时,中断标志才被设置。

2.5.4 MCU控制寄存器MCUCR

MCUCR控制寄存器包含通用MCU功能的控制位。

位	7	6	5	4	3	2	1	0	
$35($55)	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
读/写:	R/W								
初始化值:	$00								

位7——SRE:外部SRAM触发

当SRE被设置为1时,外部的SRAM被触发,且引脚AD0~AD7(A口)、A8~A15(C口)、$\overline{RD}$和$\overline{WR}$(D口)作为第二功能被触发。SRE位覆盖任何数据方向寄存器的方向设置,详见"2.3.2 内部和外部的SRAM数据存储器",该部分描述了外部SRAM的引脚功能。当SRE位被清除时,外部数据SRAM被禁止,作为通常的引脚且用于数据方向的设置。

位6——SRW:外部SRWM等待状态

当SRW设置为1时,在外部的数据SRAM访问周期中插入1个周期的等待状态。当SRW被清0时,外部数据SRAM的访问使用正常的2个周期。详见"图2.13 无等待状态的外部数据SRAM访问周期"和"图2.14 有等待状态的外部数据SRAM访问周期"。

位5——SE:休眠触发

SE 位须设为 1，以便当 SLEEP 指令执行时，使 MCU 进入休眠模式。除非出于编程的目的，为防止 MCU 进入休眠模式，只在执行 SLEEP 指令之前，才设置休眠触发 SE 位。

位 4——SM：休眠模式

这一位可选择两种休眠模式。当 SM 被清为 0 时，空闲模式被选为休眠模式；当 SM 设为 1 时，掉电模式被选为休眠模式。请参考“2.6.1 休眠状态”。

位 3，2——ISC11，ISC10：中断检测控制 1，位 1 和位 0

如果 SREG 的 I 标志位和 GIMSK 寄存器中的相应中断屏蔽已被设置，则外部中断 1 被外部引脚 INT1 触发。由表 2.4 定义触发中断的外部 INT1 引脚上的是电平还是边沿。

位 1，0——ISC01，ISC00：中断检测控制 0，位 1 和位 0

如果 SREG 的 I 标志位和 GIMSK 寄存器中的相应中断屏蔽已被设置，则外部中断 0 被外部引脚 INT0 触发。由表 2.5 定义触发中断的外部 INT0 引脚上的是电平还是边沿。

表 2.4 中断 1 检测控制

ISC11	ISC10	说 明
0	0	INT1 的低电平产生一个中断请求
0	1	INT1 的高电平产生一个中断请求
1	0	INT1 的下降沿产生一个中断请求
1	1	INT1 的上升沿产生一个中断请求

表 2.5 中断 0 检测控制

ISC01	ISC00	说 明
0	0	INT0 的低电平产生一个中断请求
0	1	INT0 的高电平产生一个中断请求
1	0	INT0 的下降沿产生一个中断请求
1	1	INT0 的上升沿产生一个中断请求

注意：当改变 ISC10/ISC00 位时，INT0 须通过清除其在 GIMSK 寄存器中的中断触发位来加以禁止；否则，当该位改变时，一个中断就会产生。

2.6 AVR 单片机的省电方式

2.6.1 休眠状态

为了进入休眠状态，MCUCR 中的 SE 位被设为 1，且须执行一条 SLEEP 指令。当 MCU 在休眠模式下发生了一个允许的中断，MCU 唤醒，执行中断程序，并恢复 SLEEP 后面指令的执行。寄存器堆的内容和 I/O 存储器没有改变。若在休眠模式下发生复位，MCU 被唤醒，并从复位向量执行。

注意：若为了从掉电状态下唤醒 MCU 而使用了某一电平触发的中断，必须保持 16 ms 的低电平，即长于晶振启动的时间；否则，在 MCU 开始执行以前，中断标志位有可能返回零值。

2.6.2 空闲模式

当 SM 位被清 0，SLEEP 指令使 MCU 进入空闲状态，这时 CPU 停止工作，而定时器/计数器、看门狗以及中断系统继续工作。这使得 CPU 可以由外部触发的中断来唤醒，亦可由内部诸如定时器溢出中断和看门狗复位唤醒，不需要通过模拟比较器中断来唤醒 CPU。这样可通过设置模拟比较器的控制状态寄存器 ACSR 中的 ACD 位来对模拟比较器进行掉电。因此在空闲状态下减少了功耗。

2.6.3 掉电模式

当 SM 位被设为 1 时，SLEEP 指令使 MCU 进入掉电状态。在此模式下，外部晶振停止工作。用户可在掉电模式下选择看门狗是否触发。若看门狗为触发，当看门狗超过超时规定的时间时，它会唤醒 MCU；若看门狗为非触发，只有外部的复位或外部电平触发中断可唤醒 MCU。

2.7 AVR 单片机定时器/计数器

AT90S8515 单片机有 2 种通用定时器/计数器，即一个 8 位的定时器/计数器、一个 16 位的定时器/计数器。定时器/计数器从 10 位的预定比例定时器获取独立的预定比例区域。2 个定时器/计数器可被用作带内部时钟的时基定时器，或被用作带可触发计数的外部引脚连接的计数器。

2.7.1 定时器/计数器预定比例器

图 2.21 所示为通用定时器/计数器的预定比例器。

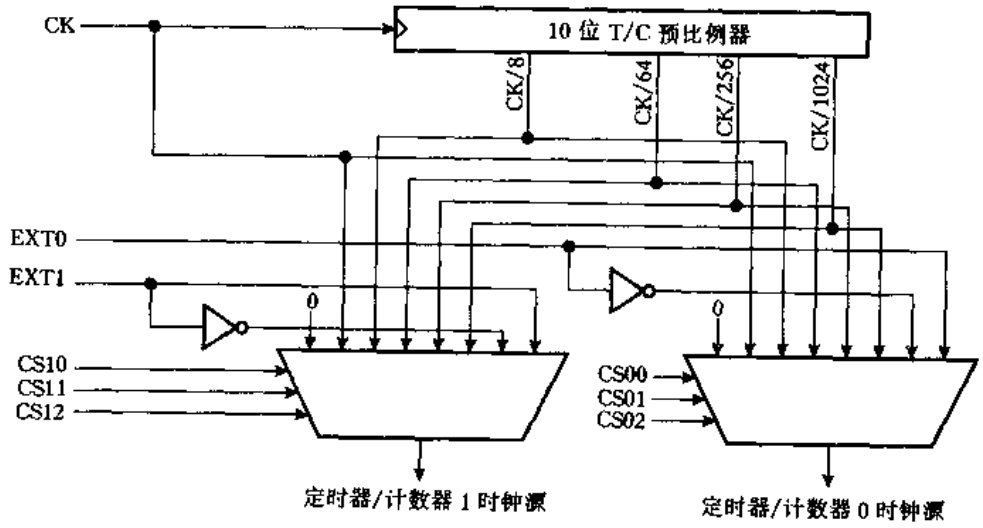


图 2.21 定时器/计数器预定比例器

4 个不同的预定比例部分为：CK/8、CK/64、CK/256 和 CK/1024，图中 CK 为晶振时钟。对于 2 个定时器/计数器，新加的部分为 CK。外部源和停止均可选为时钟源。

2.7.2 8 位定时器/计数器 0

图 2.22 所示为定时器/计数器 0 的方框图。

8 位定时器/计数器 0 可从 CK、预定比例器时钟源或外部引脚来选择时钟源。另外，它还可以像定时器/计数器 0 的控制寄存器——TCCR0 格式中所描述的那样停止。

溢出状态比例标志位在定时器/计数器中断比例标志位寄存器——TIFR 中。定时器/计数器 0 的中断触发/禁止设置在定时器/计数器中断的控制屏蔽寄存器——TIMSK 中。

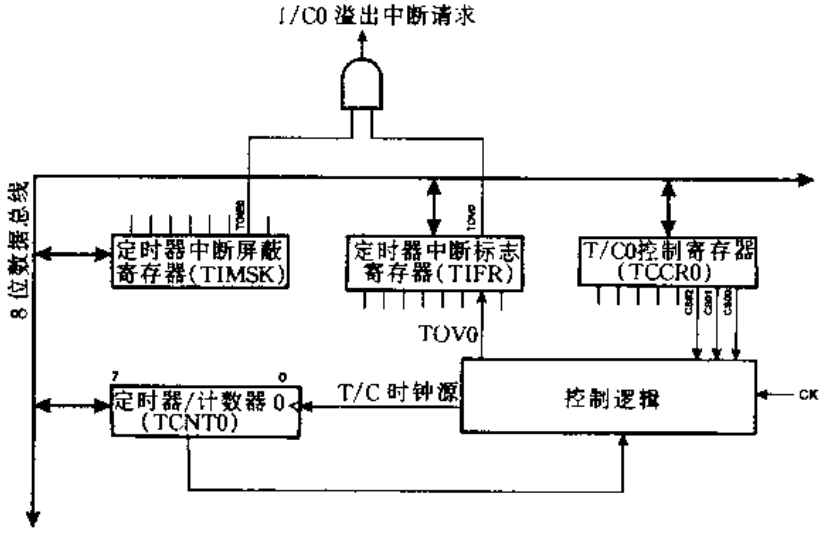


图 2.22 定时器/计数器 0 方框图

当定时器/计数器 0 被外部定时时，外部信号与 CPU 的振荡频率同步。为了确保外部时钟获取正确的采样，在两个外部时钟转换之间的最少时间必须维持一个内部 CPU 的时钟周期。外部时钟信号在内部 CPU 时钟的上升沿被采样。

8 位定时器/计数器 0 有机会用于更低的预定比例时，则具有高分辨率和高准确性的特性。类似地，高预定比例特性使得定时器/计数器 0 对低速功能，或不经常操作的精确定时功能很有用。

一、定时器/计数器 0 的控制寄存器——TCCR0

位	7	6	5	4	3	2	1	0	
$33($53)	-	-	—	—	—	CS02	CS01	CS00	TCCR0
读/写：	R	R	R	R	R	R/W	R/W	R/W	
初始化值：	$00								

位 7～3——Res：保留位

AT90 系列单片机的这些位为保留位，总读 0。

位 2、1、0——CS02，CS01，CS00：时钟选择 0 的位 2、1 和 0

时钟选择 0 的位 2、1 和 0 定义定时器 0 的预定比例源，见表 2.6。

表 2.6 时钟 0 预定选择

CS02	CS01	CS00	说 明	CS02	CS01	CS00	说 明
0	0	0	停止，定时器/计数器 0 被停止	1	0	0	CK/256
0	0	1	CK	1	0	1	CK/1 024
0	1	0	CK/8	1	1	0	外部 T0 脚，下降沿
0	1	1	CK/64	1	1	1	外部 T0 脚，上升沿

停止条件提供定时器触发/禁止功能。时钟分频的模式从晶振时钟直接换算。若使用外部引脚模式，相应设置必须在实际数据方向控制寄存器中完成。

二、定时器/计数器0——TCNT0

位	7	6	5	4	3	2	1	0	
$32($52)	MSB							LSB	TCNT0
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

定时器/计数器0是带读写访问的向上计数器。若定时器/计数器0被写入，同时时钟源正被执行，定时器/计数器0在写入操作之后继续计数。

2.7.3　16位定时器/计数器1

图2.23所示为定时器/计数器1的方框图。

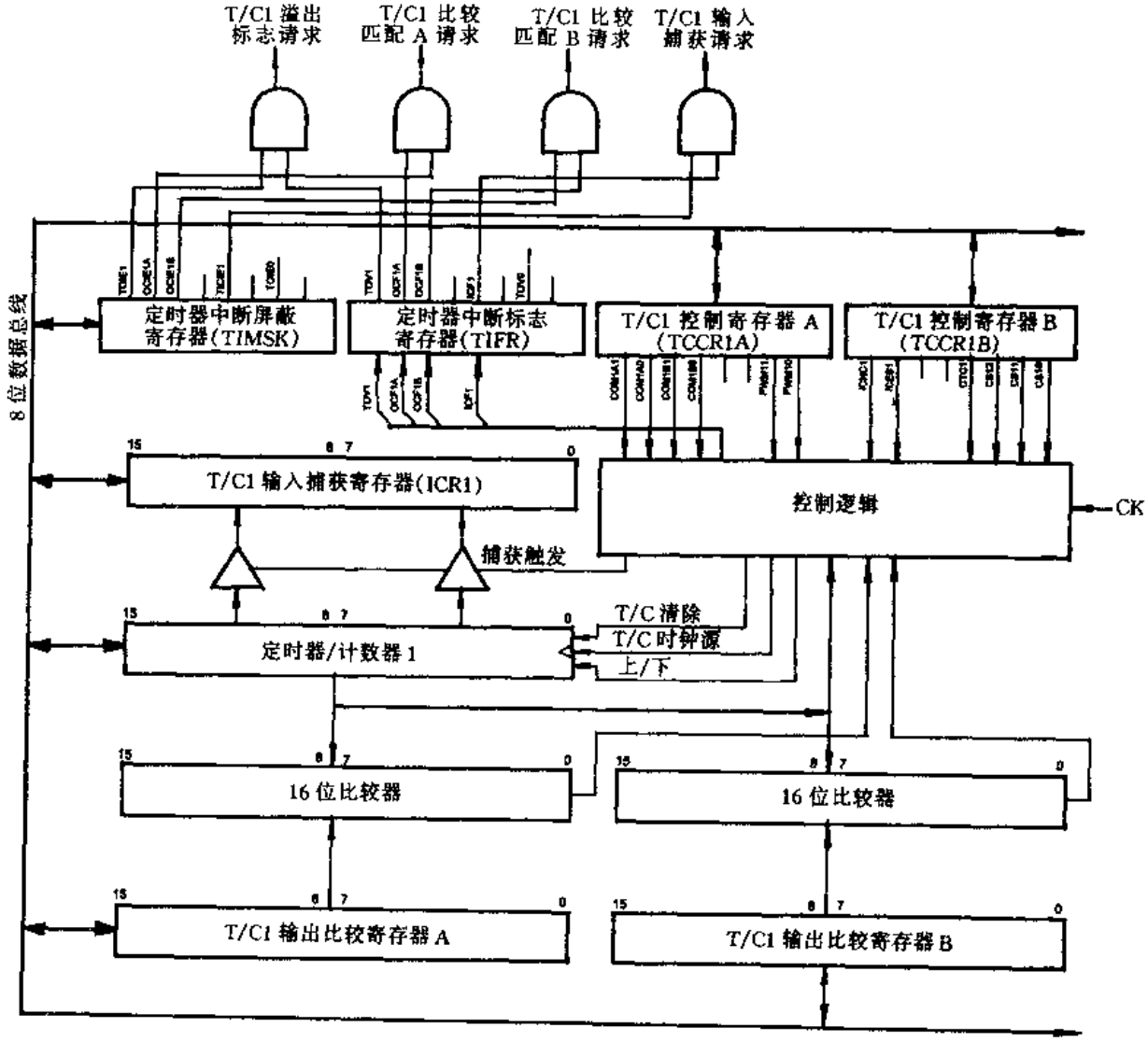


图2.23　定时器/计数器1方框图

16位定时器/计数器1可以从CK、预定比例器时钟源或外部引脚中选择时钟源。另外，它还可以像在定时器/计数器1控制寄存器TCCR1A、TCCR1B中描述的那样被停止。在定时器/计数器控制寄存器TCCR1A、TCCR1B中可以找到不同的状态标志(溢出、比较匹配以

及捕获事件)和控制信号。对定时器/计数器 1 的中断触发/禁止设置可以在定时器/计数器中断屏蔽寄存器——TIMSK 中找到。

当定时器/计数器 1 被外部定时时,外部信号与 CPU 振荡频率同步。为确保从外部时钟获取正确的采样,在 2 个外部时钟转换之间的最小时间至少为一个内部 CPU 时钟周期。外部时钟信号在内部 CPU 时钟的上升沿被采样。

16 位定时器/计数器 1 有机会用于更低的预定比例时,则具有高分辨率和高准确性的特性。类似地,高预定比例特性使得定时器/计数器 1 对低速功能,或不经常操作的精确定时功能很有用。

定时器/计数器 1 提供输出比较寄存器 1A 和 1B——OCR1A 和 OCR1B 的两个输出比较功能,作为与定时器/计数器 1 内容进行比较的数据源。输出比较功能包括比较 A 匹配的计数器可选择的清除,以及在比较匹配输出比较引脚上的操作。

定时器/计数器可被用作 8 位、9 位或 10 位脉冲调制器。在此模式下,定时器和 OCR1A/OCR1B 寄存器具有集中脉冲双抗误操作独立的 PWM 能力。

定时器/计数器的输入捕获功能提供了对定时器/计数器 1 向输入捕获寄存器 ICR1 内容的捕获,该捕获操作由在输入捕获引脚 ICP 上的外部事件触发。实际的捕获事件设置由定时器/计数器 1 的控制寄存器 TCCR1B 来定义。另外,模拟比较器可触发该输入捕获。请参照"模拟比较器"部分。ICP 的引脚逻辑如图 2.24 所示。定时器/计数器 1 的输入捕获噪声消除器示意图见图 2.25。

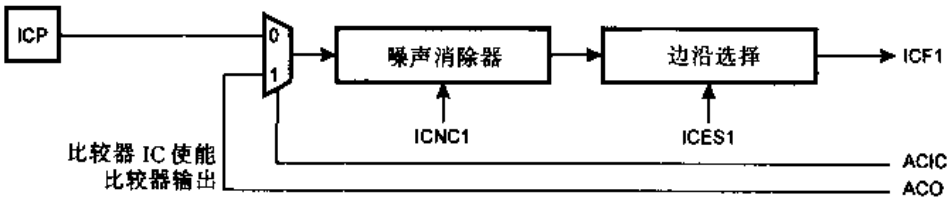


图 2.24 ICP 的引脚原理图

如果噪音清除器为触发,则捕获事件的实际触发条件在捕获被触发前受到多于 4 个采样的监测。输入引脚信号以 XTAL 的时钟频率被采样。

一、定时器/计数器 1 控制寄存器 A——TCCR1A

位	7	6	5	4	3	2	1	0	
$2F($4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	—	PWM11	PWM10	TCCR1A
读/写:	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
初始化值:	$00								

位 7,6— —COM1A1,COM1A0:比较输出模式 1,位 1 和 0

COM1A1 和 COM1A0 控制位决定了在定时器/计数器 1 中比较匹配之后的输出引脚事件。输出引脚事件影响 OC1A,即输出比较 A 引脚 1。由于这是对 I/O 口的可替换功能,相应的方向控制位必须设为 1,以便对输出引脚进行控制。控制设置如表 2.7 所示。

位 5,4——COM1B1,COM1B0:比较输出模式 1,位 1 和 0

COM1B1 和 COM1B0 控制位决定了在定时器/计数器 1 中比较匹配之后的输出引脚事件。输出引脚事件影响 OC1B,即输出比较 B 引脚 1。由于这是对 I/O 口的可替换功能,相应

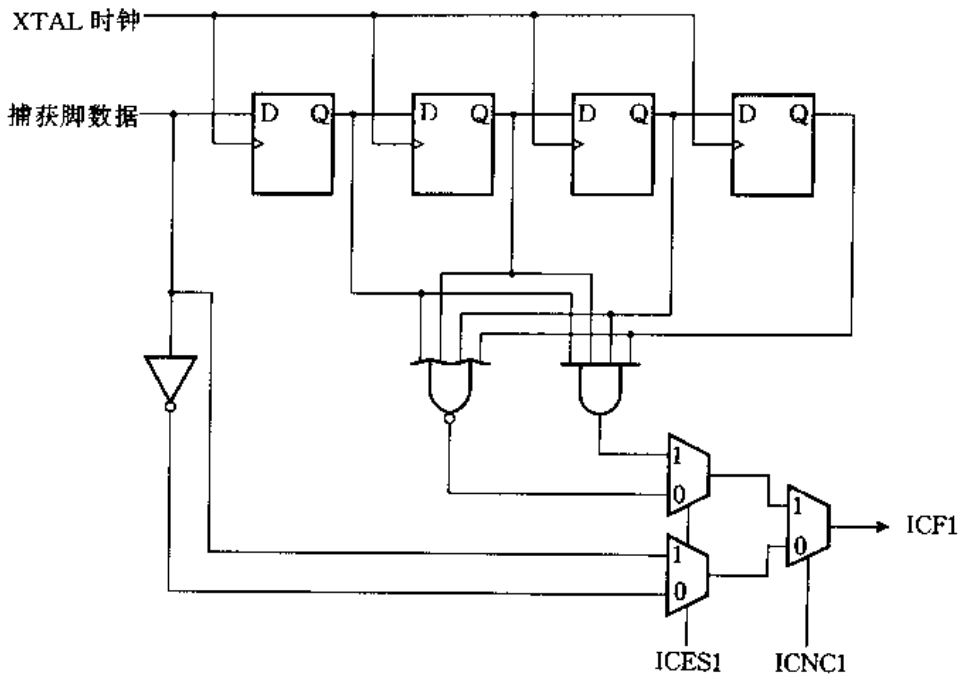


图 2.25 输入捕获噪声消除器示意图

的方向控制位必须设为 1，以便对输出引脚进行控制。控制设置如表 2.7 所示。

在 PWM 模式下，这些位有不同的功能，请参考表 2.11 的具体说明。

当变换 COM1X1 和 COM1X0 位时，输出比较器中断 1 必须通过清除 TIMSK 寄存器中的中断触发位来禁止；否则在位变换时，会发生中断。

位 3,2——Res：保留位

AT90 系列单片机的该位为保留位，总读 0。

位 1,0——PWM11,PWM10：脉冲宽度调制器选择位

这些位如表 2.8 中指明的一样选择定时器/计数器 1 的 PWM 操作。

表 2.7 比较 1 模式选择

COM1X1	COM1X0	说 明
0	0	定时器/计数器 1 与输出脚 OC1X 不连接
0	1	触发 OC1X 输出线
1	0	清除 OC1X 输出线(为 0)
1	1	设置 OC1X 输出线(为 1)

X=A 或 B。

表 2.8 PWM 模式选择

PWM11	PWM10	说 明
0	0	禁止定时器/计数器 1 的 PWM 操作
0	1	定时器/计数器 1 为 8 位 PWM
1	0	定时器/计数器 1 为 9 位 PWM
1	1	定时器/计数器 1 为 10 位 PWM

二、定时器/计数器 1 控制寄存器 B——TCCR1B

位	7	6	5	4	3	2	1	0	
$2E($4E)	ICNC1	ICES1	-	—	CTC1	CS12	CS11	CS10	TCCR1B
读/写：	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
初始化值：	$00								

位 7 —ICNC1：输入捕获噪声消除器(4CKs)

当 ICNC1 位被清为 0 时，输入捕获触发噪声消除器功能被禁止。输入捕获在指定的 ICP，即输入捕获引脚上被采样的第一个上升/下降沿处被触发。当 ICNC1 被设为 1，4 个延续的采样成为 ICP，即输入捕获引脚上的测量值，所有的采样须为高/低，取决于 ICES1 位的输入捕获触发特性。实际的采样频率为 XTAL 时钟频率。

位 6— ICES1：输入捕获 1 边沿选择

当 ICES1 位被清为 0，定时器/计数器 1 的内容被传输到输入捕获寄存器 — ICR1，即在输入捕获引脚 ICP 的下降沿。当 ICES1 位被设为 1，定时器/计数器 1 的内容被传输到输入捕获寄存器 ICR1，即在输入捕获引脚 ICP 的上升沿。

位 5，4 —Res：保留位

AT90 系列单片机的该位为保留位，总读 0。

位 3— CTC1：在比较匹配后清除定时器/计数器 1

当 CTC1 控制位被设为 1 时，在比较匹配之后，定时器/计数器 1 被复位到时钟周期中的 $0000。若 CTC1 控制位被清除时，定时器/计数器 1 继续计数，直到它被停止、清除、溢出，或被改变方向。在 PWM 模式下，该位无效。

位 2，1，0——CS12，CS11，CS10：时钟选择 1 的位 2、1 和 0

时钟选择 1 的位 2、1 和 0 定义了定时器/计数器 1 的预定比例源，见表 2.9。

表 2.9 时钟预定比例选择

CS12	CS11	CS10	说 明	CS12	CS11	CS10	说 明
0	0	0	停止，定时器/计数器 1 被停止	1	0	0	CK/256
0	0	1	CK	1	0	1	CK/1 024
0	1	0	CK/8	1	1	0	外部 T1 脚，下降沿
0	1	1	CK/64	1	1	1	外部 T1 脚，上升沿

三、定时器/计数器 1——TCNT1H 和 TCNT1L

位	15	14	13	12	11	10	9	8	
$2D($4D)	MSB								TCNT1H
$2C($4C)								LSB	TCNT1L
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00 00								

这个 16 位的寄存器包括 16 位定时器/计数器 1 的预定比例值。为确保当 CPU 访问这些寄存器时，高低字节被同时读写，使用一个 8 位的暂存寄存器(TEMP)来完成访问。

1. TCNT1 定时器/计数器 1 写入

当 CPU 向高位字节 TCNT1H 写入时，写入的数据被放入 TEMP 寄存器中；然后，当 CPU 向低位字节 TCNT1L 写入时，数据的字节被与 TEMP 寄存器中的字节数据组合，且全部的 16 位被同步地向 TCNT1 定时器/计数器 1 寄存器写入。作为结果，高字节的 TCNT1H 必须被先访问，以便完成全 16 位寄存器的写入操作。

2. TCNT1 定时器/计数器 1 读取

当 CPU 读低位字节 TCNT1L 时，TCNT1L 低字节的数据被送到 CPU，且高字节 TCNT1H 的数据被放置于 TEMP 寄存器中。当 CPU 读高位字节 TCNT1L 时，CPU 接收 TEMP 寄存器中的数据。作为结果，低字节的 TCNT1L 必须先被访问，以便完成全 16 位寄存器的读取操作。定时器/计数器 1 随着读和写访问，实行向上计数或向上/向下计数(在 PWM 方式)。如果定时器/计数器 1 已被写入且时钟源已被选择，则定时器/计数器 1 在被设置后的定时时钟周期内连续计数。

四、定时器/计数器 1 输出比较寄存器——OCR1AH 和 OCR1AL

位	15	14	13	12	11	10	9	8	
$2B($4B)	MSB								OCR1AH
$2A($4A)								LSB	OCR1AL
	7	6	5	4	3	2	1	0	
读/写：	R W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00 00								

五、定时器/计数器 1 输出比较寄存器——OCR1BH 和 OCR1BL

位	15	14	13	12	11	10	9	8	
$29($49)	MSB								OCR1BH
$28($48)								LSB	OCR1BL
	7	6	5	4	3	2	1	0	
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
读/写：	R/W	R/W	R/W	R W	R/W	R/W	R/W	R/W	
初始化值：	$00 00								

输出比较寄存器为一个 16 位的读/写寄存器。定时器/计数器 1 输出比较寄存器包括了将要连续地与定时器/计数器 1 相比较的数据。比较匹配的操作在定时器/计数器 1 的控制和状态寄存器中被区分。

由于输出比较寄存器---OCR1A 和 OCR1B 为一个 16 位的寄存器，当 OCR1A/B 被写入时，须使用临时寄存器 TEMP 来确保全部的字节被同时写入。当 CPU 写高位字节时，OCR1AH 或 OCR1BH 数据被临时地存储在寄存器 TEMP 中；当 CPU 写低位字节时，OCR1AL 或 OCR1BL、TEMP 寄存器被同步地向 OCR1AH 或 OCR1BH 写入。作为结果，高位的 OCR1AH 或 OCR1BH 必须被先写入，以便完成全部的 16 位寄存器写入操作。

六、定时器/计数器 1 输入捕获寄存器——ICR1H 和 ICR1L

位	15	14	13	12	11	10	9	8	
$25($45)	MSB								ICR1H
$24($44)								LSB	ICR1L
	7	6	5	4	3	2	1	0	
读/写：	R	R	R	R	R	R	R	R	
读/写：	R	R	R	R	R	R	R	R	
初始化值：	$00 00								

输入捕获寄存器为一个 16 位的只读寄存器。当在输入捕获引脚 ICP 上信号的上升或下

降沿(根据输入捕获沿设置——ICES1)被检测到时,定时器/计数器 1 的当前值被传输到输入捕获寄存器——ICR1。同时,输入捕获标志——ICF1 被设为 1。

由于输入捕获寄存器 ICR1 为一个 16 位的寄存器,当 ICR1 被读出时,使用了一个临时寄存器 TEMP,以便确保全部的字节被同时读出。当 CPU 读取低位字节 ICR1L 时,数据被送入 CPU 且高位字节 ICR1H 的数据被放置在 TEMP 寄存器中。当 CPU 读取高位字节 ICR1H 中的数据时,CPU 接收 TMEP 寄存器中的数据。作为结果,低位字节 ICR1L 必须先被访问到,以便完成一个全 16 位寄存器的读取操作。

七、PWM 模式下的定时器/计数器 1

当选择 PWM 模式时,定时器/计数器 1 以及输出比较寄存器 OCR1A 和输出比较寄存器 OCR1B 形成一个双 8 位、9 位或 10 位的自运行、抗误操作,且在引脚 PD5(OC1A)和 OC1B 引脚上带有输出的节拍修正 PWM。定时器/计数器 1 作为向上/向下的计数器,从 $ 0000 向上计数到 TOP(参考表 2.10),在重复循环之前,它反转,并向下再次计数到 0。

当计数器值与 OCR1A、OCR1B 的最后 10 位相配时,PD5(OC1A)/OC1B 引脚根据在定时器/计数器 1 控制寄存器 TCCR1A 中 COM1A1/COM1A0 或 COM1B1/COM1B0 位的设置而被设置或被清除,请参考表 2.11。

表 2.10 定时器 TOP 值和 PWM 频率

PWM 分辨率	定时器 TOP 值	频 率
8 位	$ 00FF(255)	$f_{TCI}/510$
9 位	$ 01FF(511)	$f_{TCI}/1\ 022$
10 位	$ 03FF(1 023)	$f_{TCI}/2\ 046$

表 2.11 在 PWM 方式时比较 1 方式选择

COM1X1	COM1X0	在 OCX1 上的作用
0	0	不连接
0	1	不连接
1	0	清比较匹配,向上计数,置比较匹配,向下计数(PWM 不翻转)
1	1	清比较匹配,向下计数,置比较匹配,向上计数(PWM 翻转)

X=A 或 B。

注意:在 PWM 模式下,当后 10 位 OCR1A/OCR1B 位被写入时,它们被送入临时地址;当定时器/计数器 1 到达 TOP 时,它们被锁存。这就防止了在非同步 OCR1A/OCR1B 写入事件中发生奇数长的 PWM 脉冲(误操作),见图 2.26 的例子。

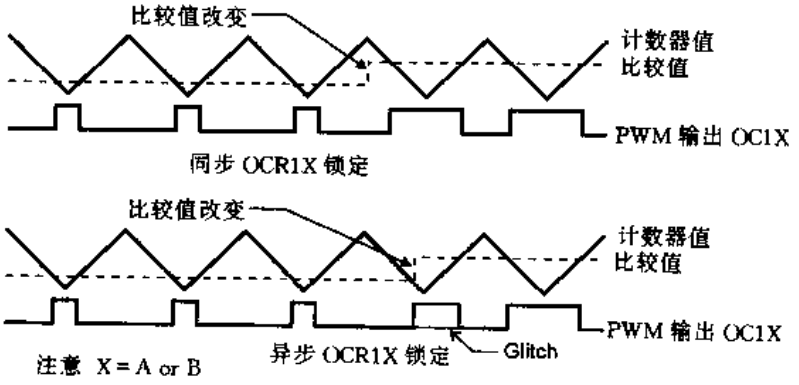


图 2.26 有效的非同步 OCR1 锁存

当 OCR1A 包含 $ 0000 或 TOP，输出 OC1A/OC1B，根据 COM1A1 和 COM1A0 或 COM1B1/COM1B0 的设置保持低或高，如表 2.12 所示。

在 PWM 模式下，定时器溢出标志 1、TOV1，当计数器在方向 $0000 时被设置。定时器溢出中断 1，以正常的定时器/计数器模式工作。比如，当 TOV1 被设置，从而提供了定时器溢出中断 1 和全局中断为触发时，它被执行。这也同样用于定时器输出比较 1 的标志和中断。

表 2.12　PWM 输出 OCR1X 好于 $ 0000 或 TOP

COM1X1	COM1X0	OCR1X	OC1X 输出
1	0	$ 0000	L
1	0	TOP	H
1	1	$ 0000	H
1	1	TOP	L

2.7.4　看门狗定时器

一、看门狗定时器

看门狗定时器由片内一个独立的 1MHz 分频的晶振定时。通过控制看门狗定时器的预定比例器，看门狗的复位间歇从 16 周期到 2 048 周期之间调整。这些值适应 $V_{CC}=5V$。请参考其它 V_{CC} 电压下的、RC 晶振频率的特性化数据。WDR，即看门狗复位指令对看门狗定时器进行复位。有 8 种不同的时钟周期可供选择，来决定在 2 个 WDR 指令之间的最大时间。这样做是为了避免看门狗定时器对 MCU 进行复位。若复位周期结束而无其它的 WDR 指令，AT90 系列单片机进行复位，并从复位向量执行。参见图 2.27。

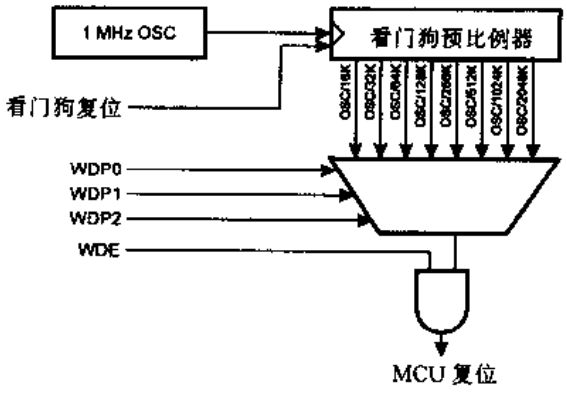


图 2.27　看门狗定时器

为了防止意外禁止看门狗定时器，当看门狗被禁止时必须服从一个特定的关断顺序，请详见看门狗的控制寄存器。

二、看门狗定时器控制寄存器——WDTCR

位	7	6	5	4	3	2	1	0	
$21($41)	—	—	—	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
读/写：	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

位 7～5—Res：保留位

AT90 系列单片机的这些位为保留位，总读 0。

位 4—WDTOE：看门狗关断触发

当 WDTOE 被清除时该位必须被置 1，否则看门狗将不会被禁止。一旦置位后，硬件将在 4 个时钟周期后清除该位。请参照看门狗禁止过程的 WDE 位的描述。

位 3—WDE：看门狗触发

当 WDE 改设为 1 时，看门狗定时器触发；若 WDE 被清为 0，看门狗定时器功能被禁止。WDE 仅在 WDTOE 位设置时被清除，为了禁止被触发的看门狗定时器，必须遵守以下过程：

(1)在同一个操作中,把 WDTOE 和 WDE 写成 1,即使在禁止操作开始前 WDE 为 1,也必须把 1 写入 WDE。

(2)在随后 4 个机器周期中,把 WDE 写为 0,这会禁止看门狗。

位 2～0—WDP2～0:看门狗定时器预定比例器 2、1、0

WDP2、WDP1、WDP0 决定了当看门狗定时器触发时,看门狗定时器的预定比例。不同的预定比例值以及它们相应的超时时间,如表 2.13 所列。

表 2.13 看门狗定时器预定比例选择(典型值 $V_{cc}=5V$)

WDP2	WDP1	WDP0	定时器输出周期/ms	WDP2	WDP1	WDP0	定时器输出周期/ms
0	0	0	16	1	0	0	256
0	0	1	32	1	0	1	512
0	1	0	64	1	1	0	1 024
0	1	1	128	1	1	1	2 048

2.8 AVR 单片机 EEPROM 读/写访问

I/O 空间中可以访问 EEPROM 寄存器。

写入访问时间在 2.5～4 ms 之间,取决于 V_{cc} 电压。自定时功能使得用户软件可检测下一个字节何时被写入。若 V_{cc} 低于一定的电平,EEPROM 降低电压被检测,防止了对 EEPROM 的写入。当 EEPROM 被读取或写入时,在下一个指令执行前,CPU 被中止达 2 个时钟周期。

一、EEPROM 地址寄存器——EEAR

位	15	14	13	12	11	10	9	8	
$1F($3F)	—	—	—	—	—	—	—	EEAR9	EEARH
$1E($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:	$00 00								

EEPROM 地址寄存器——EEARH 和 EEARL,指定了在 512 字节的 EEPROM 空间中的 EEPROM 地址。EEPROM 数据字节被在 0～511 之间线性地编址。

二、EEPROM 数据寄存器——EEDR

位	7	6	5	4	3	2	1	0	
$1D($3D)	MSB							LSB	EEDR
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:	$00								

位 7～0——EEDR7～0:EEPROM 数据

对于 EEPROM 写入操作,EEDR 寄存器包含了写入 EEPROM 的数据,由 EEAR 寄存器给出其地址;对于 EEPROM 读取操作,EEDR 包含由 EEAR 给出的 EEPROM 地址,数据将从这一地址中读出。

三、EEPROM控制寄存器——EECR

位	7	6	5	1	3	2	1	0	
$1C($3C)	—	-		-	-	EEMWE	EEWE	EERE	EECR
读 写：	R	R	R	R	R	R/W	R/W	R/W	
初始化值：	$00								

位7～3——Res:保留位

AT90系列单片机的这些位为保留位,总读0。

位2——EEMWE:EEPROM的主写触发

EEMWE位决定了设置EEWE是否导致EERPOM被写入。当EEMWE被设为1时,设置EEWE将把数据写入EEPROM所选择的地址中;如果EEMWE为0,则设置EEWE无效。当EEMWE被软件设置后,4个周期后被硬件清除。详见EEPROM写过程中的EEWE位的描述。

位1——EEWE:EEPROM写入触发

EEPROM写入触发信号EEWE是对EEPROM的写入选通。若地址和数据被正确设置,EEWE位必须被设置从而写EEPROM。当EEWE被置1时,EEMWE必须被置1;否则,不会发生EEPROM的写操作。当写入EERPOM时应服从以下的过程(第(2)步和第(3)步不是必须的):

(1)等待EEWE位变为0;

(2)把新的EEPROM地址写入EEAR(可选);

(3)把新的EEPROM数据写入EEDR(可选);

(4)在EECR中的EEMWE位写逻辑1;

(5)在设置EEMWE后的4个时钟周期内,在EEWE中写入逻辑1。

在写入访问时间(一般为5 V时2.5 ms,2.7 V时4 ms)过后,EEWE由硬件清0。用户软件在写入下一个字节之前,查询这一位,并等待零值。当EEWE被设过后,CPU在执行下一个指令前中止2个周期。

位0——EERE:EEPROM读取触发

EEPROM读取触发信号EERE是对EEPROM的读取选通。当EEAR寄存器中的地址被正确设置时,EERE必须被设置;当EERER被硬件清空时,在EEDR寄存器中可找到所需数据。EEPROM读取访问占用1个指令,无需查询EERE位。一旦EERE被设过后,CPU在执行下一个指令前中止2个周期。在开始读操作之前,用户可以查询EEWE位。如果当新的数据或地址被写到EEPROM I/O寄存器时,一个写入操作在进行,则写入操作将被中断,并且结果是定义的。

2.9 AVR单片机串行接口

2.9.1 同步串行接口SPI

同步串行接口(SPI)允许在AT90系列单片机和外设或几个AT90系列单片机之间高速同步数据传送,见图2.28。AT90系列单片机SPI的特征如下:

(1)全双工,3线同步数据传送。

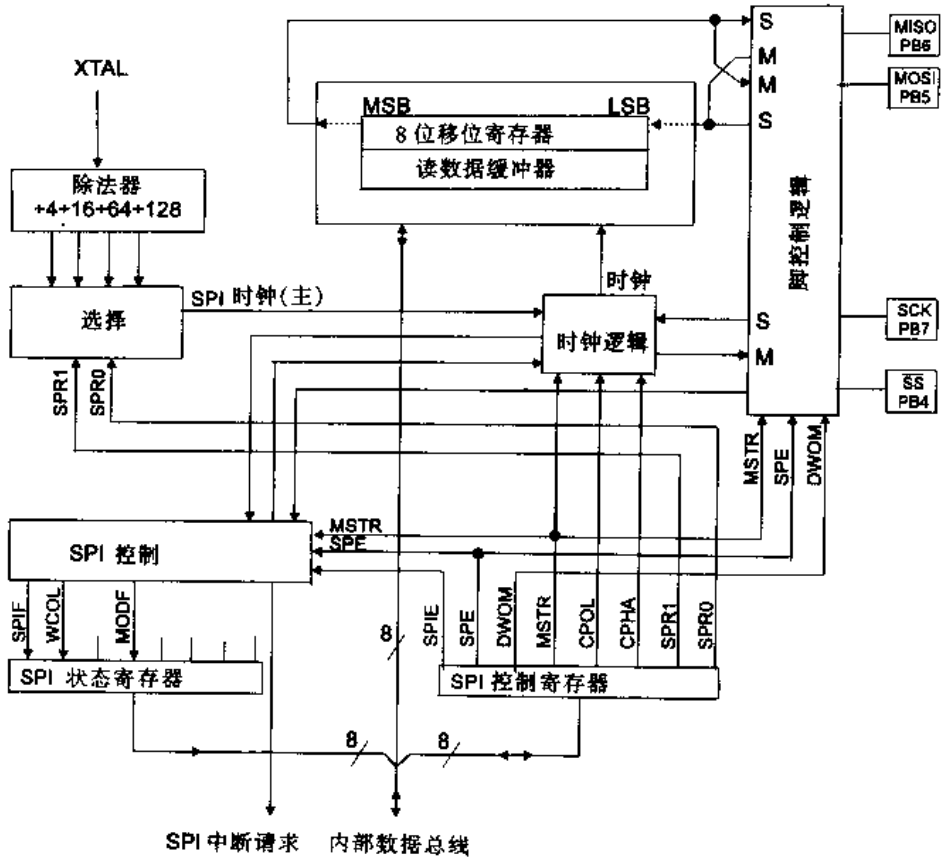


图 2.28 SPI 方框图

(2)主从操作。

(3)5Mb/s 的位传送频率(最大值)。

(4)LSB 或 MSB 在先。

(5)4 种可编程的位速率。

(6)传送停止的中断标志。

(7)写冲突标志保护。

(8)从空闲模式下唤醒(仅从模式)。

主从 CPU 之间的 SPI 连接如图 2.29 所示。PB7(SCK)引脚是主机模式的时钟输出和从机模式的时钟输入,把数据写入主 CPU 的 SPI 数据寄存器会启动 SPI 的时钟生成器,而数据从 PB5(MOSI)引脚移出和移入。在一个字节移出后,SPI 时钟生成器停止,设置传送停止标志位(SPIF)。如果 SPCR 寄存器中的中断触发位(SPIE)被设置,则生成一个中断请求。从机选择输入 PB4($\overline{SS}$),被设置为低来选择单独的 SPI 器件作为从机,主机和从机的两个移位寄存器可以被认为是一个分开的 16 位环形移位寄存器,如图 2.29 所示。当数据从主机移向从机,同时数据也移向相反的方向。这意味着在一个移位周期内,主机和从机的数据交换。

这个系统在发送方向上有一级缓冲而在接收方向有两级缓冲。这表明在全部的移位周期完成之前要被传送的字符不能被写入 SPI 数据寄存器。在接收数据时,在下一个字符被完全

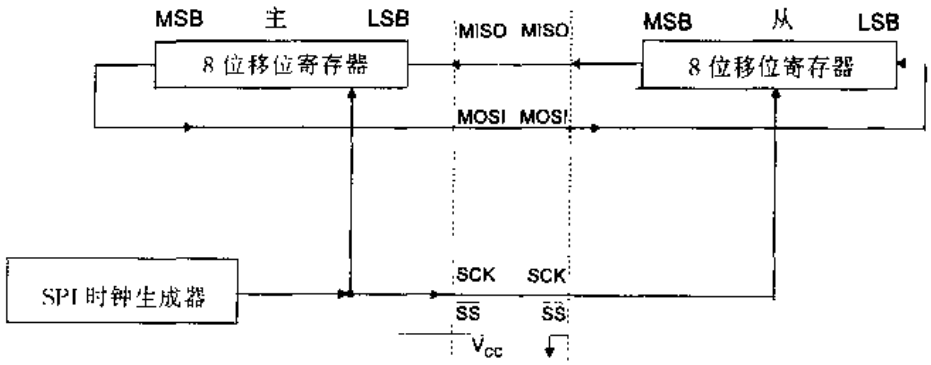


图 2.29 SPI 主从 CPU 内部连接

移入之前已经收到的数据必须从 SPI 数据寄存器中读走；否则，这个字符会丢失。

当 SPI 被触发时，MOSI、MISO、SCK 和$\overline{SS}$引脚的数据方向，按照表 2.14 来配置。

表 2.14 SPI 配置

引 脚	方向，主 SPI	方向，从 SPI	引 脚	方向，主 SPI	方向，从 SPI
MOSI	用户定义	输入	SCK	用户定义	输入
MISO	输入	用户定义	$\overline{SS}$	用户定义	输出

一、$\overline{SS}$引脚的功能

当 SPI 被配置为主机时(SPCR 的 MSTR 置 1)，用户可以决定$\overline{SS}$引脚的方向。如果$\overline{SS}$引脚被设为输出，该引脚作为通用输出不影响 SPI 系统；如果需被设为输入，则必须保持为高来保证主机 SPI 的操作。如果在主机模式下，$\overline{SS}$引脚为输入，而且被外设电路置低，则该系统认为另外的主机选择该 SPI 为它的从机并开始对它传递数据。为了防止总线相连，SPI 系统将采取以下动作：

(1)SPCR 的 MSTR 位被清除而且 SPI 系统变成从机，结果是 MOSI 和 SCK 引脚变成输入。

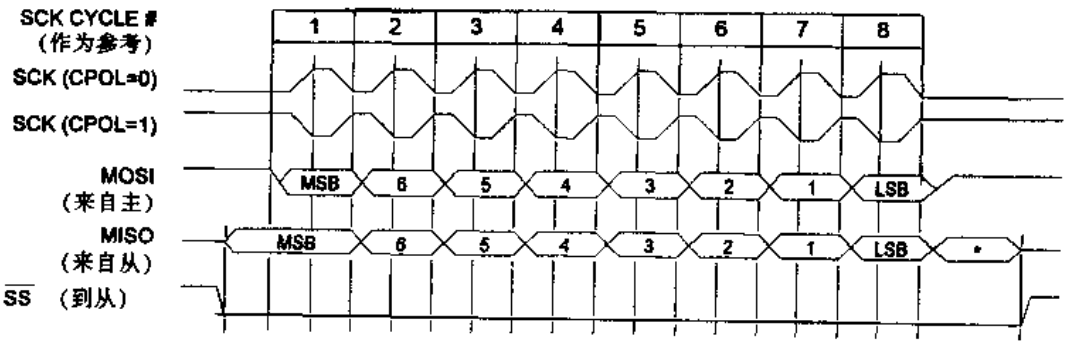
(2)SPSR 中的 SPIF 位被设置，SPI 中断被触发，中断程序被执行。

因此在主机模式下使用中断驱动的 SPI 发送时，存在$\overline{SS}$被置低的可能。中断应检查 MSTR 位是否被设置，一旦发现 MSTR 位被从机清 0，则必须被再设置。

当 SPI 被配置为从机时，$\overline{SS}$引脚应为输入。当$\overline{SS}$被置低时，SPI 被触发且 MISO 变为输出(如果被配置为输出的话)。在$\overline{SS}$被置低后，其余引脚都为输入。所有引脚都为输入，而且 SPI 是被动的，这意味着它不会接收输入的数据。

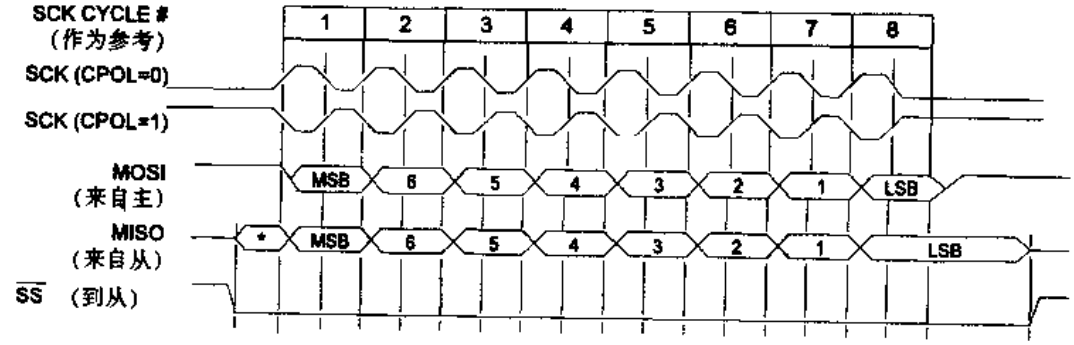
二、数据模式

对应于串行数据，SCK 相位和极性有 4 种组合，由控制位 CPHA 和 CPOL 来决定。SPI 的传送格式如图 2.30 和 2.31 所示。



* 不定义,但在接收时字符的最高有效位正常。

图 2.30 当 CPHA=0 时,SPI 传送格式



* 不定义,但在发送时,先前的最低有效位正常。

图 2.31 当 CPHA=1 时,SPI 传送格式

三、SPI 控制寄存器——SPCR

位	7	6	5	4	3	2	1	0	
$0D($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读/写:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值:	$00								

位 7——SPIE:SPI 中断触发

如果全局中断触发,该位导致设置 SPSR 寄存器的 SPIF 位来执行 SPI 中断。

位 6——SPE:SPI 触发

当该位设置时,SPI 触发,要触发 SPI 任何操作必须设置该位。

位 5——DORD:数据的顺序

当 DORD 位被置 1 时,数据的 LSB(低位)被首先传送;

当 DORD 位被置 0 时,数据的 MSB(高位)被首先传送。

位 4——MSTR:主机/从机选择

当设置为 1 时,选择主机 SPI 模式;当设置为 0 时,选择从机 SPI 模式。如果$\overline{SS}$被设置为输入且在 MSTR 被设置时被置低,则 MSTR 将被清除。而当 SPSR 中的 SPIF 位被设置,应

该设置 MSTR，再触发 SPI 主机模式。

位 3 ——CPOL：时钟极性

当该位被置 1 时，SCK 在闲置时是高电平；当 CPOL 被清 0 时，SCK 在闲置时是低电平。详见图 2.30 和图 2.31。

位 2 CPHA：时钟相位

该位的功能详见图 2.30 和图 2.31。

位 1，0——SPR1，SPR0：SPI 时钟速率选择 1 和 0

这两位控制主机模式下器件 SCK 的速率，SPR1 和 SPR0 对于从机无影响，SCK 和振荡器频率 f_{CL} 之间的关系如表 2.15 所示。

表 2.15 SCK 和振荡器频率之间关系

SPR1	SPR0	SCK 频率	SPR1	SPR0	SCK 频率
0	0	$f_{CL}/4$	1	0	$f_{CL}/64$
0	1	$f_{CL}/16$	1	1	$f_{CL}/128$

四、SPI 的状态寄存器——SPSR

位	7	6	5	4	3	2	1	0	
$0E($2E)	SPIF	WCOL	--	--		--	—	—	SPSR
读/写：	R/W	R/W	R	R	R	R	R	R/W	
初始化值： $00									

位 7——SPIF：SPI 中断标志位

当单行传送完成时，SPIF 位被设置 1，且若 SPCR 中的 SPIE 被设置 1 和全局中断触发，则生成中断。如果$\overline{SS}$被设置为输入且在 SPI 是主机模式时被置低，这将设置 SPIF 标志。SPIF 位在执行相应中断向量时被硬件清除。可选的，在首次读 SPI 状态寄存器时，如果 SPIF 为 1，则先清除它再访问 SPI 数据寄存器(SPDR)。

位 6——WCOL：写冲突位

如果在数据传送中 SPI 数据寄存器(SPDR)被写入，则设置 WCOL 位。在数据传送中，SPDR 寄存器读出的结果是不正确的，写入也没有反应。在首次读 SPI 状态寄存器时，如果 WCOL 为 1，则先清除它再访问 SPI 数据寄存器。

位 5～0——保留位

在 AT90S8515 中这几位保留，读出为 0。

AT90 系列单片机的 SPI 接口也被用于程序存储器和数据 EEPROM 的编程下载和上载，详见串行编程和校验部分。

五、SPI 数据寄存器——SPDR

位	7	6	5	4	3	2	1	0	
$0F($2F)	MSB							LSB	SPDR
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值： $00									

SPI 数据寄存器可以读/写，用于在寄存器堆和 SPI 移位寄存器之间传送数据。写入该寄存器时，初始化数据传送；读该寄存器时，读到的是移位寄存器接收缓冲区的值。

2.9.2 通用串行接口 UART

AT90 系列单片机带有一个全双工的通用串行异步收发器(UART),主要特征如下:

(1) 波特率发生器可以生成任何波特率。

(2) 在 XTAL 低频率下有高的波特率。

(3) 8 位和 9 位数据。

(4) 噪声滤波。

(5) 超越误差的探测。

(6) 帧错误探测。

(7) 错误起始位的探测。

(8) 3 个独立的中断,TX 完成、TX 数据寄存器为空、RX 完成。

一、数据传送

UART 传送器的方框示意图见图 2.32。数据传送通过把被传送的数据写入 UART I/O 寄存器 UDR 来初始化,在以下情况数据从 UDR 传送到移位寄存器中。

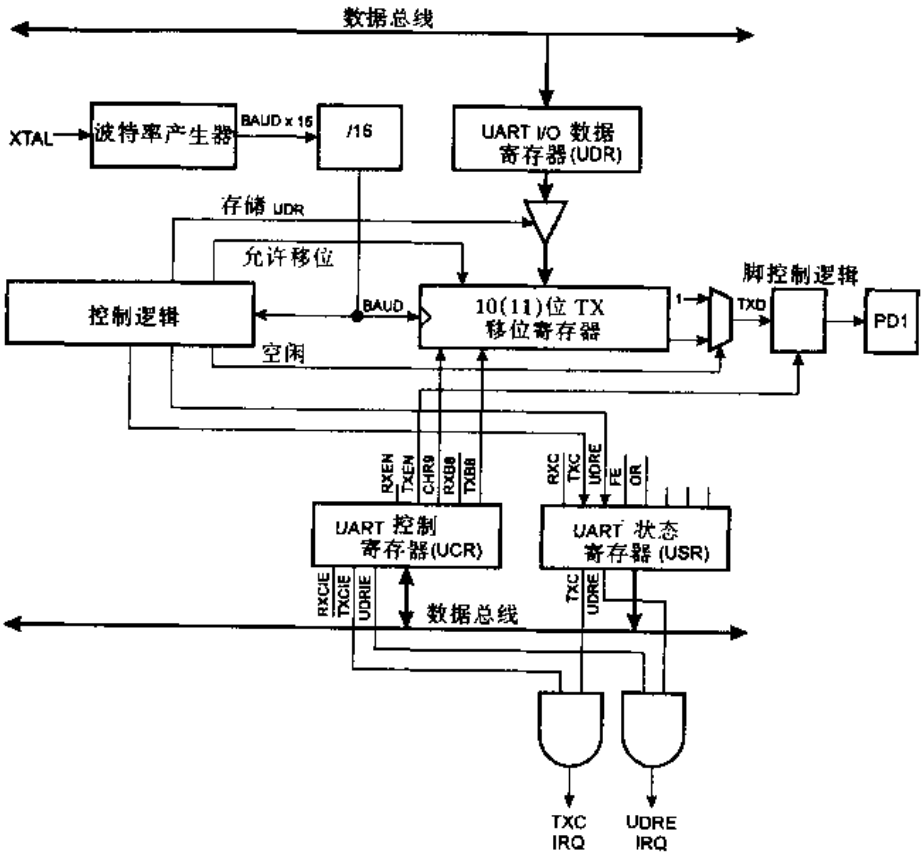


图 2.32 UART 传送器方框示意图

(1) 当前一个字符的停止位被移出后,新的字符被写入 UDR 寄存器,移位寄存器立即再被装入;

(2) 当前一个字符的停止位被移出前,新的字符被写入 UDR 寄存器,移位寄存器在当前

字符的停止位移出后被装入。

如果 10(11)位传送移位寄存器是空的，或当数据从 UDR 中传送到移位寄存器时，UART 状态寄存器、USR 的 UDRE 位(UART 状态寄存器空)被设置。当这位被设置为 1 时，UART 准备接收下一个字符；当数据从 UDR 传送到 10 位(11 位)的移位寄存器中时，移位寄存器的第 0 位(起始位)清 0，而第 9 或第 10 位置 1(停止位)。如果选择 9 位数据(UART 控制寄存器——UCR 的 CHR9 位置位)，UCR 的 TXB8 位被传送到移位寄存器的第 9 位。

在波特率时钟加载到移位寄存器的传送操作时，起始位从 TXD 引脚移出，然后是数据，最低位在先。当停止位被移出时，如果在传送中有新数据被写入 UDR 中，则被装入移位寄存器中。在装入过程中，UDRE 被设置。如果在停止位被移出时 UDR 寄存器中没有新的数据，UDRE 标志位将保持为 1 直到 UDR 被重写。当没有新的数据被写入时，而且停止位在 TXD 上保持了一位的长度，USR 的 TX 完成的标志位——TXC 被置位。

当 UCR 中的 TXEN 位被设置为 1 时，允许 UART 传送，通过清除该位，PD1 引脚可以被用于通用的 I/O。当 TXEN 被设置时，UART 将被连到 PD1 引脚，而不管 DDRD 中的 DDD1 位的是否设置。

二、数据接收

图 2.33 为 UART 接收器的示意图。

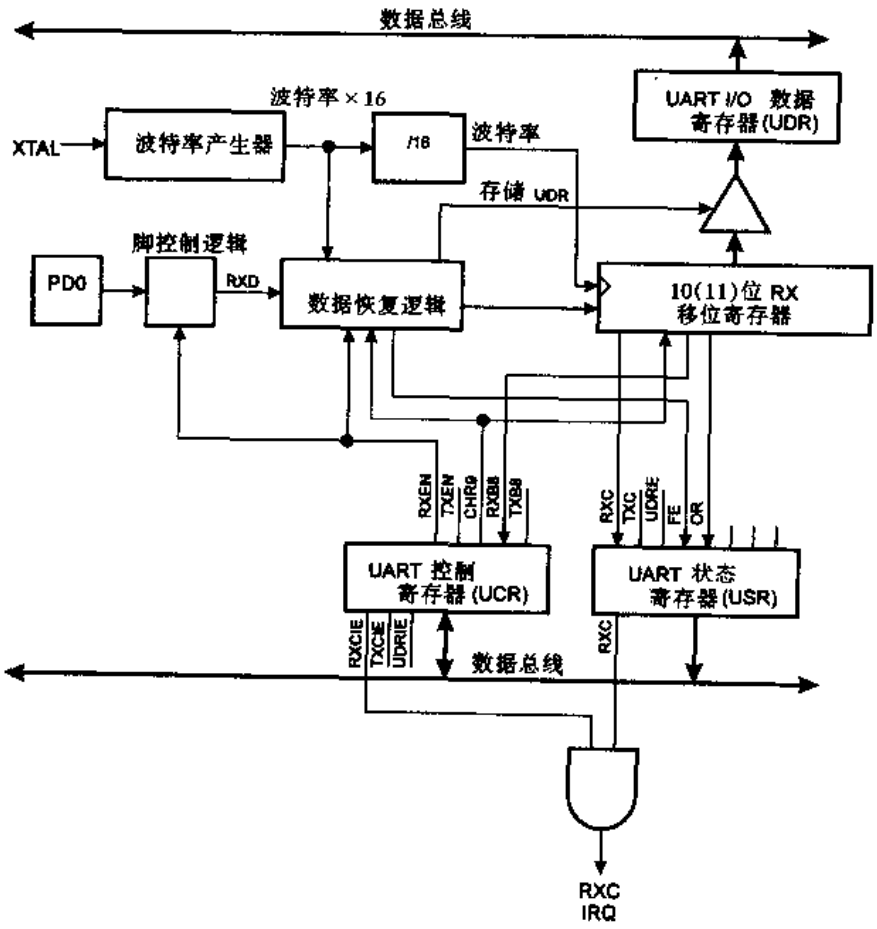


图 2.33 UART 的接收器

接收器前端的逻辑以 16 倍波特率采样 RXD 引脚的信号。当线路闲置时，一个逻辑 0 的采样将被转换为起始位的下降沿，并且起始位的探测序列被初始化。让采样 1 指示出第一个 0 采样，跟随 1 到 0 的转换之后，接收器在第 8、9 和 10 个采样点采样 RXD 引脚。如果三个采样中两个或两个以上是逻辑 1，则起始位是噪声尖峰而被拒绝，接收器继续探测下一个 1 到 0 的转换。

如果一个有效的起始位被发现，就开始起始位之后的数据位的采样。这些位也在第 8、9 和 10 处采样，3 取 2 作为该位的逻辑值，在采样的同时这些位被移入传送移位寄存器。采样输入的字符如图 2.34 所示。

图 2.34 接收器数据采样

当停止位到来时，3 个采样中的大数应为 1 才能接收该停止位。如果两个或更多为逻辑 0，UART 状态寄存器(USR)的帧错误(FE)标志被设置 1，在读 UDR 寄存器之前，用户应检查 FE 帧错误标志。一旦无效的停止位在字符接收周期的结束时被收到，数据被传送到 UDR 寄存器而 USR 的 RXC 标志位被设置。UDR 实际上是两个物理分离的寄存器，一个发送数据，一个接收数据。当读 UDR 时，接收数据寄存器被访问；当写 UDR 寄存器时，传送数据寄存器被访问。如果选择了 9 位数据(UART 控制寄存器 UCR 中的 CHR9 位被设置)，在数据被传送到 UDR 时，传送移位寄存器的第 9 位被装入到 UCR 的 RXB8 位。

如果在收到一个字符的最后一个接收后，UDR 寄存器还没有被读走，UCR 中的超越误差标志(OR)被设置。这意味着移入移位寄存器的最后的数据字节不能被送到 UDR 中而丢失。OR 位被缓冲，当 UDR 中的有效的数据字节被读出时该位被更新。因此，用户在读 UDR 后应检查 OR 位来探测任何的超越错误。

通过清除 UCR 寄存器中的 RXEN 位使接收器被禁止，这意味着 PD0 可以被用做通用的 I/O 引脚。当 RXEN 被设置，USRT 接收器被连到 PD0 引脚，而不管 DDRD 中的 DDD0 位是否设置。

三、UART 控制

1. UART I/O 数据寄存器——UDR

位	7	6	5	4	3	2	1	0	
$0C($2C)	MSB							LSB	UDR
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

UDR 寄存器是两个物理分离的寄存器分享相同的 I/O 地址。当写入寄存器时，UART 的发送数据寄存器被写入；当读 UDR 时，读的是 UART 接收寄存器。

2. UART 状态寄存器——USR

位	7	6	5	4	3	2	1	0	
$0B($2B)	RXC	TXC	UDRE	FE	OR	—	—	—	USR
读/写：	R/W	R/W	R/W	R/W	R/W	R	R	R	
初始化值：	$00								

USR 寄存器是一个只读的寄存器，提供 UART 的状态信息。

位 7 -- RXC：UART 接收完成

当收到的字符从接收移位寄存器传到 UDR 中时该位被设置。不论探测到任何的帧错误，该位都被设置。当 UCR 中的 RXCIE 位被设置后，UART 接收完成中断将被执行（当 RXC 被设置），RXC 在读 UDR 时被清除。当使用中断数据接收时，接收完成中断子程序必须读 UDR 而清除 RXC；否则，在子程序完成时会引起新的中断。

位 6 ---TXC：UART 发送完成

当发送移位寄存器的全部数据被移出后且没有新的数据被写入 UDR 时该位被设置。这个标志位在半双工的通讯接口中很有用。当完成发送后立即释放通讯总线，并必须进入接收模式。当 UCR 中的 TXCIE 位被设置后，设置 TXC 将导致 UART 发送完成中断被执行，TXC 在执行相应的中断向量时被硬件清除。TXC 位也可以通过在该位写一个逻辑 0 而被清除。

位 5 -- UDRE：UART 数据寄存器空

当写入 UDR 的字符被传送到发送移位寄存器中时该位被设置，设置该位指示出发送器准备新的数据发送。当 UCR 中的 UDRIE 位被设置时，UART 发送完成中断将被执行。只要 UDRE 被设置，UDRE 可以通过写 UDR 而清除。当使用中断驱动的数据发送时，UART 数据寄存器空的中断服务程序应该写 UDR 来清除 UDRE，否则在中断子程序完成时将发生新的中断。在复位时，UDRE 被设置为 1 指示出准备传送。

位 4 ---FE：帧出错

如果在帧出错条件被检测到时该位被设置（如当收到数据的停止位为 0 时）。FE 在收到数据的停止位为 1 时被清除。

位 3 OR：超越出错

如果超越出错条件被检测到，该位被设置（如当 UDR 寄存器中的数据没有在新的数据被移入接收移位寄存器之前被读走）。OR 位被缓冲，这意味着一旦 UDRE 中有效的数据被读走后它将被设置。OR 位在收到的数据被传送到 UDR 中时被清除。

位 2～0 Res：保留位

在 AT90S8515 中这些位被保留，读出为 0。

3. UART 控制寄存器 UCR

位	7	6	5	4	3	2	1	0	
$0A($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
初始化值：	$00								

位 7---RXCIE：RX 完成中断触发

当该位被置 1 时，如果全局中断被触发，在 USR 中设置 RXC 位将导致接收完成中断被执行。

位 6 TXCIE：TX 完成中断触发

当该位被置 1 时，如果全局中断被触发，在 USR 中设置 TXC 位将导致发送完成中断被执行。

位 5 UDRIE：UART 数据寄存器空中断触发

当该位被置 1 时，如果全局中断被触发，在 USR 中设置 UDRE 位将导致 UART 数据寄存器空中断被执行。

位 4——RXEN：接收触发

当该位被设置时允许 UART 接收。当接收器被禁止时，TXC、OR 和 FE 位状态标志位不能设置。如果这些位被设置，在把 RXEN 关闭时不能清除它们。

位 3——TXEN：发送触发

当该位被设置为 1 时允许 UART 发送。如在发送数据时禁止发送器，则在移位寄存器的数据和后续 UDR 中的数据被全部发送完成之前发送器不会被禁止。

位 2——CHR9：9 位字符

当设置该位时，发送和接收的数据是 9 位加上起始和停止位。第 9 位通过 UCR 中的 RXB8 和 TXB8 位来分别读和写。第 9 位可以作为额外的停止位和奇偶位。

位 1——RXB8：收到的数据第 8 位

当 CHR9 被设置时，RXB8 是收到数据的第 9 数据位。

位 0——TXB8：发送的数据第 8 位

当 CHR9 被设置时，TXB8 是发送数据的第 9 数据位。

4. 波特率发生器

波特率发生器是依据以下等式的分频器来产生波特率：

$$BAUD = FCK/[16(UBRR+1)]$$

其中，BAUD 表示波特率；FCK 表示晶振频率；UBRR 表示 UART 波特率寄存器的值，UBRR(0～255)。

对于标准的晶振频率，可以通过表 2.16 来设置 UBRR 而产生常用的波特率，生成与实际波特率相差小于 2%的 UBRR 的值。

表 2.16 不同晶振频率的 UBRR 设置

波特率	1 MHz	误差/%	1.843 MHz	误差/%	2 MHz	误差/%	2.457 6 MHz	误差/%
2 400	UBRR=25	0.2	UBRR=47	0.0	UBRR=51	0.2	UBRR=63	0.0
4 800	UBRR=12	0.2	UBRR=23	0.0	UBRR=25	0.2	UBRR=31	0.0
9 600	UBRR=6	7.5	UBRR=11	0.0	UBRR=12	0.2	UBRR=15	0.0
14 400	UBRR=3	7.8	UBRR=7	0.0	UBRR=8	3.7	UBRR=10	3.1
19 200	UBRR=2	7.8	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	0.0
28 800	UBRR=1	7.8	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	6.3
38 400	UBRR=1	22.9	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	0.0
57 600	UBRR=0	7.8	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	12.5
76 800	UBRR=0	22.9	UBRR=1	33.3	UBRR=1	22.9	UBRR=1	0.0
115 200	UBRR=0	84.3	UBRR=0	0.0	UBRR=0	7.8	UBRR=0	25.0
波特率	3.276 8 MHz	误差/%	3.686 4 MHz	误差/%	4 MHz	误差/%	4.608 MHz	误差/%
2 400	UBRR=84	0.4	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0
4 800	UBRR=42	0.8	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0
9 600	UBRR=20	1.6	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0
14 400	UBRR=13	1.6	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0
19 200	UBRR=10	3.1	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0

表 2.16(续)

波特率	3.276 8 MHz	误差/%	3.686 4 MHz	误差/%	4 MHz	误差/%	4.608 MHz	误差/%
28 800	UBRR=6	1.6	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0
38 400	UBRR=4	6.3	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7
57 600	UBRR=3	12.5	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0
76 800	UBRR=2	12.5	UBRR=2	0.0	UBRR=2	7.8	UBRR=3	6.7
115 200	UBRR=1	12.5	UBRR=1	0.0	UBRR=1	7.8	UBRR=2	20.0
波特率	7.372 8 MHz	误差/%	8 MHz	误差/%	9.216 MHz	误差/%	11.059 MHz	误差/%
2 400	UBRR=191	0.0	UBRR=207	0.2	UBRR=239	0.0	UBRR=287	—
4 800	UBRR=95	0.0	UBRR=103	0.2	UBRR=119	0.0	UBRR=143	0.0
9 600	UBRR=47	0.0	UBRR=51	0.2	UBRR=59	0.0	UBRR=71	0.0
14 400	UBRR=31	0.0	UBRR=34	0.8	UBRR=39	0.0	UBRR=47	0.0
19 200	UBRR=23	0.0	UBRR=25	0.2	UBRR=29	0.0	UBRR=35	0.0
28 800	UBRR=15	0.0	UBRR=16	2.1	UBRR=19	0.0	UBRR=23	0.0
38 400	UBRR=11	0.0	UBRR=12	0.2	UBRR=14	0.0	UBRR=17	0.0
57 600	UBRR=7	0.0	UBRR=8	3.7	UBRR=9	0.0	UBRR=11	0.0
76 800	UBRR=5	0.0	UBRR=6	7.5	UBRR=7	6.7	UBRR=8	0.0
115 200	UBRR=3	0.0	UBRR=3	7.8	UBRR=4	0.0	UBRR=5	0.0

5. 波特率寄存器——UBRR

位	7	6	5	4	3	2	1	0	
$09($29)	MSB							LSB	UBRR
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

UBRR 是 8 位可以读/写的寄存器，用来确定波特率。

2.10　AVR 单片机模拟比较器

2.10.1　模拟比较器

模拟比较器对正极 PB2 引脚(AIN0)和负极 PB3 引脚(AIN1)之上的输入值进行比较。当 PB2 上的电压高于 PB3 的电压时，模拟比较器输出 ACO 被设为 1。比较器的输出可以被设置为触发定时器/计数器 1 的输入捕获功能，此外，比较器的输出可以被设置为触发一个独立于模拟比较器的中断。用户可以选择比较器输出上升、下降或触发的中断触发。比较器的方框图和周围电路如图 2.35 所示。

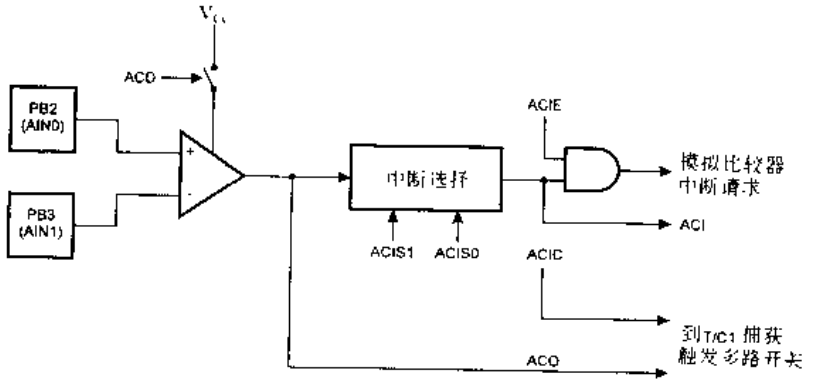


图 2.35　模拟比较器方框图

2.10.2 模拟比较器控制和状态寄存器 ACSR

位	7	6	5	4	3	2	1	0	
$08($28)	ACD		ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
读/写：	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

位 7 —— ACD：模拟比较器禁止

当该位设为 1 时，模拟比较器的电源关闭。该位可以在任何时候被设置，以便关闭模拟比较器。在空闲模式的电源消耗为临界时，它最为常用，且无需从模拟比较器唤醒。改变 ACD 位时，模拟比较器中断，必须通过清空 ACSR 中的 ACIE 位来禁止；否则，在该位改变时，会产生中断。

位 6——Res：保留位

AT90 系列单片机的该位为保留位，总读 0。

位 5——ACO：模拟比较器输出

ACO 直接与模拟比较器的输出相连。

位 4——ACI：模拟比较器中断标志位

当比较器输出事件触发由 ACI1 和 ACI0 定义的中断模式时，这一位被设为 1。若 ACIE 位被设为 1，且 SREG 中的 I 位被设为 1 时，执行模拟比较器的中断程序。当执行相应的中断处理向量时，ACI 被硬件清空。作为替换，ACI 通过对标志位写入逻辑 1 来清空。

位 3 —— ACIE：模拟比较器中断触发

当 ACIE 位设为 1，且状态寄存器中的 I 位被设为 1 时，模拟比较器中断被触发。当被清为 0 时，中断被禁止。

位 2——ACIC：模拟比较器输入捕获触发

当设置为 1 时，该位触发定时器/计数器 1 的输入捕获功能，由模拟比较器来触发。在这种情况下，模拟比较器的输出直接连到输入捕获前端逻辑，使比较器能利用定时器/计数器 1 输入捕获中断的噪声消除和边缘选择的特性。当该位被清除时，模拟比较器和输入捕获功能之间没有联系。为了使比较器触发定时器/计数器 1 的输入捕获中断，定时器中断屏蔽寄存器(TIMSK)的 TICIE1 位必须被设置。

位 1，0——ACIS1，ACIS0：模拟比较器中断模式选择

这些位决定了哪一比较器事件触发模拟比较器中断。表 2.17 所示为不同的设置。

表 2.17 ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式	ACIS1	ACIS0	中断模式
0	0	输出触发，比较器中断	1	0	下降输出沿，比较器中断
0	1	与上相反	1	1	上升输出沿，比较器中断

注意：当改变 ACIS1/ACIS0 位时，必须通过清除 ACSR 寄存器中的中断触发位来禁止模拟比较器中断；否则，当这些位改变时会发生中断。

2.11 AVR单片机I/O端口

2.11.1 端口A

一、A口特性

A口为一个8位的双向I/O口。

A口分配有3个数据存储地址，分别为数据寄存器PORTA $1B($3B)、数据方向寄存器DDRA $1A($3A)和A口的输出引脚PINA $19($39)。A口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。

所有的A口引脚都有独立可选的上拉，A口输出缓冲器可以吸收20 mA的电流以直接驱动LED显示。当PA0～PA7引脚被用作输入且被外部拉低时，若内部拉高被触发，这些引脚将成为电流源(I_{IL})。

A口引脚具有与可选的外部数据SRAM有关的第二功能，A口在访问外部数据存储器时可以配置为复用的低位地址/数据线，在该模式下，A口有内部的上拉。

当通过MCU控制寄存器MCUCR的SRE，外部SRAM触发位把A口设置为第二功能，更改的设置会覆盖数据方向存储器。

1. A口数据寄存器- PORTA

位	7	6	5	4	3	2	1	0	
$1B($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

2. A口数据方向寄存器— DDRA

位	7	6	5	4	3	2	1	0	
$1A($3A)	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0	DDRA
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

3. A口输入脚地址——PINA

位	7	6	5	4	3	2	1	0	
$19($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读/写：	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

A口的输入引脚地址PINA不是一个寄存器，该地址允许对A口的每一个引脚的物理值进行访问。当读PORTA时，读到的是PORTA的数据锁存器；当读PINA时，引脚上的逻辑值被读取。

二、A口作为通用的数字I/O

当作为数字I/O口时A口所有的8位都等效。

PAn为通用I/O引脚：DDRA寄存器的DDAn位选择引脚的方向。如果DDAn设为1，PAn被配置为输出引脚；如果DDAn设为0，PAn被配置为输入引脚；如果PORTAn被设置为1，DDAn被配置为输入引脚，则MOS上拉电阻被触发。为了关断上拉电阻，PORTAn位

必须被清除或者引脚被配置为输出引脚。A 口引脚 DDAn 的作用见表 2.18。

表 2.18 A 口引脚的 DDAn 的作用

DDAn	PORTAn	I/O	上拉	注 释
0	0	输入	否	一态(高阻)
0	1	输入	是	上拉低 PAn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,6,5,…,0 为引脚数。

三、A 口原理图

A 口原理如图 2.36 所示(注意:所有引脚是同步的,同步锁存器在图中并未列出)。

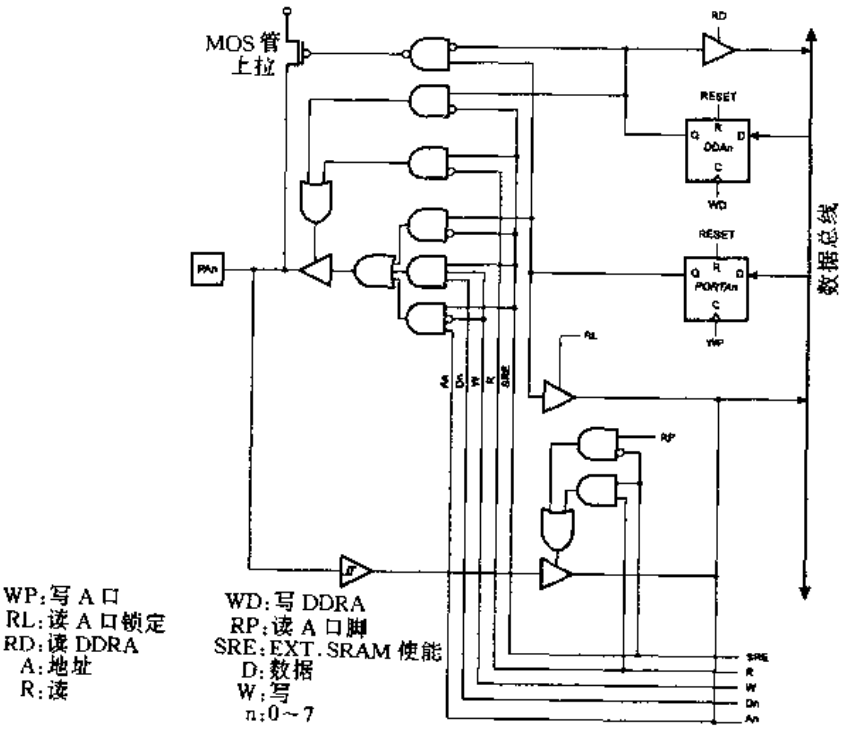


图 2.36 A 口原理图(脚 PA0~PA7)

2.11.2 端口 B

一、B 口特性

B 口为一个 8 位的双向 I/O 口。

B 口分配有 3 个数据存储地址,分别为数据寄存器 PORTB $18($38)、数据方向寄存器 DDRB $17($37)和 B 口的输出引脚 PINB $16($36)。B 口的输入引脚地址为只读,而数据寄存器和数据方向寄存器为可读写。

所有的 B 口引脚均有单独的可选择拉高。B 口输出缓冲器可以吸收 20 mA 的电流以直接驱动 LED 显示。当 PB0~PB7 引脚被用作输入,且被外部拉低时,若内部拉高被触发,这些引

脚将成为电流源(I_{tL})。

具有第二功能的 B 口引脚如表 2.19 所示。当引脚被用作第二功能时,DDRB 和 PORTB 寄存器必须根据第二功能说明来设置。

表 2.19　B 口引脚第二功能

口引脚	第二功能	口引脚	第二功能
PB0	T0(定时器/计数器 0 外部计数器输入)	PB4	$\overline{SS}$(SPI 从选择输入)
PB1	T1(定时器/计数器 1 外部计数器输入)	PB5	MOSI(SPI 总线主输出/从输入)
PB2	AIN0(模拟比较器正输入)	PB6	MOSO(SPI 总线主输出/从输入)
PB3	AIN1(模拟比较器负输入)	PB7	SCK(SPI 总线串行时钟)

1. B 口数据寄存器——PORTB

位	7	6	5	4	3	2	1	0	
$18($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

2. B 口数据方向寄存器——DDRB

位	7	6	5	4	3	2	1	0	
$17($37)	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	DDRB
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

3. B 口输入脚地址—　PINB

位	7	6	5	4	3	2	1	0	
$16($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读/写：	R	R	R	R	R	R	R	R	
初始化值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

B 口输入引脚地址 PINB 并不是一个寄存器,这一地址允许对 B 口每个引脚的物理值进行访问。

当读取 PORTB 时,PORTB 数据锁存器被读取;当读取 PINB 时,引脚上的当前逻辑值被读取。

二、B 口作为通用数字 I/O

当 B 口上的所有 8 个位用作数字 I/O 引脚时功能一样。

PBn 为通用 I/O 引脚:DDRB 寄存器中的 DDBn 位选择该引脚的方向。若 DDBn 被设为 1 时,PBn 设为输出引脚;若 DDBn 被清为 0 时,PBn 设为输入引脚。若 PORTBn 被设为 1,引脚被设为输入时,MOS 拉高电阻被触发。为关闭拉高电阻,PORTBn 必须被清为 0,或者该引脚必须被设置为输出引脚。

B 口引脚的 DDBn 作用见表 2.20。

表 2.20 B 口引脚的 DDBn 的作用

DDBn	PORTBn	I/O	上 拉	注 释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PBn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,6,…,0 为引脚数。

三、B 口的可选择性功能

SCK——B 口,位 7

SCK 为 SPI 的主时钟输出、从时钟输入端。当 SPI 被触发为从机时,该引脚被作为输入而不管 DDB7 的设置;当 SPI 被触发为主机时,该引脚的数据方向由 DDB7 来控制;当该引脚被强制作为输入时,内部的上拉仍可以被 PORTB7 位来控制,详见 SPI 口的描述。

MISO——B 口,位 6

MISO 为 SPI 的主数据输入、从数据输出端。当 SPI 被触发为主机时,该引脚被作为输入而不管 DDB6 的设置;当 SPI 被触发为从机时,该引脚的数据方向由 DDB6 来控制;当该引脚被强制作为输入时,内部的上拉仍可以被 PORTB6 位来控制,详见 SPI 口的描述。

MOSI——B 口,位 5:

MOSI 为 SPI 的主数据输出、从数据输入端。当 SPI 被触发为从机时,该引脚被作为输入而不管 DDB5 的设置;当 SPI 被触发为主机时,该引脚的数据方向由 DDB5 来控制;当该引脚被强制作为输入时,内部的上拉仍可以被 PORTB5 位来控制,详见 SPI 口的描述。

$\overline{SS}$——B 口,位 4:

$\overline{SS}$为从机端口选择信号输入端。当 SPI 被触发为从机时,该引脚被作为输入而不管 DDB5 的设置。作为从机时,当该引脚被置低时 SPI 被触发。当 SPI 被触发为主机时,该引脚的数据方向由 DDB5 来控制;当该引脚被强制作为输入时,内部的上拉仍可以被 PORTB4 位来控制,详见 SPI 口的描述。

AIN1——B 口,位 3:

AIN1 为模拟比较器负极输入。当被设置为输入(DDB3 被清为 0),且内部 MOS 拉高电阻关闭(PB3 被清为 0)时,该引脚用作片内模拟比较器的负极输入。

AIN0——B 口,位 2:

AIN0 为模拟比较器正极输入。当被设置为输入(DDB2 被清为 0),且内部 MOS 拉高电阻关闭(PB2 被清为 0)时,该引脚用作片内模拟比较器的正极输入。

T1——B 口,位 1:

T1 为定时器/计数器 1 的计数输入源,详见定时器部分。

T0—B 口,位 0:

T0 为定时器/计数器 0 的计数输入源,详见定时器部分。

四、B 口原理图

B 口原理图如图 2.37～图 2.42 所示,注意所有引脚为同步的。同步锁存器图中并未列出。

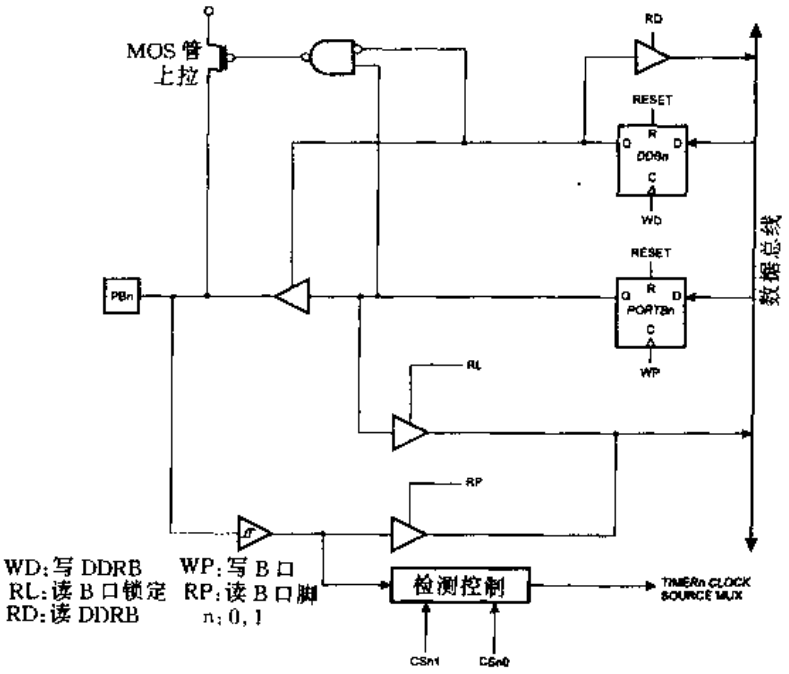


图 2.37　B 口原理图(PB0 和 PB1 脚)

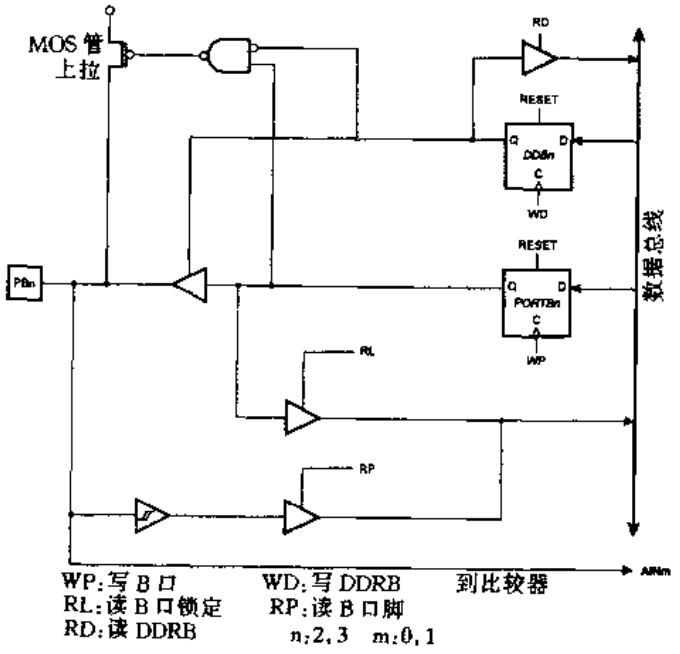


图 2.38　B 口原理图(PB2 和 PB3 脚)

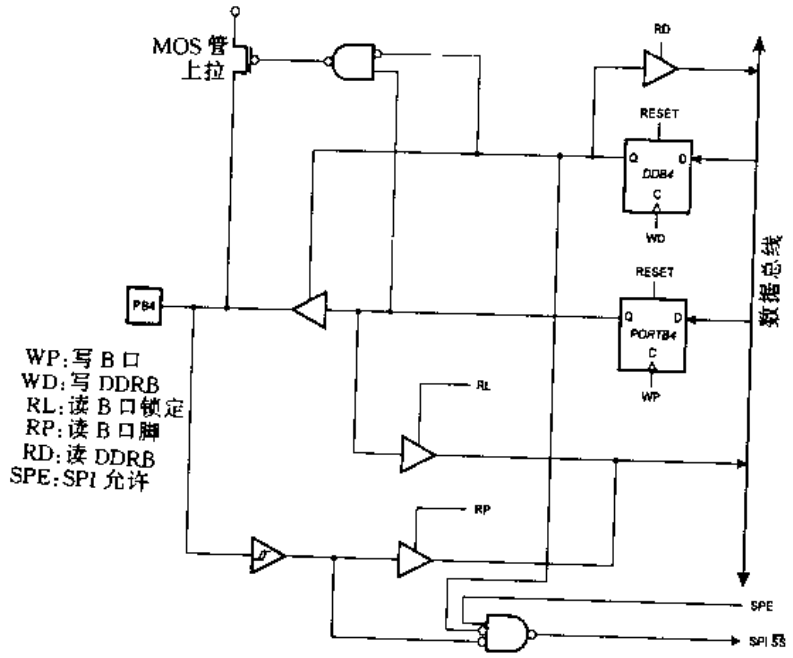


图 2.39 B 口原理图(PB4 脚)

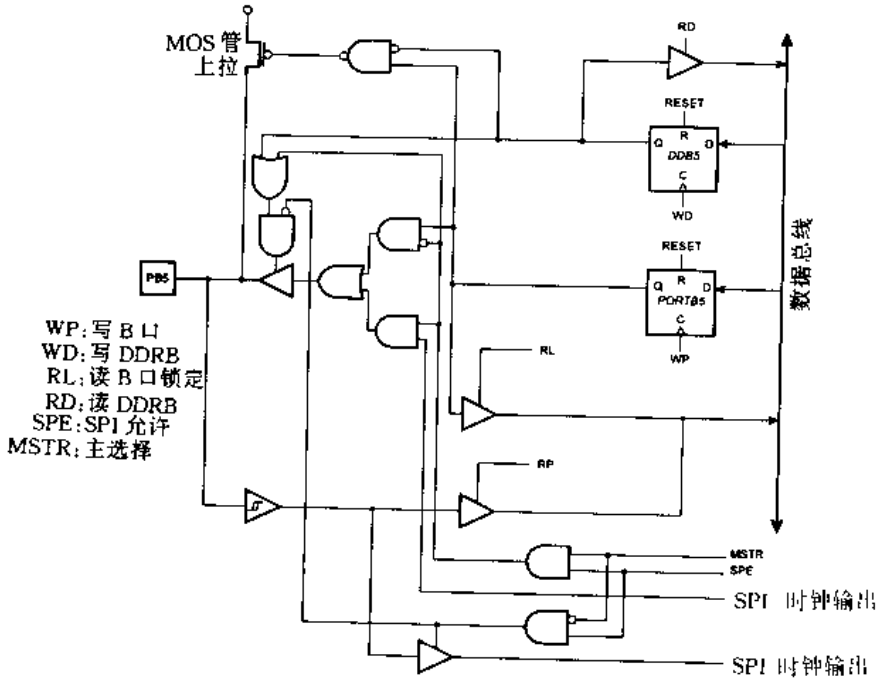


图 2.40 B 口原理图(PB5 脚)

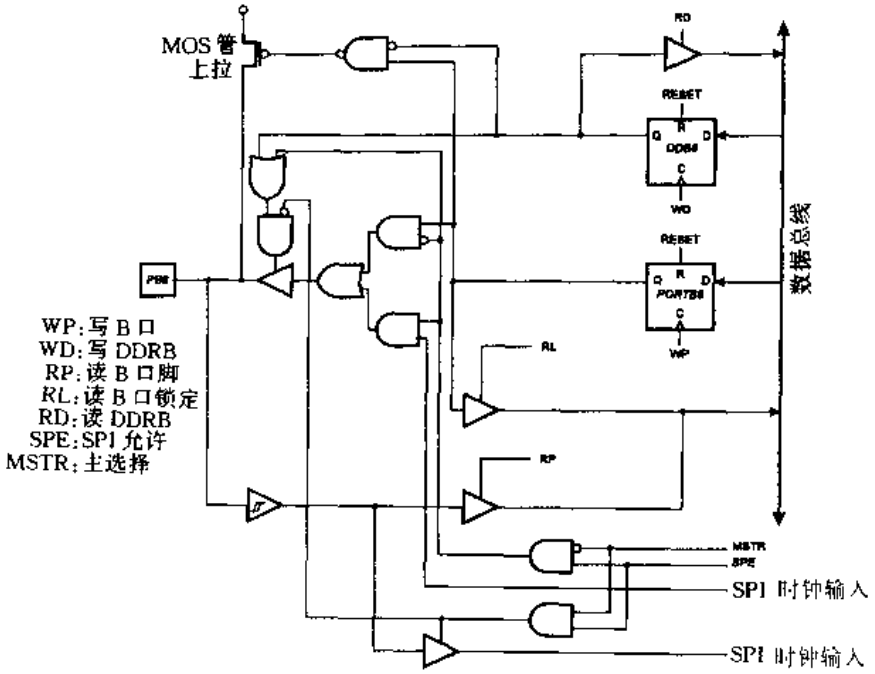


图 2.41　B 口原理图(PB6 脚)

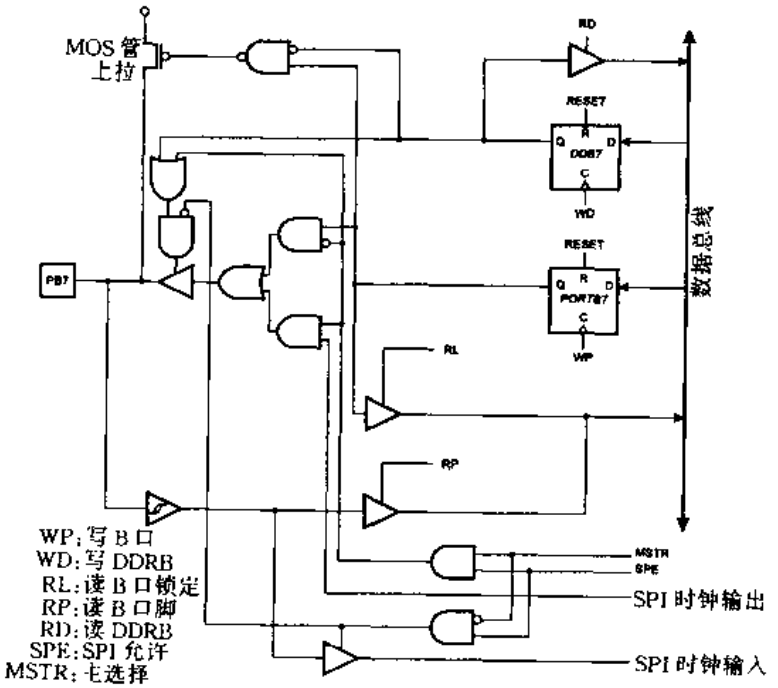


图 2.42　B 口原理图(PB7 脚)

2.11.3 端口C

一、C口

C口为一个8位的双向I/O口。

C口分配有3个数据存储地址，分别为数据寄存器PORTC $15($35)、数据方向寄存器DDRC $14($34)和C口的输出引脚PINC $13($33)。C口的输入引脚地址为只读，而数据寄存器和数据方向寄存器为可读写。

所有的C口引脚均有单独的可选择拉高。C口输出缓冲器可以吸收20 mA的电流以直接驱动LED显示。当PC0～PC7引脚被用作输入且被外部拉低时，若内部拉高被触发，这些引脚将成为电流源(I_{IL})。

C口引脚具有与可选的外部数据SRAM有关的第二功能，C口在访问外部数据存储器时可以配置为复用的高位地址线，在该模式下，C口输出1时使用内部的上拉。

当通过MCU控制寄存器MCUCR的SRE，外部SRAM触发位把C口设置为第二功能，更改的设置会覆盖数据方向存储器。

1. C口数据寄存器— PORTC

位	7	6	5	4	3	2	1	0	
$15($35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

2. C口数据方向寄存器— DDRC

位	7	6	5	4	3	2	1	0	
$14($34)	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0	DDRC
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

3. C口输入脚地址— PINC

位	7	6	5	4	3	2	1	0	
$13($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读/写：	R	R	R	R	R	R	R	R	
初始化值：	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

C口的输入引脚地址PINC不是一个寄存器，该地址允许对端口C的每一个引脚进行存取。当读PORTC时，读到的是PORTC的数据锁存器；当读PINC时，引脚上的逻辑值被读取。

二、C口作为通用的数字I/O

当作为数字I/O口时C口所有的8位都等效。

PCn为通用I/O引脚：DDRC寄存器的DDCn位选择引脚的方向。如果DDCn设为1，PCn被配置为输出引脚；如果DDCn设为0，PCn被配置为输入引脚；如果PORTCn被设置为1，DDCn被配置为输入引脚，则MOS上拉电阻被触发。为了关断上拉电阻，PORTCn位必须被清除或者引脚被配置为输出引脚。C口引脚DDCn的作用见表2.21。

表 2.21　C 口引脚的 DDCn 的作用

DDCn	PORTCn	I/O	上　拉	注　释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PCn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n:7,…,0 为引脚数。

三、C 口原理图

C 口原理图如图 2.43 所示。注意所有的引脚是同步的,同步锁存器图中并未列出。

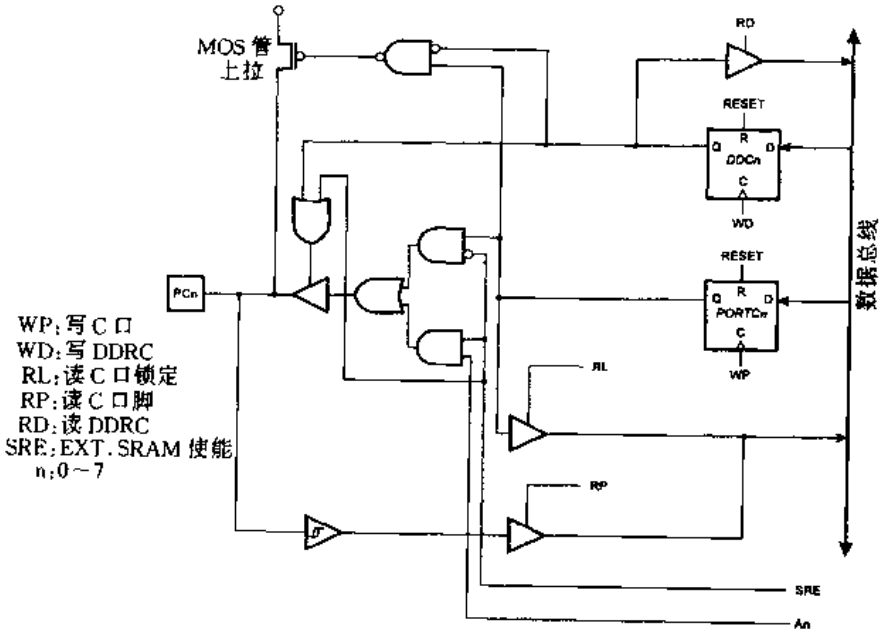


图 2.43　C 口原理图(PC0～PC7)

2.11.4　端口 D

一、D 口特性

D 口是一个带内部上拉的 8 位双向 I/O 口。

D 口占了 3 个数据存储器的地址,数据寄存器 PORTD $12($32)、数据方向寄存器 DDRD $11($31)和端口 D 输入引脚 PIND $10($30)。D 口的引脚地址是只读的,而数据寄存器和数据方向寄存器可以读写。

D 口的输出缓冲器可以吸收 20 mA 的电流。D 口的引脚在触发内部上拉时,如果外部被拉低就会成为电流源(I_{IL})。

某些 D 口的引脚有第二功能如表 2.22 所列。

表 2.22 D 口引脚第二功能

口引脚	第二功能	口引脚	第二功能
PD0	RDX(UART 输入线)	PD5	OC1A(T/C1 输入比较 A 匹配输出)
PD1	TDX(UART 输出线)	PD6	$\overline{WR}$(写选通)
PD2	INT0(外部中断 0 输入)	PD7	$\overline{RD}$(读选通)
PD3	INT1(外部中断 1 输入)		

1. D 口数据寄存器 ——PORTD

位	7	6	5	4	3	2	1	0	
$12($32)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

2. D 口数据方向寄存器——DDRD

位	7	6	5	4	3	2	1	0	
$11($31)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
读/写：	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始化值：	$00								

3. D 口输入脚地址 —PIND

位	7	6	5	4	3	2	1	0	
$10($30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
读/写：	R	R	R	R	R	R	R	R	
初始化值：	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	

D 口的输入引脚地址 PIND 不是一个寄存器，该地址允许对端口 D 的每一个引脚进行存取。当读 PORTD 时，读到的是 PORTD 的数据锁存器；当读 PIND 时，引脚上的逻辑值被读取。

二、D 口作为通用的数字 I/O

PDn，通用 I/O 引脚：DDRD 寄存器的 DDDn 位选择引脚的方向。如果 DDDn 设为 1，PDn 被配置为输出引脚；如果 DDDn 设为 0，PDn 被配置为输入引脚；如果 PORTDn 被设置为 1，DDDn 被配置为输入引脚，则 MOS 上拉电阻被触发。为了关断上拉电阻，PORTDn 位必须被清除或者引脚被配置为输出引脚。D 口引脚 DDDn 的作用见表 2.23。

表 2.23 D 口引脚的 DDDn 的作用

DDDn	PORTDn	I/O	上 拉	注 释
0	0	输入	否	三态(高阻)
0	1	输入	是	上拉低 PDn 脚输出电流
1	0	输出	否	推挽 0 输出
1	1	输出	否	推挽 1 输出

n：7，6，…，0 为引脚数。

三、D 口的第二功能

$\overline{RD}$ —PORTD，位 7

$\overline{RD}$是外部数据存储器读选通。

$\overline{WR}$ —— PORTD，位 6

$\overline{WR}$是外部数据存储器写选通。

OC1—— PORTD，位 5

OC1 表示比较匹配的输出。PD5 可以作为定时器/计数器 1 的比较匹配的外部输出。为了实现该功能，PD5 应被配置为输出(DDD5 设置为 1)。详细说明和怎样触发该输出，请见定时器/计数器 1 的描述。OC1 引脚还可以作为 PWM 模式下定时功能的输出。

INT1 —— PORTD，位 3

INT1 为外部中断源 1。PD3 引脚可作为 MCU 的外部中断源，详见中断部分。

INT0 —— PORTD，位 2

INT0 为外部中断 0。PD2 引脚可作为 MCU 的外部中断源，详见中断部分。

TXD —— PORTD，位 1

发送数据(UART 的数据输出引脚)，当 UART 数据输出允许时，该引脚被作为输出而不管 DDRD1 的值。

RXD —— PORTD，位 0

接收数据(UART 的数据输入引脚)，当 UART 数据输入允许时，该引脚被作为输出而不管 DDRD0 的值。当 UART 强制该引脚为输入时，PORTD0 的一个逻辑 1 将开启内部的上拉。

四、D 口原理图

D 口原理图如图 2.44～2.50 所示。注意所有的端口引脚都是同步的，图中未画出同步锁存器。

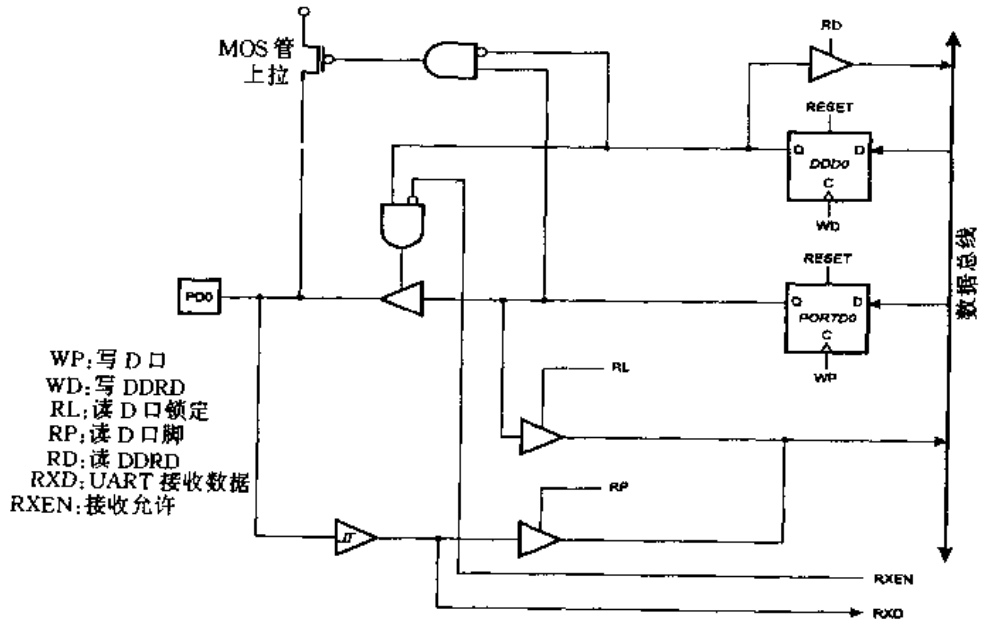


图 2.44　D 口原理图(PD0 脚)

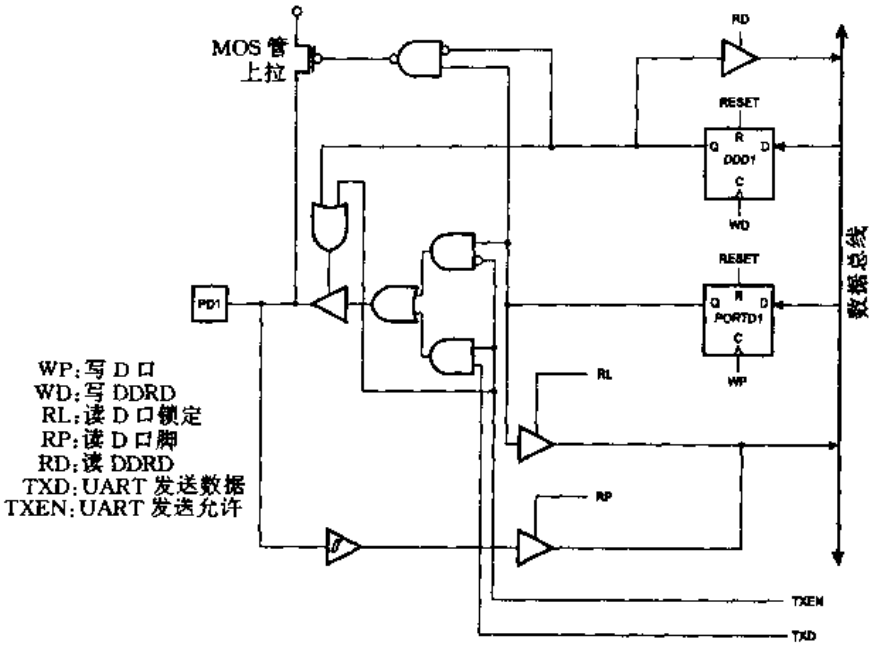


图 2.45 D 口原理图(PD1 脚)

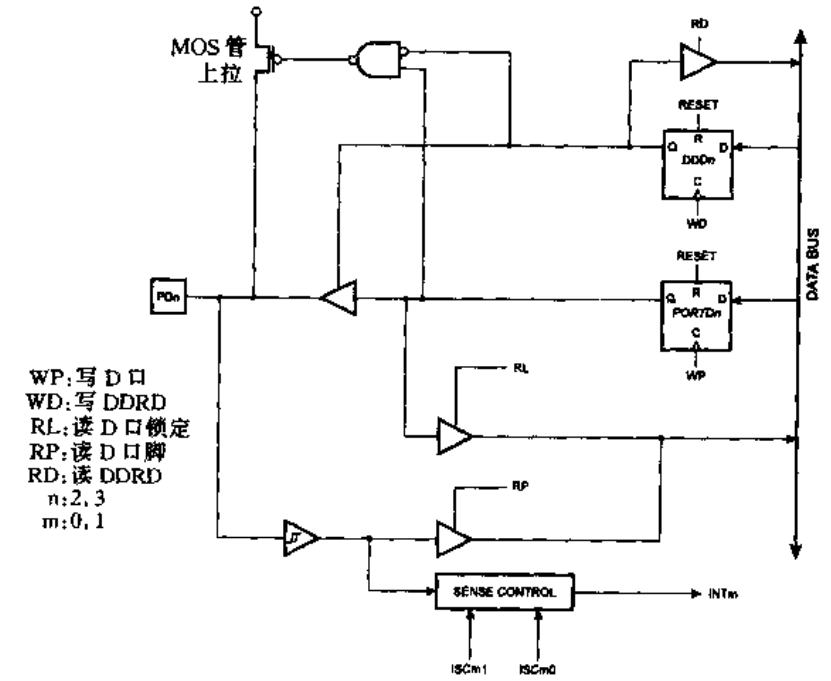


图 2.46 D 口原理图(PD2 和 PD3 脚)

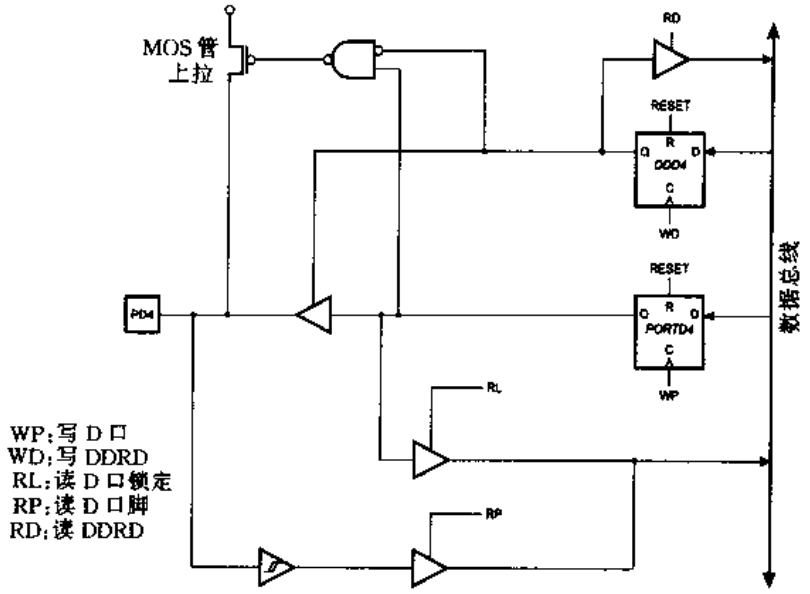


图 2.47　D 口原理图(PD4 脚)

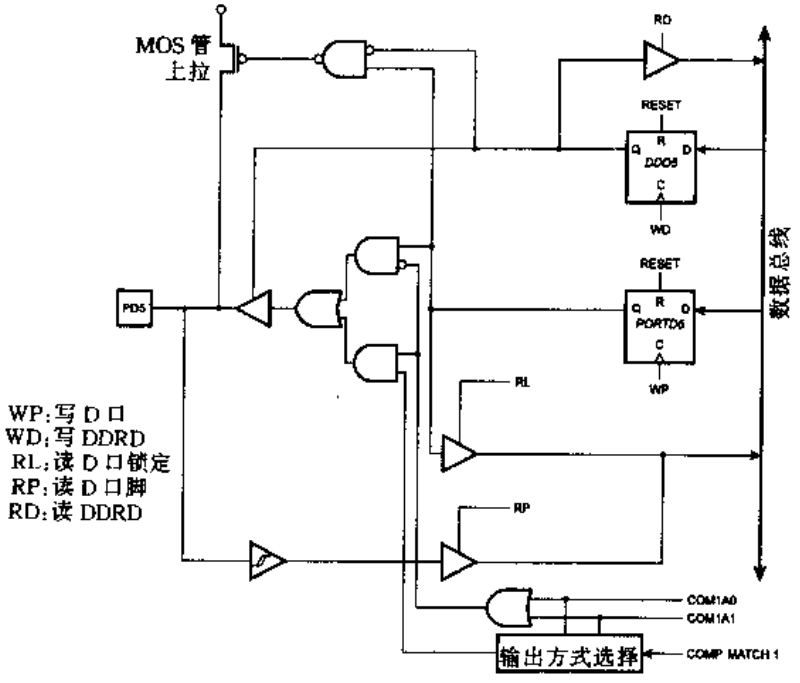


图 2.48　D 口原理图(PD5 脚)

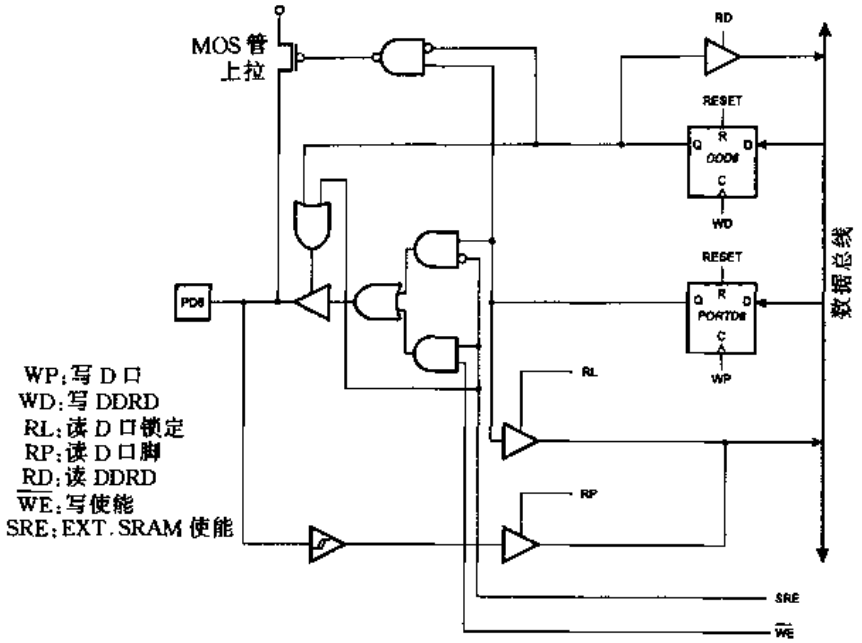


图 2.49 D 口原理图(PD6 脚)

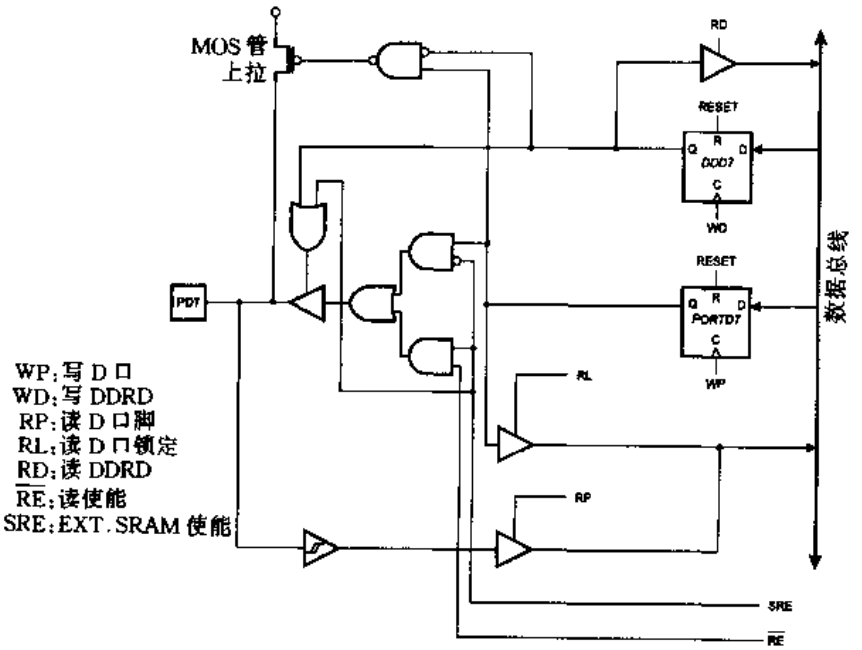


图 2.50 D 口原理图(PD7 脚)

2.12 AVR单片机存储器编程

2.12.1 编程存储器锁定位

AT90系列单片机MCU提供2个加密锁定位,可以不编程(1)或编程(0),而获得表2.24的额外特性。

表2.24 锁定位保护方式

编程锁定位			保护类型
模式	LB1	LB2	
1	1	1	无编程锁定特征
2	0	1	Flash的再编程被禁止
3	0	0	同2模式,但校验被禁止

注意:锁定位只能整片擦除时才能擦除。

2.12.2 熔断位

AT90系列单片机有两个熔断位,SPIEN和FSTRT。

当SPIEN被编程为0时,串行编程下载被允许,缺省值是被编程的(0)。

当FSTRT被编程为0时,选择短的启动时间,缺省值为擦除(1),定货时可以要求该位被编程。

这些位不能被串行编程模式访问,也不能被全片擦除所改变。

2.12.3 芯片代码

所有的ATMEL微控制器都有3字节的电子标签指定该器件的型号,该标签可以被串行模式和并行模式读出,这3个字节属于不同的地址空间。对于AT90系列单片机它们是:

(1) $000:$1E(指出厂商是ATMEL)。

(2) $001:$93(指出4K字节的Flash存储器)。

(3) $002:$01(当$001是$93时,指出是AT90S8515器件)。

2.12.4 编程Flash和EEPROM

AT90系列单片机提供了8K字节的系统在线可编程的Flash程序存储器和512字节的EEPROM数据存储器。

AT90系列单片机通常被售出时,片内的Flash程序存储器和EEPROM数据存储器阵列是被擦除的状态(即内容=$FF),而可以被编程。该器件支持高压(12V)并行编程模式和低压串行编程模式,+12 V电源仅用于编程触发,并不从该引脚获取意义上的电流。串行模式提供了对AT90系列单片机方便的在线程序和数据下载方式。

在两种模式下AT90系列单片机的程序和数据存储器阵列是按字节编程的。对于EEPROM,在串行编程模式中提供了一个自动擦除周期。

2.12.5 并行编程

该节描述了怎样并行编程和验证 AT90 系列单片机的 Flash 程序存储器、EEPROM 数据存储器及程序存储器的锁定位和熔断位。AT90S8515 的并行编程如图 2.51 所示。

一、信号名称

本节中 AT90 系列单片机的一些引脚被定义为描述并行编程的信号名称，表 2.25 中没有描述的引脚仍用引脚名来表示。

XA1/XA0 位决定了当 XTAL1 引脚给出一个正脉冲时的动作，这两位的设置见表 2.26。

当脉冲是$\overline{WR}$或$\overline{OE}$时，装入的命令决定了输入或输出的行为。该命令是一个字节，每一位的功能见表 2.27。

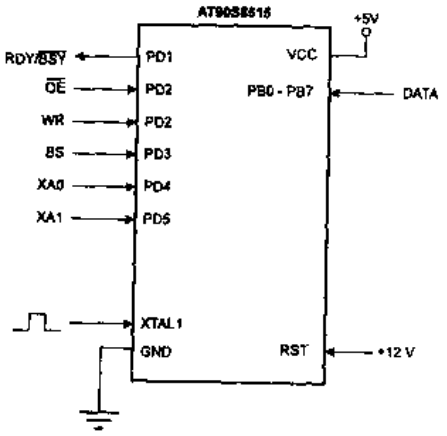


图 2.51 并行编程

二、进入编程模式

下列算法使器件进入并行编程模式：

(1) 在 V_{CC}和 GND 之间加上 4.5～5.5 V 电压。

(2) 把$\overline{RESET}$和 BS 设置为 0，并等待至少 100 ns。

(3) 把$\overline{RESET}$加到 12 V 且在改变 BS 之前至少等待 100 ns。

表 2.25 引脚名映射

编程方式的信号名	引脚名	I/O	功 能
RDY/$\overline{BSY}$	PD1	O	0：器件正在编程；1：器件就绪等待新的命令
$\overline{OE}$	PD2	I	输出允许(低激活)
$\overline{WR}$	PD3	I	写脉冲(低激活)
BS	PD4	I	字节选择
XA0	PD5	I	XTAL 用于位 0
XA1	PD6	I	XTAL 用于位 1

表 2.26 XA1 和 XA0 编程

XA1	XA0	当 Flash 被 PULSED 操作时
0	0	装入 Flash 或 EEPROM 地址(由 BS 确定的 Flash 高或低地址字节)
0	1	装入数据(由 BS 线确定的 Flash 高或低数据字节)
1	0	装入命令
1	1	闲置

表 2.27 命令字节位编码

位号	当设置时的定义
7	芯片擦除
6	写熔断位，在数据字节中定位下列位位置：D5，SPIEN 熔断丝；D0，FSTRT 熔断丝（注意，写“0”为编程，写“1”为擦除）
5	写锁定位，在数据字节中定位下列位位置：D7，LB1；D6，LB2；D5，SPIEN 熔断；D0，FSTRT 熔断（注意，“0”为编程）
4	写 Flash 或 EEPROM 中读（由位 0 测定）
3	读标记行
2	读锁定位和熔断位，在数据字节中定位下列位位置：D1，LB1；D0，LB2（注意，写“0”为编程）
1	从 Flash 或 EEPROM 中读（由位 0 测定）
0	0：Flash 访问；1：EEPROM 访问

三、全片擦除

全片的擦除功能将擦除 Flash 和 EEPROM 存储器和锁定位。锁定位在程序存储器被完全擦除之前不会被清除，熔断位并不改变。在编程之前必须执行一个全片擦除。

装载“全片擦除”命令：

(1)把 XA1 和 XA0 设置为(10)，触发命令装入。

(2)把 BS 设置为 0。

(3)设置 PB(7～0)为“1000 0000”。这是全片擦除命令。

(4)给 XTAL1 一个正脉冲，这将装入命令并且开始擦除 Flash 和 EEPROM 阵列。在 XTAL1 脉冲之后，给$\overline{WR}$一个负脉冲使得锁定位在擦除周期结束时被擦除。然后等待 10 ms 使擦除完成。全片擦除不生成 RDY/$\overline{BSY}$信号。

四、编程 Flash

装载“编程 Flash”命令：

(1)设置 XA1、XA0 为“10”，触发命令装入。

(2)把 BS 设为 0。

(3)把 PB(7～0)设为“0001 0000”。这是 Flash 编程命令。

(4)给 XTAL1 一个正脉冲，这将装入该命令。

装入地址低字节：

(1)设置 XA1、XA0 为“00”，触发地址装入。

(2)设置 BS 为 0，选择低位地址。

(3)设置 PB(7～0)＝低位地址字节($00～$FF)。

(4)给 XTAL1 一个正脉冲，这将装入地址低字节。

装入地址高字节：

(1)设置 XA1、XA0 为“00”，触发地址装入。

(2)设置 BS 为 1，选择高位地址。

(3)设置 PB(7～0)＝高位地址字节($00～$0F)。

(4)给 XTAL1 一个正脉冲,这将装入地址高字节。

装入数据字节:

(1)设置 XA1,XA0 为"01",触发数据装入。

(2)设置 PB(7~0)=数据低字节($00~$FF)。

(3)给 XTAL1 一个正脉冲,这将装入数据低字节。

写入数据低字节:

(1)设置 BS=0,选择数据低字节。

(2)给$\overline{WR}$一个负脉冲,开始编程数据低字节,RDY/$\overline{BSY}$为低电平。

(3)等待直到 RDY/$\overline{BSY}$变高时再编程另一个字节。

装入数据字节:

(1)设置 XA1、XA0 为"01",触发数据装入。

(2)设置 PB(7~0)=数据高字节($00~$FF)。

(3)给 XTAL1 一个正脉冲,这将装入数据高字节。

写入数据高字节:

(1)设置 BS=1,选择数据高字节。

(2)给$\overline{WR}$一个负脉冲,开始编程数据低字节,RDY/$\overline{BSY}$为低电平。

(3)等待直到 RDY/$\overline{BSY}$变高时再编程另一个字节。

装入的命令和地址在编程过程中保持不变,为了简化编程,请考虑如下:

- 编程 Flash 存储器的命令仅在编程第一个字节时需要被装入。
- 地址高字节仅在编程 Flash 中一个新的 256 字节页之前需要装入。

图 2.52 和图 2.53 为可编程 Flash 低字节和可编程 Flash 高字节。

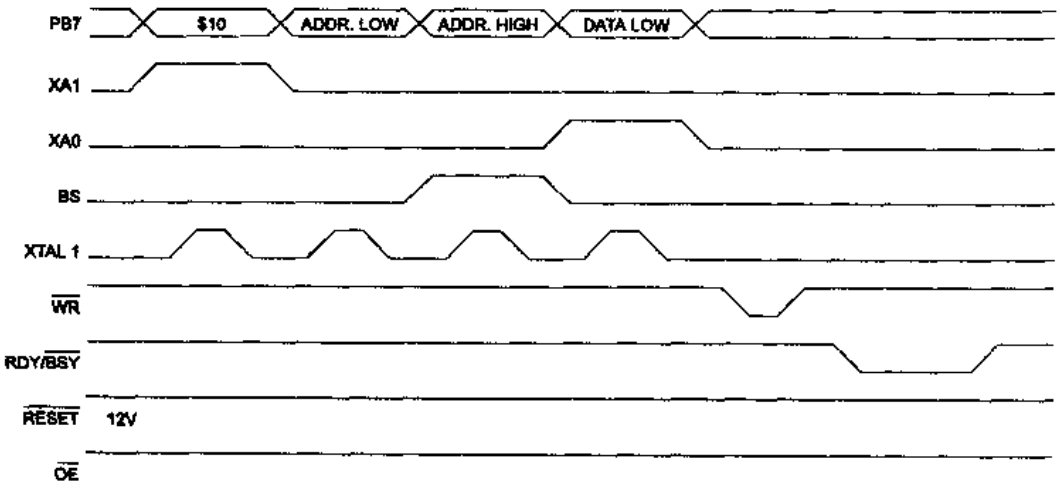


图 2.52 可编程 Flash 低字节

五、编程 EEPROM

编程 EEPROM 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1)装入命令"0001 0001"。

(2)装入低位 EEPROM 地址($00~$FF)。

(3)装入高位 EEPROM 地址($00~$01)。

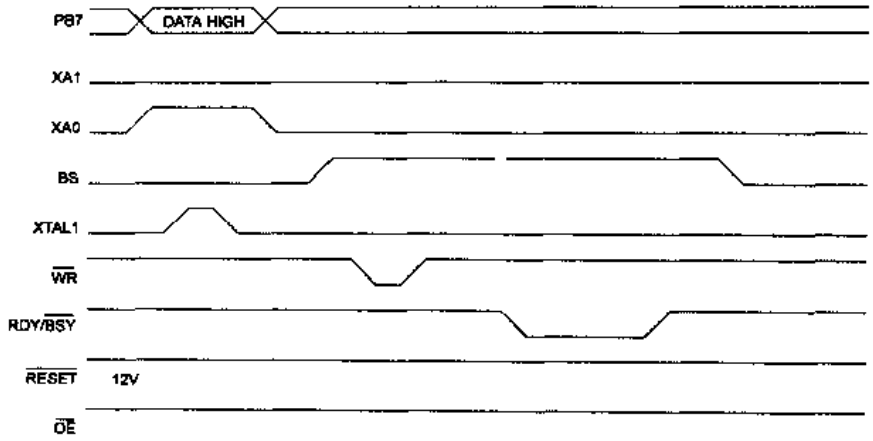


图 2.53 可编程 Flash 高字节

(4)装入低位 EEPROM 数据($00～$FF)。

(5)给$\overline{WR}$一个负脉冲并等待 RDY/$\overline{BSY}$变为高电平。

仅在编程第一个字节之前需要装入命令。

六、读 Flash

读 Flash 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1)装入命令"0000 0010"。

(2)装入低地址($00～$FF)。

(3)装入高地址($00～$0F)。

(4)设置$\overline{OE}$为 0,BS 为 0,这时数据低字节可从 PB(7～0)读出。

(5)设置 BS 为 1,这时数据高字节可从 PB(7～0)读出。

(6)设置$\overline{OE}$为 1。

仅在读第一个字节之前需要装入命令。

七、读 EEPROM

读 EEPROM 的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1)装入命令"0000 0011"。

(2)装入低地址($00～$FF)。

(3)装入高地址($00～$01)。

(4)设置$\overline{OE}$为 0,BS 为 0,这时数据低字节可从 PB(7～0)读出。

(5)设置$\overline{OE}$为 1。

仅在读第一个字节之前需要装入命令。

八、编成熔断位

编程熔断位的算法如下(参考 Flash 编程部分的命令、地址和数据的装载):

(1)装入命令"0100 0000"。

(2)装入数据:

位 5=0,编程 SPIEN 熔断位;位 5=1,擦除 SPIEN 熔断位。

位 0=0,编程 FSTRT 熔断位;位 0=1,擦除 FSTRT 熔断位。

(3)给$\overline{WR}$一个负脉冲,然后等待 RDY/$\overline{BSY}$变高。

注意：$\overline{WR}$必须保持低至少 1ms 的电平。

九、编程加密位

编程加密位的算法如下(参考 Flash 编程部分的命令、地址和数据的装载)：

(1) 装入命令“0010 0000”。

(2) 装入数据：

位 2=0，编程加密位 2；

位 1=0，编程加密位 1。

(3) 给$\overline{WR}$一个负脉冲，然后等待 RDY/$\overline{BSY}$变高。

加密位仅在全片擦除的时候被清除。

十、读熔断和加密位

读熔断和加密位的算法如下(参考 Flash 编程部分的命令、地址和数据的装载)：

(1) 装入命令“0000 0100”。

(2) 设置$\overline{OE}$为 0，BS 为 1，这时熔断和加密位的状态可从 PB(7～0)读出。

位 7：加密位 1(0 表示已被编程)。

位 6：加密位 2(0 表示已被编程)。

位 5：SPIEN 熔断位(0 表示已被编程，1 表示被擦除)。

位 0：FSTRT 熔断位(0 表示已被编程，1 表示被擦除)。

(3) 设置$\overline{OE}$为 1。

特别注意 BS 需要设置为 1。

十一、读电子标签字节

读电子标签字节的算法如下(参考 Flash 编程部分的命令、地址和数据的装载)：

(1) 装入命令“0000 1000”。

(2) 装入低位地址($00～$02)。

(3) 设置$\overline{OE}$为 0，BS 为 0，从 PB(7～0)可以读出选中的标签字节。

(4)设置$\overline{OE}$为 1。

仅在读第一个字节之前需要装入命令。

2.12.6 串行下载

一、串行下载

在$\overline{RESET}$接地时，所有的程序和数据存储器阵列都可以由串行 SPI 总线来编程。该串行接口包括引脚 SCK、MOSI(输入)、MISO(输出)。当$\overline{RESET}$设为低电平后，应先执行编程允许指令，再执行编程/擦除操作。

当编程 EEPROM 时，内部定时编程操作中包含了自动擦除周期(仅仅在串行编程模式下)，而无须先执行全片擦除指令。全片擦除指令把程序和数据存储器阵列的每一地址都变成 $FF。

程序和 EEPROM 存储器阵列的地址空间是分开的，程序存储器为 $0000～$0FFF，而 EEPROM 存储器为 $0000～$01FF。

可以用通过 XTAL1 提供的外部时钟，也可以在 XTAL1 和 XTAL2 之间加上一个晶振，串行时钟的(SCK)低电平和高电平的最小时间定义如下：

LOW：大于 1 个 XTAL1 时钟周期。

HIGH：大于 4 个 XTAL1 时钟周期。

二、串行编程算法

以串行的方式编程和校验 AT90 系列单片机，可用以下的算法（见表 2.28 的 4 字节指令格式）：

（1）上电过程：在 V_{CC} 和 GND 之间上电，同时$\overline{\text{RESET}}$和 SCK 设置为 0。（如果编程器不能保证 SCK 在上电期间为低电平，则在 SCK 为低电平时，$\overline{\text{RESET}}$必须给出一个正脉冲）。如果晶振没有被连到 XTAL1 和 XTAL2 上，则在 XTAL1 上加上 0～20MHz 的时钟。

（2）等待至少 20 ms，向 MOSI/PB5 送串行编程触发指令来触发串行编程。请参照上一节的串行时钟输入 SCK 的低电平和高电平的最小时间。

（3）如果执行了全片擦除（在擦除 Flash 时必须执行），等待 10 ms，给$\overline{\text{RESET}}$一个正脉冲，然后再从第（2）步开始。

（4）通过在相应的写指令中一齐提供地址和数据，可以一次把一个字节写入 Flash 和 EEPROM阵列中，EEPROM 存储器的每一地址在新数据写入之前被自动擦除。下一个字节 4ms 后再写入。

（5）任何存储器地址都可以通过读指令来校验，该读指令从串行输出 MISO/PB6 返回选中地址的内容。

（6）在编程结束时，$\overline{\text{RESET}}$可以设置为高电平来开始正常操作。

（7）下电过程（如果需要的话）：设置 XTAL1 为 0（如果未用晶振的话）；设置$\overline{\text{RESET}}$为 1；把 V_{CC} 断开。

表 2.28　串行可编程指令设置

指　令	指令格式				操　作
	Byte1	Byte2	Byte3	Byte4	
编程允许	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	在复位变低后，允许串行编程
芯片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	芯片擦除存储器阵列为 8K 或 512K 字节
读程序存储器	0010 h000	bbbb bbbb	bbbb bbbb	oooo oooo	从字地址 a:b 处的程序存储器读 h（高或低）数据
写程序存储器	0100 h000	xxxx aaaa	bbbb bbbb	iiii iiii	写 h（高或低）数据 i 到从字地址 a:b 处的程序存储器中
读 EEPROM 存储器	1010 0000	xxxx aaaa	bbbb bbbb	oooo oooo	从地址 a:b 处的 EEPROM 中读数据
写 EEPEOM 存储器	1100 0000	xxxx xxx0	bbbb bbbb	iiii iiii	写数据 i 到地址 a:b 处的程序存储器中
写锁定位	1010 1100	111x x21x	xxxx xxxx	xxxx xxxx	写锁定位，置位 1，2＝'0'到程序锁定位
读器件代码	0011 0000	xxxx xxxx	xxxx xxxb	oooo oooo	从地址 b 处读器件代码

注：a＝高位地址；b＝低位地址；h＝0 为低字节，h1 为高字节；o－数据输出；i＝数据输入；

x＝任意；1＝加密位 1；2－加密位 2。

2.12.7　可编程特性

当把串行数据写入 AT90 系列单片机时，数据在 CLK 的上升沿被输入。

当从 AT90 系列单片机中读数据时，数据在 CLK 的下降沿输出，见图 2.54 的解释。图 2.55 是串行可编程和校验示意图。

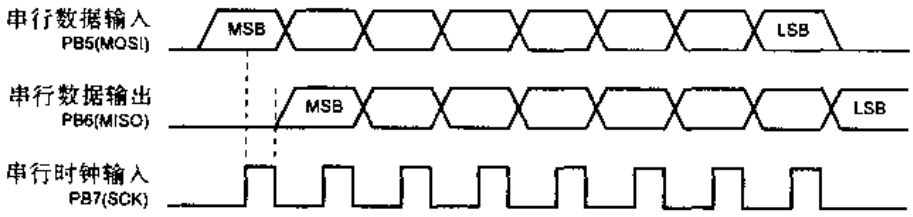


图 2.54 串行下载波形图

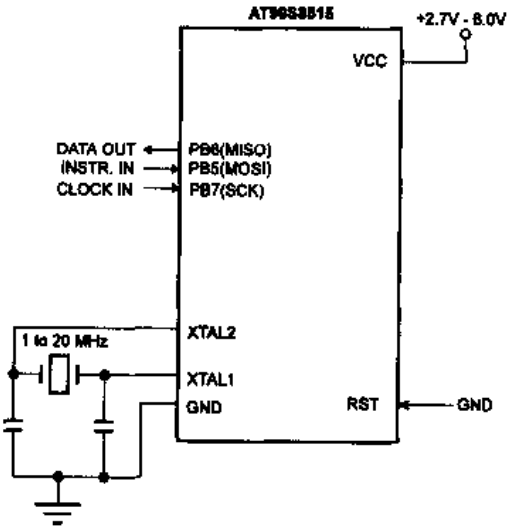
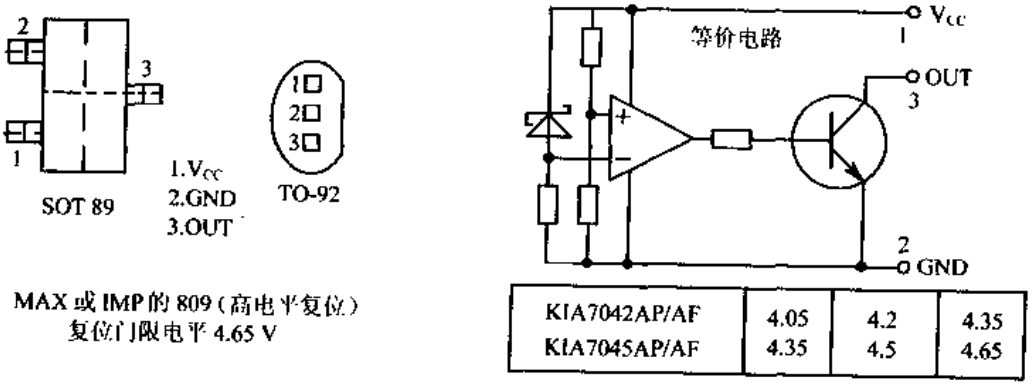


图 2.55 串行可编程和校验

图 2.56 是复位检测器件引脚接线图。



KIA7042AP/AF	4.05	4.2	4.35
KIA7045AP/AF	4.35	4.5	4.65

图 2.56 复位检测器件引脚接线图

第三章　AVR 单片机开发工具

AVR 的开发工具可分为硬件仿真器、软件模拟仿真器(开发实验器/评估板),均在 AVR 集成开发环境(IDE)下工作,有多种 C 高级语言支持。

3.1　AVR 实时在线仿真器 ICE200

为了使 AVR 单片机在我国迅速得到应用,广州市天河双龙电子有限公司及时引进美国原装 AVR 实时在线仿真器 ICE200,见图 3.1。它可在线仿真的 AVR 器件有 ATtiny10/11/12(V/L)、AT90S1200/2313、AT90(L)S2333/4433、AT90S4414/8515、AT90(L)S4434/8535。由于仿真器的电源不对外,所以 ICE200 也支持低电压器件。ICE200 采用 AVR 专用仿真 CPU 与监控 CPU 独立设计的方案,充分提供各种调试手段,真实再现被仿 AVR 的各种特性。

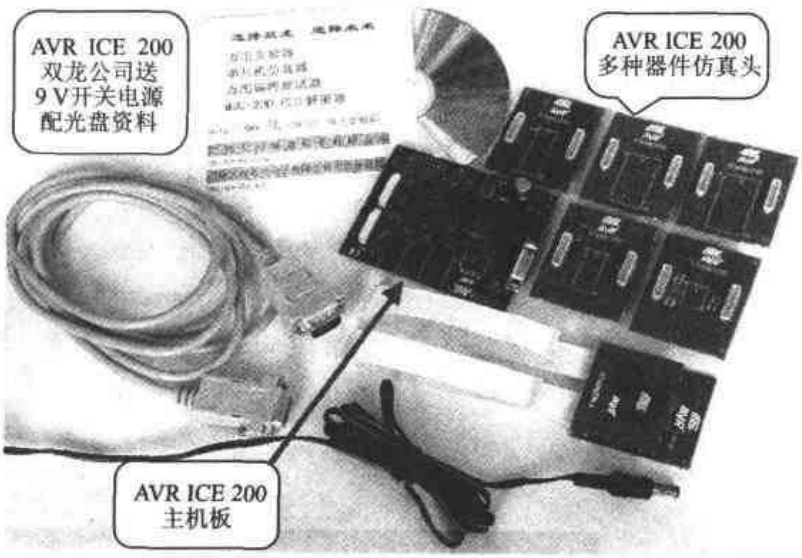


图 3.1　实时在线仿真器 ICE200

ICE200 的仿真软件最新版集成环境(IDE)为 Stuido 3.5X。在支持以上 11 种 AVR 以外,还可模拟其它 AVR 器件的运行,支持汇编、C 高级语言调试,JTAG ICE 仿真及程序下载等操作。其中,汇编级编译器免费提供;C 编译器只提供 Image Craft Inc. 的 30 天免费试用版 ICCAVR demo,30 天后转为 2KB 限止版。该软件及其升级版均可从互联网(www.image-craft.com)上免费获得。

3.2　JTAG ICE 仿真器

AVR 高档单片机 ATmega 系列 ATmega16/32/64/128/323 等具有 JTAG 仿真接口,可

以实现在线实时仿真、编程下载。这在 8 位单片机中属于领先技术。JTAG ICE 仿真器，见图 3.2。JTAG ICE 与用户机 AVR 的接线图，见图 3.3。JTAG ICE 仿真，对不同封装器件可以先焊接，然后进行实时仿真、编程。这比用传统的芯片仿真头，大大降低了仿真器价格。

图 3.2 JTAG ICE 仿真器

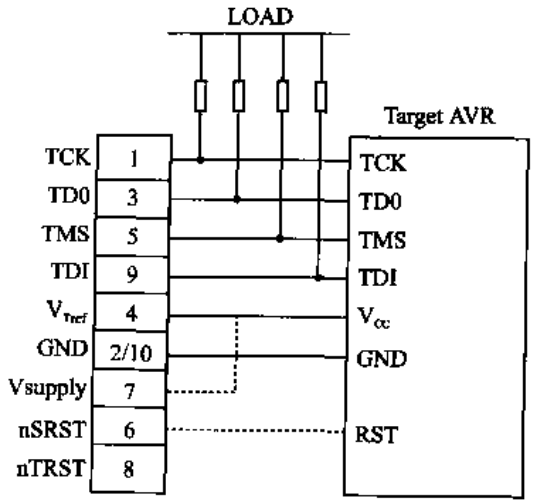


图 3.3 JTAG ICE 与用户机 AVR 的接线

在图 3.3 中，虚线为用户板合用 V_{CC}、RST；LOAD 电阻为用户板上拉电阻。

3.3 AVR 嵌入式单片机开发下载实验器 SL-AVR

“四合一”SL-AVR 等于 AVR 编程器＋模拟仿真器＋实验器＋科研样机，见图 3.4。它的硬

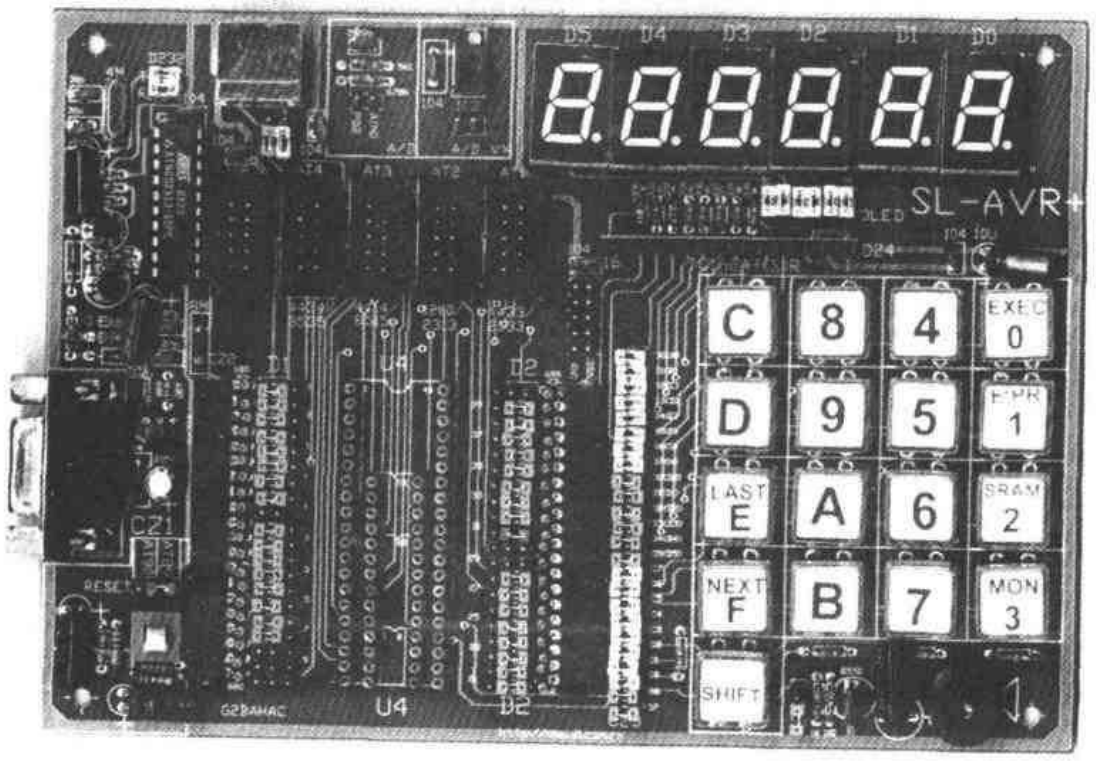


图 3.4 SL-AVR＋图

件采用模块化设计，便于用户灵活组成科研项目所需的各种硬件结构。硬件有：RS232 通信接口；串行下载监控；DIP8/20/28/40 通用锁紧插座，DIP40 端口用短路块连接作输出，用 LED 发光二极管显示器件引脚高低电平，也可用短路块断开作输入或其它用途；6 位 LED 数码管作显示；2×16 点阵 LCD 液晶显示器；17 键的键盘；模拟比较输入电路；音响电路；单片机复位电路；模拟电压输入电路等。随机附 120mm×170mm 万通实验板及一片 AT90S8515 器件。

SL-AVR 适用于所有具有串行 ISP 下载编程功能的 AVR 单片机，用户板上的器件无需拆下即可编程，同时还可做 AVR 单片机的 I/O 口、A/D、D/A、LED、LCD、键盘输入、音频输出、模拟比较等开发实验。本机提供功能强大的 ATMEL AVR Studio 3.53 以上集成开发环境(IDE)，包括：① AVR Assembler 编译器；② AVR Studio 调试功能；③ AVR Prog 串行、并行下载功能；④ JTAG ICE 在线仿真等功能。同时，也提供限时版的 ICC AVR C 编译器。对初学 AVR 单片机的设计者，可暂时节省购买较昂贵的实时仿真器及万用编程器的费用。新版 SL-AVR 下载插座 AT1～AT4 改为 DC10 插座；也可使用 SL-AVRISP 并行高速下载线，对大容量器件 ISP 下载编程带来方便。本机工作晶振可插拔更换，便于用户超频、降频实验。SL-AVR 开发实验器提供的几十个实用实验程序，用户也可改变硬件接口、修改程序，实现源程序的其它功能。这对大专院校学生发挥其创造性思维及动手能力的培养都特别有用，可改变我国传统教育下的"高分低能"的弊病。本开发实验器也可当做科研样机使用。

SL-AVR 硬件接口电路，见图 3.5。各器件说明如下：

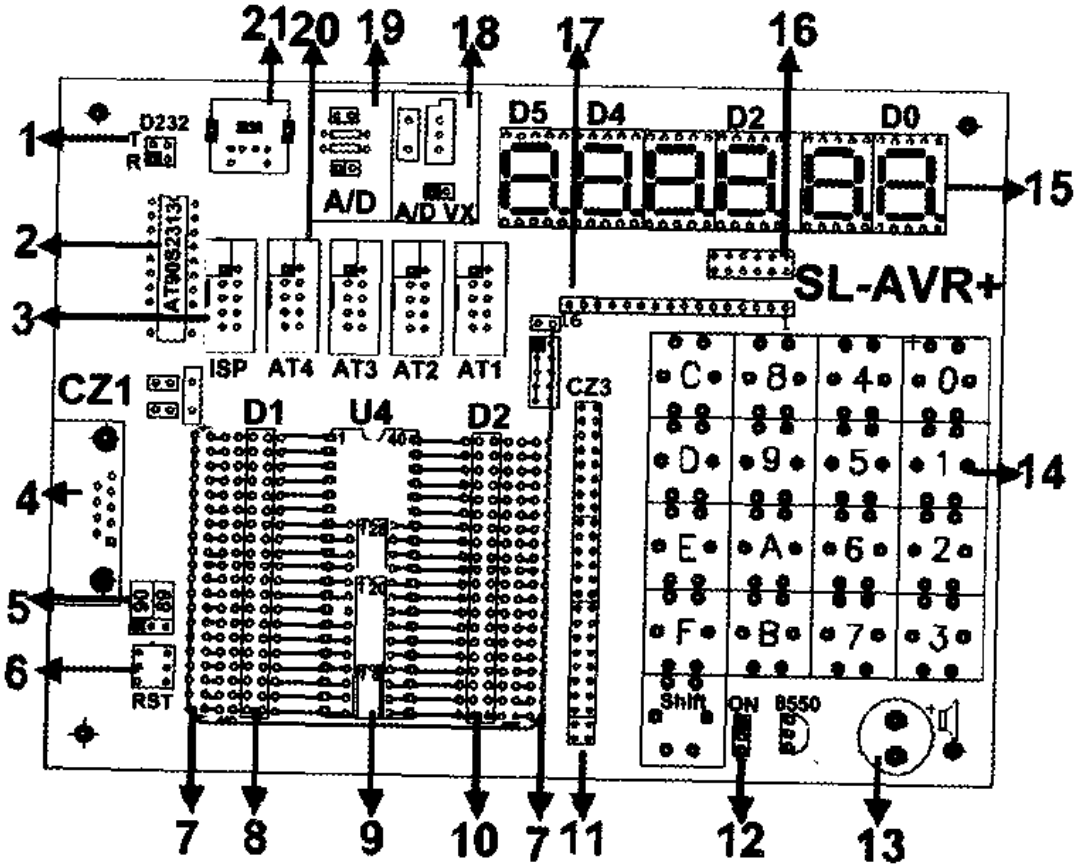


图 3.5　SL-AVR+硬件接口电路

1. D232

D232 通信线短路块：插上短路块，接到 AT90S1200；拔出短路块，可从 T、R 端用插针线接到单片机通信口，做单片机异步通信 UART 或主从同步通信 SPI 或 ISP 下载通信。

2. AT90S2313

AT90S2313 为 AVR 开发实验器监控芯片。

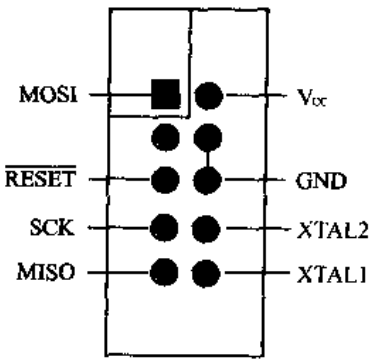


图 3.6 ISP 信号插座引脚功能

3. ISP

该列 DC10 的（ISP）插座，即 AVR 单片机的下载信号插座，见图 3.6。引脚功能分别为 V_{CC}，GND，XTAL2，XTAL1，MOSI，RESET，SCK，MISO。随机附有一条 10 线信号线，由用户接插到对应 AVR 单片机（AT1～AT4）的信号脚上。ISP 也可接到用户板作 AVR 单片机（或用转接线）的串行下载编程用。如用户板有晶振，则 XTAL1/XTAL2 两信号线无需接出。

本开发实验器配一片 AT90S8515 器件，绝大多数实验使用该器件，硬件（用短路块）连接出厂时也按该器件连接，其它器件作为选购件。

4. CZ1

电源及通信下载插座，电源线为地及＋5V，通信电缆一头接 CZ1，另一头接计算机 RS232 九针插座。

POWER：红色 LED，为电源指示；BUSY：绿色 LED，为下载通信工作忙指示。

5. 复位选择

用短路块选择 AVR 还是 AT89S 单片机做开发下载实验。

6. 单片机 RST 复位按钮

作为程序下载后再启动复位用，一般情况程序下载结束或开机通电时程序就自动执行。

7. LED 发光二极管

低电平有效，当 I/O 口输出指示。

8. D1 短路块列

使单片机引脚作输出或输入用。插上短路块引脚作输出，用 LED 显示高低电平，低电平 LED 灯亮；不插短路块，器件引脚可作输入用或作其它用途，如电源引线等。

警告：① D1、D2 短路块中如有对应器件的晶振脚或复位脚应拔出插短路块！

② AT90S4433/2333/4434/8535/mega103 等器件下载时把 DLED 短路块拔出。因为这些器件的晶振脚或复位脚相连，电平被 LED 电阻拉低了，会造成下载出错！

9. U4

作为 AVR 单片机四种 DIP（DIP8/20/28/40）封装器件下载插座。器件插放时，缺口向上，向下插座底部对齐。

10. D2 短路块列

D2 短路块列作用同 D1 短路块列。

11. CZ3 接线排针

为 AT90S8515 对应引脚 PB0～PB7，PD0～PD5，PC7～PC0。插上短路块，接到键盘（17

键)和LED/LCD显示器;拔出短路块,用接插线可把键盘与显示改为其它I/O口。也可改用其它器件(如AT90S2313/8535等)接到器件相应引脚上(D1、D2短路块处)。

12. SPEAKER

音响输入端,接相应器件的右下脚(5/11/15/21)。插上短路块,接通AT90S8515的PC0的音响信号;也可用接插线接到单片机任何I/O脚上作音响输出。

13. 无源音响器

由单片机发出音响。

14. 17键键盘模块

接AT90S8515的PC口,16个数字键;另一个SHIFT换档键,接AT90S8515的PD7。当先按下SHIFT键,再按数字键,即实现数字键上的命令键功能。当然,根据程序的要求,这些数字的名称可以重新定义,不妨一试。

15. LED模块

由6只数码管组成,D5～D2作地址,D1～D0作数据。数码管字位口对应接AT90S8515的PD5～PD0口;数码管的字形(abcdefgh)对应接AT90S8515的PB7～PB0口。

16. DLED短路块

LED位线,用DLED短路块连接;断开短路块,位线也可作它用途。注意:固化AT90S4433/8535/mega103等,应拔出短路块DLED!

17. CZ4接线座

为2行X16字LCD液晶显示模块插座,用AT90S8515的PB、PD口。

18. A/D VX

多圈电位器作为模拟信号输入用,两边分别接上V_{cc}、GND,中间头(VX)用连接线接到单片机作模拟信号输入脚(如AIN1)。

19. A/D

为片内模拟比较器,作A/D转换外部元件电路(最好R精度为1%,C精度为5%,所接阻值仅供参考),AIN0、PD2为接线端。

20. AT1～AT4

对应器件下载插座。

AT1插针座为AT90S4433/AT90S2333/ATmega8 ISP下载信号线座;

AT2插针座为AT90S1200/2313 ISP下载信号线座;

AT3插针座为AT90S4414/8515,AT89S8252/53 ISP下载信号线座;

AT4插针座为AT90S4443/8535 ISP下载信号线座。

如用于其它AVR单片机,用转接线接到其它AVR单片机下载信号引脚上。

21. PC机键盘插座

PD2接小口键盘。

为了便于用户能使用与自己工作样机相一致的晶振及提高SL-AVR工作可靠性,特别对晶振设置作了改进。在SL-AVR工作时,随机所附晶振接插小印制板(附插8 MHz晶振),应插在对应器件的晶振引脚上(对应器件旁插针)。晶振数值可根据实际工作改变(如超频、降频等)。

图3.7为SL-AVR+电原理图。

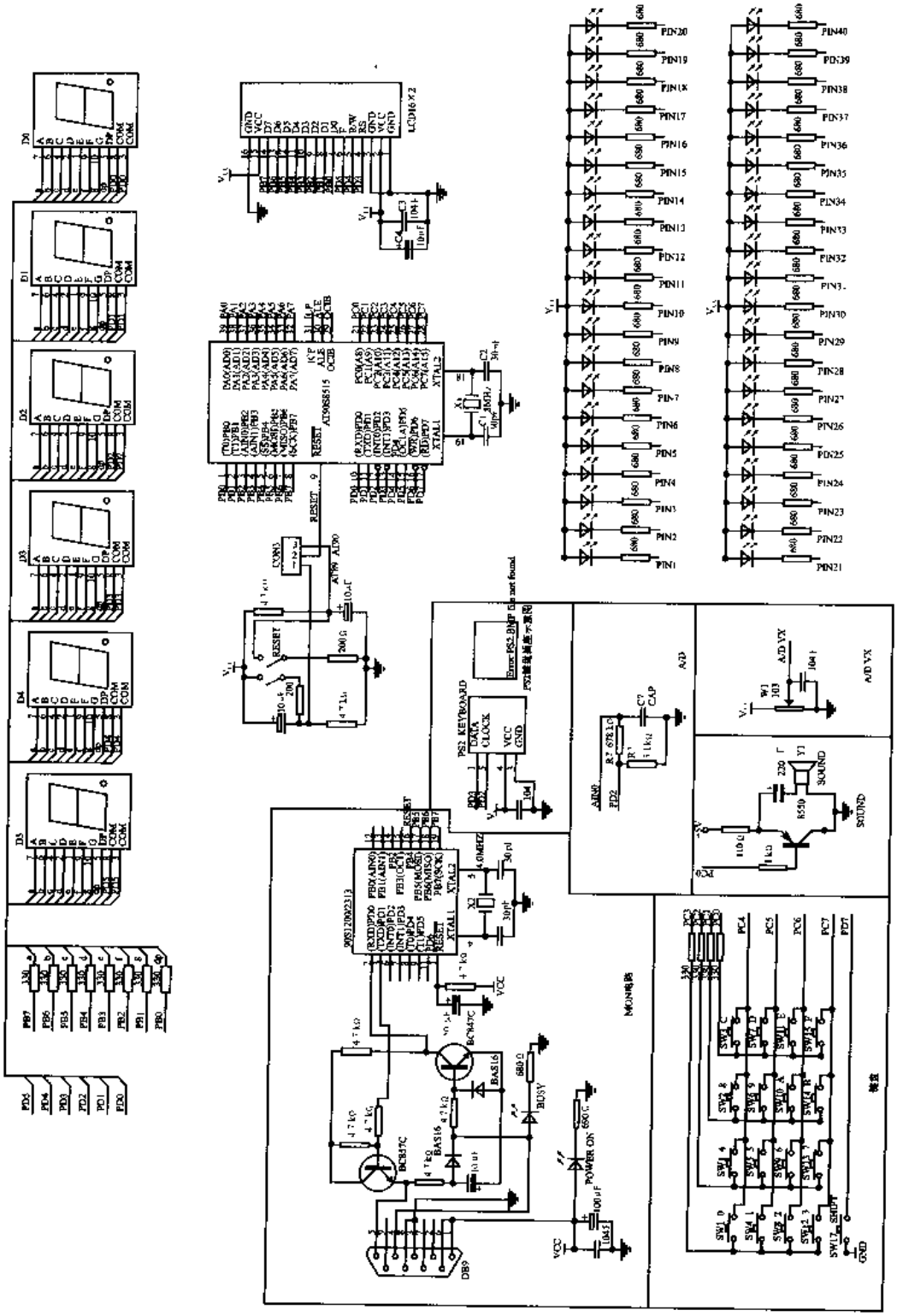


图 3.7 SL－AVR＋电原理图

3.4　AVR集成开发环境(IDE)

ATMEL AVR Studio 3.53集成开发环境(IDE)，包括：① AVR Assembler编译器；② AVR Studio调试功能；③ AVR Prog串行、并行下载功能；④ JTAG ICE仿真等功能。AVR IDE安装后，双击AVR Studio图标，则出现AVR Studio 3.53集成开发环境窗口，见图3.8。

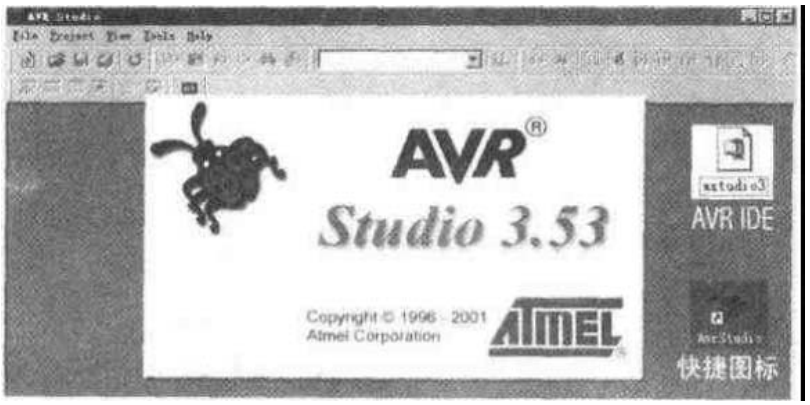


图3.8　AVR Studio 3.53集成开发环境窗口

3.4.1　AVR Assembler编译器

有源文件编辑、汇编(生成.OBJ/.HEX/.LIS文件)、搜寻、选项(生成汇编文件格式)、窗口、帮助等操作；若汇编出错则有错误定位、错误指示，便于源文件排错。

1. 建立工程项目

① Project→New→出现Select new project窗口，新建工程项目；② 必须选择工程项目名字和项目的类型输入，例：SL.APR工程项目(.APR也可缺省，则默认为.APR)；③ 选择存放工程项目路径；④ 用鼠标选中AVR Assembler汇编；⑤ 再按OK键，即自动新建工程项目，见图3.9。

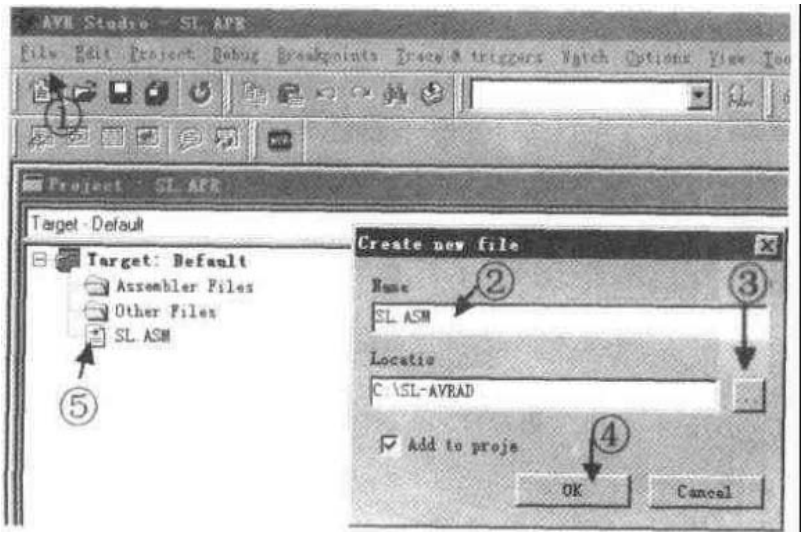


图3.9　建立工程项目

2. 打开已保存的工程项目

File→Open 选择路径，打开已保存的工程项目。

3. 新建汇编文件名

① File→New text file→出现新建文件窗口 Create new file；② 输入汇编文件名，例 SL. ASM；③ 选择存放路径；④ 按 OK 键，即源文件添加到工程项目中，并出现新的源文件编辑窗口，可编辑新的源程序，见图 3.10。

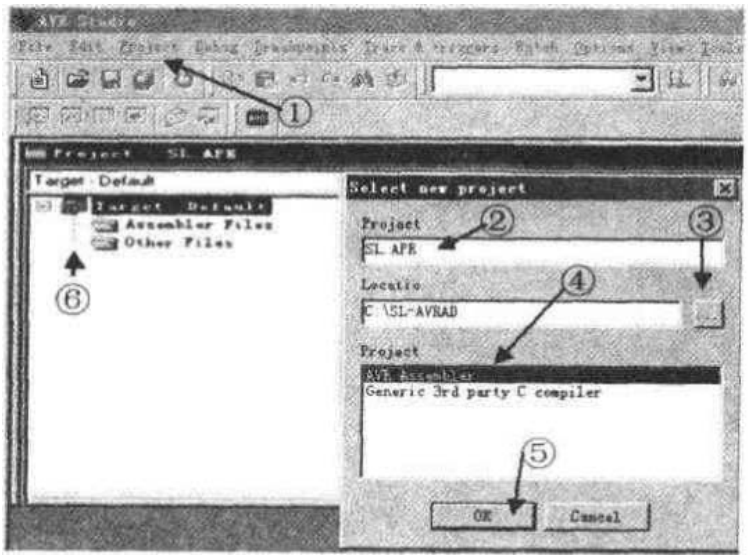


图 3.10 新建汇编文件名

4. 打开已保存的汇编文件

File→Open 选择路径，打开已保存的汇编文件。

5. 源文件编译选项

① 选 Project 菜单；② 选 Project Settings 编译项目设置，出现 AVR Assembler Options 选择窗口；③ 选 Output file 中 Intel Intellec8/MDS... 生成 Intel 格式 hex 文件；④ 按 OK 键，设置完成，见图 3.11。

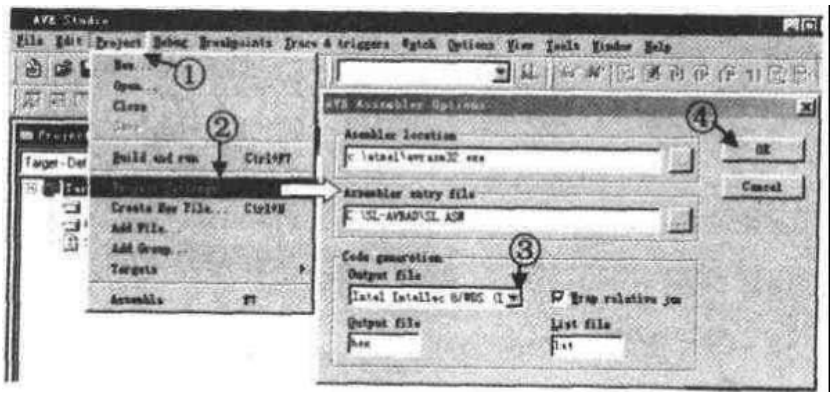


图 3.11 源文件编译选项

6. 新文件的编辑与编译

① 窗口输入源文件 SL. ASM；② 选 Project 菜单；③ 选 Project 下拉菜单的 Assemble 编译选项，进行编译；④ 编译通过，显示编译提示，见图 3.12。

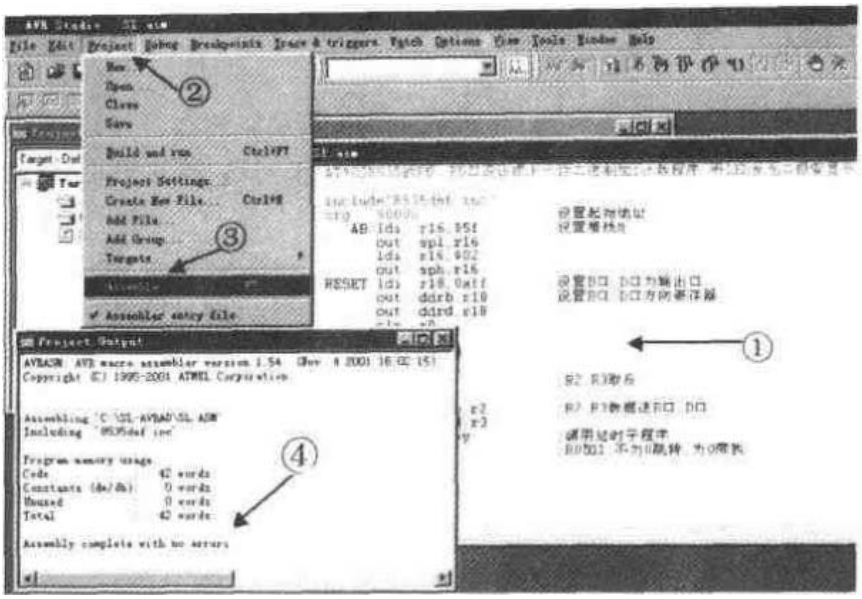


图 3.12　新文件的编辑与编译

3.4.2　AVR Studio

对源文件 DBUG 调试(装入.OBJ 目标文件,以源文件格式显示调试;如装入.HEX 文件,以反汇编格式显示地址、机器码、指令等格式显示调试)、排错、断点、单步、自动单步、触发、注视、选项、查看、窗口、帮助等操作;调试中可打开多种窗口,有 I/O 窗口、源文件窗口、CPU 窗口、记录窗口、数据窗口等,见"AVR 集成软件调试窗口"的图示。

选择菜单 Debug→G0 进入 Debug 调试选项窗口:① 源文件调试窗口;② 打开 Processor 观察窗口,可观察 CPU 的各种参数变化情况;③ 打开 Standard 观察窗口,可观察 I/O 口电平变化情况;④ Debug 调试快捷按钮;⑤ 观察窗口快捷按钮。AVR Studio 调试窗口见图 3.13。根据源程序调试要求,还可打开更多的观察窗口。

图 3.13　AVR Studio 调试窗口

3.4.3 AVR Prog

串行下载软件，当用户接通下载线，一头接 PC 机 RS232 串行口，另一头接 SL－AVR 下载开发实验器，并接上 5V 电源(红色线接＋5V，黑色线接地)。若连机正确，则出现下载提示窗口，按提示窗口要求操作。

1. AVR ISP/JTAG 编程

启动 AVR Stuio 3.53 程序。如果对芯片编程，可以启动 TOOLS 菜单中的 STK500/AVR ISP/JTAG ICE，进入串行下载操作窗口。如图 3.14 所示，根据窗口提示操作。

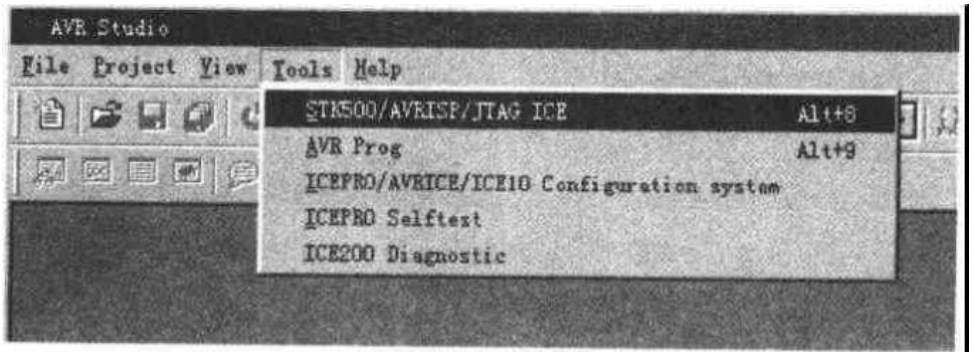


图 3.14 进入 AVR ISP/JTAG ICE

2. AVR Prog 串行编程

启动 AVR Stuio 3.53 程序。如果对芯片编程，可以启动 TOOLS 菜单中的 AVR Prog 或 Alt＋9 按钮，进入串行下载操作窗口，如图 3.15 所示，根据窗口提示操作。双龙光盘里另外提供串行、并行汉化 ISP 下载软件。

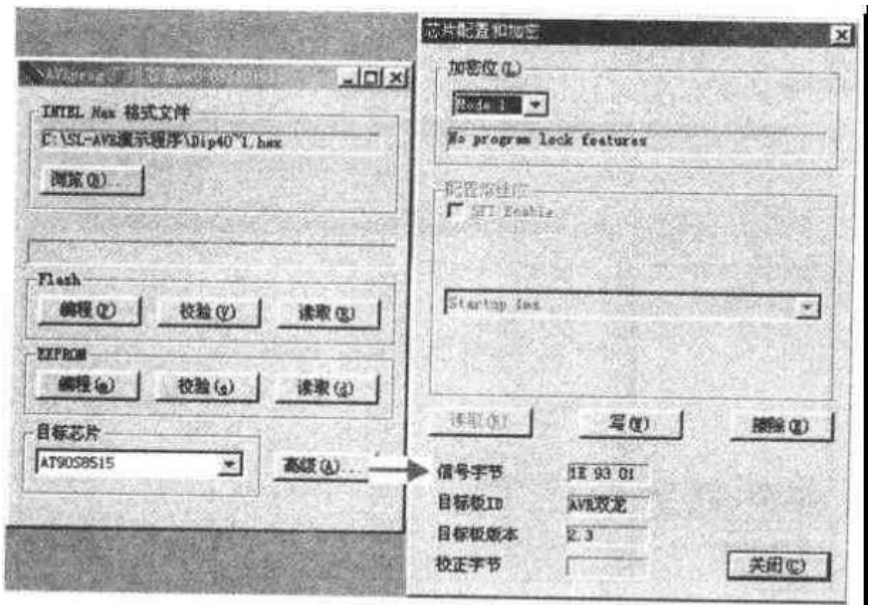


图 3.15 AVR ISP 下载操作窗口

3. AVR 单片机串行下载操作

(1) 在光盘 * :\双龙工作软件\AVR\AVR PROG 下，双击图标  进入串行下载窗口，也可把图标移到桌面成快捷菜单，双击图标进入串行下载操作窗口，如图 3.15 所示，根据窗口提示操作。

(2) 按图 3.14 单击 AVR Prog 或同时按 PC 机 Alt+9 按键，进入 ISP 下载操作窗口，如图 3.15 所示，根据窗口提示操作。

注意：SL-AVR 编程开发实验器，必须在通电情况下，并连接 PC 机，才会出现 ISP 下载操作窗口！

3.5　SL-AVR 系列组态开发实验系统

SL AVR 系列组态开发实验系统，指双龙电子的 SL-AVR/SL-AVR+ /SL-AVRAD 等在 AVR 单片机开发实验器上，可以增加组态系统软件和实验器组态监控，变为 AVR 单片机组态开发实验系统(选购件)。这样充分利用 PC 机资源，使 AVR 单片机教学以生动、形象、直观、有声有色地了解单片机 I/O 口基本功能、I/O 口的扩展功能、A/D 采样显示及单片机组态开发的复杂应用。单片机组态开发实验系统，使用户在娱乐中学习单片机，也可以使单片机启蒙学习从中小学开始，循序渐进。SL-AVR 单片机组态互动演示实验系统，如 PC 机屏幕控件开关，可控制单片机开发实验器上的 LED 灯，单片机实验器上的开关；也可控制 PC 机屏幕上的控件灯，当然 PC 机上控件开关也可控制 PC 机上的控件灯；还可学习工控组态方法，使单片机开发学习与现实社会工程应用接轨。

MCGS 组态系统是一套基于 Microsoft Windows95/98 和 Microsoft Windows NT 平台，用于快速构造和生成上位机监控系统的组态软件系统。它为用户提供了从数据采集到数据处理、报警处理、流程控制、动画显示、报表输出等解决实际工程问题的完整方案和操作工具。

MCGS 的主要功能和特性如下：

● 概念简单，易于理解和使用。普通工程人员经过短时间的培训就能正确掌握、快速完成多数简单工程项目的监控程序设计和运行操作。用户可避开复杂的计算机软硬件问题，集中精力解决工程本身的问题。按照系统的规定，组态配置出高性能、高可靠性、高度专业化的上位机监控系统。

● 功能齐全，便于方案设计。MCGS 为解决工程监控问题提供了丰富多样的手段，从设备驱动(数据采集)到数据处理、报警处理、流程控制、动画显示、报表输出、曲线显示等各个环节，均有丰富的功能组件和常用图形库可供选用。用户只需根据工程作业的需要和特点，进行方案设计和组态配置，即可生成用户应用软件系统。

● 实时性与并行处理。MCGS 充分利用了 Windows 操作平台的多任务、按优先级分时操作的功能，使 PC 机广泛应用于工程测控领域成为可能。工程作业中，大量的数据和信息需要及时收集、即时处理，在计算机测控技术领域称其为实时性关键任务。如数据采集、设备驱动和异常处理等。另外，许多工作则是非实时性的，或称为非时间关键任务。如画面显示，可在主机运行周期时间内插空进行。而像打印数据一类的工作，可运行于后台，称为脱机作业。MCGS 是真正的 32 位系统，可同时运行于 Microsoft Windows95/98 和 Microsoft Windows

NT 平台，以线程为单位进行分时并行处理。

● 建立实时数据库，便于用户分部组态，保证系统安全可靠运行。MCGS 由五部分组成，其中的“实时数据库”是整个系统的核心。在生成用户应用系统时，每一部分均可分别进行组态配置，独立建造，互不相干；而在系统运行过程中，各个部分都通过实时数据库交换数据，形成互相关联的整体。实时数据库是一个数据处理中心，是系统各个部分及其各种功能性构件的公用数据区。各个部件独立地向实时数据库输入和输出数据，并完成自己的差错控制。

● 设立“设备工具箱”。针对外部设备的特征，用户从中选择某种“构件”，设置于设备窗口内，赋予相关的属性，建立系统与外部设备的连接关系，即可实现对该种设备的驱动和控制。不同的设备对应于不同的构件，所有的设备构件均通过实时数据库建立联系；而建立时又是相互独立的，即对某一构件的操作或改动，不影响其它构件和整个系统的结构。从这一意义上讲，MCGS 是一个与“设备无关”的系统，用户不必因外部设备的局部改动，而影响整个系统。

● “面向窗口”的设计方法，增加了可视性和可操作性。以窗口为单位，构造用户运行系统的图形界面，使得 MCGS 的组态工作既简单直观又灵活多变。用户可以使用系统的缺省构架，也可以根据需要自己组态配置，生成各种类型和风格的图形界面，包括 DOS 风格的图形界面、标准 Windows 风格的图形界面以及带有动画效果的工具条和状态条。

● 利用丰富的“动画组态”功能，快速构造各种复杂生动的动态画面。以图像、图符、数据、曲线等多种形式，为操作员及时提供系统运行中的的状态、品质及异常报警等有关信息。用变化大小、改变颜色、明暗闪烁、移动翻转等多种手段，增强画面的动态显示效果。图元、图符对象定义相应的状态属性，即可实现动画效果。同时，MCGS 为用户提供了丰富的动画构件、模拟工程控制与实时监测作业中常用的物理器件的动作和功能。每个动画构件都对应一个特定的动画功能，如：实时曲线构件、历史曲线构件、报警显示构件、自由表格构件等。

● 引入“运行策略”的概念。复杂的工程作业，运行流程都是多分支的。用传统的编程方法实现，既繁琐又容易出错。MCGS 开辟了“策略窗口”，用户可以选用系统提供的各种条件和功能的“策略构件”，用图形化的方法构造多分支的应用程序，实现自由、精确地控制运行流程。按照设定的条件和顺序，操作外部设备，控制窗口的打开或关闭，与实时数据库进行数据交换。同时，也可以由用户创建新的策略构件，扩展系统的功能。

● MCGS 系统由五大功能模块组成，主要的功能模块以构件的形式来构造。不同的构件有着不同的功能，且各自独立。三种基本类型的构件（设备构件、动画构件、策略构件）完成了 MCGS 系统三大部分（设备驱动、动画显示和流程控制）的所有工作。用户也可以根据需要，定制特定类型的构件，使 MCGS 系统的功能得到扩充。这种充分利用“面向对象”的技术，大大提高了系统的可维护性和可扩充性。

● 支持 OLE Automation 技术。MCGS 允许用户在 Visual Basic 中操作 MCGS 中的对象，提供了一套开放的可扩充接口，用户可根据自己的需要用 VB 编制特定的功能构件来扩充系统的功能。

● MCGS 中数据的存储不再使用普通的文件，而是用数据库来管理一切。组态时，系统生成的组态结果是一个数据库；运行时，数据对象、报警信息的存储也是一个数据库。利用数据库来保存数据和处理数据，提高了系统的可靠性和运行效率；同时，也使其它应用软件系统能直接处理数据库中的存盘数据。

● 设立“对象元件库”，解决了组态结果的积累和重新利用问题。所谓对象元件库，实际

上是分类存储各种组态对象的图库。组态时，可把制作完好的对象(包括图形对象、窗口对象、策略对象，以至位图文件等等)以元件的形式存入图库中；也可把元件库中的各种对象取出，直接为当前的工程所用。随着工作的积累，对象元件库将日益扩大和丰富，组态工作将会变得越来越简单方便。

更详细资料查阅“SL-AVR单片机组态开发实验系统说明”等光盘资料。

3.6 SL-AVR*.ASM源文件说明

大量AVR应用实验源程序及编译、调试、下载工作软件可上网下载，网址为WWW.SL.COM.CN。

光盘中提供多种源文件：

(1) ATMEL公司提供的见光盘文件*:\AVR\AVR\asmpack\appnotes\AVR*.ASM(实验程序)；

(2) 双龙公司提供的见光盘文件*:\AVR\AVR\SL-AVR*.ASM(实验应用程序，也适用SL-AVR+)；

(3) 双龙公司提供的见光盘文件*:\AVR\AVR\指令ASM*.ASM(指令实验调试程序)；

(4) 双龙公司提供的见光盘文件*:\AVR\AVR\SL-AVRAD\程汇*.ASM(实验程序)。

(5) 双龙公司提供的见光盘文件*:\AVR\AVR\SL-AVR+*.ASM(实验程序)。

注意：光盘文件拷到用户计算机里是只读属性，只有去掉只读属性才能进行修改、运行等操作。

第四章　AVR 单片机指令系统

AVR 单片机每条指令对应的实验源程序见文件夹《指令 ASM》。

计算机的指令系统是一套控制计算机操作的代码，称之为机器语言。计算机只能识别和执行机器语言的指令。为了便于人们理解、记忆和使用，通常用汇编语言指令来描述计算机的指令系统。汇编语言指令可通过汇编器翻译成计算机能识别的机器语言。

AVR 单片机指令系统是 RISC 结构的精简指令集，是一种简明易掌握、效率高的指令系统。

AVR 单片机指令系统速查表中，不同器件使用不同的指令表，见附录 3。

(1) 89 条指令器件：对应器件 AT90S1200 是 最基本指令，见附录表 3.1。

(2) 90 条指令器件(□)：对应器件 Attiny11/12/15/22；　90 条指令＝□＋89 条基本指令。

(3) 118 条指令器件(◇)：对应器件 AT90S2313/2333/4414/4433/4434/8515/8534/8535；　118 条指令＝◇＋ 90 条指令，见附录表 3.2。

(4) 121 条指令器件(△)：对应器件 ATmega603/103；　121 条指令＝△＋ 118 条指令。

(5) 130 条指令器件(☆)：对应器件 ATmega161；　130 条指令＝☆＋121 条指令。

附录表 3.3 为各种器件指令比较表，也为指令速查表。

AVR 单片机的大多数指令执行时间为单个时钟周期。这一章主要分析 AVR 单片机指令系统的功能和使用方法。表 4.1 为常用 AVR 器件指令表。

表 4.1　AVR 器件 118 条指令速查表

AT90S2313/2333/4414　1133/4434/8515/8534/8535

算术和逻辑指令		BRCC k	C 清零转	位指令和位测试指令	
ADD Rd,Rr	加法	BRSH k	≥转	SBI P,b	置位 I/O 位
ADC Rd,Rr	带进位加	BRLO k	小于转(无符号)	CBI P,b	清零 I/O 位
◇ADIW Rdl,K	加立即数	BRMI k	负数转移	LSL Rd	左移
SUB Rd,Rr	减法	BRPL k	正数转移	LSR Rd	右移
SUBI Rd,Rr	减立即数	BRGE k	≥转(带符号)	ROL Rd	带进位左循环
SBC Rd,Rr	带进位减	BRLT k	小于转(带符号)	ROR Rd	带进位右循环
SBCI Rd,K	带 C 减立即数	BRHS k	H 置位转移	ASR Rd	算术右移
◇SBIW Rdl,K	减立即数	BRHC k	H 清零转移	SWAP Rd	半字节交换
AND Rd,Rr	与	BRTS k	T 置位转移	BSET s	置位 SREG
ANDI Rd,K	与立即数	BRTC k	T 清零转移	BCLR s	清零 SREG
OR Rd,Rr	或	BRVS k	V 置位转移	BST Rr,b	Rr 的 b 位送 T
ORI Rd,K	或立即数	BRVC k	V 清零转移	BLD Rd	T 送 Rr 的 b 位
EOR Rd,Rr	异或	BRIE k	中断位置位转移	SEC	置位 C
COM Rd	取反	BRID k	中断位清零转移	CLC	清零 C
NEG Rd	取补	数据传送指令		SEN	置位 N
SBR Rd,K	寄存器位置位	MOV Rd,Rr	寄存器传送	CLN	清零 N
CBR Rd,K	寄存器位清零	◇ LDI Rd,Rr	装入立即数	SEZ	置位 Z
INC Rd	加 1	◇ LD Rd,X	X 间接取数	CLZ	清零 Z
DEC Rd	减 1	◇ LD Rd,X+	X 间接取数后+	SEI	置位 I
TST Rd	测试零或负	◇ LD Rd,－X	X 间接取数先－	CLI	清零 I
CLR Rd	寄存器清零	◇ LD Rd,Y	Y 间接取数	SES	置位 S
SER Rd	寄存器置 FF	◇ LD Rd,Y+	Y 间接取数后+	CLS	清零 S
条件转移指令		◇ LD Rd,－Y	Y 间接取数先－	SEV	置位 V
RJMP k	相对转移	◇LDD Rd,Y+q	Y 间接取数先+q	CLV	清零 V
◇IJMP	间接转移(Z)	LD Rd,Z	Z 间接取数	SET	置位 T
RCALL k	相对调用	◇LD Rd,Z+	Z 间接取数后+	CLT	清零 T
◇ICALL	间接调用(Z)	◇LD Rd,－Z	Z 间接取数先－	SEH	置位 H
RET	子程序返回	◇LDD Rd,Z+q	Z 间接取数+q	CLH	清零 H
RETI	中断返回	◇LDS Rd,K	从 SRAM 装入	NOP	空操作
CPSE Rd,Rr	比较相等跳行	◇ST X,Rr	X 间接存数	SLEEP	休眠指令
CP Rd,Rr	比较	◇ST X+,Rr	X 间接存数后+	WDR	看门狗复位
CPC Rd,Rr	带进位比较	◇ST －X,Rr	X 间接存数先－	90 条指令器件为 ATtiny11/12/15/22= □+89 条基本指令器件是 AT90S1200	
CPI Rd,K	与立即数比较	◇ST Y,Rr	Y 间接存数		
SBRC Rr,b	位置位跳行	◇ST Y+,Rr	Y 间接存数后+		
SBRS Rr,b	I/O 位清零跳行	◇ST －Y,Rr	Y 间接存数先－		
SBIC P,b	I/O 位置位跳行	◇STD Y+q,Rr	Y 间接存数+q		
SBIS P,b	SREG 位置位转	ST Z,Rr	Z 间接存数		
BRBC s,k	SREG 位清零转	◇ST Z+,Rr	Z 间接存数后+	118 条指令器件= ◇+90 条指令器件	
BREQ k	相等转移	◇ST －Z,Rr	Z 间接存数先－		
BRNE k	不相等转移	◇STD Z+q,Rr	Z 间接存数+q		
BRNE k	不相等转移	◇STS k,Rr	数据送 SRAM		
BRCS k	C 置位转	□LPM	从程序区取数		
		IN Rd,P	从 I/O 口取数		
		OUT P,Rdr	存数 I/O 口		
		PUSH Rr	压栈		
		POP Rd,	出栈		

4.1 指令格式

4.1.1 汇编指令

汇编语言源文件是由汇编语言代码和汇编程序指令所组成的 ASCII 字符文件。

一、汇编语言源文件

汇编语言源文件包括指令助记符、标号和伪指令。指令助记符和伪指令常带操作数。

每条程序输入行首先是标号,标号为字母数字串,并带一个冒号。使用标号的目的是为了跳转和转移指令及在程序存储器和 SRAM 中定义变量名。

程序输入行有下列 4 种形式:

(1)【标号:】伪指令【操作数】【注释】

(2)【标号:】指令【操作数】【注释】

(3)注释

(4)空行

注释有下列形式:【文字】

括号内的项是任选的。用于注释的分号(;)及到行结尾的文字,汇编器是忽略的。标号、指令和伪指令在后面有详细说明。

例子:

```
Label:. EQU Var1=100      ;置 Var1 等于 100(伪指令)
       . EQU Var2=200      ;置 Var2 等于 200
test:  rjmp  test          ;无限循环(指令)
                           ;纯注释行
                           ;另一个注释行
```

注意:不限制有关标号、伪指令、注释或指令的列位置。

二、指令助记符

汇编器认可指令集中的指令助记符。指令集中综合了助记符并给出了参数。

操作数有下列形式:

Rd:R0～R31 或 R16～R31(取决于指令)。

Rr:R0～R31。

b:常数(0～7),可能是常数表达式。

s:常数(0～7),可能是常数表达式。

p:常数(0～31/63),可能是常数表达式。

K:常数(0～255),可能是常数表达式。

k:常数,值范围取决于指令,可能是常数表达式。

q:常数(0～63),可能是常数表达式。

4.1.2 汇编器伪指令

汇编器提供一些伪指令,伪指令并不直接转换成操作数,而是用于调整存储器中程序的位置、定义宏、初始化存储器等。全部伪指令在表 4.2 中给出。

表 4.2 伪指令表

序 号	伪指令	说 明	序 号	伪指令	说 明
1	BYTE	保存字节到变量	10	ESEG	EEPROM 段
2	CSEG	代码段	11	EXIT	退出文件
3	DB	定义字节常数	12	INCLUDE	从指定文件开始读
4	DEF	设置寄存器的符号名	13	LIST	打开列表文件生成器
5	DEVICE	定义被汇编的文件	14	LISTMAC	打开宏表达式
6	DSEG	数据段	15	MACRO	宏开始
7	DW	定义字常数	16	NOLIST	关闭列表文件生成器
8	ENDMACRO	宏结束	17	ORG	设置程序起始位置
9	EQU	符号相等于表达式	18	SET	赋值给一个标号

1. BYTE ——保存字节到变量

BYTE 伪指令保存存储的内容到 SRAM 中。为了能提供所要保存的位置，BYTE 伪指令前应有标号。该伪指令带一个表征被保存字节数的参数。该伪指令仅用在数据段内(见伪指令 CSEG，DSEG 和 ESEG)。注意：必须带一个参数，字节数的位置不需要初始化。

语法：LABEL：. BYTE 表达式

2. CSEG ——代码段

CSEG 伪指令定义代码段的开始位置。一个汇编文件包含几个代码段，这些代码段在汇编时被连接成一个代码段。在代码段中不能使用 BYTE 伪指令，典型的缺省段为代码段。代码段有一个字定位计数器。ORG 伪指令用于放置代码段和放置程序存储器指定位置的常数。CSEG 伪指令不带参数。

语法：. CSEG

3. DB——在程序存储器或 EEPROM 存储器中定义字节常数

DB 伪指令保存数据到程序存储器或 EEPROM 存储器中。为了提供被保存的位置，在 DB 伪指令前必须有标号。DB 伪指令可带一个表达式表，至少有一个表达式。DB 伪指令必须放在代码段或 EEPROM 段。表达式表是一系列表达式，用逗号分隔，每个表达式必须是一个 128～255 之间的有效值。如果表达式有效值是负数，则用 8 位 2 的补码放在程序存储器或 EEPROM 存储器中。如果 DB 伪指令用在代码段，并且表达式表多于一个表达式，则以 2 个字节组合成一个字放在程序存储器中。如果表达式表是奇数，那么最后一个表达式将独自以字格式放在程序存储器中，而不管下一行汇编代码是否是单个 DB 伪指令。

语法：LABEL：. DB 表达式

4. DEF——设置寄存器的符号名

DEF 伪指令允许寄存器用符号代替。一个定义的符号用在程序中，并指定一个寄存器，一个寄存器可以赋几个符号。符号在后面程序中能再定义。

语法：. DEF 符号＝寄存器

5. DEVICE——定义被汇编的文件

DEVICE 伪指令允许用户告知汇编器被执行的代码使用哪种文件。如果使用该伪指令，

若在代码中有指定的文件不提供的指令，则提示一个警告。如果代码段或 EEPROM 段的尺寸大于被指定文件的尺寸，也提示警告。如果不使用 DEVICE 伪指令，则假定文件提供所有的指令，也不限制存储器尺寸。

语法：. DEVICE AT90S1200|AT90S2313|AT90S4414|AT90S8515

6. DSEG——数据段

DSEG 伪指令定义数据段的开始。一个汇编文件能包含几个数据段，这些数据段在汇编时被连接成一个数据段。一个数据段正常仅由 BYTE 伪指令(和标号)组成。数据段有自己的定位字节计数器。ORG 伪指令被用于 SRAM 指定位置放置变量。DSEG 伪指令不带参数。

语法：. DSEG

7. DW——在程序存储器和 EEPROM 存储器中定义字常数

DW 伪指令保存代码到程序存储器或 EEPROM 存储器，为了提供被保存的位置，在 DW 伪指令前必须有标号。DW 伪指令可带一个表达式表，至少有一个表达式。DW 伪指令必须放在代码段或 EEPROM 段。表达式表是一系列表达式，用逗号分隔，每个表达式必须是—32 768～65 535 的有效值。如果表达式有效值是负数，则用 16 位 2 的补码放在程序存储器中。

语法：LABEL：. DW 表达式表

8. ENDMACRO——宏结束

ENDMACRO 伪指令定义宏定义的结束。该伪指令并不带参数，参见 MACRO 伪指令。

语法：. ENDMACRO

9. EQU——设置一个符号相等于一个表达式

EQU 伪指令赋一个值到标号，该标号用于后面的表达式，用 EQU 伪指令赋值的标号是一个常数，不能改变或重定义。

语法：. EQU 标号＝表达式

10. ESEG——EEPROM 段

ESEG 伪指令定义 EEPROM 段的开始位置。一个汇编文件包含几个 EEPROM 段，这些 EEPROM 段在汇编时被连接成一个 EEPROM 段。在 EEPROM 段中不能使用 BYTE 伪指令。EEPROM 段有一个字节定位计数器。ORG 伪指令用于放置 EEPROM 存储器指定位置的常数。ESEG 伪指令不带参数。

语法：. ESEG

11. EXIT——退出文件

EXIT 伪指令告诉汇编器停止汇编该文件。正常情况下，汇编器汇编到文件的结束，如果 EXIT 出现在包括文件中，则汇编器从文件中 INCLUDE 伪指令行继续汇编。

语法：. EXIT

12. INCLUDE——从指定文件开始读

INCLUDE 伪指令告诉汇编器从指定的文件开始读。然后汇编器汇编指定的文件，直到文件结束或遇到 EXIT 伪指令。一个包括文件也可能自己用 INCLUDE 伪指令来表示。

语法：. INCLUDE“文件名”

13. LIST——打开列表文件生成器

LIST 伪指令告诉汇编器打开列表文件生成器。汇编器生成一个汇编源代码、地址和操作代码的文件列表。列表文件生成器缺省值是打开。该伪指令总是与 NOLIST 伪指令一起

出现，用于生成列表或汇编源文件有选择的列表。

语法：.LIST

14. LISTMAC——打开宏表达式

LISTMAC伪指令告诉汇编器，当调用宏时，用列表生成器在列表文件中显示宏表达式。缺省值仅是在列表文件中显示宏调用参数。

语法：.LISTMAC

15. MACRO——宏开始

MACRO伪指令告诉汇编器这是宏开始。MACRO伪指令带宏名和参数。当后面的程序中写了宏名，被表达的宏程序在指定位置被调用。一个宏可带10个参数。这些参数在宏定义中用@0～@9代表。当调用一个宏时，参数用逗号分隔。宏定义用ENDMACRO伪指令结束。缺省值为汇编器的列表生成器，仅列表宏调用。为了在列表文件中包括宏表达式，必须使用LISTMAC伪指令。在列表文件的操作代码域内宏用a+作记号。

语法：.MACRO 宏名

16. NOLIST · 关闭列表文件生成器

NOLIST伪指令告诉汇编器关闭列表文件生成器。正常情况下，汇编器生成一个汇编源代码、地址和操作代码文件列表。缺省时为打开列表文件，但是可用该伪指令禁止列表。为了使被选择的汇编源文件部分产生列表文件，该伪指令可以与LIST伪指令一起使用。

语法：.NOLIST

17. ORG——设置程序起始位置

ORG伪指令设置定位计数器一个绝对值。设置的值为一个参数。如果ORG伪指令放在数据段，则设置SRAM定位计数器；如果该伪指令放在代码段，则设置程序存储器计数器；如果该伪指令放在EEPROM段，则设置EEPROM定位计数器。如果该伪指令前带标号(在相同的源代码行)，则标号由参数值给出，代码和EEPROM定位计数器的缺省值是零；而当汇编器启动时，SRAM定位计数器的缺省值是32(因为寄存器占有地址0～31)。注意，EEPROM和SRAM定位计数器按字节计数，而程序存储器定位计数器按字计数。

语法：.ORG 表达式

18. SET——赋值给一个标号

SET伪指令赋值给一个标号。这个标号能用在后面的表达式中。用SET伪指令赋值的标号在后面的程序中能改变。

语法：.SET 标号＝表达式

4.1.3 表达式

汇编器包括一些表达式，表达式由操作数、运算符和函数组成。表达式内部都为32位。

一、操作数

(1) 用户定义的标号，该标号给出了放置标号位置的定位计数器的值。

(2) 用户用SET伪指令定义的变量。

(3) 用户用EQU伪指令定义的常数。

(4) 整数常数，包括下列几种形式：

● 十进制数(缺省值)：10，255

- 十六进制数(二进制表示法):0x0a,$0a,0xff,$ff
- 二进制数:0b00001010,0b11111111

(5) PC:程序存储器定位计数器的当前值。

二、函 数

(1)LOW(表达式):返回1个表达式的低字节。

(2)HIGH(表达式):返回1个表达式的第2个字节。

(3)BYTE2(表达式):与HIGH函数相同。

(4)BYTE3(表达式):返回1个表达式的第3个字节。

(5)BYTE4(表达式):返回1个表达式的第4个字节。

(6)LWRD(表达式):返回1个表达式的0～15位。

(7)HWRD(表达式):返回1个表达式的16～31位。

(8)PAGE(表达式):返回1个表达式的16～21位。

(9)EXP2(表达式):返回2^表达式。

(10)LOG2(表达式):返回LOG2(表达式)的整数部分。

三、运算符

汇编器提供的部分运算符见表4.3。越高的运算符,优先级越高。表达式可以用括号括起来,并且与括号外任意表达式所组合的表达式总是有效的。

表4.3 部分运算符表

序 号	名 称	符 号	优先级	说 明
1	逻辑非	!	14	一元运算符,表达式是0返回1,而表达式为非0则返回0
2	逐位非	~	14	一元运算符,输入表达式的所有位倒置
3	负号	-	14	一元运算符,使表达式为算术负
4	乘法	*	13	二进制运算符,两个表达式相乘
5	除法	/	13	二进制运算符,左边表达式除以右边表达式,得整数的商值
6	加法	+	12	二进制运算符,两个表达式相加
7	减法	-	12	二进制运算符,左边表达式减去右边表达式
8	左移	<<	11	二进制运算符,左边表达式左移右边表达式给出的次数
9	右移	>>	11	二进制运算符,左边表达式右移右边表达式给出的次数
10	小于	<	10	二进制运算符,左边带符号表达式小于右边带符号表达式,则为1;否则为0
11	小于等于	<=	10	二进制运算符,左边带符号表达式小于或等于右边带符号表达式,则为1;否则为0
12	大于	>	10	二进制运算符,左边带符号表达式大于右边带符号表达式,则为1;否则为0
13	大于等于	>=	10	二进制运算符,左边带符号表达式大于或等于右边带符号表达式,则为1;否则为0

表 4.3(续)

序　号	名　称	符　号	优先级	说　明
14	等于	==	9	二进制运算符，左边带符号表达式等于右边带符号表达式，则为1；否则为0
15	不等于	!=	9	二进制运算符，左边带符号表达式不等于右边带符号表达式，则为1；否则为0
16	逐位与	&	8	二进制运算符，两个表达式之间逐位与
17	逐位异或	^	7	二进制运算符，两个表达式之间逐位异或
18	逐位或	|	6	二进制运算符，两个表达式之间逐位或
19	逻辑与	&&	5	二进制运算符，两个表达式逻辑与，非0则为1；否则为0
20	逻辑或	||	4	二进制运算符，两个表达式逻辑或，非0则为1；否则为0

4.2　寻址方式

指令的一个重要组成部分是操作数，指令给出参与运算的数据方式称为寻址方式。AVR单片机指令操作数的寻址方式有以下多种：单寄存器直接寻址、双寄存器直接寻址、I/O直接寻址、数据直接寻址、带位移的数据间接寻址、数据间接寻址、带预减量的数据间接寻址、带后增量的数据间接寻址、常量寻址、程序直接寻址、程序间接寻址、程序相关寻址。

一、单寄存器直接寻址

由指令指出一个寄存器的内容作为操作数，这种寻址方式称为单寄存器直接寻址。寄存器寻址所选的工作寄存器为寄存器堆中的0～31区域。指令字的低5位(D0～D4位)指出所用的寄存器Rd。图4.1为单寄存器直接寻址示意图。

二、双寄存器直接寻址

双寄存器直接寻址方式同单寄存器直接寻址方式，指令字中指出2个寄存器Rd和Rr。指令字的D0～D4位指出Rd寄存器，D5～D9位指出Rr寄存器。结果存在Rd寄存器中。图4.2为双寄存器直接寻址示意图。

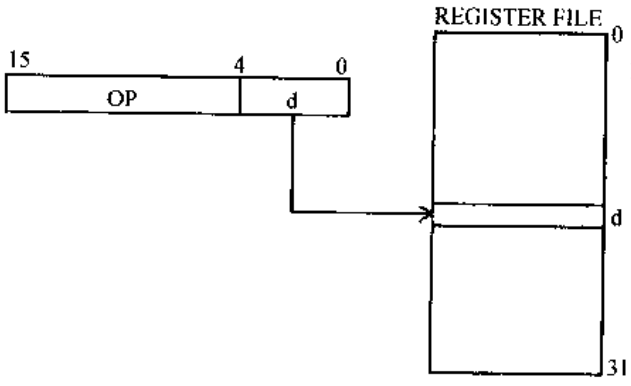


图4.1　单寄存器直接寻址示意图

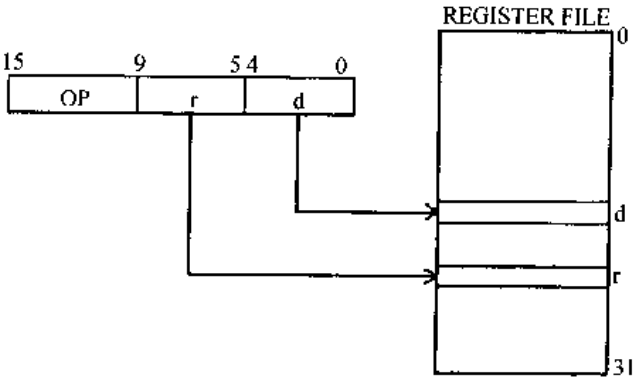


图4.2　双寄存器直接寻址示意图

三、I/O直接寻址

在AVR单片机的寄存器区中映射有I/O寄存器。指令可以直接对I/O空间进行操作。操作数包含在指令字中的D0～D5位中，同时指令字中还包含了目的或源寄存器地址n。图4.3为I/O直接寻址示意图。

四、数据直接寻址

数据直接寻址方式便于直接从SRAM存储器中存取数据。数据直接寻址为双字指令，1个16位的数据地址放在低字指令中，高字指令中的Rd/Rr指定了目的寄存器或源寄存器。存储器存取的范围限制在SRAM当前64字节页。图4.4为数据直接寻址示意图。

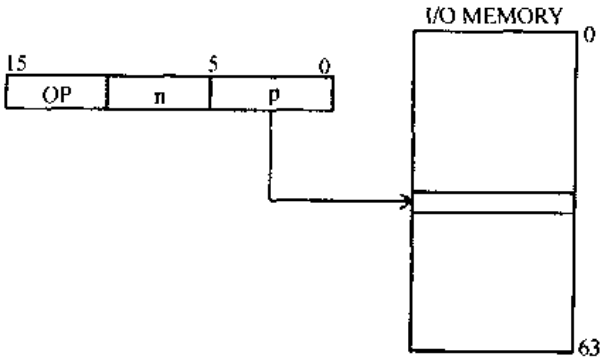


图4.3 I/O直接寻址示意图

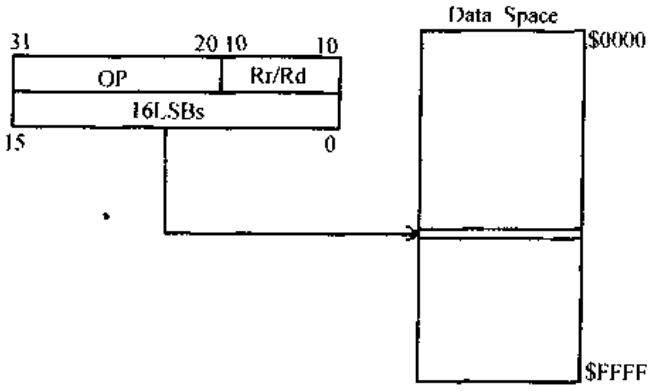


图4.4 数据直接寻址示意图

五、带位移的数据间接寻址

带位移的数据间接寻址方式是利用变址寄存器(Y或Z)及指令字中的位移量共同决定被存取SRAM存储器的地址。操作数的地址由Y或Z寄存器的内容加上指令字D0～D5位给出的位移量a给出。指令字中的n为目的或源寄存器地址。图4.5为带位移的数据间接寻址示意图。

六、数据间接寻址

由指令指出某一个寄存器的内容作为操作数的地址，该寻址方式称为寄存器间接寻址。AVR单片机中用变址寄存器X、Y或Z作为规定的寄存器，并对SRAM存储器存取操作，称为数据间接寻址。操作数的地址在变址寄存器(X、Y或Z)中。图4.6为数据间接寻址示意图。

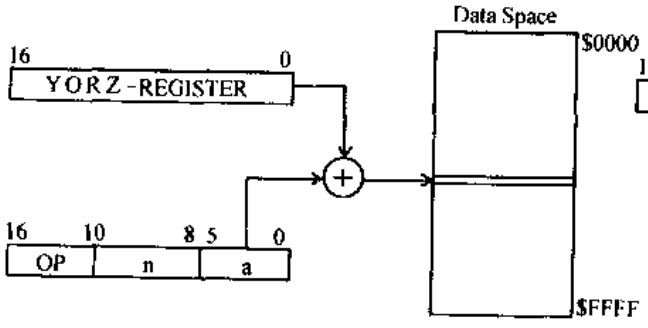


图4.5 带位移的数据间接寻址示意图

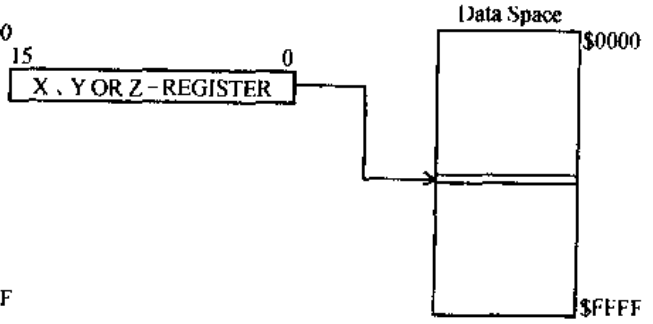


图4.6 数据间接寻址示意图

七、带预减量数据间接寻址

同数据间接寻址，但寄存器 X、Y 或 Z 的内容在操作之前先被减 1，相减后的内容为操作数的地址。这种寻址方式特别适用于访问矩阵、查表等应用。图 4.7 为带预减量的数据间接寻址示意图。

八、带后增量的数据间接寻址

同数据间接寻址方式，但寄存器 X、Y 或 Z 的内容在操作后被加 1，而操作数地址的内容为寄存器增量之前的内容。这种寻址方式特别适用于访问矩阵、查表等应用。图 4.8 为带后增量的数据间接寻址示意图。

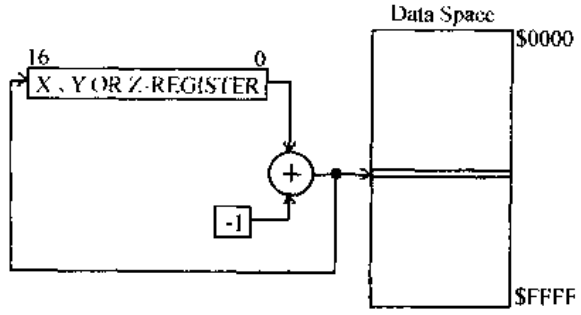


图 4.7　带预减量的数据间接寻址示意图

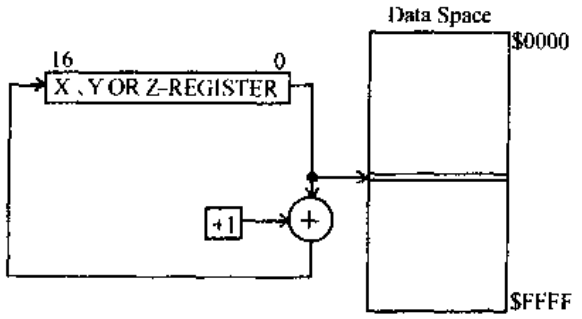


图 4.8　带后增量的数据间接寻址示意图

九、常量寻址

常量寻址主要从程序存储器取常量。程序存储器中放常量字节的地址由寄存器 Z 的内容确定。Z 寄存器的高 15 位用于选择字地址(0～4K)，而 Z 寄存器的最低位(D0)用于写字地址的高低字节。若最低位被清除(LSB=0)，则选择低字节；若最低位被置位(LSB=1)，则选择高字节，例如 LPM 指令。图 4.9 为常量寻址示意图。

十、程序直接寻址

程序直接寻址方式中操作数包含在指令字中，即操作数以指令字的形式存放于程序存储器中。程序在双指令字中操作数指定的立即地址处执行，如 JMP、CALL 指令。图 4.10 为程序直接寻址示意图。

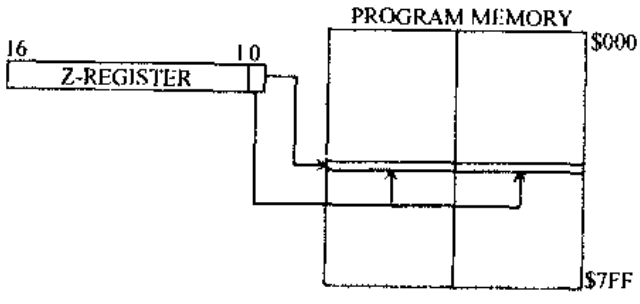


图 4.9　常量寻址示意图

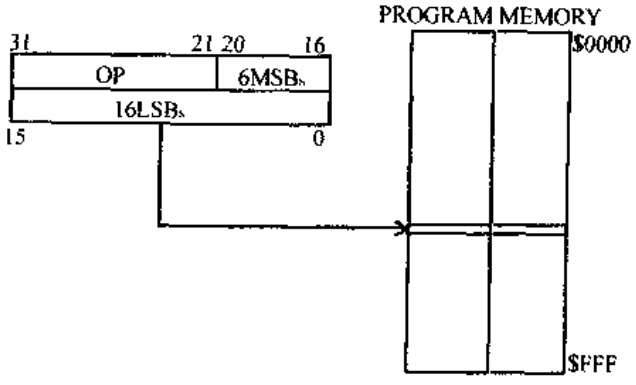


图 4.10　程序直接寻址示意图

十一、程序间接寻址

程序间接寻址方式中，使用Z寄存器存放要执行程序的地址。程序在Z寄存器的内容指定的地址处继续执行，即用寄存器Z的内容代替PC的值，如IJMP、ICALL指令。图4.11为程序间接寻址示意图。

十二、程序相关寻址

程序间接寻址方式中，在指定字中包含了相关地址K。执行程序时，首先将PC值与相关地址K相加，得出程序需要继续执行的下一条指令的地址。然后程序在地址PC+K处继续执行。其范围$-2K\sim(2K-1)$，如RJMP、RCALL指令。图4.12为程序相关寻址示意图。

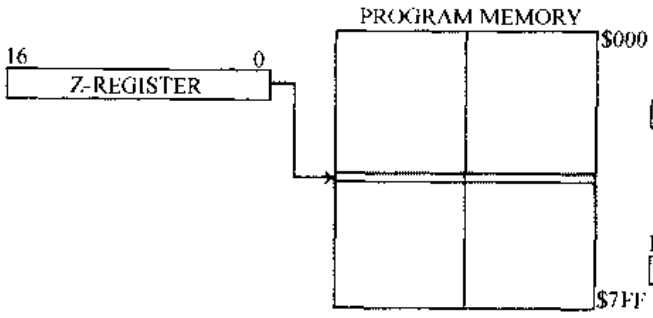


图4.11 程序间接寻址示意图

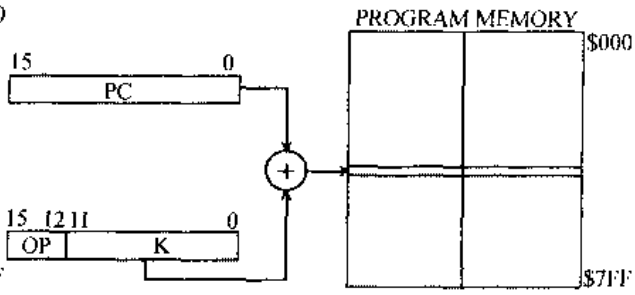


图4.12 程序相关寻址示意图

4.3 数据操作和指令类型

4.3.1 数据操作

AVR单片机是一个增强型RISC微控制器，具有高性能的数据处理能力，能对位、半字节、字节和双字节数据进行各种操作。它们包括算术和逻辑运算、数据传送、布尔处理和控制转移等操作。

4.3.2 指令类型

AVR单片机共有89～130条指令。如果118条指令按功能分类，则有22条算术和逻辑指令、34条条件转移指令、31条数据传送指令、31条位指令和位测试指令。

4.3.3 指令集名词

1. 状态寄存器

SREG：状态寄存器。
C：进位标志位。
Z：零标志位。
N：负数标志位。
V：2的补码溢出指示位。
S：N⊕V，符号测试位。
H：半进位标志位。
T：用于BLD和BST指令传送位。
I：全局中断触发/禁止标志位。

2. 寄存器和操作码

Rd：寄存器区中的目的(或源)寄存器。

Rr：寄存器区中的源寄存器。

R：指令执行后的结果。

K：常数项或字节数据(8位)。

k：程序计数器的常量地址数据。

b：在寄存器区中或I/O寄存器(3位)中的位。

s：在状态寄存器(3位)中的位。

X，Y，Z：间接地址寄存器(X=R27，R26；Y=R29，R28；Z=R31，R30)。

P：I/O口地址。

q：直接寻址的偏移(6位)。

3. I/O寄存器

RAMPX、RAMPY、RAMPZ：X、Y、Z寄存器的级联寄存器，允许MCU在相连多于64K字节的SRAM整个范围内间接寻址。

4. 堆栈

STACK：作为返回地址和压栈寄存器的堆栈。

SP：STACK的堆栈指针。

5. 标志

⇔：由指令引起的有效标志。

0：由指令清除的标志。

1：由指令置位的标志。

－：由指令引起的无效标志。

说明：为了对每条指令有一个具体的了解，以便对单片机映像空间(通用工作寄存器空间、I/O寄存器空间、片内片外SRAM空间、程序存储器空间、EEPROM数据存储器空间)认识更清楚(软件即完成在单片机映像空间之间或自身之间的传送、运算、检测、处理等操作)，我们对每条指令均编一些简单的测试指令，约定它们的程序编号为第四章第四节第几段第几题。例4.4.1加法指令的"1.不带进位加法"，程序编号为4411.ASM。其余以此类推。

AVR单片机的指令系统对不同器件有不同指令，选用器件时应注意这一点(详见本书附录3)。某种器件应使用哪些指令，详细资料请阅相应器件(电子书光盘中)英文指令表。以下讲述130条指令功能及应用。

4.4　算术和逻辑指令

AVR的算术运算指令有加、减、乘、取反、取补、比较、增量和减量指令。逻辑运算指令有与、或、异或指令等。图4.13为状态寄存器功能示意图。

4.4.1　加法指令

1. 不带进位加法

ADD——不带进位加

说明：两个寄存器不带进位C标志加，结果送目的寄存器Rd。

操作：Rd←Rd＋Rr

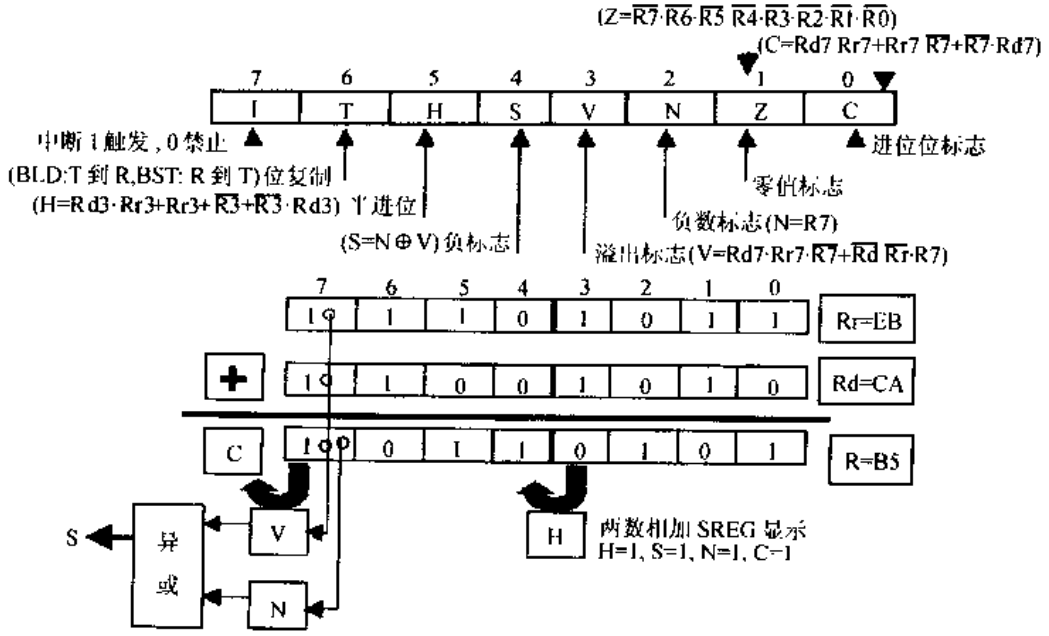


图 4.13 状态寄存器功能示意图

语法： 操作码： 程序计数器：

ADD Rd,Rr $0 \leqslant d \leqslant 31, 0 \leqslant r \leqslant 31$ PC←PC+1

16 位操作码：

0000	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
--	—	⇔	⇔	⇔	⇔	⇔	⇔

H:$Rd3 \cdot Rr3 + Rr3 \cdot \overline{R3} + \overline{R3} \cdot Rd3$ N:R7

S:$N \oplus V$, Z:$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V:$Rd7 \cdot Rr7 \cdot \overline{R7} + \overline{Rd7} \cdot \overline{Rr7} \cdot R7$ C:$Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + \overline{R7} \cdot Rd7$

说明：以下所有例子仅说明指令书写方法，不能直接汇编，因为程序缺器件配制文件及程序执行地址。实践操作例子 *.ASM 可以汇编，可以调试，可以修改(修改需改变文件属性，因为只读文件无法修改)！

约定：注释寄存器的内容(数据)用括号表示，例：(R16)= $16，表示寄存器的内容(数据)为十六进制数 16H！图 4.14 为寄存器窗口数据表示法。

例子：(实践操作程序 4411.ASM) 实践操作例子 *.ASM，必须编译生成 *.OBJ 文件才可调试；如要修改 *.ASM，必须修改文件属性，去掉 *.ASM 只读文件属性。

```
    ldi r16,$11          ;LDI 立即数装入指令，要求寄存器必须符合 16≤d≤31 条件
    ldi r20,$22
    ldi r28,0XAA         ;$,0X 均为十六进制表示法
lp: add r16,r20          ;也可单步执行到此行
                         ;在调试窗口的对应寄存器 R16,R20 输入数据
                         ;(R16) =      ,(SREG) =
```

图 4.14　寄存器窗口数据表示法

```
    add r28,r28          ;也可单步执行到此行,在调试窗口的对应寄存器 R28 输入数据
                         ;(R28) =     ,(SREG) =
    rjmp lp              ;反复做实验
```

Words:1(2bytes)

Cycles:1

2. 带进位加法

ADC——带进位加

说明:两个寄存器和 C 标志的内容相加,结果送目的寄存器 Rd。

操作: Rd←Rd+Rr+C

语法:	操作码:	程序计数器:
ADC Rd,Rr	0≤d≤31,0≤r≤31	PC←PC+1

16 位操作码:

0000	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H:$Rd3 \cdot Rr3 + Rr3 \cdot \overline{R3} + \overline{R3} \cdot Rd3$　　N:R7

S:N ⊕ V,　　Z:$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V:$Rd7 \cdot Rr7 \cdot \overline{R7} \cdot \overline{Rd7} \cdot \overline{Rr7} \cdot R7$　　C:$Rd7 \cdot Rr7 + Rr7 \cdot \overline{R7} + \overline{R7} \cdot Rd7$

例子:(实践操作程序 4412.ASM)

```
    ldi r20,$77          ; LDI 立即数装入指令,要求寄存器必须符合 16≤d≤31 条件
    ldi r21,$99
    LDI R22,0X77
    LDI R23,0X11
lp: add r22,r20          ;(r22) =     ,(SREG) =
    ADC R23,R21          ;也可单步执行到此行,在调试窗口的对应寄存器 R23,R21 输入数据
                         ;(R23) =     ,(SREG) =
    rjmp lp              ;反复做实验
```

Words: 1(2 bytes)

Cycles: 1

3. 立即数加法(字)

ADIW——立即数加法

说明：寄存器对于立即数值(0～63)相加，结果放到寄存器对。

操作：Rdh:Rdl←Rdh:Rdl+K

语法：	操作码：	程序计数器：
ADIW Rdl, K	dl∈{ 24 26 28 30 },0≤K≤63	PC←PC+ 1

16 位操作码：

1001	0110	KKdd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
--	—	--	⇔	⇔	⇔	⇔	⇔

S:N ⊕ V

V:Rdh7 · R15

N: R15

Z:$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

C:$\overline{R15}\cdot$Rdh7

例子：(实践操作程序 4413. ASM)

```
lp:adiw r24, 1          ;也可单步执行到此行，在调试窗口的对应寄存器 R24 输入数据
ADIW R30, 63            ;也可单步执行到此行，在调试窗口的对应寄存器 R30 输入数据
rjmp lp                 ;反复测试
```

Words: 1(2 bytes)

Cycles: 2

4. 加 1 指令

INC——加 1

说明：寄存器 Rd 的内容加 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 INC 指令允许在多倍字长计算中用作循环计数。当对无符号数操作时，仅有 BREQ(相等转移)和 BRNE(不为零转移)指令有效；当对二进制补码值操作时，所有的带符号转移指令都有效。

操作：Rd←Rd+ 1

语法：	操作码：	程序计数器：
INC Rd	0≤d≤31	PC←PC+ 1

16 位操作码：

1001	010d	dddd	0011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	--	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V

V:R7 · $\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

N:R7

Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：(实践操作程序 4414. ASM)

```
        clr   r22          ;寄存器 R22 清零
lp:     inc   r22          ;+1，也可单步执行到此行，在调试窗口的对应寄存器 R22 输入数据
lp1:    INC R22
        cpi   r22,$04      ;(R22)与立即数 $04 比较
        brnq lp1           ;不相等转移，相等则按顺序执行(观察状态寄存器 Z 标志变化)
        rjmp lp            ;反复测试
```

Word：1(2 bytes)

Cycles：1

4.4.2　减法指令

1. 不带进位减法

SUB——不带进位减

说明：两个寄存器相减，结果送目的寄存器 Rd 中。

操作：Rd←Rd−Rr

语法：	操作码：	程序计数器：
SUB Rd,Rr	0≤d≤31,0≤r≤31	PC←PC+1

16 位操作码：

0001	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—		⇔	⇔	⇔	⇔	⇔	⇔

H：$\overline{Rd3}\cdot Rr3+Rr3\cdot R3+R3\cdot\overline{Rd3}$　　N：R7

S：N ⊕ V　　Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{Rr3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：$Rd7\cdot\overline{Rr7}\cdot\overline{R7}+\overline{Rd7}\cdot Rr7\cdot R7$　　C：$\overline{Rd7}\cdot Rr7+Rr7\cdot R7+R7\cdot\overline{Rd7}$

例子：(实践操作程序 4421. ASM)

```
lp1: ldi    r23,$44        ;寄存器装入立即数
     ldi    r22,$11        ;寄存器装入立即数
lp:  sub r23,r22           ;减法 ，也可单步执行到此行
                           ;在调试窗口的对应寄存器 R23,R22 输入数据
     brne lp               ;r23 内容不为 0 转，为 0 顺执
     rjmp lp1              ;反复测试
```

Words：1(2 bytes)

Cycles：1

2. 立即数减法(字节)

SUBI——立即数减

说明：一个寄存器和常数相减，结果送目的寄存器 Rd。该指令工作于寄存器 R16 到 R31 之间，非常适合 X、Y 和 Z 指针的操作。

操作：Rd←Rd−K

语法： 操作码： 程序计数器：

SUBI Rd,K 16≤d≤31,0≤k≤255 PC←PC+ 1

16 位操作码：

0101	kkkk	dddd	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H:$\overline{Rd3}\cdot K3+K3\cdot R3+R3\cdot\overline{Rd3}$ N:R7

S:N⊕V Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{Rr3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V:$Rd7\cdot\overline{K7}\cdot\overline{R7}+\overline{Rd7}\cdot K7\cdot R7$ C:$\overline{Rd7}\cdot Rr7+Rr7\cdot R7+R7\cdot Rd7$

例子：(实践操作程序 4422. ASM)

```
LP: LDI   R22, $55
LP1:SUBI  R22 , $11      ;也可单步执行到此行,在调试窗口的对应寄存器 R22 输入数据
    BRNE  LP1            ;不为 0 转,为 0 顺执
    RJMP  LP             ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

3. 带进位减法

SBC——带进位减

说明：两个寄存器随着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：Rd←Rd−Rr−C

语法： 操作码： 程序计数器：

SBC Rd Rr 0≤d≤31, 0≤r≤31 PC←PC+ 1

16 位操作码：

0000	lord	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\cdot Rr3+Rr3\cdot R3+R3\cdot\overline{Rd3}$ N: R7

S: N⊕V Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

V: $Rd7\cdot\overline{Rr7}\cdot\overline{R7}+\overline{Rd7}\cdot Rr7\cdot R7$ C:$\overline{Rd7}\cdot Rr7+Rr7\cdot R7+R7\cdot\overline{Rd7}$

例子：(实践操作程序 4423. ASM)

```
LP: LDI R22, $80        ;寄存器装数(R22)= $80
    LDI R20, $80        ;(R20)= $80
    LDI R23, $23        ;(R23)= $23
    LDI R21, $11        ;(R21)= $11
    ADD R22,R20         ;(R22)= $00,C=1
LP1:SBC R23,R21         ;也可单步执行到此行,在调试窗口的对应寄存器 R23,R21 输入数据
                        ;(R23)−(R21)−(C)
```

```
        CPI R23,$00          ;R23 的内容与立即数 $00 比较,
        BRNE LP1             ;R23 的内容不为 0 转,为 0 顺执
        RJMP LP              ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

4. 带进位立即数减

SBCI- －带进位立即数减

说明:寄存器和立即数随着 C 标志相减,结果放到目的寄存器 Rd 中。

操作: Rd←Rd－K－C

语法:	操作码:	程序计数器:
SBCI Rd,K	16≤d≤31,0≤K≤255	PC←PC+1

16 位操作码:

0100	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H:$\overline{Rd3}\cdot K3+K3\cdot R3+R3\cdot\overline{Rd3}$　　N: R7

S:N ⊕ V　　Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}\cdot Z$

V:$Rd7\cdot\overline{K7}\cdot\overline{R7}+\overline{Rd7}\cdot K7\cdot R7$　　C:$\overline{Rd7}\cdot K7+K7\cdot R7+R7\cdot\overline{Rd7}$

例子:(实践操作程序 4424.ASM)

```
LP: SEC                  ;(C)=1
    LDI   R16,$44        ;(R16)=$44
    SUBI  R16,$22        ;(R16)减立即数 $22
LP1:SBCI  R16, $11       ;也可单步执行到此行,在调试窗口的对应寄存器 R16 输入数据
                         ;(R16)－$11－1
    CPI   R16,$00        ;比较 R16 内容是否为 $00
    BRNE  LP1            ;(R16)不为 0 转,为 0 顺执
    RJMP  LP             ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

5. 立即数减法(字)

SBIW—— 立即数减法

说明:双寄存器与立即数(0～63)减,结果送双寄存器。该指令操作于 4 个以上的寄存器对,比较适合对指针寄存器操作。

操作:Rdh:Rdl←Rdh:Rdl－K

语法:	操作码:	程序计数器:
SBIW Rdl, K	dl∈{24,26,28,30}, 0≤K≤63	PC←PC+ 1

16 位操作码:

1001	0111	KKdd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S：N⊕V　　V：Rdh7 · $\overline{R15}$　　N：R15　　C：R15 · $\overline{Rdh7}$

Z：$\overline{R15}$ · $\overline{R14}$ · $\overline{R13}$ · $\overline{R12}$ · $\overline{R11}$ · $\overline{R10}$ · $\overline{R9}$ · $\overline{R8}$ · $\overline{R7}$ · $\overline{R6}$ · $\overline{R5}$ · $\overline{R4}$ · $\overline{R3}$ · $\overline{R2}$ · $\overline{R1}$ · $\overline{R0}$ · Z

例子：(实践操作程序 4425. ASM)

```
LP: LDI   R24,5      ;寄存器装入立即数,寄存器必须符合 16≤R≤31
    LDI   R28,63     ;5,63 为十进制数,十六进制为 0X3F
    sbiw  r24,1      ;
    sbiw  r28,60     ;60 为十进制数
    RJMP  LP         ;反复测试
```

Words：1(2 bytes)

Cycles：1

6. 减 1 指令

DEC——减 1

说明：寄存器 Rd 的内容减 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 DEC 指令允许在多倍字长计算中用作循环计数。当对无符号值操作时，仅有 BREQ(不相等转移)和 BRNE(不为零转移)指令有效；当对二进制补码值操作时，所有的带符号转移指令都有效。

操作：Rd←Rd－1

语法：	操作码：	程序计数器：
DEC Rd	0≤d≤31	PC←PC+1

16 位操作码：

1001	010d	dddd	1010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S：N⊕V　　N：R7

V：$\overline{R7}$ · R6 · R5 · R4 · R3 · R2 · R1 · R0　　Z：$\overline{R7}$ · $\overline{R6}$ · $\overline{R5}$ · $\overline{R4}$ · $\overline{R3}$ · $\overline{R2}$ · $\overline{R1}$ · $\overline{R0}$

例子：(实践操作程序 4426. ASM)

```
   LP: LDI R16,$04    ;多级－1,可作延时子程序用
LOOP: DEC R16         ;也可单步执行到此行,在调试窗口的对应寄存器 R16 输入数据
                      ;(R16)－1
      BRNE  LOOP      ;(R16)不为 0 转,为 0 顺执
      DEC   R16       ;第一次[(R16)= $00]－1= $FF
      BRNE  LP        ;(R16)不为 0 转,为 0 顺执
      RET             ;子程序返回
```

Words：1(2 bytes)

Cycles：1

4.4.3　乘法指令

注意：AVR 单片机只有 ATmega161 和 Atmeg103L 等乘法指令。

MUL　—乘法

说明：该指令完成 8 位×8 位→16 位的无符号数乘法操作。被乘数 Rr 和乘数 Rd 是两个寄存器。16 位结果放在 R1(高字节)和 R0(低字节)中。注意，如果被乘数和乘数选择了 R0 或 R1，则当进行乘法后，结果将溢出。

操作：R1，R0←Rr * Rd

语法：	操作码：	程序计数器：
MUL　Rd，Rr	0≤d≤31，0≤r≤31	PC←PC+ 1

16 位操作码：

0001	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	·	—	⇔

C：R15

例子：(实践操作程序 4431. ASM，因无相应器件配制文件 *. inc，所以汇编时有出错提示)

```
lp:ldi r6, $04
ldi r5, #05
mul r6,r5          ;乘法
mov  r6,r1         ;保存乘积高位
mov  r5,r0         ;保存乘积低位
rjmp  lp
```

Words：1(2 bytes)

Cycles：2

4.4.4　取反码指令

COM——取二进制反码

说明：该指令完成寄存器 Rd 的二进制反码操作。

操作：　Rd←$FF—Rd

语法：	操作码：	程序计数器：
COM　Rd	0≤d≤31	PC←PC+ 1

16 位操作码：

1001	010d	dddd	0000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	·	·	⇔	0	⇔	⇔	1

S:N ⊕ V　　Z:$\overline{R7}$. $\overline{R6}$. $\overline{R5}$. $\overline{R4}$. $\overline{R3}$. $\overline{R2}$. $\overline{R1}$. $\overline{R0}$

V:0　　C:1　　N:R7

例子:(实践操作程序 4441. ASM)

```
lp: ldi    R24, $AA
    com   r24              ;取反
    cpi   r24, $aa         ;(r24)与 $aa 比较相等吗?
    breq lp1
lp1: COM   R24             ;取反
    cpi   r24, $aa         ;(r24)与 $aa 比较相等吗?
    breq lp                ;相等,反复测试
    BREQ LP
```

Words: 1(2 bytes)

Cycles: 1

4.4.5 取补指令

NEG——二进制补码

说明:寄存器 Rd 的内容转换成二进制补码,值 $80 是不改变的。

操作:R←$00−Rd

语法:	操作码:	程序计数器:
NEG　Rd	0≤d≤31	PC←PC+ 1

16 位操作码:

1001	010d	dddd	0001

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	—	⇔	⇔	⇔	⇔	⇔	⇔

H: R3 • $\overline{Rd3}$

S:N ⊕ V

V:R7 • $\overline{R6}$ • $\overline{R5}$ • $\overline{R4}$ • $\overline{R3}$ • $\overline{R2}$ • $\overline{R1}$ • $\overline{R0}$

N:R7

Z:$\overline{R7}$ • $\overline{R6}$ • $\overline{R5}$ • $\overline{R4}$ • $\overline{R3}$ • $\overline{R2}$ • $\overline{R1}$ • $\overline{R0}$

C:R7+ R6+ R5+ R4+ R3+ R2+ R1+ R0

例子:(实践操作程序 4451. ASM)

```
LP: SUB   R11,R0      ;开始时在调试窗口中的对应寄存器输入数据
                      ;设(r11)=0b1010101= $AA    (r0)=0b10011001= $99
    BRPL LP1          ;(r11)为正数转移
LP1:NEG   R11
    BRMI LP           ;(r11)为负数转移
```

Words: 1(2 bytes)

Cycles: 1

4.4.6　比较指令

1. 寄存器比较

CP——比较

说明：该指令完成两个寄存器 Rd 和 Rr 相比较操作，而寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作：Rd－Rr

语法：	操作码：	程序计数器：
CP　Rd　Rr	0≤d≤31，0≤r≤31	PC←PC＋1

16 位操作码：

1001	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—		⇔	⇔	⇔	⇔	⇔	⇔

H：$\overline{Rd3}\cdot Rr3+Rr3\cdot R3+R3\cdot\overline{Rd3}$　　N：R7

S：N ⊕ V　　Z：$Rd7\cdot Rr7\cdot\overline{Rr7}\cdot\overline{R7}+\overline{R7}\cdot\overline{Rd7}$

V：$Rd7\cdot\overline{Rd7}\cdot\overline{Rr7}+\overline{Rd7}\cdot Rr7\cdot R7$　　C：$\overline{Rd7}\cdot Rr7+Rr7\cdot R7+R7\cdot\overline{Rd7}$

例子：(实践操作程序 4461.ASM)

```
lp: cp r24,r19      ;单步执行到此行，开始时在调试窗口的对应寄存器输入数据
                    ;第一次操作设：(r24)＝$AA (r19)＝$55
                    ;第二次操作设：(r24)＝$11 (r19)＝$55
    brsh  lp1       ;(r24)≥(r19)则转 lp1，(r24)小于(r19)顺执
    rjmp lp2        ;
lp1: sub  r24,r19   ;(r24)相减(r19)
    brne  lp        ;(r24)不为 0 转，为 0 顺执
lp2: adiw r24,$11   ;立即数加，要求 d∈{ 24 26 28 30 }，0≤K≤63
    rjmp lp         ;反复测试
```

Words：1(2 bytes)

Cycles：2

2. 带进位比较

CPC——带进位比较

说明：该指令完成寄存器 Rd 的值和寄存器 Rr 加前位进位的值相比较操作。而寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作：Rd－Rr－C

语法：	操作码：	程序计数器：
CPC　Rd Rr	0≤d≤31，0≤r≤31	PC←PC＋1

16 位操作码：

0000	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \cdot Rr3 + Rr3 \cdot R3 + R3 \cdot \overline{Rd3}$　　N: R7

S: $N \oplus V$　　Z: $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0} \cdot Z$

V: $Rd7 \cdot \overline{Rr7} \cdot \overline{R7} + \overline{Rd7} \cdot Rr7 \cdot R7$　　C: $\overline{Rd7} \cdot Rr7 + Rr7 \cdot R7 + R7 \cdot \overline{Rd7}$

例子:(实践操作程序 4462. ASM)

```
    cp r2,r0          ;单步执行到此行,开始时在调试窗口的对应寄存器输入数据
                      ;设:(r2)= $ AA,(r0)= $ 55
lp: sec               ;(c)=1
    cpc r3,r1         ;第一次操作设:(r3)= $ 11, (r1)= $ 10
                      ;第二次操作设:(r3)= $ 12,(R1)= $ 10
    brsh  lp1         ;(r3)≥(r1)+(c)则转,小于则顺执
    rjmp  lp2         ;相对转移
lp1: inc r1           ;+1
    rjmp lp
lp2: dec  r1          ;-1
    rjmp lp           ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

3. 立即数比较

CPI- -带立即数比较

说明:该指令完成寄存器 Rd 和常数的比较操作。寄存器的内容不改变。该指令后能使用所有条件转移指令。

操作:Rd−K

语法:	操作码:	程序计数器:
CPI Rd, K	16≤d≤31, 0≤K≤255	PC←PC+ 1

16 位操作码:

0011	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	—	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \cdot K3 + K3 \cdot R3 + R3 \cdot \overline{Rd3}$　　N: R7

S: $N \oplus V$　　Z: $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V: $Rd7 \cdot \overline{K7} \cdot \overline{R7} + \overline{Rd7} \cdot K7 \cdot R7$　　C: $\overline{Rd7} \cdot K7 + K7 \cdot R7 + R7 \cdot \overline{Rd7}$

例子:(实践操作程序 4463. ASM)

```
LP: CPI   R19,3       ;单步执行到此行,开始时在调试窗口的对应寄存器输入数据
                      ;设(R19)=4
    BRNE LP1          ;不相等转,相等顺执
    INC R19           ;(R19)+1
    RJMP LP           ;反复测试
```

```
LP1: DEC R19          ;(R19)-1
     RJMP LP          ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

4.4.7　逻辑与指令

表 4.4 为逻辑与指令表。

表 4.4　逻辑与指令

逻辑与输入		输　出
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

1. 寄存器逻辑与

AND—　逻辑与;全 1 为 1,有 0 即 0。

说明:寄存器 Rd 和寄存器 Rr 的内容为逻辑与,结果送目的寄存器 Rd。

应用:清 0,使某位为 0,用 0 去与;保留,用 1 去逻辑与;代硬件与门。

操作:Rd←Rd · Rr

语法:	操作码:	程序计数器:
AND　Rd,Rr	0≤d≤31,0≤r≤31	PC←PC+1

16 位操作码:

0010	00rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	-	⇔	⇔	⇔	⇔	⇔	—

S:N ⊕ V　　N:R7

V:0　　Z:$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子:(实践操作程序 4471. ASM)

```
LP: add r2,r3          ;单步执行到此行,开始时在调试窗口的对应寄存器输入数据
                       ;设:(r2)=0B1000 0000,(r3)=0B0001 0011
    LDI r16,1          ;(r16)= 0B0000 0001,
    and  r2,r16        ;(r2)=
    RJMP LP            ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

2. 带立即数与

ANDI——立即数逻辑与;全 1 为 1,有 0 即 0。

说明：寄存器 Rd 的内容与常数逻辑与，结果送目的寄存器 Rd。

应用：清 0，使某位为 0，用 0 去与；保留，用 1 去逻辑与；代硬件与门。

操作：Rd←Rd • K

语法：	操作码：	程序计数器：
ANDI Rd K	16≤d≤31，0≤K≤255	PC←PC+1

16 位操作码：

0001	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
	—		⇔	0	⇔	⇔	—

S：N ⊕ V　　N：R7

V：0　　Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子：(实践操作程序 4472. ASM)

```
LP: andi   r17, $0F      ;单步执行到此行，开始时在调试窗口的对应寄存器输入数据
                         ;设：(r17)= $F0 ,(R18)= $EF ,(r19)= $FF,单步执行后，(r17)=
    andi   r18, $10      ;(r18)=
    andi   r19, $AA      ;(r19)=
    RJMP   LP            ;反复测试
```

Words：1(2 bytes)

Cycles：1

3. 清除寄存器位

CBR……清除寄存器指定位

说明：清除寄存器 Rd 中的指定位。利用寄存器 Rd 的内容与常数表征码 K 的补码相与完成的，其结果放在寄存器 Rd 中。

操作：Rd←Rd • ($FF−K)

语法：	操作码：	程序计数器：
CBR Rd,K	16≤d≤31，0≤K≤255	PC←PC+1

16 位操作码：

0111	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	--	⇔	0	⇔	⇔	—

S：N ⊕ V　　N：R7

V：0　　Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子：(实践操作程序 4473. ASM)

```
lp: cbr R16, $F0         ;单步执行到此行，在调试窗口的对应寄存器输入数据
                         ;(r16)= $FF
    CBR R18,1            ;(r18)= $80
    INC R16
```

```
    INC R18
    rjmp lp              ;反复输入数据测试
```

Words：1(2 bytes)

Cycles：1

4. 测试零或负

TST　--测试零或负

说明：测试寄存器是零或是负。完成同一寄存器之间的逻辑与操作，寄存器内容不改变。

操作：Rd←Rd · Rd

语法：	操作码：	程序计数器
TST　Rd	0≤d≤31	PC←PC+1

16位操作码：

0011	00dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	·	⇔	0	⇔	⇔	-

S：N ⊕ V　　N：R7

V：0　　Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子：(实践操作程序 4474.ASM)

```
    sub r0,r2        ;单步执行到此行，在调试窗口的对应寄存器输入数据
                     ;并打开状态寄存器 SREG 观察窗口
                     ;(r0)=$aa,(r2)=$aa
                     ;(r0)=$77,(r2)=$80
lp: tst   r0         ;测试寄存器 r0 是零或是负
    breq  lp1        ;如果零标志(Z)位为1，则转移；为0顺执
    dec r0           ;-1
    rjmp lp          ;再测试
lp1: inc r0          ;+1
    rjmp lp          ;再测试
```

Words：1(2 bytes)

Cycles：1

4.4.8　逻辑或指令

表4.5为逻辑或指令表。

表4.5　逻辑或

逻辑或输入		输　出
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

1. 寄存器逻辑或

OR —逻辑或;有 1 即 1,全 0 为 0,

应用: 置数,使某位为 1,用 1 去或;保留,用 0 去逻辑或;代硬件或门。

说明:完成寄存器 Rd 与寄存器 Rr 的内容逻辑或操作,结果送目的寄存器 Rd 中。

操作:Rd←Rd ∨ Rr

语法:	操作码:	程序计数器:
OR Rd,Rr	0≤d≤31, 0≤r≤31	PC←PC+ 1

16 位操作码:

0010	10rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—		—	⇔	0	⇔	⇔	-

S:N ⊕ V　　N:R7

V:0　　Z:$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子:(实践操作程序 4481.ASM)

```
    or   r19,r16        ;逻辑或,单步执行到此行,在调试窗口的对应寄存器输入数据
                        ;(r19)= $ AA, (r16)= $ 44, (r18)=0B0100 1111= $ 4F
                        ;                          (r18)=0B0000 1111= $ 0F
LP:bst   r18,6          ;r18 中的 6 位内容到 SREG 中 T 标志,观察 SREG 中 T 标志的变化
   brts ok              ;SREG 中标志为 1 转移,为 0 顺执
   SER R18              ;置位 r18
   RJMP LP              ;反复测试
ok:CLR R18              ;清零 r18
   RJMP LP              ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

2. 带立即数或

ORI—立即数逻辑或;功能: 保留(屏蔽)数据,置数(使某位为 1)。

说明:完成寄存器 Rd 的内容与常量逻辑或操作,结果送目的寄存器 Rd 中。

操作: Rd←Rd ∨ K

语法:	操作码:	程序计数器:
ORI Rd,K	16≤d≤31, 0≤K≤255	PC←PC+ 1

16 位操作码:

0110	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	-	⇔	0	⇔	⇔	—

S:N ⊕ V　　N:R7

V:0　　Z: $\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

例子：(实践操作程序 4482. ASM)

```
LP:ori r19, $ F0        ;立即数或,单步执行到此行,在调试窗口的对应寄存器输入数据
                        ;(r19)= $ A0    , (r17) = $ 45
                        ;(r19)=0B0110 1111= $ 6F   , (r17)=0B1111 0000= $ F0
   ori r17,1            ;立即数或
   lsl r19              ;r19 逻辑左移
   lsr r17              ;r17 逻辑右移
   RJMP LP              ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

3. 置寄存器位

SBR----寄存器位置位

说明：对寄存器 Rd 中指定位置位。完成寄存器 Rd 和常数表征码 K 之间的逻辑直接数或(ORI)，结果送目的寄存器 Rd。

操作：Rd←Rd ∨ K

语法：	操作码：	程序计数器：
SBR　Rd,K	16≤d≤31,0≤K≤255	PC←PC+1

16 位操作码：

0110	KKKK	dddd	KKKK

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S:N ⊕ V　　N:R7

V:0　　Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子：(实践操作程序 4483. ASM)

```
                        ;设:(r16)= $ F0,(r17)= $ 80
lp: sbr r16,3           ;对 r16 的(0B0000 0011)第 1,0 位置为 1,执行后(r16 )=
   sbr r17, $ 17        ;对 r17 的(0B0001 0111)第 4,2,1,0 位置 1,执行后(r17)=
   INC R16              ;+1
   DEC R17              ;-1
   rjmp lp              ;反复试验
```

Words: 1(2 bytes)

Cycles: 1

4. 置寄存器

SER——置位寄存器的所有位

说明：直接装入 $ FF 到寄存器 Rd。

操作：Rd←$ FF

语法：	操作码：	程序计数器：
SER　Rd	16≤d≤31	PC←PC+ 1

16 位操作码：

1110	KKKK	dddd	1111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	—	—	—	—	—	—

例子：(实践操作程序 4484. ASM)

```
LP:CLR R16          ;(r16)清零
   ser r17          ;(r17)置 $FF
   out $18,r16      ;输出到 B 口,打开 I/O 寄存器窗口看 $18 变化
   out $18,r17      ;输出到 B 口,打开 I/O 寄存器窗口看 $18 变化
   INC R16          ;+1
   INC R17          ;+1
   RJMP LP          ;反复实验
```

Words: 1(2 bytes)

Cycles: 1

4.4.9 逻辑异或指令

1. 寄存器异或

EOR ——异或；输入相同输出为 0，输入不同输出为 1；称同或(清零)；也称互斥(置 1)，见真值表 4.6。

表 4.6 异 或

异或输入		输 出
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

说明：完成寄存器 Rd 和寄存器 Rr 的内容相逻辑异或操作，结果送目的寄存器 Rd。

操作：Rd←Rd ⊕ Rr

语法：	操作码：	程序计数器：
EOR Rd,Rr	0≤d≤31, 0≤r≤31	PC←PC+ 1

16 位操作码：

0010	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	--	--	⇔	0	⇔	⇔	-

S:N ⊕ V　　N:R7

V:0　　Z:$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

例子:(实践操作程序 4491.ASM)

```
LP:eor   r4,r4          ;设:(r4)=0B1010 0011,相同数异或为清零,也可称同或清零
   eor   r0,r22         ;设:(r0)=0B1010 0101 设:(r22)=0B0101 0011
   SWAP R4              ;半字节交换
   SWAP R0              ;半字节交换
   SWAP R22             ;半字节交换
   RJMP LP              ;反复实验
```

Words:1(2 bytes)

Cycles: 1

2. 清除寄存器

CLR—— 寄存器清零

说明:寄存器清零。该指令采用寄存器 Rd 与自己的内容相异或实现的。寄存器的所有位都被清零。

操作: Rd←Rd ⊕ Rd

语法:	操作码:	程序计数器:
CLR　Rd	0≤d≤31	PC←PC+ 1

16 位操作码:

0001	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
·	↕	—	0	0	0	1	—

S:0　　V:0　　N:0　　Z:1

例子:(实践操作程序 4492.ASM)

```
 LP: CLR   R18          ;R18 清零
Lp1: inc r18            ;+1
     CPI R18,$05        ;R18 内容与立即数比较
     brne lp1           ;不相等转,相等顺执
     RJMP LP            ;循环测试
```

Words: 1(2 bytes)

Cycles: 1

4.5　转移指令

4.5.1　无条件转移指令

1. 相对跳转

RJMP——相对跳转

说明:相对跳转到 PC－2K 和 PC+2K(字)范围内的地址。在汇编程序中,标号用于替代相对操作。AVR 微控制器的程序存储器空间不超过 4K 字(8K 字节),该指令能寻址整个存储器空间的每个地址位置。

严格地讲：在 PC+1 后的 $-2K \leqslant k \leqslant 2K$ 范围内转移才能正确执行！

操作：PC←(PC+1)+ k

语法：	操作码：	程序计数器：
RJMP k	$-2K \leqslant k \leqslant 2K$	PC←(PC+1)+k

16 位操作码：

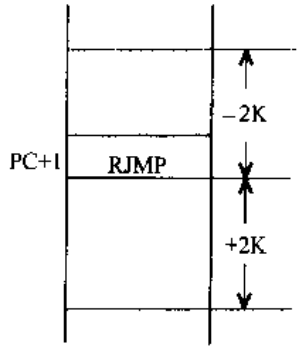
1100	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	·	··	—	—

例子：(实践操作程序 4511. ASM)

```
      .ORG 0X0010
LP:  cpi r16, $42        ;设：(r16)= $42
      brne LP2           ;不相等转移，相等顺执
      rjmp LP1           ;转移不超过+2K
LP2:  ADD R16,R17
      inc  r16
      .ORG $080F         ;ORG 为 $0810，则超过-2K；ORG
                         ;为 $080F 不超过-2K
                         ;因为 PC+1( $080F+1= $0810)，从 $0010 到 $0810 正好等于-2K
                         ;如 PC+1( $0810+1= $0811)，从 $0010 到 $0811 正好超过-2K
                         ;汇编时出错提示
LP1:  RJMP LP            ;请修改 ORG 为 $0810 一试，打开程序存储器窗口观察
                         ;用 RJMP 相对转移好处为作为子程序模块搬家(移动)较方便，不必修改
                         ;转移指令；缺点为子程序大小限在 PC←(PC+1)+k
```



Words：1(2 bytes)

Cycles：2

2. 间接跳转

IJMP——间接跳转

说明：间接跳转到由寄存器区中的 Z(16 位)指针寄存器指向的地址。Z 指针寄存器是 16 位宽，允许在当前程序存储器空间 64K 字(128K 字节)内跳转。

IJMP 间接跳转优点为转移范围大；缺点为作为子程序模块，移值时需修改转移地址，希望在子程序中不要使用！

注意：只能到设计的硬件电路所具有的空间。

操作：PC←Z(15－0)

PC(15－0)←Z(15－0)

语法：	操作码：	程序计数器：
IJMP	None	See Operation

16 位操作码：

1001	0100	XXXX	1001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4512.ASM)

```
    ORG $0010
LP: MOV R30,R0          ;设(R0)为Z寄存器低8位地址,R30为Z间接地址低8位
    MOV R31,R1          ;设(R1)为Z寄存器高8位地址,R31为Z间接地址高8位
    ijmp LP             ;根据Z寄存器的内容跳转
    .ORG $0200
    LDI R20,$11         ;地址$0201,如设(R0)=$01,(R1)=02,执行IJMP到此行
    LDI R21,$22         ;地址$0202,如设(R0)=$02,(R1)=02,执行IJMP到此行
    LDI R22,$33         ;地址$0203,如设(R0)=$03,(R1)=02,执行IJMP到此行
    LDI R23,$44         ;地址$0204,如设(R0)=$04,(R1)=02,执行IJMP到此行
    LDI R24,$55         ;地址$0205,如设(R0)=$05,(R1)=02,执行IJMP到此行
    LDI R25,$66         ;地址$0206,如设(R0)=$06,(R1)=02,执行IJMP到此行
    IJMP LP             ;地址$0207,如设(R0)=$07,(R1)=02,执行IJMP到此行
                        ;又根据Z寄存器的内容跳转到指定地址
```

Words: 1(2 bytes)

Cycles: 1

3. 长跳转

JMP----跳转

说明:在整个程序存储空间4M(字)内跳转,见RJMP。

JMP跳转优点为可跳转到程序存储器空间4M(字)任何地方;缺点为不适宜子程序模块中使用。

注意:只能到设计的硬件电路所具有的空间。

操作:PC←k

语法:	操作码:	程序计数器:
JMP k	0≤k≤4M	PC←k

32位操作码:

1001	010k	kkkk	1110k
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4513.ASM)

```
    equ lp1=$0f00
lp: mov r0,r1           ;设(r1)=12
    jmp lp2             ;长跳转
                        ;AT90S8515等AVR单片机无此指令
                        ;现在用AVR的MEG103单片机可使用该条指令
    .org $0f00
```

```
lp2: swap r0          ;半字节交换
     jmp lp           ;跳回反复实验
```

Words: 2(4 bytes)

Cycles: 3

4.5.2 条件转移指令

条件转移指令是以某种特定的条件转移的指令。条件满足则转移，条件不满足时则顺序执行下面的指令。

BRBS,BRBC 为基本指令，其余为根据 SREG 位内容派生出来的指令。

一、测试条件符合转移指令

1. 状态寄存器中位置位转移

BRBS——SREG 中的位被置位转移

说明：条件相对转移，测试 SREG 的某一位，如果该位被置位，则相对 PC 值转移。这条指令相对 PC 转移的方向为：PC−64≤目的寄存器≤PC+63。参数 K 为 PC 的偏移，用 2 的补码表示。

操作：if SREG(S)=1 then PC←(PC+1)+k, else PC←PC+1

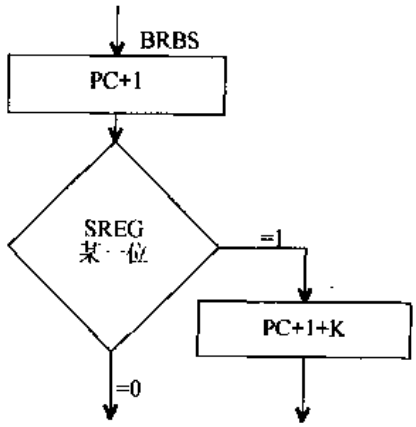
语法：	操作码：	程序计数器：
BRBS S, k	0≤S≤7	PC←(PC+ 1)+ k
	−64≤k≤+63	PC←PC+ 1

16 位操作码：

1111	00kk	kkkk	ksss

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子：(实践操作程序 4521. ASM)

```
LP: bst r0,3          ;设:(r0)=0B0000 1000;r0 中
                      ;第 3 位到 SREG 中的 T 标志
                      ;设:(r0)=0B0100 0100;r0 中
                      ;第 3 位到 SREG 中的 T 标志
    brbs 6,LP1        ; SREG 中的 6 位被置位(T=1)则转移,若为零顺执
    SET               ;置 T 标志为 1
    RJMP LP           ;反复测试
LP1: CLT              ;T 标志清零
    RJMP LP           ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if conditlon is true

2. 状态寄存器中位清零转移

BRBC——SREG 中的位被清零转移

说明：条件相对转移，测试 SREG 的某一位，如果该位被清零，则相对 PC 值转移。这条指

令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示。

操作：if　SREG(S)＝0　then　PC←(PC＋1)＋k，else　PC←PC＋1

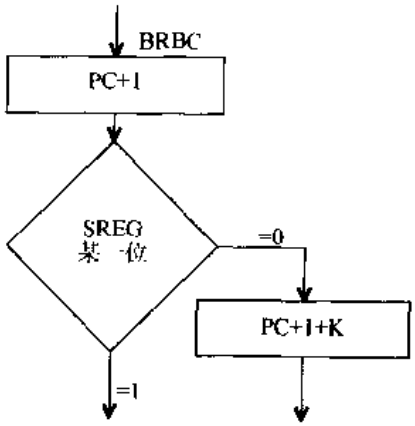
语法：	操作码：	程序计数器：
BRBC　S，k	0≤S ≤7	PC←(PC＋1)＋ k
	－64≤k≤＋63	PC←PC＋1

16 位操作码：

1111	01kk	kkkk	ksss

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
	—	·	—	—	—	—	—



例子：(实践操作程序 4522. ASM)

```
    CPI R20,5          ;R20 中内容与立即数 05 比，
                       ;设(R20)＝6 或 (R20)＝5
LP: BRBC 1,LP1         ;SREG 中位被清零则转移
                       ;为 1 顺执
    CLZ
    RJMP LP            ;反复测试
LP1: SEZ               ;Z＝1
    RJMP LP            ;反复测试
```

Words：1(2 bytes)

Cycles：1 if condition is false

2 if condition is true

3. 相等转移

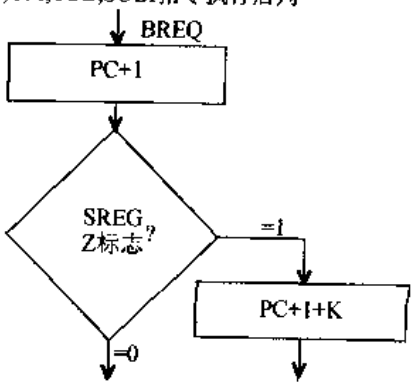
BREQ——相等转移

说明：条件相对转移，测试零标志(Z)，如果 Z 位被置位，则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当寄存器 Rd 中无符号或有符号二进制数与寄存器 Rr 中无符号或有符号二进制数相等时，转移将发生。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 1K)。

操作：if　Rd＝Rr(z＝1)　then　PC←(PC＋1)＋k，PC←PC＋1

语法：	操作码：	程序计数器：
BREQ　k	－64≤k≤＋63	PC←(PC＋ 1)＋ k
		PC←PC＋1

CP,CPI,SUB,SUBI指令执行后判



16 位操作码：

1111	00kk	kkkk	K001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	·	—		—		—	

例子:(实践操作程序 4523. ASM)

```
lp: cp r1,r2        ;设:(r1)=$AA,(r2)=$AA
    breq lp1        ;相等转移,不相等顺执,请同时观察 Z 标志
    inc r1          ;+1
    rjmp lp         ;反复验证
lp1: dec r1         ;-1
    rjmp lp         ;反复验证
```

Words: 1(2 bytes)

Cycles: 1 if condition is fslse

2 if condition is true

4. 不相等转移

BRNE——不相等转移

说明:条件相对转移,测试零标志(Z),如果 Z 位被清零,则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令,且当在寄存器 Rd 中的无符号或带符号二进制数不等于寄存器 Rr 中的无符号或带符号二进制数时,转移将发生。该指令相对 PC 转移的方向为:PC−64≤目的≤PC+63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBC 1K)。

操作: if Rd≠Rr(Z=0) then PC←(PC+1)+k, eles PC←PC+1

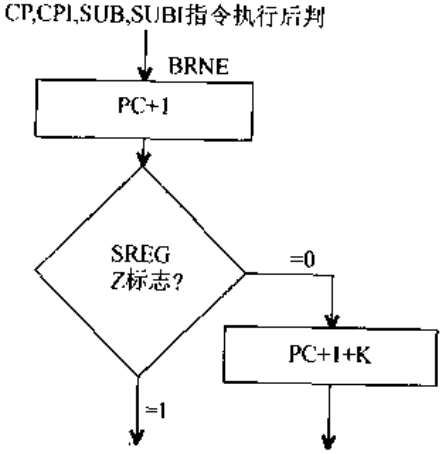
语法:	操作码:	程序计数器:
BRNE k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码:

1111	01kk	kkkk	K001

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
·	—	—	—	—	—	-	—



例子:(实践操作程序 4524. ASM)

```
lp: cp r1,r2        ;设:(r1)=$AB,(r2)=$AA
    brne lp1        ;不相等转移,相等顺执
                    ;请同时观察 Z 标志
    inc r1          ;+1
    rjmp lp         ;反复验证
lp1: dec r1         ;-1
    rjmp lp         ;反复验证
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

5. C 标志位置位转移

BRCS——进位位置位转移

说明:条件相对转移,测试进位标志(C),如果 C 位被置位,则相对 PC 值转移。这条指令

相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 0,K)。

操作：if C＝1 then　PC←(PC＋1)＋k,eles　PC←PC＋ 1

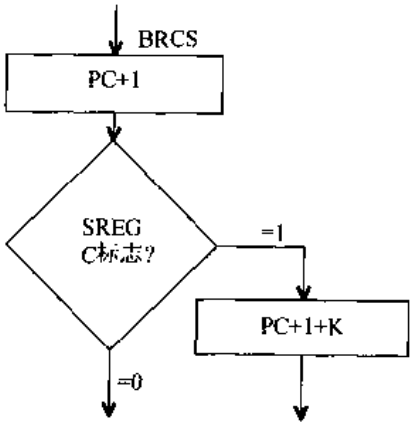
语法：	操作码：	程序计数器：
BRCS　k	－64≤ k≤＋63	PC←(PC＋1)＋k
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	K000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	·	—	--	—	·	—



例子：(实践操作程序 4525. ASM)

```
      SEC          ;C=1,请同时观察 C 标志
LP:  BRCS LP1      ;C=1 转,C=0 顺执
      SEC          ;C=1
      RJMP LP      ;重复实验
LP1: CLC           ;C=0
      RJMP LP      ;重复实验
```

Words：1(2 bytes)

Cycles：1　if condition is false

　　2　if condition is true

6. C 标志位清除转移

BRCC—— 进位位清除转移

说明：条件相对转移，测试进位标志(C)，如果 C 位被清除，则相对 PC 值转移。这条指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC0,K)。

操作：if C＝ 0 then　PC←(PC＋1)＋k,eles PC←PC＋1

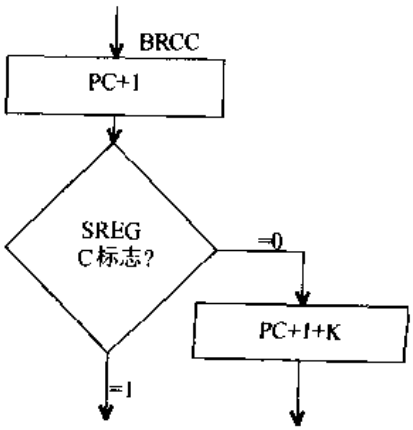
语法：	操作码：	程序计数器：
BRCC　k	－64≤ k≤＋63	PC←(PC＋1)＋k
		PC←PC＋1

16 位操作码：

1111	00kk	kkkk	K000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	--		·	—



例子：(实践操作程序 4526. ASM)

```
      CLC          ;C=0,请同时观察 C 标志
LP: BRCC LP1       ;C=0 转,C=1 顺执
      CLC          ;C=0
```

```
        RJMP LP     ;重复试验
LP1:    SEC         ;C=0
        RJMP LP     ;重复试验
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

7. 大于或等于转移

BRSH——大于等于转移(无符号)

说明:条件相对转移,测试进位标志(C),如果 C 位被清零,则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令,且当在寄存器 Rd 中无符号二进制数大于等于寄存器 Rr 中无符号二进制数时,转移将发生。该指令相对 PC 转移的方向为:PC−64≤目的≤PC+63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBS 0, K)。

操作:if Rd≥Rr(C=0)then PC←(PC+1)+k,eles PC←PC+1

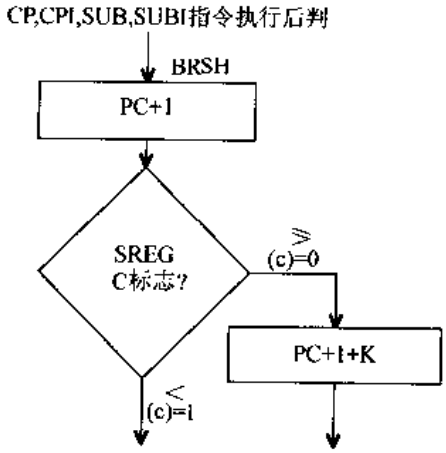
语法:	操作码:	程序计数器:
BRSH k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+ 1

16 位操作码:

1111	01kk	kkkk	K000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子:(实践操作程序 4527. ASM)

```
lp:  subi r19,2      ;带进位位立即数减
                     ;设:(r19)=1
     brsh lp1        ;大于等于转,小于顺执
                     ;请同时观察 C 标志
     inc r19         ;(r19)+1
     inc r19
     inc r19
     rjmp lp         ;反复实验
lp1: inc r19         ;(r19)+1
     rjmp lp         ;反复实验
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

8. 低于转移

BRLO——小于转移(无符号)

说明:条件相对转移,测试进位标志(C),如果 C 位被置位,则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令,且当在寄存器 Rd 中无符号二进制数小于

在寄存器 Rr 中无符号二进制数时，转移将发生。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 0，K)。

操作：if Rd< Rr(C=1)then PC←(PC+1)+k，eles PC←PC+1

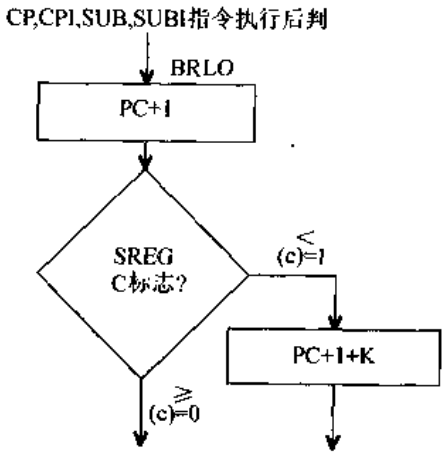
语法：	操作码：	程序计数器：
BRLO k	−64≤k≤+ 63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码：

1111	00kk	kkkk	k000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	-	—	—	—	—	—	—



例子：(实践操作程序 4528. ASM)

```
lp: subi r19,2      ;带进位位立即数减
                    ;设:(r19)=2
    brlo lp1        ;小于转,大于等于顺执
                    ;请同时观察 C 标志
    inc r19         ;(r19)−1
    rjmp lp         ;反复实验
lp1: inc r19        ;(r19)+1
    inc  r19
    inc  r19
    inc  r19
    rjmp lp         ;反复实验
```

Words：1(2 bytes)

Cycles：1 if condition is false

2 if condition is true

9. 负数转移

BRMI——负数转移

说明：条件相对转移，测试负号标志(N)，如果 N 被置位，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋ 63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS2，K)。

操作：if N=1 then PC←(PC+1)+ k，eles PC←PC+1

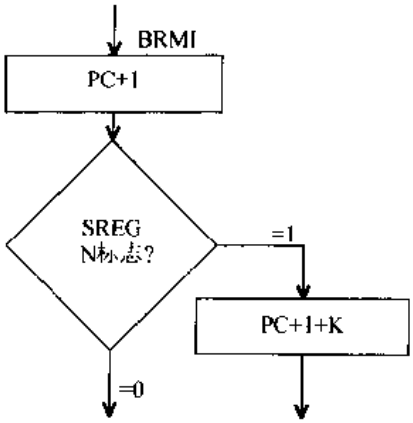
语法：	操作码：	程序计数器：
BRMI k	−64≤k≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码：

1111	00kk	kkkk	K010

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子:(实践操作程序 4529. ASM)

```
lp: subi   r18,2      ;带进位位立即数减,设:(r18)=2
    brmi lp1          ;为负转 N=1,为正顺执 N=0
                      ;请同时观察 N 标志
    inc r18           ;(r18)+1
    rjmp lp           ;反复实验
lp1: inc r18          ;(r18)+1
    inc r18
    inc r18
    inc r18
    rjmp lp           ;反复实验
```

Words: 1(2 bytes)

Cycles: 1 if cocdition is false

2 if condition is true

10. 正数转移

BRPL——正数转移

说明:条件相对转移,测试负号标志(N),如果 N 被清零,则相对 PC 值转移。该指令相对 PC 转移的方向为: PC−64≤目的≤PC+63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBC 2,K)。

操作:if N= 0 then PC←(PC + 1)+ k, eles PC←PC+ 1

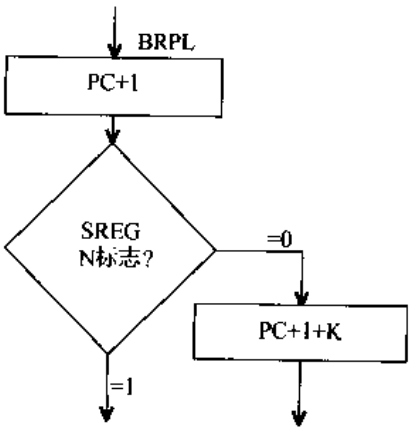
语法:	操作码:	程序计数器:
BRPL k	−64≤k≤+63	PC←PC+1)+ k
		PC←PC+1

16 位操作码:

1111	01kk	kkkk	K010

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子:(实践操作程序 45210. ASM)

```
lp: subi   r18,2      ;带进位位立即数减,设:(r18)=2
    BRPL lp1          ;为正转 N=0,为负顺执 N=1,请
                      同时观察 N 标志
    inc r18           ;(r18)+1
    inc r18
    inc r18
    inc r18
    rjmp lp           ;反复实验
lp1: dec r18          ;(r18)+1
```

```
        dec r18
        rjmp lp          ;反复实验
positive:nop
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

11. 大于或等于转移

BRGE——大于或等于转移(带符号)

说明:条件相对转移,测试符号标志(S),如果S位被清零,则相对PC值转移。如果在执行CP、CPI、SUB或SUBI指令后立即执行该指令,且当在寄存器Rd中带符号二进制数大于或等于寄存器Rr中带符号二进制数时,转移将发生。该指令相对PC转移的方向为:PC−64≤目的≤PC+63。参数K为PC的偏移,用2的补码表示(相当于指令BRBC 4,K)。

操作: if Rd≥Rr (N⊕V= 0) then PC←(PC+1)+k,eles PC← PC+1

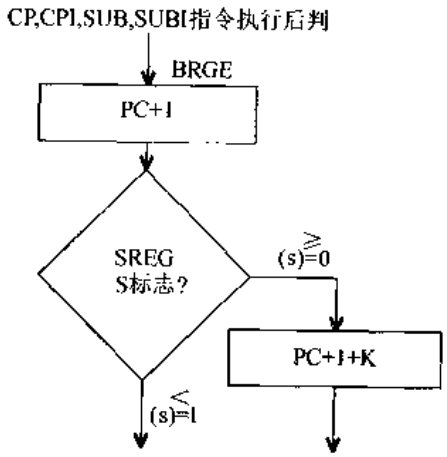
语法:	操作码:	程序计数器:
BRGE k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16位操作码:

1111	01kk	kkkk	K100

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	-	—	—	—	—	—



例子:(实践操作程序45211.ASM)

```
lp: subi  r18,2          ;带进位位立即数减
                         ;设:(r18)=2
    brge lp1             ;为大于或等于转 s=0,小
                          于顺执 s=1 请同时观察
                          s标志
    inc r18              ;(r18)+1
    inc r18
    inc r18
    inc r18
    rjmp lp              ;反复实验
lp1: dec r18             ;(r18)+1
    dec r18
    rjmp lp              ;反复实验
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

12. 小于转移

BRLT——小于转移(有符号)

说明：条件相对转移，测试符号标志(S)，如果 S 位被置位，则相对 PC 值转移。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中带符号二进制数小于在寄存器 Rr 中带符号二进制数时，转移将发生。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 4，K)。

操作：if Rd< Rf (N ⊕ V=1) then PC←(PC+1)+k, eles PC←PC+1

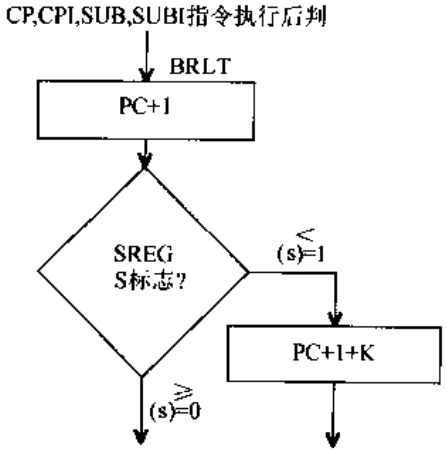
语法：	操作码：	程序计数器：
BRLT k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码：

1111	00kk	kkkk	k100

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	·	—



例子：(实践操作程序 45212. ASM)

```
lp: subi   r18,2      ;带进位位立即数减
                      ;设:(r18)=2
    brlt lp1          ;为小于转 S=0，大于或等于
                      ;顺执 S=1,请同时观察 S 标志
    inc r18           ;(r18)+1
    inc r18
    inc r18
    inc r18
    rjmp lp           ;反复实验
lp1: dec r18          ;(r18)-1
    dec r18
    rjmp lp           ;反复实验
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if conditlon is true

13. 半进位标志置位转移

BRHS——半进位标志置位转移

说明：条件相对转移，测试半进位标志(H)，如果 H 被置位，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS 5,K)。

操作：if H=1 then PC←(PC+1)+k, eles PC←PC+1

语法：	操作码：	程序计数器；
BRHS k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码：

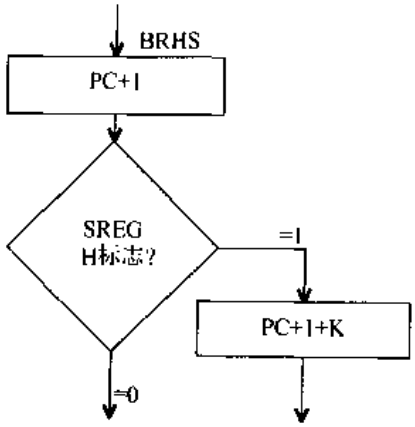
1111	00kk	kkkk	k101

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—		—	—	--

例子：(实践操作程序 45213. ASM)

```
lp: add r10,r11     ;加法设:(r10)=5 ,(r11)=5 ,(r12)=5
    brhs    lp1     ;半进位标志(H)=1 转移
                    ;(H)=0 顺执
    dec r10         ;(r10)-1
    dec r10
    rjmp    lp      ;反复实验
lp1: add r10,r12    ;加法
    rjmp lp         ;反复实验
```



Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

14. 半进位标志清零转移

BRHC-- —半进位标志被清零转移

说明：条件相对转移，测试半进位标志(H)，如果 H 位被清零，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBC 5,K)。

操作：if H=0 then PC←(PC+1)+k, eles PC←PC+ 1

语法：	操作码：	程序计数器：
BRHC k	−64≤ k ≤+ 63	PC←(PC+ 1)+ k
		PC←PC+ 1

16 位操作码：

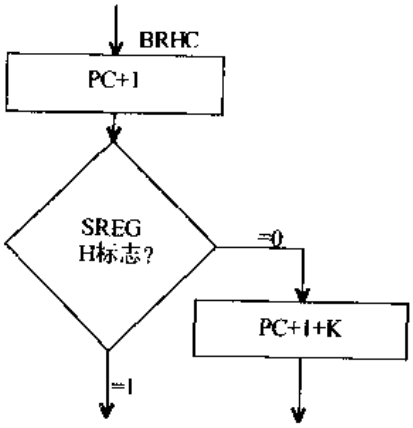
1111	01kk	kkkk	k101

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	…	-	—	—	-	-

例子：(实践操作程序 45214. ASM)

```
lp: add r10,r11     ;加法设:(r10)=5 ,(r11)=5
                    ;(r12)=5
    brhc    lp1     ;半进位标志(H)-0 转移,
                    ;(H)=1 顺执
    dec r10         ;(r10)-1
    dec r10
    rjmp    lp      ;反复实验
lp1: add r10,r12    ;加法
    rjmp lp         ;反复实验
```



Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if conditlon is true

15. T 标志置位转移

BRTS— T 标志被置位转移

说明:条件相对转移,测试 T 标志,如果 T 被置位,则相对 PC 值转移。该指令相对 PC 转移的方向为:PC−64≤目的≤PC+63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBS 6, K)。

操作:if T=1 then PC←(PC+1)+k, eles PC←PC+1

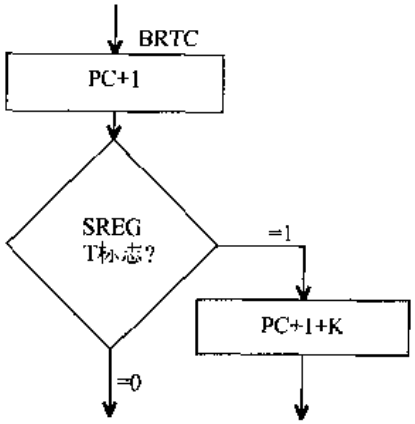
语法:	操作码:	程序计数器:
BRTS k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码:

1111	00kk	kkkk	k110

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	-	—	—	-	—



例子:(实践操作程序 45215. ASM)

```
lp: bst   r3,5      ;r3 中的 5 位到 SREG 中 T 标志
                    ;设:(r3)= $ AA
    brts LP1        ;(T)=1 转移,(T)=0 顺执
    rol r3          ;r3 通过进位位左循环
    rjmp lp         ;反复实验
lp1: rol r3         ;r3 通过进位位左循环
    rjmp lp         ;反复实验
```

Words: 1(2 bytes)

Cycles : 1 if condition is false

2 if conditlon is true

16. T 标志清零转移

BRTC—— T 标志被清零转移

说明:条件相对转移,测试 T 标志,如果 T 被清零,则相对 PC 值转移。该指令相对 PC 转移的方向为:PC−64≤目的≤PC+63。参数 K 为 PC 的偏移,用 2 的补码表示(相当于指令 BRBC 6,K)。

操作:if T=0 then PC←(PC+1)+k, eles PC←PC+1

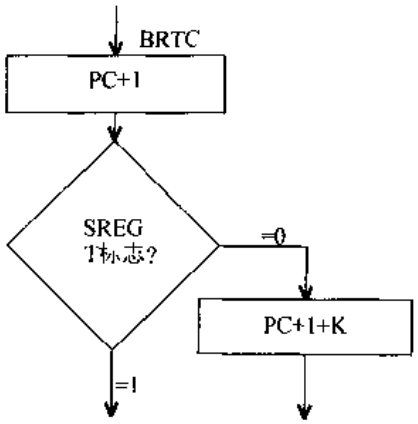
语法:	操作码:	程序计数器:
BRTC k	−64≤k≤+63	PC←(PC+1)+k
		PC←PC+1

16 位操作码:

1111	01kk	kkkk	k110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	·	·	·	—



例子：(实践操作程序 45216. ASM)

```
    bst   r3,5        ;r3 中的 5 位到 SREG 中 T
                      ;标志，设 r3= $55
lp: brtc  lp1         ;T 标志置 0 转移，置 1 顺执
    clt               ;(T)=0
    rjmp  lp          ;反复测试
lp1: set              ;(T)=1
    rjmp  lp          ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

17. 溢出标志置位转移

BRVS　- 溢出标志被置位转移

说明：条件相对转移，测试溢出标志(V)，如果 V 被置位，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC−64≤目的≤PC+63。参数 K 为 PC 的偏移，用 2 的补码表示(相当于指令 BRBS3,K)。

操作：if　V=1　then　PC←(PC+1)+k, eles　PC←PC+1

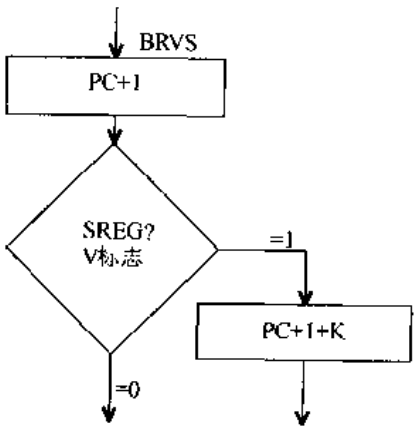
语法：	操作码：	程序计数器：
BRVS　k,	−64≤k≤+63	PC←(PC+1)+k
		PC←PC+1

16 位操作码：

1111	00kk	kkkk	k011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
·	—	—	—	·	—	—	—



例子：(实践操作程序 45217. ASM)

```
lp: brvs lp1      ;溢出标志位(V)=1 转移，(V) =0 顺执
    sev           ;溢出标志位(V)置 1
    rjmp lp       ;反复测试
lp1: clv          ;溢出标志位(V)置 0
    rjmp lp       ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

18. 溢出标志清零转移

BRVC—　溢出标志被清零转移

说明：条件相对转移，测试溢出标志(V)，如果 V 被清零，则相对 PC 值转移。该指令相对

PC 转移的方向为：PC−64≤目的≤PC+63。参数 K 为 PC 的偏移，用 2 的补码表示（相当于指令 BRBC3，K）。

操作：if V=0 then PC←(PC+1)+ k，eles PC←PC+1

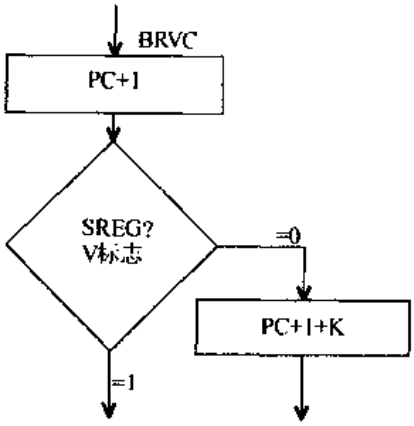
语法：	操作码：	程序计数器：
BRVC k	−64≤k≤+63	PC←(PC+1)+k PC←PC+1

16 位操作码：

1111	01kk	kkkk	k011

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	‥	‥	—	‥



例于：(实践操作程序 45218. ASM)

```
lp: brvc lp1      ;溢出标志位(V)=0 转移，(V)=1 顺执
    clv           ;溢出标志位(V)置 0
    rjmp lp       ;反复测试
lp1: sev          ;溢出标志位(V)置 1
    rjmp lp       ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if conition is true

19. 中断标志触发转移

BRIE——全局中断被触发转移

说明：条件相对转移，测试全局中断标志(I)，如果 I 被置位 1，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC−64≤目的≤PC+63。参数 K 为 PC 的偏移，用 2 的补码表示（相当于指令 BRBS7，K）。

操作：if I=1 then PC←(PC+1)+k，eles PC←PC+1

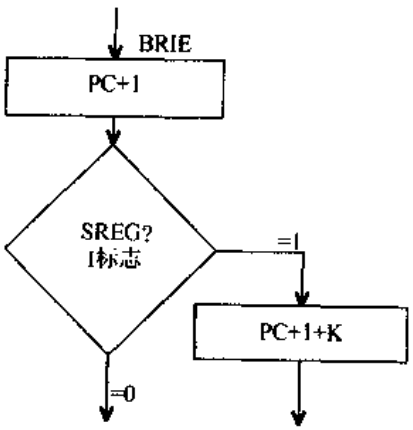
语法：	操作码：	程序计数器：
BRIE k	−64≤k≤+63	PC←(PC+1)+k PC←PC+1

16 位操作码：

1111	00kk	kkkk	k111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子：(实践操作程序 45219. ASM)

```
BRIE LP1    ;全局中断标志(I)=1 转移，
            ;用鼠标在状态寄存器窗口中 I 标志
            ;点一下
            ;(I)=0 顺执
```

```
    SEI          ;(I)=1
    RJMP LP      ;反复测试
LP1:CLI          ;(I)=0
    RJMP LP      ;反复测试
```

words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

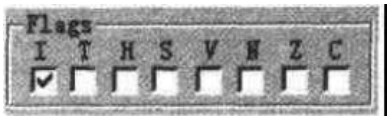


图 4.14　状态寄存器标志位

20. 中断标志禁止转移

BRID——全局中断被禁止转移

说明：条件相对转移，测试全局中断标志(I)，如果 I 被清零，则相对 PC 值转移。该指令相对 PC 转移的方向为：PC－64≤目的≤PC＋63。参数 K 为 PC 的 b 偏移，用 2 的补码表示(相当于指令 BRBC 7，K)。

操作：if　I=0　then　PC←PC+1)+k，else　PC←PC+ 1

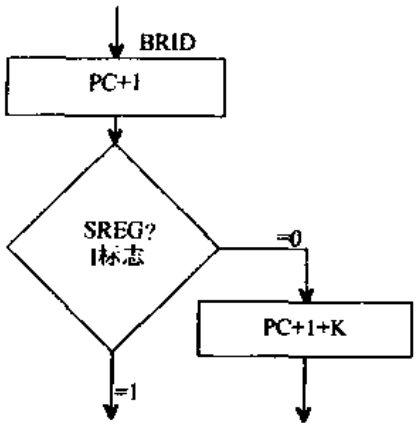
语法：	操作码：	程序计数器：
BRID　k	−64≤ k ≤+63	PC←(PC+1)+ k
		PC←PC+1

16 位操作码：

1111	01kk	kkkk	k111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子：(实践操作程序 45220. ASM)

```
LP: BRID LP1     ;全局中断标志(I)=0 转移
    CLI          ;(I)=1 顺执;(I)=0
    RJMP LP      ;反复测试
LP1: SEI         ;(I)=1
    RJMP LP      ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false

2 if condition is true

二、测试条件符合跳行转移指令

21. 相等跳行

CPSE——比较相等跳行

说明：该指令完成两个寄存器 Rd 和 Rr 的比较，若 Rd=Rr，则跳行执行指令。

操作：　if　Rd=Rr　then　PC←PC+2(or 3)　else　PC←PC+1

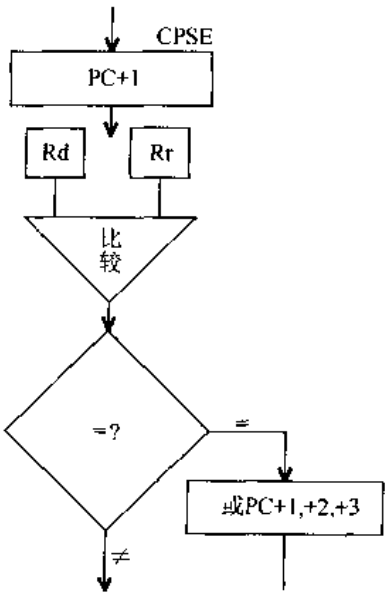
语法：	操作码：	程序计数器：
CPSE　Rd，Rr	0≤d≤31，0≤r≤31	PC←PC+ 1
		PC←PC+ 2
		PC←PC+ 3

16 位操作码：

0001	01rd	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—



例子：(实践操作程序 45221. ASM)

```
lp: inc     r4          ;设(r4)=$77 ,(r0)=$77
                        ;设(r4)=$76 ,(r0)=$77
lp1: cpse r4,r0         ;r4 与 r0 比较相等跳行,不等顺执
     lds r10,$0100      ;这是四字节指令把 SRAM 地址
                         $0100 的数据装入 R10
     nop
     dec r4             ;(r4)-1
     cpse r4,r0         ; r4 与 r0 比较相等跳行,不等顺执
     rjmp lp1           ;这是二字节指令
     mov r10,r0         ;拷贝
     rjmp lp            ;反复测试
```

Words: 1(2 bytes)

Cycles: 1

22. 寄存器位清零跳行

SBRC——寄存器位被清零跳行

说明：该指令测试寄存器某位，如果该位被清零，则跳下一行执行指令。

操作：if Rd(b)=0 then PC←PC+2(or 3) eles PC←PC+ 1

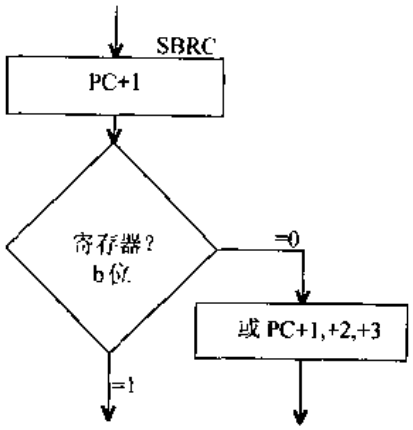
语法：	操作码：	程序计数器：
SBRC Rr,b	0≤r≤31	PC←PC+1
	0≤b≤7	PC←PC+ 2
		PC←PC+ 3

16 位操作码：

1111	110r	rrrr	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	--	—	--	-	—	--	--



例子：(实践操作程序 45222. ASM)

```
lp: sub r0,r1       ;设:(r0)=$80,(r1)=$70
    sbrc r0,7       ;r0 的 7 位清零跳行,为 1 顺执
    sub r0,r1       ;这是二字节指令
    swap r0         ;r0 半字节交换
    rjmp lp         ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is executed)

23. 寄存器位置位跳行

SBRS　—寄存器位置位跳行

说明：该指令测试寄存器某位，如果该位被置位，则跳下一行执行指令。

操作：if　Rr(b)=1　then　PC←PC+2(or 3)　eles　PC←PC+1

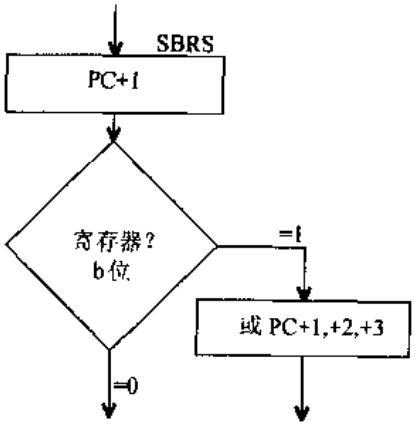
语法：	操作码：	程序计数器：
SBRS　Rr，b	0≤r≤31	PC←PC+1
	0≤b≤7	PC←PC+2
		PC←PC+3

16 位操作码：

1111	111r	rrrr	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
	-	—	-	—		—	—



例子：(实践操作程序 45223. ASM)

```
lp: sub r0,r1      ;设:(r0)=$80,(r1)=$70
    sbrs r0,7      ;r0 的 7 位置位跳行,为 0 顺执
    sub r0,r1      ;这是二字节指令
    swap r0        ;r0 半字节交换
    rjmp lp        ;反复测试
```

Words: 1(2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is executed)

24. I/O 寄存器位清零跳行

SBIC——I/O 寄存器的位清零跳行

说明：该指令测试 I/O 寄存器某位，如果该位被清零，则跳一行执行指令。该指令在低 32 个 I/O 寄存器内操作，地址为：0～31。

操作：if　I/O P，b= 0　then　PC←PC+2(or 3)　eles　PC←PC+1

语法：	操作码：	程序计数器：
SBIC　P，b	0≤P≤31，0≤b≤7	PC←PC+1
		PC←PC+2
		PC←PC+3

16 位操作码：

1001	1001	PPPP	Pbbb

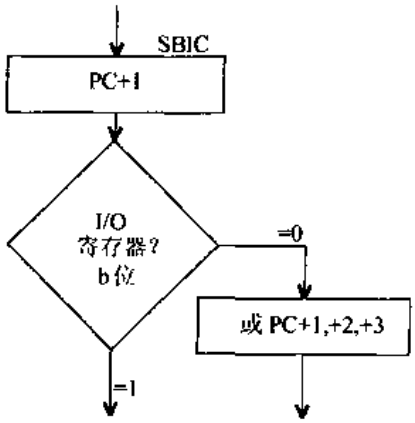
状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	-	—	-	—

例子：(实践操作程序 45224. ASM)

图 4.15 为 I/O 地址数据窗口。

```
lp: sbic $1c,1      ;复位后(EECR)=$00,.equ EECR
                    ;=$1c,测试1位,为0跳行,为1顺
                    ;执
                    ;请打开I/O窗口观察
;注意:这条指令只能连续测试1次,第2次会改变$1C
      数据(反复测试不能用该指令)
   rjmp lp          ;这是二字节指令
   ldi R16,$0f      ;
   OUT $1C,R16      ;置I/O寄存器1位为1
   rjmp lp
```



Words: 1(2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is exectued)

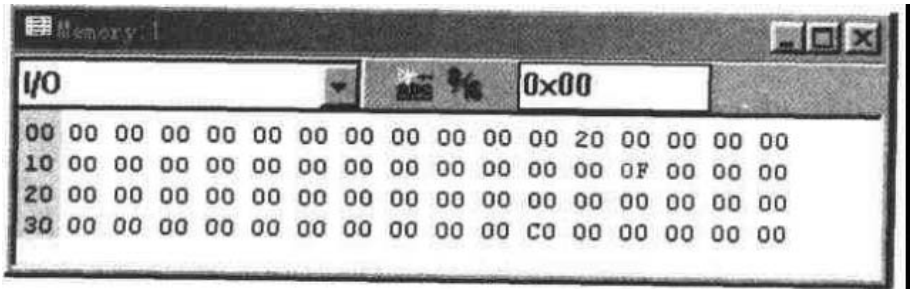


图 4.15　I/O 地址数据窗口

25. I/O 寄存器位置位跳行

SBIS——I/O 寄存器的位置位跳行

说明:该指令测试 I/O 寄存器某位,如果该位被置位,则跳一行执行指令。该指令在低 32 个 I/O 寄存器内操作,地址为 0～31。

操作: if　I/O P, b=1　then　PC←PC+2(or 3)　else　PC←PC+1

语法:	操作码:	程序计数器:
SBIS　P ,b	0≤P≤31, 0≤b≤7	PC←PC+ 1
		PC←PC+ 2
		PC←PC+ 3

16 位操作码:

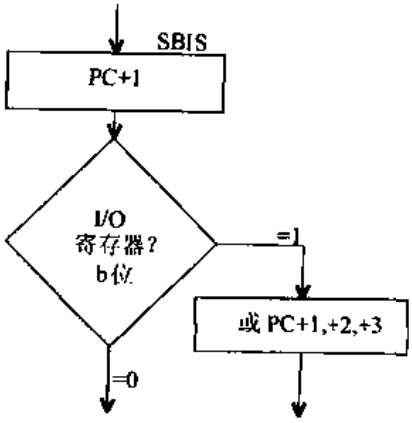
1001	1011	PPPP	Pbbb

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 45225.ASM)

```
lp: ldi R16,$0f     ;
    OUT $1C,R16     ;置I/O寄存器1位为1
Lp1:sbis $1c,I      ;复位后(EECR)=$00,.equ EE-
```



```
                    ;CR= $ 1c,测试 1 位,为 1 跳行,为 0 顺执
                    ;请打开 I/O 窗口观察
;注意:这条指令只能连续测试 1 次,第 2 次会改变 $ 1C 数据(反复测试不能用该指令),见 45225B. ASM
    rjmp lp         ;这是二字节指令
    ldi R16, $00    ;
    OUT $ 1C,R16    ;置 I/O 寄存器 1 位为 0
    rjmp lp1
```

Words: 1(2 bytes)

Cycles: 1 if condition is false(no skip)

2 if condition is true(skip is executed)

三、调用和返回指令

在程序设计中通常把具有一定功能模块的公用程序段定义为子程序。为了实现调用子程序的功能,指令系统中都有调用指令,也称转移子程序指令。它与转移指令的区别如下:执行调用子程序时,把下一条指令地址(PC 值)保留堆栈中,即断点保护,然后把子程序的起始地址置入 PC;子程序执行完毕,再从断点返回,从断点处继续执行源程序。而转移指令既不保护断点,也不返回源程序。在每个子程序中都必须有返回指令,返回指令的功能就是把调用前压入堆栈的断点弹出置入 PC,恢复调用前的源程序。

在一个程序中,子程序中还会调用别的子程序,这称为子程序嵌套。每次调用子程序时必须将下条指令地址保存起来,返回时按后进先出原则依次取出旧 PC 值。堆栈就是按后进先出规律存取数据的,调用指令和返回指令具有自动的保存和恢复 PC 内容的功能,即自动进栈、自动出栈。

26. 相对调用

RCALL——相对调用子程序

说明:在 PC+1 后[2K 字(4K 字节)范围内调用子程序。返回地址(RCALL 后的指令地址)存储到堆栈(见 CALL)。

操作:PC←(PC+1)+k

语法:	操作码:	程序计数器:
RCALL k	−2K≤k≤2K	PC←(PC+1)+ k

16 位操作码:

1101	kkkk	kkkk	kkkk

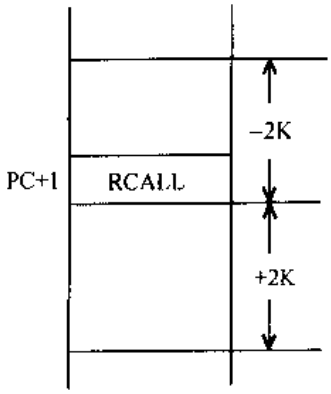
状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 45226. ASM)

```
ser     temp                ;直接装入 $ FF,
out     DDRA,     temp      ;口的方向寄存器设定,为输出
forever:
clr     temp                ;硬件设低电平 LED 灯亮
out     PORTA,    temp      ;PORTA 口 LED 灯亮
ldi     temp,1200           ;装延时常数(十进制),灯亮延时,
```

```
rcall    delay_ms          ;调用延时子程序
ser      temp              ;硬件设高电平 LED灯灭
out      PORTA,    temp    ;PORTA 口 LED灯灭
ldi      temp,1200         ;装延时常数,灯灭延时,可修改该参数
rcall    delay_ms          ;调用延时子程序
rjmp     forever           ;无限循环
         delay_ms:         ;延时子程序
ldi      zl,low(2500)      ;装入延时参数,参数可修改
ldi      zh,high(2500)     ;参数可修改
delay_loop:
dec      zl                ;-1
brne     delay_loop        ;不相等(Z=0)转移,相等(Z=1)顺执
dec      zh                ;-1
brne     delay_loop        ;不相等(Z=0)转移,相等(Z=1)顺执
dec      temp              ;-1
brne     delay_ms          ;不相等(Z=0)转移,相等(Z=1)顺执
ret                        ;子程序返回
```



Words: 1(2 bytes)

Cycles: 3

27. 间接调用

ICALL——间接调用子程序

说明:间接调用由寄存器区中的 Z(16 位)指针寄存器指向的子程序。Z 指针寄存器是 16 位宽,允许调用当前程序存储空间 64K 字(128 字节)内的子程序。

操作: PC(15—0)←Z(15—0)

PC(15—0)←Z(15—0)

语法:	操作码:	程序计数器:
ICALL	None	See operation

16 位操作码:

1001	0101	XXXX	1001

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—		—	—	—

例子:(实践操作程序 45227. ASM)

```
    .ORG $0010
LP: MOV R30,R0          ;设(R0)为 Z 寄存器低 8 位地址,R30 为 Z 间接地址低 8 位
    MOV R31,R1          ;设(R1)为 Z 寄存器高 8 位地址,R31 为 Z 间接地址高 8 位

    ICALL               ;根据 Z 寄存器的内容调用,该步用进入子程序图标调试
    .ORG $0200
    LDI R20,$11         ;地址 $0200 ,如设(R0)= $00,(R1)=02,执行 ICALL 到此行
    LDI R21,$22         ;地址 $0201 ,如设(R0)= $01,(R1)=02,执行 ICALL 到此行
```

```
LDI R22, $33          ;地址 $0202 ,如设(R0)= $02,(R1)=02,执行 ICALL 到此行
LDI R23, $44          ;地址 $0203 ,如设(R0)= $03,(R1)=02,执行 ICALL 到此行
LDI R24, $55          ;地址 $0204 ,如设(R0)= $04,(R1)=02,执行 ICALL 到此行
LDI R25, $66          ;地址 $0205 ,如设(R0)= $05,(R1)=02,执行 ICALL 到此行
ICALL                 ;地址 $0205 ,如设(R0)= $06,(R1)=02,执行 ICALL 到此行
                      ;又根据 Z 寄存器的内容调用到指定地址
move  r30, r0
icall
```

Words: 1(2 bytes)

Cycles: 3

28. 长调用

CALL——子程序长调用

说明:在整个程序存储器区内调用子程序。返回地址(调用后返回的指令地址)将存储在堆栈(见 RCALL 指令)中。

操作:PC←k

PC←k

语法:	操作码:	程序计数器:
CALL k	0≤k ≤64K	PC←k TACK←PC+ 2 SP←SP−2
CALL k	0≤k ≤4M	PC←k STACK←PC+ 2 SP←SP−3

32 位操作码:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 45228. ASM)

```
move    r16,r0

call    delay_ms          ;长调用延时子程序,该步用进入子程序图标[redacted]调试
.ORG $0200
delay_ms:                 ;延时子程序
ldi     zl,low(1990)
ldi     zh,high(1990)
delay_loop:
dec     zl
brne    delay_loop
dec     zh
brne    delay_loop
dec     temp
```

```
brne    delay_ms
ret                          ;子程序返回
```

Words：2(4 bytes)

Cycles：4

29. 从子程序返回

RET——子程序返回

说明：从子程序返回。返回地址从堆栈中弹出。

操作：PC(15～0)←STACK

PC(21～0)←STACK

语法：	操作码：	程序计数器：	堆栈：
RET	None	See Operation	SP←SP+ 2
RET	None	See Operation	SP←SP+ 3

16 位操作码：

1001	0101	0XX0	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 45229. ASM)

同实践操作程序 45228. ASM。

Words：1(2 bytes)

Cyclse：4

30. 从中断程序返回

RETI——中断程序返回

说明：从中断程序中返回。返回地址从堆栈中弹出，且全局中断标志被置位。

注意：(1) 主程序应跳过中断区，防止修改补充中断程序带来麻烦；

(2) 不用的中断入口地址写上 RETI——中断返回，有抗干扰作用。

操作：PC(15～0)← STACK

PC(21～0)←STACK

语法：	操作码：	程序计数器：	堆栈：
RETI	None	See Operation	SP←SP+2
RETI	None	See Operation	SP←SP+3

16 位操作码：

1001	0101	0XX0	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	-	—	-	—	-	—	-

例子：(程序 45230. ASM，摘自“乐曲. ASM”部分程序，仅供参考)，能执行程序请阅“乐曲. ASM”程序及“AVR 单片机在儿童智能玩具中的应用——音乐玩具(电脑放音机)”一文

```
            .cseg
            .org    0x06              ;timerl 中断入口地址
inttl:      RJMP    OUTPM             ;转中断服务子程序
            .cseg
            .org    0x010             ;
OUTPM:      OUT     TCNT1H,TONH       ;中断服务子程序
            OUT     TCNT1L,TONL       ;
            SBIS    PORTC,00          ;
            RJMP    SETOP1            ;
SETOP0:     CBI     PORTC,00          ;
            LDI     MUSN,$00          ;
            RET1                      ;中断返回
SETOP1:     SBI     PORTC,00          ;
            LDI     MUSN,$01          ;
            RETI                      ;中断返回
extint:push r0 pop    r0
            reti
Words: 1(2 bytes)
Cyclse: 4
```

4.6　数据传送指令

数据传送指令是在编程时使用最频繁的一类指令。数据传送指令是否灵活、快速对程序的执行速度产生很大影响。数据传送指令是执行寄存器与寄存器、寄存器与数据存储器(SRAM)、寄存器与 I/O 端口之间的数据传送。另外还有从程序存储器直接取数指令 LPM 以及 PUSH(压栈)和 POP(出栈)的堆栈指令。

4.6.1　直接数据传送指令

1. 寄存器拷贝数据

MOV——寄存器拷贝

说明:该指令将一个寄存器拷贝到另一个寄存器。源寄存器 Rr 的内容不改变,而目的寄存器 Rd 拷贝了 Rr 的内容。

操作:Rd←Rr

语法:	操作码:	程序计数器:
MOV　Rd, Rr	0≤d≤31, 0≤r≤31	PC←PC+ 1

16 位操作码:

0010	11rd	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	--	—	--	—	--	—

例子:(实践操作程序 4611. ASM)

```
lp: mov r16,r0      ;把 r0 的内容传送到 r16 中
    rcall check     ;调用子程序
    rjmp lp         ;反复实验
    .org $0100      ;子程序首址
    check:swap r0   ; r0 半字节交换
    ret             ;子程序返回
```

Words: 1(2 bytes)

Cycles: 1

2. SRAM 数据直接送寄存器

LDS——直接从 SRAM 装入数据

说明:把 SRAM 中 1 个字节装入到寄存器,必须提供一个 16 位地址。存储器访问被限制在当前 64K 字节的 SRAM 页,超过 64K 字节时 LDS 指令使用 RAMPZ 寄存器访问。

操作:Rd←(k)

语法:	操作码:	程序计数器:
LDS Rd, k	0≤d≤31,0≤k≤65 535	PC←PC+2

32 位操作码:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4612. ASM)

图 4.16 为片内 RAM 数据窗口。

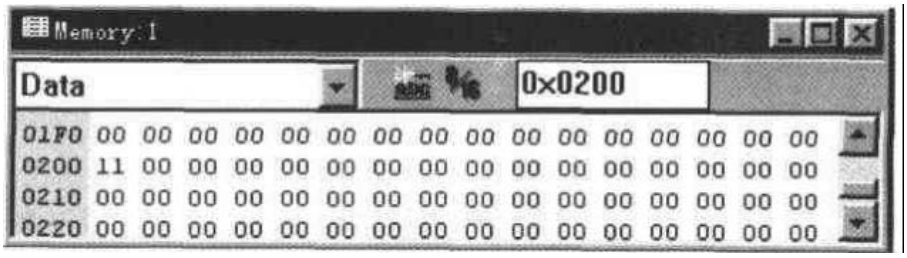


图 4.16 片内 RAM 数据窗口

```
LP: lds r2, $0200     ;把片内 SRAM 地址 $0200 数据装入,设:($0200)= $11
    add r2,r1         ;r2 与 r1 内容相加,结果存于 R2 中,设:(r1)= $88
    sts $0200,r2      ;r2 的内容送到片内 SRAM 地址 $0200
    RJMP LP           ;打开片内 SRAM 窗口观察,反复测试
```

Words: 2(4 bytes)

Cycles: 3

3. 寄存器数据直接送 SRAM

STS——寄存器数据直接送 SRAM

说明：将寄存器的内容直接存储到SRAM，必须提供一个16位的地址。存储器访问被限制在当前64K字节的SRAM页，STS指令使用RAMPZ寄存器访问存储器可超过64K字节。

操作：(k)←Rr

语法：	操作码：	程序计数器：
STS　k，Rr	0≤r≤31，0≤k≤65 535	PC←PC+2

32位操作码：

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	·		

例子：(实践操作程序4613.ASM与4612.ASM相同)程序略

4. 立即数送寄存器

LDI——装入立即数

说明：装入一个8位立即数到寄存器R16～R31中。

操作：Rd←K

语法：	操作码：	程序计数器：
LDI　Rd，K	16≤d≤31，0≤K≤255	PC←PC+ 2

16位操作码：

1110	KKKK	dddd	KKKK

状态寄存器(SRE)和布尔格式：

I	T	H	S	V	N	Z	C
—	· ·	—	—		—	—	

例子：(实践操作程序4614.ASM)

```
clr   r31          ;r31清零
ldi   r30,$F0      ;立即数送r30中，r16≤r≤r31
lpm                ;装入程序存储器，请观察Z寄存器内容
```

Words：1(2 bytes)

Cycles：1

4.6.2　间接数据传送指令

一、使用X寄存器间接传送数据

1. 使用变址X间接将SRAM中内容送入到寄存器

LD— —使用变址X间接将SRAM中内容送入到寄存器

说明：从SRAM中送入一个字节到寄存器，SRAM中的位置由寄存器区中的X(16位)指针寄存器指出。存储器访问被限制在当前64K字节的SRAM页中，为访问另外SRAM页，则I/O范围内的寄存器RAMPX需改变。在指令执行中，X指针寄存器值要么不改变，要么就加1或减1操作。使用X指针寄存器的这些特性，特别适合于访问矩阵、表和堆栈指针等。

操作：Rd←(X)　　；送数，X指针寄存器值不改变

Rd←(X) X←X+1 ;先送数,后 X 指针寄存器值加 1

X←X−1 Rd←(X) ;先 X 指针寄存器值减 1,后送数

语法：	操作码：	操作流程：	程序计数器：
LD Rd, X	0≤d≤31	送数,X 指针不改变	PC←PC+1
LD Rd,X+	0≤d≤31	先送数,后 X 指针加 1	PC←PC+1
LD Rd,−X	0≤d≤31	先 X 指针减 1,后送数	PC←PC+1

16 位操作码：

1	1001	000d	dddd	1100
2	1001	000d	dddd	1101
3	1001	000d	dddd	1110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4621. ASM)

图 4.17 为寄存器 CPU 数据窗口。

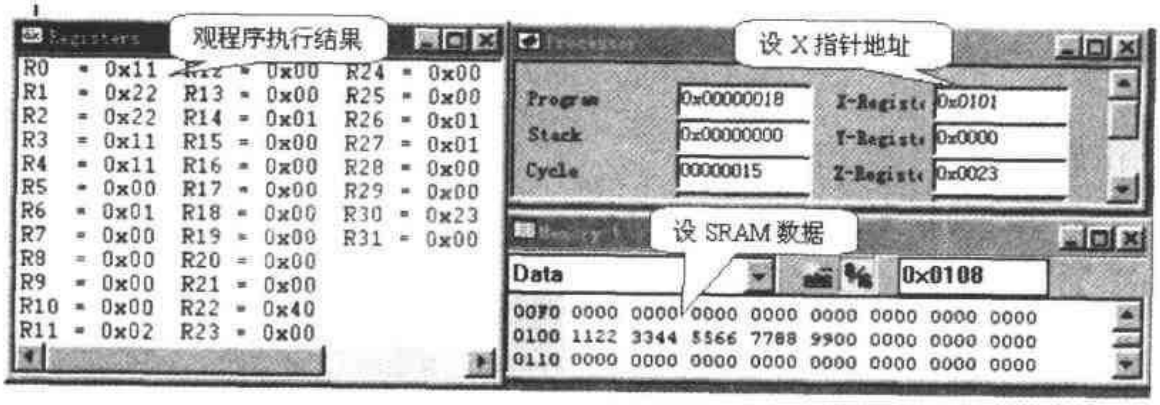


图 4.17 寄存器 CPU 数据窗口

```
LP: clr   r31       ;清零 Z 寄存器高位
    ldi   r30,$20   ;$20 送 Z 寄存器低位
    ld    r0,X+     ;执行 X 寄存器,X+1 为地址的 SRAM 内容送 R0,设:SRAM($0100)=
                    ;$11
                    ;($0101)=$22 ,($0102)=$33;再设 X 寄存器(X)=$0100
                    ;这时(X)=$0100,先单步执行(R0)=$11 ,然后(X)+1=$0101
                    ;X 寄存器高位地址为 R27,低位地址为 R26, SRAM 地址≥$60
    ld    r1,X      ;这时(X)=$0101 单步执行后,(R1)=$22
    ldi   r30,$23   ;单步执行后(R30)=$23
    ld    r2,X      ;这时(X)=$0101,单步执行后(R2)=$22
    ld    r3,-X     ;单步先执行(X)-1=$0100,然后执行(R3)=$11
    ld    r4,X+     ;先单步执行,这时(X)=$0100,(R4)=$11,然后执行(X)+1=$0101
    NOP             ;再从寄存器窗口看程序执行情况
    RJMP LP
```

Words：1(2 bytes)

Cycles：2

2. 使用变址 X 间接将寄存器内容传送到 SRAM

ST—— 使用变址 X 间接将寄存器内容传送到 SRAM

说明：间接将寄存器的一个字节传送到 SRAM。SRAM 的位置由寄存器区中的 X(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围的寄存器 RAMPX 将被修改。在操作之后，X 指针寄存器要么不改变，要么是加 1 或减 1。使用 X 指针寄存器的这些特性，特别适合用作堆栈指针。

操作：(X)←Rr

X ← Rr　　X←X+1

X ← X−1　　(X)←Rr

语法：	操作码：	操作流程：	程序计数器：
ST X,Rr	0≤d≤31	送数，X 指针不改变	PC←PC+1
ST X+,Rr	0≤d≤31	先送数，后 X 指针加 1	PC←PC+1
ST −X,Rr	0≤d≤31	先 X 指针减 1，后送数	PC←PC+1

16 位操作码：

1	1001	001r	rrrr	1100
2	1001	001r	rrrr	1101
3	1001	001r	rrrr	1110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	-	—	—		—	—	-

例子：(实践操作程序 4622.ASM)

```
lp: clr   r27          ;X 寄存器高位清零，X 寄存器高位地址为 R27，低位地址为 R26
    ldi   r26,$70      ;$70 送 X 寄存器低位，SRAM 地址≥$60
    st    X+,r0        ;设(R0)=$11，先单步执行送 SRAM(X)=$0070，然后(X)+1=$0071
    st    X,r1         ;设(R1)=$22，这时(X)=$0071，单步执行后 SRAM($0071)=$22
    ldi   r26,$80      ;$80 送 X 寄存器低位
    st    x,r2         ;设(R2)=$33，这时(X)=$0080，单步执行后 SRAM($0080)=$33
    st    −x,r3        ;设(R3)=$44，这时先单步执行(X)−1=$007F，然后 R3 内容送 SRAM
                       ;($007F)
    nop                ;
    rjmp  lp           ;反复测试
```

Words：1(2 bytes)

Cycles：2

二、使用 Y 寄存器间接传送数据

3. 使用变址 Y 间接将 SRAM 中的内容传送到寄存器

LD(LDD)——使用变址 Y 间接将 SRAM 中的内容传送到寄存器

说明：带或不带偏移间接从 SRAM 中传送一个字节到寄存器，SRAM 中的位置由寄存器区中的 Y(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访

问另外 SRAM 页，则 I/O 范围内的寄存器 RAMPY 需改变。在指令执行后，Y 指针寄存器值要么不改变，要么就加 1 或减 1 操作。使用 Y 指针寄存器的这些特性，特别适合于访问矩阵、表和堆栈指针等。

操作：Rd←(Y)

Rd←(Y)

Y←Y−1　　Y←Y+ 1

Rd←(Y+q)　　Rd←(Y)

语法：	操作码：	操作流程：	程序计数器：
LD Rd,Y	0≤d≤31	送数，Y 指针不改变	PC←PC+1
LD Rd Y+	0≤d≤31	先送数，后 Y 指针加 1	PC←PC+1
LD Rd,−Y	0≤d≤31	先 Y 指针减 1，后送数	PC←PC+1
LDD Rd Y+q	0≤d≤31，	先 Y 指针加 q，后送数	PC←PC+1
	0≤q≤63	执行后 Y 指针(Y 不含 q)不变	

16 位操作码：

1	1000	000d	dddd	1000
2	1001	000d	dddd	1001
3	1001	000d	dddd	1010
4	10q0	qq0d	dddd	1qqq

状态寄存器(SRE)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	- -	—		—	-	·

例子：(实践操作程序 4623. ASM)

```
LP: clr   r29        ;Y 寄存器高位清零，Y 寄存器高位地址为 R29，低位地址为 R28
    ldi   r28,$60    ;将 $60 送 Y 寄存器低位，SRAM 地址≥$60
    ld    r0,y+      ;执行 Y 寄存器，Y+1 为地址的 SRAM 内容送 R0，设：SRAM($0060)=$11
                     ;($0061)=$22 ,($0062)=$33;这时(Y)=$0060，先单步执行 (R0)=
                     ;$11 ，然后(Y)+1=$0061
    ld    r1,y       ;这时(Y)=$0061，单步执行后，(R1)=$22
    ldi   r28,$70    ;将 $70 送 Y 寄存器低位
    ld    r2,y       ;这时(Y)=$0070，设($0070)=$44，($006F)=$55，($0071)=$66
                     ;单步执行后(R2)=$44
    ld    r3,-y      ;单步先执行(Y)−1=$006F，然后单步执行(R3)=$55
    ldd   r4,y+2     ;单步执行，先找增量地址(Y+Q)=(Y+2)=($006F+2)=$071，后送数
                     ;(R4)=$66，
                     ;但 Y 指针内容不变(Y 不含 q)(Y)=$006F
    nop              ;
    rjmp  lp         ;反复测试
```

Words：1(2 bytes)

Cycles：2

4. 使用变址 Y 间接将寄存器内容传送到 SRAM

ST(STD)——使用变址 Y 间接将寄存器内容传送到 SRAM

说明:间接将带或不带偏移的寄存器的一个字节传送到 SRAM 。SRAM 的位置由寄存器区中的 Y(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页。为访问另外 SRAM 页,则 I/O 范围的寄存器 RAMPY 将被修改。在操作之后,Y 指针寄存器要么不改变,要么是加 1 或减 1。使用 Y 指针寄存器的这些特性,特别适合用作堆栈指针。

操作:(Y)←Rr

(Y)←Rr　　Y←Y+ 1

Y← Y－1　　(Y)←Rr

(Y+ q)←Rr

语法:	操作码:	操作流程:	程序计数器:
ST Y,Rr	0≤d≤31	送数,Y 指针不改变	PC←PC+1
ST Y+,Rr	0≤d≤31	先送数,后 Y 指针加 1	PC←PC+1
ST －Y,Rr	0≤d≤31	先 Y 指针减 1,后送数	PC←PC+1
STD Y+q,Rr	0≤d≤31, 0≤q≤63	先 Y 指针加 q,后送数 执行后 Y 指针(Y 不含 q)不变	PC←PC+1

16 位操作码:

1	1000	001r	rrrr	1000
2	1001	001r	rrrr	1001
3	1001	001r	rrrr	1010
4	10q0	Qq1r	rrrr	1qqq

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	--	—	—	---	—	~	—

例子:(实践操作程序 4624. ASM)

```
LP: clr   r29        ;Y 寄存器高位清零,Y 寄存器高位地址为 R29,低位地址为 R28
    ldi   r28,$70    ;$70 送 Y 寄存器低位, SRAM 地址≥$60
    st    y+,r0      ;设(R0)=$11,先单步执行送 SRAM(Y)=$0070,然后(Y)+1=$0071
    st    y,r1       ;设(R1)=$22,这时(Y)=$0071,单步执行后 SRAM($0071)=$22
    ldi   r28,$80    ;$80 送 Y 寄存器低位
    st    y,r2       ;设(R2)=$33,这时(Y)=$0080,单步执行后 SRAM($0080)=$33
    st    -y,r3      ;设(R3)=$44,这时先单步执行(Y)-1=$007F,然后 R3 内容送 SRAM
                     ;($007F)
    std   y+2,r4     ;设(R4)=$55
                     ;单步执行,先计算出增量地址(Y+q)=(y+2)=($007F+2)=$0081,后传送
                     ;数据 SRAM($0081)=$55
                     ;但这时 Y 指针内容不变(Y 不含 q)(Y)=$007F
    nop              ;
    rjmp  lp         ;反复测试
Words: 1( 2 bytes)
Cycles: 2
```

三、使用 Z 寄存器间接传送数据

5. 使用变址 Z 间接将 SRAM 中的内容传送到寄存器

LD(LDD) … 使用变址 Z 间接将 SRAM 中的内容传送到寄存器

说明：带或不带偏移间接从 SRAM 中传送一个字节到寄存器，SRAM 中的位置由寄存器区中的 Z(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页中。为访问另外 SRAM 页，则 I/O 范围内的寄存器 RAMPZ 需改变。在指令执行后，Z 指针寄存器值要么不改变，要么就加 1 或减 1 操作。使用 Z 指针寄存器的这些特性，特别适合于堆栈指针，因为 Z 指针寄存器能用于直接子程序调用，直接跳转和查表。Z 指针寄存器作为专用堆栈指针要比 X、Y 指针方便。

用 Z 指针在程序存储器中查表，可参见 LPM 指令。

操作：Rd←(Z)

Rd←(Z)

Z← Z−1

Rd←(Z+q)

语法：	操作码：	操作流程：	程序计数器：
LD Rd,Z	0≤d≤31	送数，Z 指针不改变	PC←PC+1
LD Rd Z+	0≤d≤31	先送数，后 Z 指针加 1	PC←PC+1
LD Rd,−Z	0≤d≤31	先 Z 指针减 1，后送数	PC←PC+1
LDD Rd Z+q	0≤d≤31，	先 Z 指针加 q，后送数	PC←PC+1
	0≤q≤63	执行后 Z 指针(Z 不含 q)不变	

16 位操作码：

1	1000	000d	dddd	0000
2	1001	000d	dddd	0001
3	1001	000d	dddd	0010
4	10q0	qq0d	dddd	0qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	·	—	—

例子：(实践操作程序 4625. ASM)

```
LP: clr   r31          ;Z 寄存器高位清零，Z 寄存器高位地址为 R31，低位地址为 R30
    ldi   r30,$60      ;将 $60 送 Z 寄存器低位，SRAM 地址≥$60
    ld    r0,Z+        ;执行 Z 寄存器，Z+1 为地址的 SRAM 内容送 R0，设：SRAM($0060)=$11
                       ;($0061)=$22，($0062)=$33；这时(Z)=$0060，先单步执行 (R0)=$11
                       ;然后(Z)+1=$0061
    ld    r1,Z         ;这时(Z)=$0061，单步执行后，(R1)=$22
    ldi   r30,$70      ;将 $70 送 Y 寄存器低位
    ld    r2,Z         ;这时(Z)=$0070，设($0070)=$44，($006F)=$55，($0071)=$66
                       ;单步执行后(R2)=$44
    ld    r3,−Z        ;单步先执行(Z)−1=$006F，然后单步执行(R3)=$55
    ldd   r4,Z+2       ;单步执行，先找增量地址(Z+Q)=(Z+2)=($006F+2)=$071，后送数(R4)
                       ;=$66，
                       ;但 Z 指针内容不变(Z 不含 q)(Z)=$006F
    nop                ;
```

```
    rjmp lp          ;反复测试
```

Words：1(2 bytes)

Cycles：2

6. 使用变址 Z 间接将寄存器内容传送到 SRAM

ST(STD)——使用变址 Z 间接将寄存器内容传送到 SRAM

说明：间接将带或不带偏移的寄存器的一个字节传送到 SRAM。SRAM 的位置由寄存器区中的 Z(16 位)指针寄存器指出。存储器访问被限制在当前 64K 字节的 SRAM 页。为访问另外 SRAM 页，则 I/O 范围的寄存器 RAMPZ 将被修改。在操作之后，Z 指针寄存器要么不改变，要么是加 1 或减 1。使用 Z 指针寄存器的这些特性，特别适合用作堆栈指针。因为 Z 指针寄存器能适用于间接子程序调用。间接跳转和查表。所以 Z 指针寄存器像一个专用堆栈指针，用起来比 X 和 Y 指针更方便。

操作：(Z)←Rr

(Z)←Rr　　Z←Z+1

Z←Z−1　　(Z)←Rr

(Z+q)←Rr

语法：	操作码：	操作流程：	程序计数器：
ST Z,Rr	0≤d≤31	送数，Z 指针不改变	PC←PC+1
ST Z+,Rr	0≤d≤31	先送数，后 Z 指针加 1	PC←PC+1
ST −Z,Rr	0≤d≤31	先 Z 指针减 1，后送数	PC←PC+1
STD Z+q,Rr	0≤d≤31， 0≤q≤63	先 Z 指针加 q，后送数 执行后 Z 指针(Z 不含 q)不变	PC←PC+1

16 位操作码：

1	1000	001r	rrrr	0000
2	1001	001r	rrrr	0001
3	1001	001r	rrrr	0010
4	10q0	qq1r	rrrr	1qqq

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 4626. ASM)

```
LP: clr   r31       ;Z 寄存器高位清零，Z 寄存器高位地址为 R31，低位地址为 R30
    ldi   r30,$70   ;$70 送 Z 寄存器低位，SRAM 地址≥$60
    st    Z+,r0     ;设(R0)=$11，先单步执行送 SRAM(Y)=$0070，然后(Z)+1=$0071
    st    Z,r1      ;设(R1)=$22，这时(Z)=$0071，单步执行后 SRAM($0071)=$22
    ldi   r30,$80   ;$80 送 Z 寄存器低位
    st    Z,r2      ;设(R2)=$33，这时(Z)=$0080，单步执行后 SRAM($0080)=$33
    st    −Z,r3     ;设(R3)=$44，这时先单步执行(Z)−1=$007F，然后 R3 内容送 SRAM($007F)
    std   Z+2,r4    ;设(R4)=$55
                    ;单步执行，先计算出增量地址(Z+q)=(Z+2)=($007F+2)=$0081，后传送数
                    ;据 SRAM($0081)=$55
```

```
                ;但这时 Z 指针内容不变(Z 不含 q)(Z)= $007F
    nop         ;
    rjmp lp     ;反复测试
```

Words：1(2 bytes)

Cycles：2

4.6.3 从程序存储器直接取数据指令

LPM——装入程序存储器

说明：将 Z 寄存器指向的一个字节传送到寄存器 0(R0)。该指令使 100%空间有效，常量初始化或常数取数特别有用。程序存储器被编为 16 位字，Z(16 位)指针的最低位(LSB)选择为 0 是低字节，选择为 1 是高字节。该指令能寻址程序存储器第一个 64K 字节(32 字)。

操作：R0←(Z)

语法：	操作码：	程序计数器：
LPM	None	PC←PC+ 1

16 位操作码：

1001	0101	110X	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	-	—	—

例子：(实践操作程序 4631. ASM，更详细资料阅“按钮猜数. ASM”)

```
    .org $0010          ;主程序实际地址为 $0020
    clr  r31            ;Z 寄存器高位，存放程序存储器高位地址
    ldi r30, $F0        ;Z 寄存器低位，存放程序存储器低位地址
    CLR  R29            ;清零 Y 寄存器高位
    LDI R28, $F0        ;将 $F0 装入 Y 寄存器低位
LP: NOP                 ;也可设程序存储器( $000F0)= $00，以此类推至设( $00FF)= $FF
    lPM                 ;将 Z 寄存器低位数送 R0
    ST Y+, R0           ;Y 变址先将 R0 送 SRAM( $00F0)，后 Y 地址 Y=(Y+1)
    INC R30             ;Y 变址将 R0 送 SRAM( $00F1)，这时 Y=(Y+1)
    CPI R30, $00        ;R30 内容(Z 寄存器低位)与立即数 $00 比较
    BRNE LP             ;R30 内容不为 0 转，为 0 顺执
    INC R31             ;Z 寄存器高位加 1 ，(Z)= $0100
    RJMP LP             ;反复取数送数，修改程序，使数据存储器与
                        ;程序存储器数据大小排列相同
    .ORG    $0078       ;实际存放数据地址为 $00F0
    .DW 0X1122,0X2233,0X4455,0X6677,0X8899,0XAABB,0XCCDD,0XEEFF
    .DW $0208, $5510, $1910, $8750, $5012, $8757, $8872, $8757, $8852
```

Words：1(2 bytes)

Cycles：3

4.6.4 I/O 口数据传送指令

1. I/O 口数据传送到寄存器

IN——I/O 口数据传送到寄存器

说明:将 I/O 空间(口、定时器、配置寄存器等)的数据传送到寄存器区中的寄存器 Rd 中。

操作:Rd←P

语法:	操作码:	程序计数器:
IN Rd,P	0≤d≤31, 0≤ P≤63	PC←PC+ 1

16 位操作码:

1011	0PPd	dddd	PPPP

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	—	—	—	—	—	—	—

例子:(实践操作程序 4641. ASM)

```
LP: IN R25 , $1B        ;A 口的内容送入 R25 中
    LDI R23 , $26       ;将 26 送入 R23 中
    ADD R23 ,R25        ;R23 与 R25 相加送入 R23
    RJMP   LP           ;循环
```

Words: 1(2 bytes)

Cycles: 1

2. 寄存器数据送 I/ O 口

OUT——寄存器数据送 I/O 口

说明:将寄存器区中寄存器 Rr 的数据传送到 I/O 空间(口、定时器、配置寄存器等)。

操作: P←Rr

语法:	操作码:	程序计数器:
OUT P, Rr	0≤r≤31, 0≤P≤63	PC←PC + 1

16 位操作码:

1011	1PPr	rrrr	PPPP

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4642. ASM)

```
;AT90S1200 的 PB 口、PD 口接 LED 发光二极管,用单步模拟测试 I/O 口工作情况
        .DEVICE    AT90S1200        ;定义被汇编的器件为 AT90S1200
        .EQU    PORTB    =0X18
        .EQU    DDRB     =0X17      ; B 口数据方向寄存器
        .EQU    PORTD    =0X12
        .EQU    DDRD     =0X11      ;D 口数据方向寄存器
        .ORG    0X0000
```

```
LOP: RJMP   LOP1              ;跳转
     .ORG   0X0010
LOP1: LDI   R20,0XFF         ;硬件设定低电平 LED 灯亮,高电平 LED 灭
     OUT    0X17,R20         ;送 B 口数据方向寄存器
     OUT    0X11,R20         ;送 D 口数据方向寄存器
     CLR    R20              ;清零 LED 灯亮
     OUT    0X18,R20         ;送 B 口数据寄存器
     OUT    0X12,R20         ;送 D 口数据寄存器
     LDI    R20,0X64
     LDI    R20,0XFF         ;关 LED 灯
     OUT    0X18,R20         ;送 B 口 PORTB
     OUT    0X12,R20         ;送 D 口 PORTD
     LDI    R20,0X64
     RJMP   LOP1             ;循环
```

Words: 1(2 bytes)

Cycles: 1

4.6.5 堆栈操作指令

AVR 单片机的特殊功能寄存器中有一个堆栈指针 SP。它指出栈顶的位置,在指令系统中有 2 条用于数据传送的栈操作指令。

1. 进栈指令

PUSH——压寄存器到堆栈

说明:该指令存储寄存器 Rr 的内容到堆栈。

操作:STACK←Rr

语法:	操作码:	程序计数器:
PUSH Rr	0≤ d ≤31	PC←PC+1
		SP←SP−1

16 位操作码:

1001	001d	dddd	1111

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4651. ASM)

```
LP: rcall routine          ;调用 ROUTINE 子程序
    routine:push r13       ;把 R13 压入堆栈
    push r14               ;把 R14 压入堆栈
    push r15               ;把 R15 压入堆栈
    pop r15                ;从堆栈中取出数据放入 R15
    pop r14                ;从堆栈中取出数据放入 R14
    pop r13                ;从堆栈中取出数据放入 R13
    ret                    ;返回 R13,R14,R15 的数据
```

```
        rjmp lp                    ;循环继续做
```

Words：1(2 bytes)

Cycles：2

2. 出栈指令

POP——堆栈弹出到寄存器

说明：该指令将堆栈中的字节装入到寄存器 Rd 中。

操作：Rd←STACK

语法：	操作码：	程序计数器：
POP　Rd	0 ≤d ≤31	PC←PC+1　SP←SP+ 1

16 位操作码：

1001	000d	dddd	1111

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

例子：(实践操作程序 4652. ASM)

```
LP: rcall routine          ;调用子程序 ROUTINE
    routine:
    ldi r16, $01           ;把立即数 01 送入到 R16 中
    ldi r17, $02           ;把立即数 02 送入到 R17 中
    push r16               ;把 R16 压入堆栈内
    push r17               ;把 R17 压入堆栈内
    pop r17                ;出栈
    pop r16                ;出栈
    ret                    ;返回
```

Words：1(2 bytes)

Cycles：2

4.7　位指令和位测试指令

AVR 单片机指令系统中有 1/4 的指令为位和位测试指令。位指令的灵活应用，极大地提高了系统的逻辑控制和处理能力。

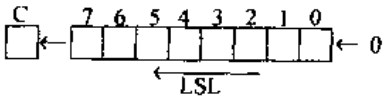
4.7.1　带进位逻辑操作指令

1. 逻辑左移

LSL——逻辑左移

说明：寄存器 Rd 中所有位左移 1 位。第 0 位被清零，第 7 位移到 SREG 中的 C 标志。该指令完成一个无符号数乘 2 的操作。

操作：



语法： 操作码： 程序计数器：

LSL Rd 0≤d ≤31 PC←PC+ 1

16 位操作码：

0000	11dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	—	⇔	⇔	⇔	⇔	⇔	⇔

H：Rd3 N：R7

S：N⊕V Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C C：Rd7

例子：(实践操作程序 4711. ASM)

```
LP: ldi r16, $01        ;将 01 送入到 R16 中
    ldi r17, $02        ;将 02 送入到 R17 中
    add r16,r17         ;R16 与 R17 相加
    lsl r16             ;加后再左移一位(01+02=03,左一位,即乘以 2。所以 r=06)
    rjmp lp             ;继续实验
```

Words：1(2 bytes)

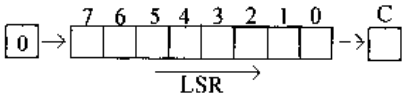
Cycles：1

2. 逻辑右移

LSR——逻辑右移

说明：寄存器 Rd 中所有位右移 1 位。第 7 位被清零，第 0 位移到 SREG 中的 C 标志。该指令完成一个无符号数除 2 的操作。C 标志被用于结果舍入。

操作：



语法： 操作码： 程序计数器：

LSR Rd 0≤d ≤31 PC←PC+ 1

16 位操作码：

1001	010d	dddd	0110

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	—		⇔	⇔	⇔	⇔	⇔

S：N ⊕ V Z：$\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

V：N⊕C C：Rd0 N：0

例子：(实践操作程序 4712. ASM)

```
LP: add r0,r1           ;将 R0 与 R1 的内容相加 (r0)= ,(r1)=
    lsr r0              ;将 R0 的内容右转移一位
    rjmp lp             ;循环继续做
```

Words：1(2 bytes)

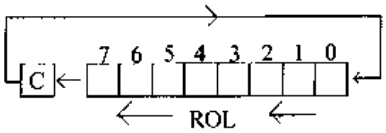
Cycles：1

3. 通过进位左循环

ROL　--通过进位左循环

说明：寄存器 Rd 的位都左移 1 位，C 标志被移到 Rd 的第 0 位，Rd 的第 7 位移到 C 标志。

操作：



语法：	操作码：	程序计数器：
ROL　Rd	0 ≤d ≤31	PC←PC+ 1

16 位操作码：

0001	11dd	dddd	dddd

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H：Rd3　　N：R7

S：N ⊕ V　　Z：$\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V：N⊕C　　C：Rd7

例子：(实践操作程序 4713. ASM)

```
LP: rol r15      ;将 R15 中的内容通过进位位循环左移一位，设：(r15)=
    rol r15      ;通过进位位循环左移一位
    nop          ;
    rjmp  lp     ;
```

Words：1(2 bytes)

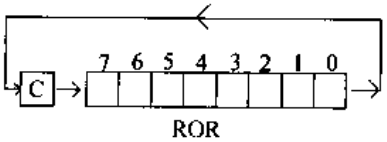
Cycles：1

4. 通过进位右循环

ROR——通过进位右循环

说明：寄存器 Rd 的位都右移 1 位，C 标志被移到 Rd 的第 7 位，Rd 的第 0 位移到 C 标志。

操作：



语法：	操作码：	程序计数器：
ROR　Rd	0≤d≤31	PC←PC+ 1

S：N ⊕ V　　Z：$\overline{R7} \cdot \overline{R6} \cdot . R \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

V：N ⊕ C　　C：Rd0　　N：R7

例子：(实践操作程序 4714. ASM)

```
LP: ror r15      ;将 R15 中的内容通过进位循环右移，设：(r15)=
    ror r15
    nop          ;
```

```
rjmp  lp        ;
```

Words: 1(2 bytes)

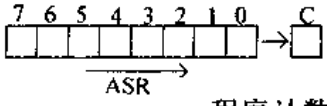
Cycles: 1

5. 算术右移

ASR—— 算术右移

说明:寄存器 Rd 中的所有位右移 1 位,而位 7 保持常量,位 0 被装入 SREG 的 C 标志位。这个操作实现 2 的补码值除 2,而不改变符号,进位标志用于结果的舍入。

操作:



语法:	操作码:	程序计数器:
ASR Rd	0≤d ≤31	PC←PC+ 1

16 位操作码:

1001	010d	dddd	0101

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S:N ⊕ V　　Z:R7 • R6 • R5 • R4 • R3 • R2 • R1 • R0

V:N ⊕ C　　C:Rd0　　N:R7

例子:(实践操作程序 4715. ASM)

```
LP: ldi r16, $10
    ldi r17, $fc
    asr r16
    asr r17
    rjmp lp
```

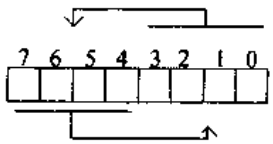
Words: 1(2 bytes)

Cycles: 1

6. 半字节交换

SWAP——半字节交换

说明:寄存器中的高半字节和低半字节交换。



操作:R(7~4)←Rd(3~0),R(3~0)←Rd(7~4)

语法:	操作码:	程序计数器:
SWAP Rd	0≤d≤31	PC←PC+ 1

16 位操作码:

1001	010d	dddd	0010

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4716. ASM)

```
LP: inc r1
```

```
inc r2
swap r1
swap r2
rjmp lp
```

Words：1(2 bytes)

Cycles：1

4.7.2 位变量传送指令

1. 寄存器中的位存储到SREG中的T标志

BST——寄存器中的位存储到SREG中的T标志

说明：把寄存器中的位b存储到SREG(状态寄存器)中的T标志。

操作：T←Rd(b)

语法：	操作码：	程序计数器：
BST Rd, b	0≤d≤31, 0≤b≤7	PC←PC+1

16位操作码：

1111	101d	dddd	Xbbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	⇔	—	—	—	—	—	—

T：0 if bit b in Rd is cleared. Set to 1 otherwise.

例子：(实践操作程序4721.ASM)

```
bst   r1,2          ;T←R1(2)
bld   r0,4          ;R0(4)←T
```

Words：1(2 bytes)

Cycles：1

2. SREG中的T标志装入寄存器中的某一位

BLD——位装入，将SREG中的T标志装入到寄存器中的某一位

说明：拷贝SREG(状态寄存器)的T标志到寄存器Rd中的位b。

操作：Rd(b)←T

语法：	操作码：	程序计数器：
BLD Rd, d	0≤d≤31, 0≤b≤7	PC←PC+1

16位操作码：

1111	100d	dddd	0bbb

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序4722.ASM)

```
bst   r1,2          ;T←R1(2)
bld   r0,4          ;R0(4)←T
```

Words：1(2 bytes)

Cycles：1

4.7.3 位变量修改指令

BSET,BCLR为基本指令,其它位指令可根据状态字SREG内容派生出来的(即5.～20.的位指令)。

1. 置状态寄存器的位

BSET——置状态寄存器的位

说明:置状态寄存器(SREG)的某一标志或某一位。

操作:SREG(S)←1

语法：	操作码：	程序计数器：
BSET s	0≤s≤7	PC←PC+ 1

16位操作码：

1001	0100	0sss	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I：1 if s=7　　V：1 if s=3

T：1 if s=6　　N：1 if s=2

H：1 if s=5　　Z：1 if s=1

S：1 if s=4　　C：1 if s=0

例子:(实践操作程序4731.ASM)

```
bset    6           ;SREG(6)←1
bset    7           ;SREG(7)←1
```

Words：1(2 bytes)

Cycles：1

2. 清状态寄存器的位

BCLR——SREG中的位清零

说明:清零SREG状态寄存器中的一个标志位。

操作：SREG(S)← 0

语法：	操作码：	程序计数器：
BCLR s	0≤s≤7	PC←PC+1

16位操作码：

1001	0100	1sss	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I：0 if s=7　　V:0 if s=3

T:0 if s=6　　N:0 if s=2

H:0　if　s=5　　Z:0　if　s=1

S:0　if　s=4　　C:0　if　s=0

例子:(实践操作程序 4732.ASM)

```
bclr    0      ;SREG(0)←0
bclr    7      ;SREG(7)←0
```

Words: 1(2 bytes)

Cycles: 1

3. 置 I/O 寄存器的位

SBI——置 I/O 寄存器的位

说明:对 I/O 寄存器指定的位置位,该指令在低 32 个 I/O 寄存器内操作,I/O 寄存器地址为 0～31。

操作:I/O(P,b)←1

语法:	操作码:	程序计数器:
SBI　P, b	0≤P≤31,0≤b≤7	PC←PC+1

16 位操作码:

1001	1010	PPPP	Pbbb

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	-		—	—

例子:(实践操作程序 4733.ASM)

```
out     $1e,r0        ;(EEARL 寄存器)←(R0)
sbi     $1c,0         ;(EECR 寄存器 0 位)←1
in      rl,$1d        ;(R0)←(EEDR 寄存器)
```

Words: 1(2 bytes)

Cycles: 2

4. 清 I/O 寄存器的位

CBI—— 清 I/O 寄存器的位

说明:清零 I/O 寄存器中的指定位,该指令用在寄存器最低的 32 个 I/O 寄存器上,I/O 寄存器地址为 0～31。

操作:I/O(P,b)←0

语法:	操作码:	程序计数器:
CBI　P, b	0≤P≤31, 0≤b≤7	PC←PC + 1

16 位操作码:

1001	1000	PPPP	Pbbb

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—		—	—	—

C:1

例子:(实践操作程序 4734.ASM)

```
cbi   $18,7      ;I/O(PORTB 寄存器的 7 位)←0
```

Words: 1(2 bytes)

Cycles: 2

5. 置进位位

SEC——置位进位标志

说明:置位 SREG(状态寄存器)中的进位标志(C)。

操作:C←1

语法:	操作码:	程序计数器:
SEC	None	PC←PC+ 1

16 位操作码:

1001	0100	0000	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

C:1

例子:(实践操作程序 4735. ASM)

```
sec                ;C←1
adc   r0, r1       ;带进位位加
```

Words: 1(2 bytes)

Cycles: 1

6. 清进位位

CLC——清零进位标志

说明:清零 SREG(状态寄存器)中的进位标志(C)。

操作:C←0

语法:	操作码:	程序计数器:
CLC	None	PC←PC+ 1

16 位操作码:

1001	0100	1000	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	0

C:0

例子:(实践操作程序 4936. ASM)

```
add   r0, r0       ;加
clc                ;C←0
```

Words: 1(2 bytes)

Cycles: 1

7. 置位负标志位

SEN——置位负数标志

说明:置位 SREG(状态寄存器)中的负数标志(N)。

操作:N←1

语法:	操作码:	程序计数器:
SEN	None	PC←PC+ 1

16 位操作码:

1001	0100	0010	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-		-	1	-	-

N:1

例子:(实践操作程序 4737. ASM)

```
add   r2,r19          ;加
sen                   ;N←1
```

Words: 1(2 bytes)

Cycles: 1

8. 清负标志位

CLN——清零负数标志

说明:清零 SREG(状态寄存器)中的负数标志(N)。

操作:N←0

语法:	操作码:	程序计数器:
CLN	None	PC←PC+ 1

16 位操作码:

1001	0100	1010	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
-	-	-	-	-	0	-	-

N:0

例子:(实践操作程序 4738. ASM)

```
add r2,r3             ;加
cln                   ;N←0
```

Words: 1(2 bytes)

Cycles: 1

9. 置零标志位

SEZ——置位零标志

说明:置位 SREG(状态寄存器)中的零标志(Z)。

操作:Z←1

语法:	操作码:	程序计数器:
SEZ	None	PC←PC+ 1

16 位操作码:

1001	0100	0001	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	—	—	—	—	—	1	—

Z:1

例子:(实践操作程序 4739. ASM)

```
add  r2,r19          ;加
sez                  ;Z←1
```

Words: 1← (2 bytes)

Cycles: 1

10. 清零标志位

CLZ——清零零标志

说明:清零 SREG(状态寄存器)中的零标志(Z)。

操作:Z←0

语法：	操作码：	程序计数器：
CLZ	None	PC←PC+ 1

16 位操作码：

1001	0100	1001	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	0	—

Z: 0

例子:(实践操作程序 47310. ASM)

```
add r2,r3            ;加
clz                  ;Z←0
```

Words: 1(2 bytes)

Cycles: 1

11. 触发全局中断位

SEI——置位全局中断标志

说明:置位 SREG(状态寄存器)中的全局中断标志(I)。

操作:I←1

语法：	操作码：	程序计数器：
SEI	None	PC←PC+ 1

16 位操作码：

1001	0100	0111	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
1	—	—	—	—	—	—	—

I:1

例子:(实践操作程序 47311. ASM)

```
cli                          ;I← 0
in   r13,$16                 ;(r13)←(PINB寄存器数据)
set                          ;I←1
```

Words: 1(2 bytes)

Cycles: 1

12. 禁止全局中断位

CLI——清零全局中断标志

说明:清除 SREG(状态寄存器)中的全局中断标志(I)。

操作:I←0

语法:	操作码:	程序计数器:
CLI	None	PC←PC+ 1

16 位操作码:

1001	0100	1111	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
0	·	—	—	—	·	—	-

I:0

例子:(实践操作程序 47312. ASM)

```
cli                          ;I←0
in   r13,$16                 ;(r13)←(PINB寄存器数据)
set                          ;I←1
```

Words: 1(2 bytes)

Cycles: 1

13. 置 S 标志位

SES——置位符号标志

说明:置位 SREG(状态寄存器)中的符号标志(S)。

操作:s←1

语法:	操作码:	程序计数器:
SES	None	PC←PC+1

16 位操作码:

1001	0100	0100	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
	- -	-	1	—	—	·	

S:1

例子:(实践操作程序 47313. ASM)

```
add  r2,r19                  ;加
ses                          ;s←1
```

Words：1(2 bytes)

Cycles：1

14. 清S标志位

CLS——清零符号标志

说明：清零 SREG(状态寄存器)中的符号标志(S)。

操作：S←0

语法：	操作码：	程序计数器：
CLS	None	PC←PC+ 1

16 位操作码：

1001	0100	1100	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	0	—	—	—	—

S:0

例子：(实践操作程序 47314. ASM)

```
add   r2, r3          ;加
cls                   ;S←0
```

Words：1(2 bytes)

Cycles：1

15. 置溢出标志位

SEV——置位溢出标志

说明：置位 SREG(状态寄存器)中的溢出标志(V)。

操作：V←1

语法：	操作码：	程序计数器：
SEV	None	PC←PC + 1

16 位操作码：

1001	0100	0011	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	1	—	—	—

V:1

例子：(实践操作程序 47315. ASM)

```
add     r2 ,r19       ;加
sev                   ;V←1
```

Words：1(2 bytes)

Cycles：1

16. 清溢出标志位

CLV——清零溢出标志

说明：清零 SREG(状态寄存器)中的溢出标志(V)。

操作：V←0

语法：	操作码：	程序计数器：
CLV	None	PC←PC+ 1

16 位操作码：

1001	0100	1011	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	-	—	0	-	—	—

V：0

例子：(实践操作程序 47316. ASM)

```
add     r2,r3          ;加
clv                    ;V←0
```

Words：1(2 bytes)

Cycles：1

17. 置 T 标志位

SET——置位 T 标志

说明：置位 SREG(状态寄存器)中的 T 标志。

操作：T←1

语法：	操作码：	程序计数器：
SET	None	PC←PC+ 1

16 位操作码：

1001	0100	0110	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	1	—	—	—	—	—	—

T：1

例子：(实践操作程序 47317. ASM)

```
set                    ;T←1
```

Words：1(2 bytes)

Cycles：1

18. 清 T 标志位

CLT——清零 T 标志

说明：清零 SREG(状态寄存器)中的 T 标志。

操作：T←0

语法：	操作码：	程序计数器：
CLT	None	PC←PC+ 1

16 位操作码：

1001	0100	1110	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	0	—	—	—	—	—	—

T:0

例子:(实践操作程序 47318.ASM)

```
clt                    ;T←0
```

Words: 1(2 bytes)

Cycles: 1

19. 置半进位标志

SEH——置位半进位标志

说明:置位 SREG(状态寄存器)中的半进位标志(H)。

操作:H←1

语法：	操作码：	程序计数器：
SEH	None	PC←PC+ 1

16 位操作码：

1001	0100	0101	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	1	—	—	—	—	—

H:1

例子:(实践操作程序 47319.ASM)

```
seh                    ;H←1
```

Words: 1(2 bytes)

Cycles: 1

20. 清半进位标志

CLH——清零半进位标志

说明:清零 SREG(状态寄存器)中的半进位标志(H)。

操作:H←0

语法：	操作码：	程序计数器：
CLH	None	PC←PC+ 1

16 位操作码：

1001	0100	1101	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	0	—	—	—	—	—

H:0

例子:(实践操作程序 47320.ASM)

```
clh            ;H←0
```

Words：1(2 bytes)

Cycles：1

4.7.4 其它指令

1. 空指令

NOP——空操作

说明：该指令完成一个单周期空操作。

应用：延时等待、产生方波、抗干扰、在无程序单元写上空操作。空操作指令最后转到$000H。

操作：No

语法：	操作码：	程序计数器：
NOP	None	PC←PC+ 1

16 位操作码：

0000	0000	0000	0000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 4741. ASM)

```
nop                 ;空操作
add r0 ,r1          ;R1 与 R0 相加
nop                 ;空操作
add r0, r1          ;R1 与 R0 相加
rjmp lp             ;循环
```

Words：1(2 bytes)

Cycles：1

2. 休眠指令

SLEEP——休眠

说明：该指令设置电路休眠模式，由 MCU 控制寄存器定义。当在休眠状态由一个中断唤醒时，在中断程序执行后，紧跟在休眠指令后的指令被执行。

应用：省电，尤其对便携式仪器特别有用。

操作：

语法：	操作码：	程序计数器：
SLEEP	None	PC←PC+ 1

16 位操作码：

1001	0101	100x	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4742. ASM)

```
mov   ro, rll          ;拷贝
sleep                  ;休眠
```

Words: 1(2 bytes)

Cycles: 1

3. 看门狗复位

WDR—— 看门狗复位

说明:该指令复位看门狗定时器,在 WD 预定比例器给出限定时间内必须执行。参见看门狗定时器硬件部分。应用于抗干扰、延时。

操作: WD timer restart.

语法:	操作码:	程序计数器:
WDR	None	PC←PC+ 1

16 位操作码:

1001	0101	101x	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 4743. ASM)

```
PLYDEL:   LDI    TEMP,185     ;用 WDR 做延时子程序
DT3:      LDI    TEMP1,04     ;
DT2:      LDI    TEMP2,250
DT1:      WDR                 ;1T
          WDR                 ;2T
          WDR                 ;3T
          WDR                 ;4T
          WDR                 ;5T
          DEC    TEMP2        ;
          BRNE   DT1          ;
          DEC    TEMP1        ;
          BRNE   DT2          ;
          DEC    TEMP         ;
          BRNE   DT3          ;
          RET
```

Words: 1(2 bytes)

Cycles: 1

4.8 新增指令(新器件)

4.8.1 EICALL—— 延长间接调用子程序

说明:间接调用由寄存器文件中的 Z(16 位) 指针和 I/O 端口中的 EIND 寄存器指向的子程序。该指令允许调用整个程序存储器空间内的子程序,但在双字节 PC 的设备中是无效的,

见 ICALL。在 EICALL 指令执行期间，堆栈指针使用一种后进先出的设置。

操作：PC←(15：0)←Z(15：0)

PC(21：16)←EIND

语法：	操作码：	程序计数器：	堆栈：
EICALL	None	See Operation	STACK←PC + 1 SP←SP−3 (3 bytes, 22 bits)

16 位操作码：

1001	0101	0001	1001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 481. ASM)

```
ldi r16, $05              ;设置 EIND 和 Z 指针
out EIND,r16
ldi r30, $00
ldi r31, $10
eicall                    ;调用 $051000
```

Words：1 (2 bytes)

Cycles：4 (仅在带 22 位 PC 的器件上执行)

4.8.2　EIJMP——扩展间接跳转

说明：间接跳转到由寄存器文件中的 Z (16 位) 指针和 I/O 端口中的 EIND 寄存器指向的地址。该指令允许间接跳转到整个程序存储器空间。

操作：PC(15：0) ←Z(15：0)

PC(21：16)←EIND

语法：	操作码：	程序计数器：	堆栈：
EIJMP	None	See Operation	不影响

16 位操作码：

1001	0100	0001	1001

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 482. ASM)

```
ldi r16, $05              ;设置 EIND 和 Z 指针
out EIND,r16
ldi r30, $00
ldi r31, $10
eijmp                     ; 跳转到 $051000
```

Words:1 (2 bytes)

Cycles:2

4.8.3 ELPM——扩展装载程序存储器

说明:装入由 Z 寄存器和 I/O 端口中的 RAMPZ 寄存器指向的一个字节,将这一字节装入目的寄存器 Rd。该指令描述一个 100%空间有效的常量初始化或常量数据引出。程序存储器按 16 位字组织起来,Z 寄存器最重要的位选择低字节(0)或高字节(1)。该指令能寻址整个程序存储器空间。该操作不改变 Z 指针寄存器,或使之增加。增加适用于 RAMPZ 和 Z 指针寄存器的整个 24 位串联。

这些合并的结果是不确定的:

```
ELPM r30, Z+
ELPM r31, Z+
```

操作:

```
R0←(RAMPZ : Z)                                  ;RAMPZ : Z : Unchanged,R0 im-
                                                ;plied destination register
Rd←(RAMPZ : Z)                                  ;RAMPZ : Z : Unchanged
Rd←(RAMPZ : Z) (RAMPZ:Z)←(RAMPZ:Z)+1 ;RAMPZ : Z : Post incremented
```

语法:	操作码:	程序计数器:
ELPM	None,R0 implied	PC←PC+1
ELPM Rd,Z	0≤d≤31	PC←PC+1
ELPM Rd,Z+	0≤d≤31	PC←PC+1

16 位操作码:

(i)	1001	0101	1101	1000
(ii)	1001	000d	dddd	0110
(iii)	1001	000d	dddd	0111

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—		—		—	—	·	—

例子:(实践操作程序 483.ASM)

```
clr r16                 ;RAMPZ 寄存器清零
out RAMPZ, r16
clr r31                 ;清除 Z 指针寄存器的高字节
ldi r30, $F0            ;Z 指针寄存器的低字节置 1
elpm r16, Z+            ;从程序装入常数
                        ;RAMPZ : Z (r31 : r30)指向的存储器
```

Words:1 (2 bytes)

Cycles:3

4.8.4 ESPM——扩展存储程序存储器

说明:ESPM 指令用来擦除程序存储器中的一页、写程序存储器中的一页(那是已经被擦

除的)和设置引导装载器锁位。在一些器件中,程序存储器一次写一字节,另一些器件中在先填充一个暂时页缓冲器后能同时编程一整页。就一般情况而论,程序存储器必须一次被擦除一页。擦除程序存储器时,RAMPZ 和 Z 寄存器被用来页寻址;写程序存储器时,RAMPZ 和 Z 寄存器被用作页或字寻址,R1:R0 寄存器组被当作数据。设置引导装载锁位时,R1:R0 寄存器组被当作数据。参考器件文件可获得 ESPM 用法的详细说明。该指令可寻址整个程序存储器。

操作:

(RAMPZ:Z)←$ffff ;擦除程序存储器页

(RAMPZ:Z)←R1:R0 ;写程序存储器字

(RAMPZ:Z)←R1:R0 ;写暂时页缓冲器

(RAMPZ:Z)←TEMP ;写暂时页缓冲器到程序存储器

BLBITS←R1:R0 ;设置引导装载器锁位

语法:	操作码:	程序计数器:
ESPM	None	PC←PC + 1

16 位操作码:

1001	0101	1111	1000

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子:(实践操作程序 484.ASM)

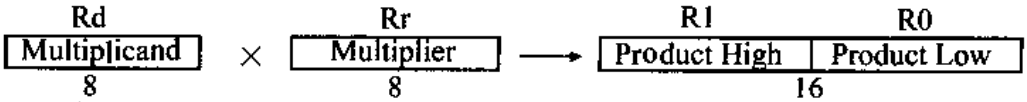
```
;此例说明了对于带页写的器件的字 ESPM 写
clr   r31              ;Z 寄存器的高字节清零
clr   r30              ; Z 寄存器的低字节清零
ldi   r16,    $F0      ; 装入 RAMPZ 寄存器
out   RAMPZ,  r16      ;
ldi   r16,    $CF      ;存入数据
mov r1,       r16
ldi   r16,    $FF
mov r0,       r16
ldi   r16,    $03      ; ESPM 使能,擦除页
out   SPMCR,  r16      ;
espm                   ; 从 $F00000 开始擦除页
ldi   r16,    $01      ; ESPM 使能,将 R1:R0 存入暂时缓冲器
out   SPMCR,  r16      ;
espm                   ; 执行 ESPM,将 R1:R0 存入暂时缓冲器的 $F00000 处
ldi   r16,    $05      ; ESPM 使能,写页
out   SPMCR,  r16      ;
espm                   ; 执行 ESPM,将暂时缓冲器存入从程序存储器地址 $F00000 开始的页
```

Words:1 (2 bytes)

Cycles:依操作而定

4.8.5 FMUL——小数乘法

说明：该指令完成 8 位×8 位→16 位的无符号数乘法操作，并把结果左移一位。



(N. Q) 表示一个小数点左边有 N 个二进制数位、右边有 Q 个二进制数位的小数。以(N1. Q1)和(N2. Q2)为格式的两个数相乘，产生格式为[(N1+N2). (Q1+Q2)]的结果。为处理符号，以(1. 7)格式作输入，产生(2. 14) 格式的结果。结果的高字节要左移一位以使结果的格式与输入的一致。FMUL 指令在与 MUL 指令相同的周期内合并了左移操作。

被乘数 Rd 和乘数 Rr 是两个包含无符号小数的寄存器，固定的小数位在第 6 位和第 7 位之间。16 位无符号小数结果的固定小数位在第 14 位和第 15 位之间，即存放在 R1(高字节)和 R0(低字节)。

操作：

R1：R0←Rd×Rr [unsigned(1. 15)←unsigned(1. 7)×unsigned(1. 7)]

语法：	操作码：	程序计数器：
FMUL Rd,Rr	16≤d≤23,16≤r≤23	PC←PC+1

16 位操作码：

0000	0011	0ddd	1rrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	⇔	⇔	⇔	⇔	—	⇔	⇔

C：R16

如果结果的第 15 位在左移前被置 1 则置 1，否则清除。

Z：$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

如果结果是 $0000 则置 1，否则清除。

R (结果)操作后等于 R1,R0。

例子：(实践操作程序 485. ASM)

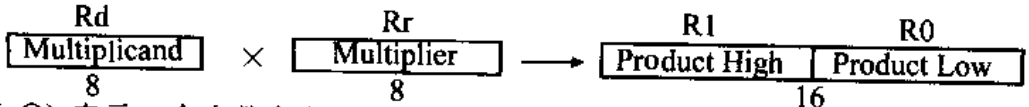
```
fmul r23,r22          ;无符号数 r23 和 r22 以(1.7)格式相乘，产生(1.15)格式的结果
movw r22,r0           ;复制结果回 r23:r22
```

Words：1 (2 bytes)

Cycles：2

4.8.6 FMULS——有符号数乘法

说明：该指令完成 8 位×8 位→16 位的有符号数乘法操作，并把结果左移一位。



(N. Q) 表示一个小数点左边有 N 个二进制数位、右边有 Q 个二进制数位的小数。以

(N1.Q1)和(N2.Q2)为格式的两个数相乘产生格式为[(N1+N2).(Q1+Q2)]的结果。为处理符号,以(1.7)格式作输入,产生(2.14)格式的结果。结果的高字节要左移一位以使结果的格式与输入的一致。FMULS指令在与MULS指令相同的周期内合并了左移操作。

被乘数Rd和乘数Rr是2个包含有符号小数的寄存器,固定的小数位在第6位和第7位之间。16位有符号小数结果的固定小数位在第14位和第15位之间,即存放在R1(高字节)和R0(低字节)。

操作:

R1:R0←Rd×Rr (signed(1.15)←signed(1.7)×signed(1.7))

语法:	操作码:	程序计数器:
FMUL Rd,Rr	16≤d≤23,16≤r≤23	PC←PC+1

16位操作码:

0000	0011	1ddd	0rrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	⇔	⇔	⇔	⇔	—	⇔	⇔

C:R16

如果结果的第15位在左移前被置1则置1,否则清除。

Z:$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

如果结果是$0000则置1,否则清除。

R(结果)操作后等于R1,R0。

例子:(实践操作程序486.ASM)

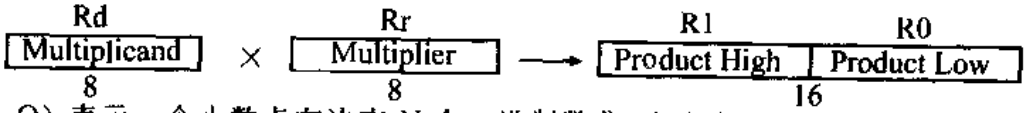
```
fmuls r23,r22          ;有符号数 r23 和 r22 以(1.7)格式相乘,产生(1.15)格式的结果
movw r22,r0            ;复制结果回 r23:r22
```

Words: 1 (2 bytes)

Cycles:2

4.8.7 FMULSU——有符号小数和无符号小数乘法

说明:该指令完成8位×8位→16位的有符号数乘法操作,并把结果左移一位。



(N.Q)表示一个小数点左边有N个二进制数位、右边有Q个二进制数位的小数。以(N1.Q1)和(N2.Q2)为格式的两个数相乘产生格式为[(N1+N2).(Q1+Q2)]的结果。为处理符号,以(1.7)格式作输入,产生(2.14)格式的结果。结果的高字节要左移一位以使结果的格式与输入的一致。FMULSU指令在与MULSU指令相同的周期内合并了左移操作。被乘数Rd和乘数Rr是2个包含小数的寄存器,暗含的小数位在第6位和第7位之间。被乘数Rd是一个有符号小数,乘数Rr是一个无符号小数。16位有符号小数结果暗含的小数位在第14位和第15位之间,即存放在R1(高字节)和R0(低字节)。

操作：

R1：R0←Rd×Rr (signed(1.15)←signed(1.7)×unsigned(1.7))

语法：	操作码：	程序计数器：
FMULSU Rd,Rr	16≤d≤23,16≤r≤23	PC←PC+1

16位操作码：

0000	0011	1ddd	1rrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	·	—	—	—	⇔	⇔

C：R16

如果结果的第15位在左移前被置1则置1，否则清除。

Z：$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

如果结果是$0000则置1，否则清除。

R(结果)操作后等于R1,R0。

例子：(实践操作程序487.ASM)

```
fmulSU r23,r22          ;有符号数r23和无符号数r22以(1.7)格式相乘，产生(1.15)格式的结果
movw r22,r0             ;复制结果回r23:r22
```

Words:1 (2 bytes)

Cycles:2

4.8.8 MOVW——拷贝寄存器字

说明：该指令完成将一个寄存器组拷贝到另一个寄存器组的操作。源寄存器组Rr+1:Rr不改变，目的寄存器组Rd+1：Rd则是Rr+1：Rr所含内容的拷贝。

操作：

Rd+1：Rd←Rr+1：Rr

语法：	操作码：	程序计数器：
MOVW Rd,Rr	d={0,2,…,30},r={0,2,…30}	PC←PC+1

16位操作码：

0000	0001	dddd	rrrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	-	—	—	--	—	—	-

例子：(实践操作程序488.ASM)

```
        movw r16,r0         ; 拷贝r1：r0到r17：r16
        call check          ; 调用子程序
        ...
check:  cpi r16;$11         ; r16 - $11
        ...
```

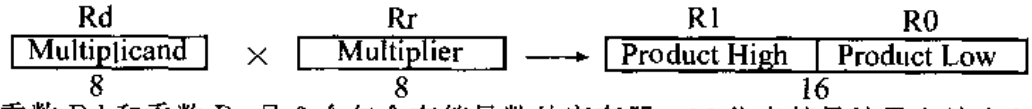
```
cpi r17,$32          ;r17 - $32
...
ret                  ;子程序返回
```

Words:1 (2 bytes)

Cycles:1

4.8.9 MULS——有符号数乘法

说明:该指令完成8位×8位→16位有符号数乘法的操作。



被乘数Rd和乘数Rr是2个包含有符号数的寄存器。16位有符号结果存放在R1(高字节)和R0(低字节)中。

操作:

R1:R0←Rd×Rr (signed←signed×signed)

语法:	操作码:	程序计数器:
MULS Rd,Rr	16≤d≤31,16≤r≤31	PC←PC+1

16位操作码:

0000	0010	dddd	rrrr

状态寄存器(SREG)和布尔格式:

I	T	H	S	V	N	Z	C
—	—	‥	—	—	—	⇔	⇔

C:R15

如果结果的第15位被置1则置1,否则清除。

Z:$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

如果结果是$0000则置1,否则清除。

R(结果)操作后等于R1,R0。

例子:(实践操作程序489.ASM)

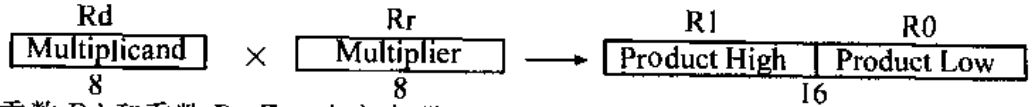
```
muls r21,r20         ;有符号数r21和r20相乘
movw r20,r0          ;拷贝结果回r21:r20
```

Words:1 (2 bytes)

Cycles:2

4.8.10 MULSU——有符号数与无符号数乘法

说明:该指令完成8位×8位→16位的一个有符号数和一个无符号数乘法的操作。



被乘数Rd和乘数Rr是2个寄存器。被乘数Rd是有符号数,乘数Rr是无符号数。16

位有符号结果存放在 R1（高字节）和 R0（低字节）中。

操作：

R1:R0←Rd×Rr （signed←signed×signed）

语法：	操作码：	程序计数器：
MULSU Rd,Rr	16≤d≤23,16≤r≤23	PC←PC+1

16 位操作码：

0000	0011	0ddd	0rrr

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
-	—	—	—	—	—	⇔	⇔

C：R15

如果结果的第 15 位被置 1 则置 1，否则清除。

Z：$\overline{R15}\cdot\overline{R14}\cdot\overline{R13}\cdot\overline{R12}\cdot\overline{R11}\cdot\overline{R10}\cdot\overline{R9}\cdot\overline{R8}\cdot\overline{R7}\cdot\overline{R6}\cdot\overline{R5}\cdot\overline{R4}\cdot\overline{R3}\cdot\overline{R2}\cdot\overline{R1}\cdot\overline{R0}$

如果结果是 $0000 则置 1，否则清除。

R（结果）操作后等于 R1,R0。

例子：(实践操作程序 4810.ASM)

```
mulsu r21,r20          ;有符号数 r21 和无符号数 r20 相乘得有符号数结果
movw r20,r0            ;拷贝结果回 r21:r20
```

Words:1 (2 bytes)

Cycles:2

4.8.11 SPM——存储程序存储器

说明：SPM 指令可用来擦除程序存储器的一页、写程序存储器中的一页（是已经被擦除的）和设置引导装载器锁位。在一些器件中，程序存储器一次写一字节，另一些器件中在先填充一个暂时页缓冲器后能同时编程一整页。就一般情况而论，程序存储器必须一次被擦除一页。擦除程序存储器时，Z 寄存器被用来页寻址；写程序存储器时，Z 寄存器被用作页或字寻址，R1：R0 寄存器组被当作数据。设置引导装载器锁位时，R1：R0 寄存器组被当作数据。参考器件文件可获得 SPM 用法的详细说明。该指令可寻址程序存储器的前 64K 字节(32K 字)。

操作：

```
(Z)←$ffff              ;擦除程序存储器页
(Z)←R1：R0             ;写程序存储器字
(Z)←R1：R0             ;写暂时页缓冲器
(Z)←TEMP               ;写暂时页缓冲器到程序存储器
BLBITS←R1：R0          ;设置引导装载器锁位
```

语法：	操作码：	程序计数器：
SPM	None	PC←PC+1

16 位操作码：

1001	0101	1110	1000

状态寄存器(SREG)和布尔格式：

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

例子：(实践操作程序 4811. ASM)

```
; 此例显示对带字写的器件的 SPM 字写
ldi r31, $F0          ; 装入 Z 的高字节
clr r30               ; 清除 Z 的低字节
ldi r16, $CF          ; 装入数据以便存储
mov r1, r16
ldi r16, $FF
mov r0, r16
ldi r16, $03          ; SPM 使能，擦除页
out SPMCR, r16        ;
spm                   ; 从 $F000 开始擦除页
ldi r16, $01          ; SPM 使能，存储到存储器
out SPMCR, r16        ;
spm                   ;执行 SPM 操作，存储 R1:R0 到存储器的 $F000 单元
```

Words：1 (2 bytes)

Cycles：依据操作而定。

第五章　AVR单片机AT90系列

本章重点叙述AT90S1200及AT90S8535单片机，有关AT90S8515已在第二章中详述。简要介绍其它AVR单片机，其性能特点、引脚图及硬件结构框图。

5.1　AT90S1200

5.1.1　特　点

(1) AVR RISC结构：
——高性能、低功耗RISC结构；
——89条指令，大多数为单指令周期；
——32个8位通用(工作)寄存器；
——工作在12MHz时具有12MIPS的性能。
(2) 数据和非易失性程序内存：
——1K字节的在线可编程Flash(擦除次数为1 000次)；
——64字节在线可编程EEPROM(寿命为100 000次)；
——程序加密位。
(3) 外围(Peripheral)特点：
——1个可预分频(Prescale)的8位定时器/计数器；
——片内模拟比较器；
——可编程的看门狗定时器(由片内振荡器生成)；
——用于下载程序的SPI口。
(4) MCU特点：
——低功耗空闲和掉电模式；
——内外部中断源；
——可选的片内RC振荡器。
(5) 规范(Specification)：
——低功耗、高速CMOS工艺；
——全静态工作。
(6) 在4MHz、3V、25℃条件下的功耗：
——工作模式为2.0mA；
——空闲模式为0.4mA；
——掉电模式为<1μA。
(7) I/O和封装：
——15个可编程的I/O脚；
——20脚PDIP和SOIC封装。

(8) 工作电压：

2.7～6.0V(AT90S1200－4)；

4.0～6.0V(AT90S1200－12)。

(9) 速度：

——0～4MHz(AT90S1200－4)；

——0～12MHz(AT90S1200－12)。

5.1.2 描　述

图 5.1 为 AT90S1200 结构框图。AT90S1200 是一款基于 AVR RISC 的、低功耗 CMOS

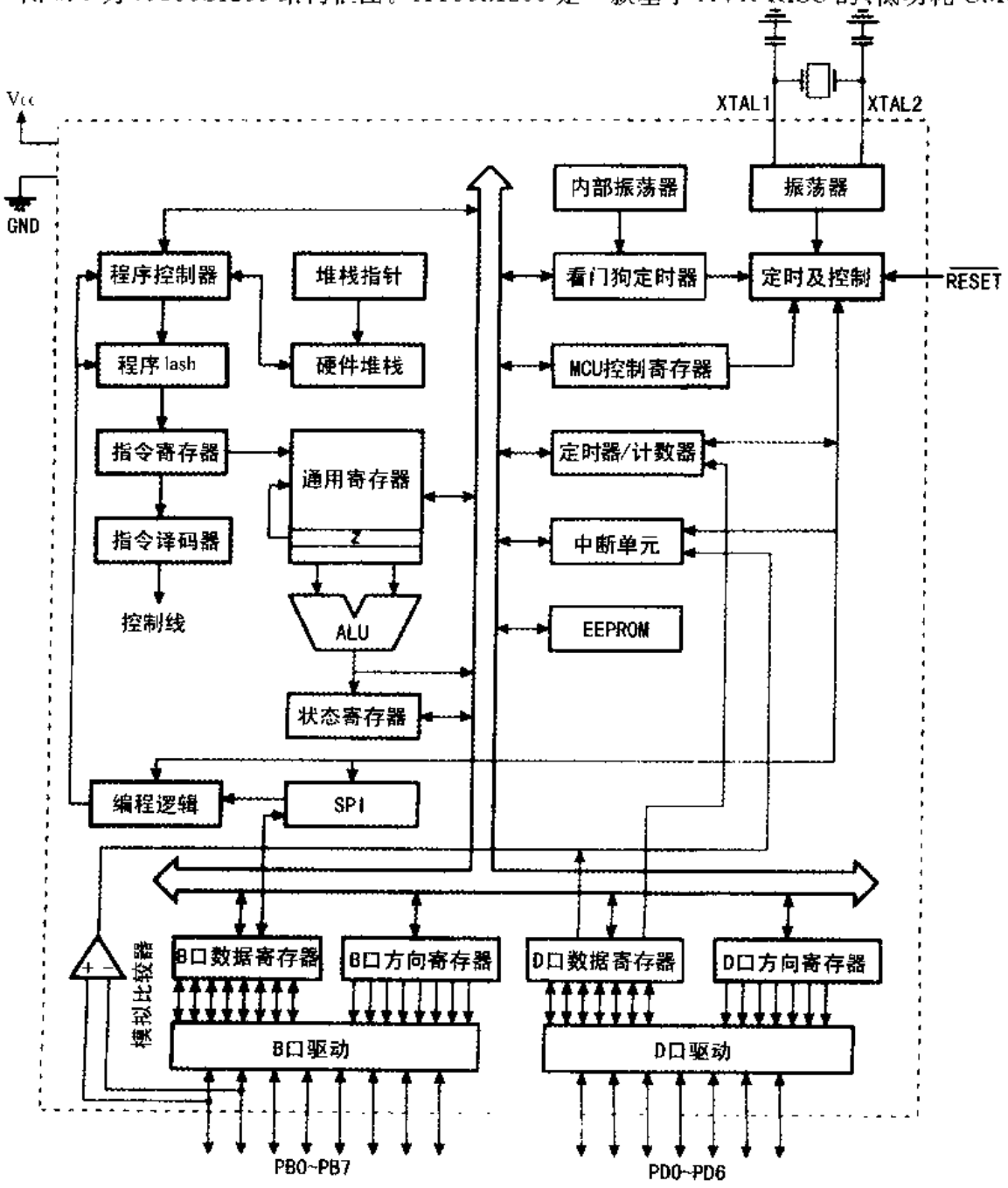


图 5.1　AT90S1200 结构方框图

8 位单片机。通过在一个时钟周期内执行一条指令，AT90S1200 可以取得接近 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU（算逻单元）直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

这种结构可以有效地支持高级语言编程，同时保持代码密度紧凑。AT90S1200 具有以下特点：1K 字节 Flash；64 字节 EEPROM；15 个通用 I/O 口；32 个通用工作寄存器；内外中断源；可编程的看门狗定时器；下载程序用的 SPI 口以及 2 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而寄存器、定时器/计数器、看门狗和中断系统继续工作；掉电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有外部低电平中断或硬件复位才可以退出此状态。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 允许多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，AT90S1200 为许多嵌入式控制应用提供了灵活而低成本的方案。

AT90S1200 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和 SL-AVR 编程开发实验器。

5.1.3 引脚配置

AT90S1200 引脚配置如图 5.2 所示。

一、引脚定义

V_{CC}、GND：电源。

B 口（PB7～PB0）：B 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻（可单独选择）。PB0 和 PB1 还可作为片内模拟比较器的正（AIN0）负（AIN1）输入端。B 口的输出缓冲器能够吸收 20mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。B 口作为特殊功能口的使用方法见以后章节。

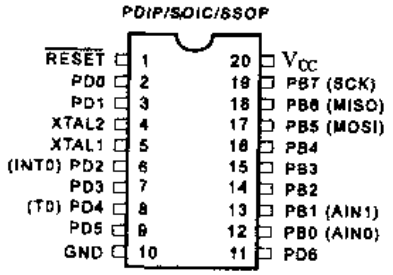


图 5.2 AT90S1200 引脚配置

D 口（PD6～PD0）：D 口是一个带内部上拉电阻的 7 位双向 I/O 口。输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。D 口作为特殊功能口的使用方法见以后章节。

$\overline{\text{RESET}}$：复位输入。超过 50ns 的低电平将引起系统复位；低于 50ns 的脉冲不能保证可靠复位。

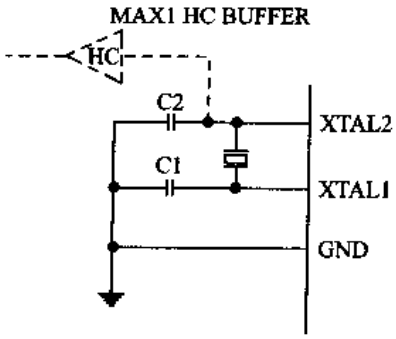
XTAL1：振荡器放大器的输入端。

XTAL2：振荡器放大器的输出端。

二、晶体振荡器

XTAL1 和 XTAL2 分别是片内振荡器的输入、输出端，可使用晶体振荡器或陶瓷振荡器。

当使用外部时钟时，XTAL2 应悬空。图 5.3 和图 5.4 分别为振荡器连接图和外部时钟驱动配置图。



注：若要利用 MCU 的振荡器作为外围器件的时钟，应在上图连接一个 HC 缓冲器。

图 5.3 振荡器连接

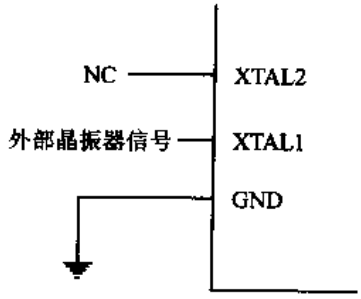


图 5.4 外部时钟驱动配置

三、片内 RC 振荡器

可以选择片内的 RC 振荡器(频率为 1MHz)作为 MCU 的时钟，从而 AT90S1200 可以在“零外围”的情况下工作。当 RCEN 位编程为“0”时，RC 振荡器即成为 MCU 时钟。RCEN 位只能在并行编程模式下改变，如果要利用 RC 振荡器来进行串行下载，首先要并行改变 RCEN。为了方便起见，产品 AT90S1200A 在出厂时已经将 RCEN 编程。

5.1.4 结构纵览

一、结 构

AVR AT90S1200 结构如图 5.5 所示。快速访问寄存器堆包含 32 个 8 位可单周期访问

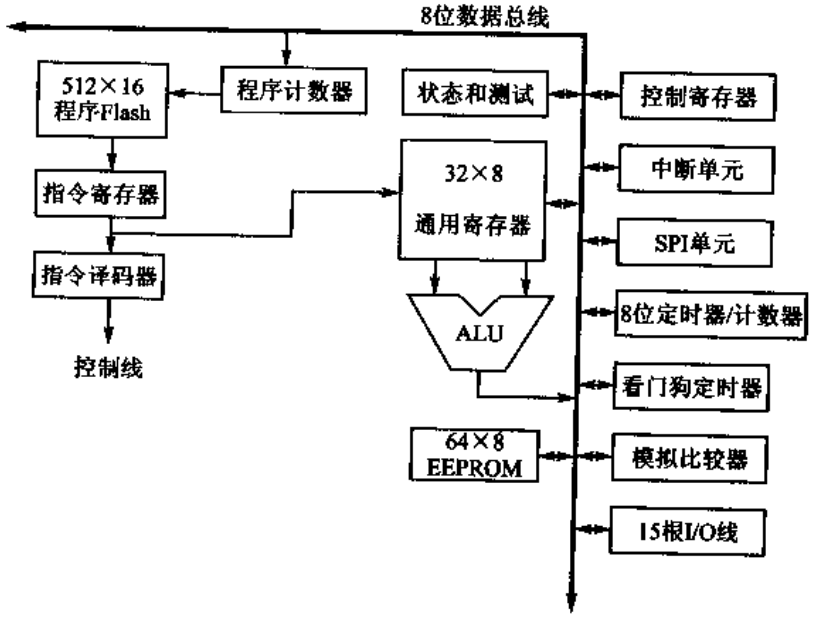


图 5.5 AT90S1200AVR RISC 结构

的通用寄存器。这意味着在一个时钟周期内，ALU 可以完成一次如下操作：读取寄存器堆中的两个操作数；执行操作；将结果存回到寄存器堆。

ALU 支持 2 个寄存器之间、寄存器和常数之间的算术和逻辑操作，以及单寄存器的操作。AVR 采用了 HARVARD 结构：程序和数据总线分离。程序内存通过两段式的管道(Pipeline)进行访问：当 CPU 在执行一条指令的同时，就去取下一条指令。这种预取指的概念使得指令可以在一个时钟完成。

相对跳转和相对调用指令可以直接访问 512 个地址空间。所有的 AVR 指令都为 16 位长，也就是说，每一个程序内存地址都包含一条 16 位的指令。

当执行中断和子程序调用时，返回地址存储于堆栈中。AT90S1200 的堆栈为 3 级硬件堆栈。I/O 内存空间包含 64 个 CPU 外围的地址，如控制寄存器、T/CO 等。AVR 结构的内存空间是线性的。

中断模块由 I/O 空间中的控制寄存器和状态寄存器中的全局中断触发位组成。每个中断都具有一个中断向量，由中断向量组成的中断向量表位于程序存储区的最前面。中断向量地址低的中断具有高的优先级。

二、通用工作寄存器

图 5.6 所示为在 CPU 中，32 个通用工作寄存器文件的结构图。

所有的寄存器操作指令都可以单指令的形式直接访问所有的寄存器。例外情况为 5 条涉及常数操作的指令：带进位位减立即数 SBCI、减立即数 SUBI、与立即数比较 CPI、与立即数 ANDI 和或立即数 ORI。这些与立即数有关的指令只能访问通用寄存器堆的后半部分：R16～R31。寄存器 R30 也作为寄存器间接寻址的 8 位指针。

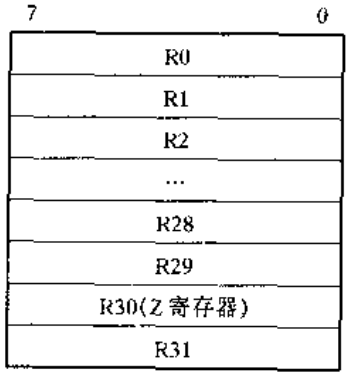


图 5.6 AVR CPU 通用工作寄存器

三、ALU 算术、逻辑操作部件

AVR ALU 与 32 个通用工作寄存器直接相连。ALU 操作分为 3 类：算术、逻辑和位操作。

四、在线可编程 Flash

AT90S1200 具有 1K 字节的 Flash。因为所有的指令为 16 位宽，所以 Flash 结构为 512×16。Flash 的擦除次数至少为 1 000 次。

AT90S1200 的程序计数器(PC)为 9 位宽，可以寻址到 512 个字的 Flash 程序区。

五、程序和数据寻址方式

AT90S1200 AVR 单片机，只有 5 种寻址方式，89 条指令，是 AVR 系列最基本的单片机。这 5 种寻址方式为单寄存器直接寻址、单寄存器间接寻址、双寄存器直接寻址、I/O 直接寻址、程序相对寻址。

六、子程序和中断硬件堆栈

AT90S1200 使用 3 级硬件堆栈，宽度为 9 位。

RCALL 相对调用指令和中断将 PC 推入堆栈 0，而堆栈 0 和 1 原有的内容进入深一级的位置。执行子程序返回 RET 或中断返回 RETI 后，堆栈 0 的 PC 将出栈，而堆栈 1 和 2 的内容上升一级。

如果有多于 3 个的子程序或中断连续(嵌套)发生,则第一个进栈的内容将丢失。这是使用 AT90S1200 特别要注意的地方。

七、EEPROM

AT90S1200 包含 64 字节的 EEPROM,其写寿命为 100 000 次。具体操作见第二章2.12。

八、指令操作时序

指令执行和内存访问时序。见第二章 2.3.4 存储器访问和指令执行时序。

九、I/O 存储器

表 5.1 为 AT90S1200 的 I/O 空间。

表 5.1 AT90S1200 的 I/O 空间

地址(十六进制)	名 称	功 能
$3F	SREG	状态寄存器
$3B	GIMSK	通用中断屏蔽寄存器
$39	TIMSK	T/C 中断屏蔽寄存器
$38	TIFR	T/C 中断标志寄存器
$35	MCUCR	MCU 控制寄存器
$33	TCCR0	T/C0 控制寄存器
$32	TCNT0	T/C0(8 位)
$21	WDTCR	看门狗控制寄存器
$1E	EEAR	EEPROM 地址寄存器
$1D	EEDR	EEPROM 数据寄存器
$1C	EECR	EEPROM 控制寄存器
$18	PORTB	B 口数据寄存器
$17	DDRB	B 口数据方向寄存器
$16	PINB	B 口输入引脚
$12	PORTD	D 口数据寄存器
$11	DDRD	D 口数据方向寄存器
$10	PIND	D 口输入引脚
$08	ACSR	模拟比较器控制及状态寄存器

AT90S1200 的所有 I/O 和外围都被放置在 I/O 空间。IN 和 OUT 指令用来访问不同的 I/O 地址,以及在 32 个通用寄存器之间传输数据。地址为 $00～$1F 的 I/O 寄存器还可用置位 I/O 位 SBI 指令和清零 I/O 位 CBI 指令进行位寻址,而 SBIC(I/O 位清零跳过一条指令)和 SBIS(I/O 位置位跳过一条指令)则用来检查单个位置位与否。

为了与后续产品兼容,保留未用的位应写"0",而保留的 I/O 寄存器则不应写。

一些状态标志位的清除是通过写"1"来实现的。清零 I/O 位 CBI 指令和置位 I/O 位 SBI 指令读取已置位的标志位时,会回写"1",因此会清除这些标志位。

I/O 寄存器和外围控制寄存器在后续章节介绍。

状态寄存器 SREG(Status Register)

BIT	7	6	5	4	3	2	1	0
$3F	I	T	H	S	V	N	Z	C
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

I:全局中断触发。置位时触发全局中断。单独的中断触发由中断屏蔽寄存器 GIMSK/TIMSK 控制。如果 I 清零,则不论单独中断标志置位与否,都不会产生中断。I 在复位时清零,RETI 指令执行后置位。

T:位拷贝存储。位拷贝指令 BLD(T 送 Rr 的 b 位)和 BST(Rr 的 b 位送 T)利用 T 作为目的或源地址。BST 把寄存器的某一位拷贝到 T,而 BLD 把 T 拷贝到寄存器的某一位。

H:半加标志位。运算中低 4 位向高位进位时,H 置位 1。

S:符号位。总是 N 与 V 的异或。

V:二进制补码溢出标志位。

N:负数标志位。

Z:零标志位。

C:进位标志位。

状态寄存器在进入中断和退出中断时并不自动进行存储和恢复,这项工作由软件完成。

十、复位和中断处理

AT90S1200 有 3 个中断源,每个中断源在程序空间都有一个独立的中断向量。这 3 个中断事件有自己的触发位。当触发位置位,且 I 也置位的情况下,中断可以发生。

器件复位后,程序空间的最低位置自动定义为中断向量。完整的中断表见表 5.2。在中断向量表中处于低地址的中断具有高的优先级。所以,RESET 具有最高的优先级。

表 5.2 复位与中断向量

向量号	程序地址	来 源	中断定义
1	$000	RESET	硬件引脚、上电复位和看门狗复位
2	$001	INT0	外部中断 0
3	$002	TIMER0,OVF0	T/C0 溢出
4	$003	ANA_COMP	模拟比较器

设置中断向量地址最典型的方法如下:

```
地址    标号    代码          注释
$000            RJMP  RESET   ;复位
$001            RJMP  EXT_INT0 ;IRQ0
$002            RJMP  TIM0_OVF ;T0 溢出
$003            RJMP  ANA_COMP ;模拟比较器
;
$004    MAIN:   <指令>  XXX    ;主程序开始
—       —       —       —
```

1. 复位源

AT90S1200 有 3 个复位源：

- 上电复位。当电源电压低于上电门限 V_{POT} 时，MCU 复位。
- 外部复位。当$\overline{\text{RESET}}$引脚上的低电平超过 50ns 时，MCU 复位。
- 看门狗复位。看门狗定时器超时后，MCU 复位。

图 5.7 的电路图说明了复位逻辑。

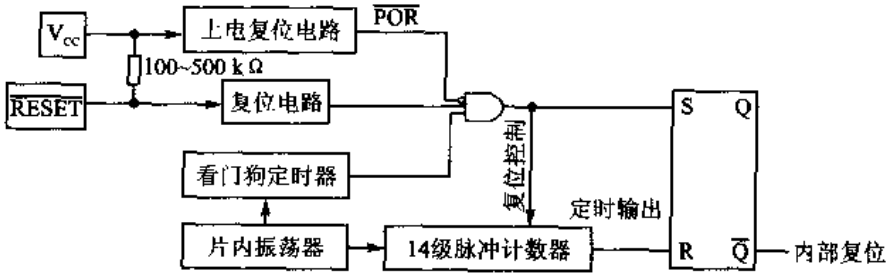


图 5.7　复位电路的逻辑图

在复位期间，所有的 I/O 寄存器被设置为初始值，程序从地址 $000 开始执行。$000 地址中放置的指令必须为 RJMP——相对跳转指令——跳转到复位处理例程。若程序永远不需中断，则中断向量就可放置通常的程序代码。表 5.3 定义了复位电路的时序和电参数。

表 5.3　复位电参数(V_{CC} = 5.0V)

符　号	参　数	最小值	典型值	最大值	单位
V_{POT}*	上电复位电压门限(上升)	0.8	1.2	1.6	V
	上电复位电压门限(下降)	0.2	0.4	0.6	V
V_{RST}	引脚门限电压		Vcc/2	0.85V_{CC}	V
t_{POR}	上电复位周期	2	3	4	ms
t_{TOUT}	复位延迟周期(等于 16K 个 WDT 时钟)	11	16	21	ms

* 除非电源电压低于 V_{POT}，否则上电复位不会发生。

2. 中断处理

AT90S1200 有 2 个中断屏蔽控制寄存器：GIMSK —通用中断屏蔽寄存器和 TIMSK——T/C 中断屏蔽寄存器。

一个中断产生后，全局中断触发位 I 将被清零，后续中断被屏蔽。用户可以在中断例程里对 I 置位，从而开放中断。执行 RETI 后 I 重新置位。

当程序计数器指向实际中断向量开始执行相应的中断例程时，硬件清除对应的中断标志。一些中断标志位也可以通过软件写"1"来清除。

当一个符合条件的中断发生后，如果相应的中断触发位为"0"，则中断标志位置位，并一直保持到中断执行，或者被软件清除。

如果全局中断标志被清零，则所有的中断都不会被执行，直到 I 置位。

注意：外部电平中断没有中断标志位，因此当电平变为非中断电平后，中断条件即终止。进入中断和退出中断时 MCU 不会自动保存或恢复状态寄存器，故需由软件处理。

● **通用中断屏蔽寄存器——GIMSK**

BIT	7	6	5	4	3	2	1	0
$3B	—	INT0	—	—	—		—	—
读/写	R	R/W	R	R	R	R	R	R
初始值	0	0	0	0	0	0	0	0

位 7、5…0：保留。

INT0：外部中断 0 请求触发。

当 INT0 和 I 都为“1”时，外部引脚中断触发。MCU 控制寄存器(MCUCR)中的中断检测控制位 1/0(ISC01 和 ISC00)定义中断 0 是上升沿中断还是下降沿中断，或者是低电平中断。即使引脚被定义为输出，中断仍可产生。

● **T/C 中断屏蔽寄存器——TIMSK**

BIT	7	6	5	4	3	2	1	0
$39	—	—		—	—	—	TOIE0	
读/写	R	R	R	R	R	R	R/W	R
初始值	0	0	0	0	0	0	0	0

位 7…2、0：保留。

TOIE0：T/C0 溢出中断触发。

当 TOIE0 和 I 都为“1”时，T/C0 溢出中断触发。当 T/C0 溢出或 TIFR 中的 TOV0 位置位时，中断例程($002)得到执行。

● **T/C 中断标志寄存器——TIFR**

BIT	7	6	5	4	3	2	1	0
$38		—	—	-	-	—	TOV0	—
读/写	R	R	R	R	R	R	R/W	R
初始值	0	0	0	0	0	0	0	0

位 7…2、0：保留。

TOV0：T/C0 溢出中断标志位。

当 T/C0 溢出时，TOV0 置位。执行相应的中断例程后此位硬件清零。此外，TOV0 也可以通过写“1”来清零。当 SREG 中的位 I、TOIE0 和 TOV0 一同置位时，中断例程得到执行。

3. 外部中断

外部中断由 INT0 引脚触发，触发方式可以为上升沿、下降沿或低电平。这些设置由 MCU 控制寄存器 MCUCR 决定。当 INT0 设置为低电平触发时，只要电平为低，中断就一直挂起。即使 INT0 配置为输出，中断也会发生。这种特性可以用来实现软件中断。

用户不能直接访问中断标志位。如果怀疑有一个边沿触发中断被挂起，则标志位可以通过如下步骤清除：

- 清除 GIMSK 的 INT0 位以禁止外部中断；
- 选择电平触发中断方式；
- 选择需要的中断沿；
- 置位 INT0，重新触发外部中断。

4. 中断响应时间

AVR 中断响应时间最少为 4 个时钟周期。在这 4 个时钟期间，PC 自动入栈。在通常情

况下，中断向量为一个相对跳转指令，此跳转要花 2 个时钟周期。如果中断在一个多周期指令执行期间发生，则在此一个多周期指令执行完后 MCU 才会执行中断程序。

中断返回亦需 4 个时钟。在此期间，PC 将被弹出栈，SREG 的位 I 被置位。如果在中断期间发生了其他中断，则 AVR 在退出中断程序后，要执行一条主程序指令之后才能再响应被挂起的中断。

要注意 AT90S1200 只有一个 3 级硬件堆栈。如果 3 个以上例程嵌套发生，则只有最后的 3 个返回地址得到保留，其它的将丢失。

● **MCU 控制寄存器——MCUCR**

BIT	7	6	5	4	3	2	1	0
$35	—	—	SE	SM		—	ISC01	ISC00
读,写	R	R	R/W	R/W	R	R	R/W	R/W
初始值	0	0	0	0	0	0	0	0

位 7、6、3、2：保留。

SE：休眠触发。执行 SLEEP 指令时，SE 必须置位才能使 MCU 进入休眠模式。为了防止无意间使 MCU 进入休眠，建议与 SLEEP 指令相连使用。

SM：休眠模式。此位用于选择休眠模式。SM 为“0”时，为空闲模式；SM 为“1”时，为掉电模式。

ISC01，ISC00：中断检测控制位。选择 INT0 中断的边沿或电平，如表 5.4 所列。

注意：改变 ISC01/ISC00 时，首先要禁止 INT0(清除 GIMSK 的 INT0 位)，否则可能引发不必要的中断。INT0 引脚的电平在检测边沿之前采样。如果边沿中断触发，则大于一个 MCU 时钟的脉冲将触发中断；如果选择了低电平触发，则此电平必须保持到当前执行的指令结束。

表 5.4　中断 0 检测控制

ISC01	ISC00	描　述
0	0	低电平中断
0	1	保留
1	0	下降沿中断
1	1	上升沿中断

十一、省电方式

见第二章 2.6 AVR 单片机省电方式。

十二、定时器/计数器 0(T/C0)

见第二章 2.7 AVR 单片机定时器/计数器。

十三、EEPROM 读/写

写 EEP 的时间与电压有关，在 2.5～4ms 之间。自定时功能可以监测何时开始写下一字节。如果操作 EEPROM，应当注意如下问题：在电源滤波时间常数比较大的电路中，上电/下电时，V_{cc} 上升/下降会比较慢。此时 MCU 将工作于低于晶振所要求的电源电压。在这种情况下，程序指针有可能跑飞，并执行 EEP 写指令。为了保证 EEP 数据的完整性，建议使用低电压复位电路。

十四、模拟比较器

模拟比较器比较正输入端 PB0(AIN0)和负输入端 PB1(AIN1)的值。如果 PB0(AIN0)的电压高于 PB1(AIN1)的值，比较器的输出 AC0 将置位。此输出可用来触发模拟比较器中断(上升沿、下降沿或电平变换)。应用实例见第七章 7.3.7 廉价的 A/D 转换器及 7.3.8 高精度廉价的 A/D 转换器。

十五、I/O 口

所有的 AVR I/O 端口都具有真正的读-修改-写功能。这意味着用 SBI 或 CBI 指令改变某些引脚的方向(值、禁止/触发、上拉)时不会无意地改变其它引脚的方向(值、禁止/触发、上拉)。

1. B 口

B 口是 8 位双向 I/O 口。

B 口有 3 个 I/O 地址:数据寄存器——PORTB($18)、数据方向寄存器——DDRB($17)和输入引脚——PINB($16)。PORTB 和 DDRB 可读可写,PINB 只可读。

所有的引脚都可以单独选择上拉电阻。引脚缓冲器可以吸收 20mA 的电流,能够直接驱动 LED。当引脚被拉低时,如果上拉电阻已经激活,则引脚会输出电流。

B 口的第二功能如表 5.5 所列。

表 5.5 B 口第二功能

引 脚	第二功能
PB0	AIN0(模拟比较器正输入端)
PB1	AIN1(模拟比较器负输入端)
PB5	MOSI(程序下载时的数据输入线)
PB6	MISO(程序下载时的数据输出线)
PB7	SCK(串行时钟)

注:当使用 B 口的第二功能时,DDRB 和 PORTB 要设置成对应的值。

2. D 口

D 口有 3 个 I/O 地址:数据寄存器——PORTD($12)、数据方向寄存器——DDRD($11)和输入引脚——PIND($10)。PORTD 和 DDRD 可读可写,PIND 只可读。

D 口有 7 个带上拉电阻的双向 I/O 引脚,PD6～PD0。引脚缓冲器可以吸收 20mA 的电流,能够直接驱动 LED。当引脚被拉低时,如果上拉电阻已经激活,则引脚会输出电流。

D 口的第二功能如表 5.6 所列。

表 5.6 D 口第二功能

引 脚	第二功能
PD2	INT0(外部中断 0 输入)
PD4	T0(T/C0 的外部输入)

十六、程序编程

AT90S1200 具有 2 个锁定位,如表 5.7 所列。锁定位只能通过片擦除命令擦除。

表 5.7 锁定保护模式

程序锁定位			保护类型
模式	LB1	LB2	
1	1	1	无锁定功能
2	0	1	禁止编程*
3	0	0	禁止校验

* 在并行编程模式下,熔断位编程也被禁止。要先编程熔断位,然后编程锁定位。

5.2 AT90S2313

5.2.1 特 点

(1) AVR RISC 结构：

高性能、低功耗 RISC 结构；

——118 条指令，大多数为单指令周期；

——32 个 8 位通用(工作)寄存器；

——工作在 10MHz 时具有 10MIPS 的性能。

(2) 数据和非易失性程序内存：

——2K 字节的在线可编程 Flash(擦除次数为 1 000 次)；

——128 字节 SRAM；

——128 字节在线可编程 EEPROM(寿命为 100 000 次)；

——程序加密位。

(3) 外围(Peripheral)特点：

——1 个可预分频(Prescale)的 8 位定时器/计数器；

——1 个可预分频、具有比较、捕捉和 8/9/10 位 PWM 功能的 16 位定时器/计数器；

——片内模拟比较器；

——可编程的看门狗定时器(由片内振荡器生成)；

——用于下载程序的 SPI 口；

——全双工 UART。

(4) MCU 特点：

——低功耗空闲和掉电模式；

——内外部中断源。

(5) 规范(Specification)：

——低功耗、高速 CMOS 工艺；

——全静态工作。

(6) 在 4MHz、3V、25℃条件下的功耗：

——工作模式为 2.8mA；

——空闲模式为 0.8mA；

——掉电模式为$<1\mu A$。

(7) I/O 和封装：

——15 个可编程的 I/O 脚；

——20 脚 PDIP 和 SOIC 封装。

(8) 工作电压：

——2.7～6.0V(AT90S2313－4)；

——4.0～6.0V(AT90S2313－10)。

(9) 速度：

——0～4MHz(AT90S2313－4)；

— 0～10MHz(AT90S2313－10)。

5.2.2 描 述

图 5.8 为 AT90S2313 结构图。AT90S2313 是一款基于 AVR RISC 的低功耗、CMOS 8 位单片机。通过在一个时钟周期内执行一条指令，AT90S2313 可以取得接近 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。

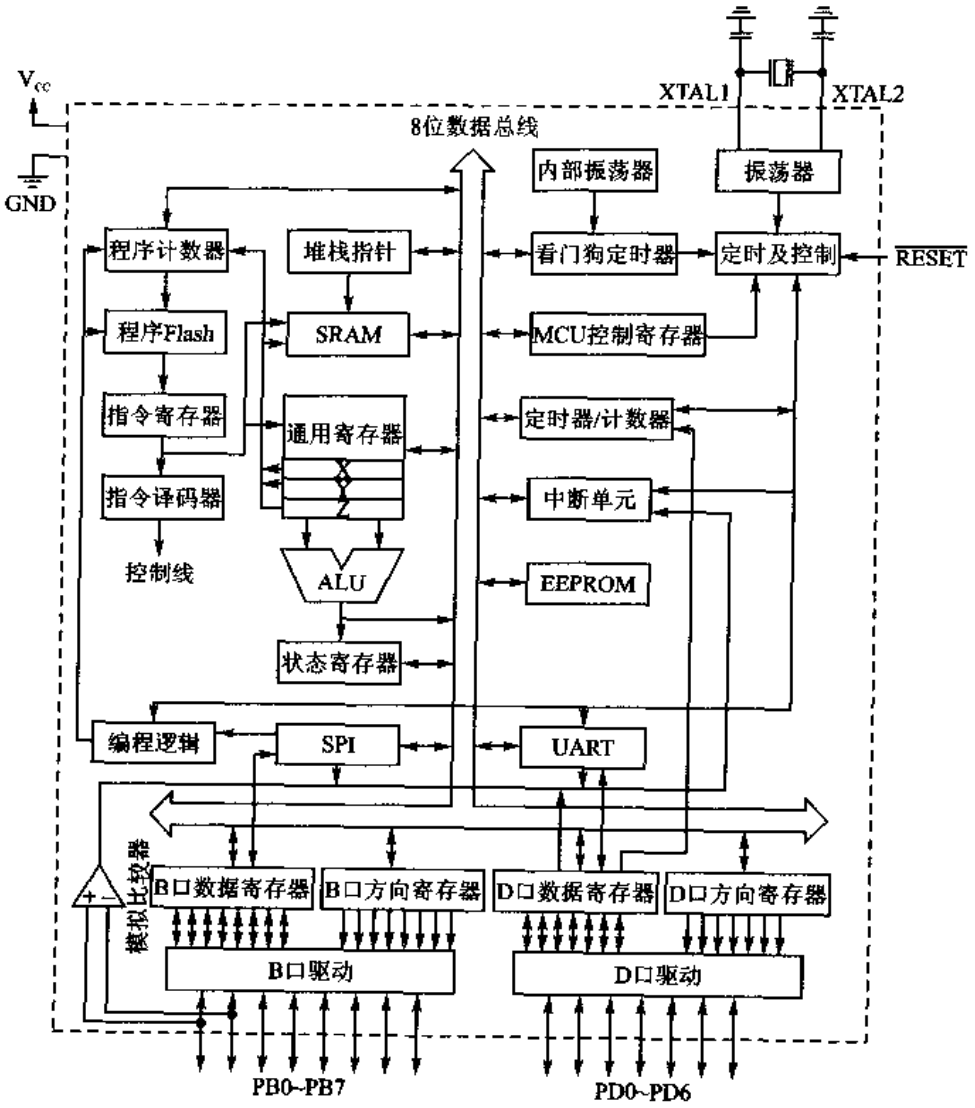


图 5.8 AT90S2313 结构方框图

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU (算逻单元)直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

AT90S2313 具有以下特点：2K 字节 Flash；128 字节 EEPROM；128 字节 SRAM；15 个通用 I/O 口；32 个通用工作寄存器；具有比较模式的灵活的定时器/计数器；内外中断源；可编程的 UART；可编程的看门狗定时器；下载程序用的 SPI 口以及 2 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而寄存器、定时器/计数器、看门狗和中断系统继续

工作;掉电模式时振荡器停止工作,所有功能都被禁止,而寄存器内容得到保留。只有外部低电平中断或硬件复位才可以退出此状态。

器件是以ATMEL的高密度、非易失性内存技术生产的。片内Flash可以通过ISP接口或通用编程器多次编程。通过将增强的RISC 8位CPU与Flash集成在一个芯片内,AT90S2313为许多嵌入式控制应用提供了灵活而低成本的方案。

AT90S2313具有一整套的编程和系统开发工具:宏汇编、调试/仿真器、在线仿真器和SL-AVR编程开发实验器。

5.2.3 引脚配置

AT90S2313引脚配置如图5.9所示。

引脚定义

V_{CC}、GND:电源。

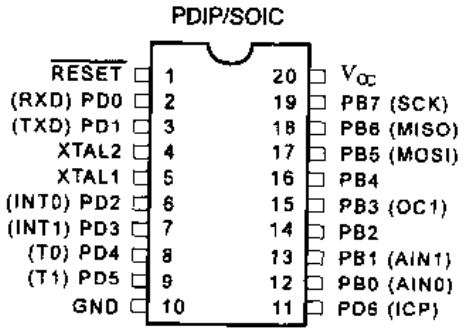


图5.9 AT90S2313引脚配置图

B口(PB7~PB0):B口是一个8位双向I/O口,每一个引脚都有内部上拉电阻(可单独选择)。PB0和PB1还可作为片内模拟比较器的正(AIN0)负(AIN1)输入端。B口的输出缓冲器能够吸收20mA的电流,可直接驱动LED。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,B口为三态,即使此时时钟还未起振。B口作为特殊功能口的使用方法见光盘文件。

D口(PD6~PD0):D口是一个带内部上拉电阻的7位双向I/O口。输出缓冲器能够吸收20mA的电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,D口为三态,即使此时时钟还未起振。D口作为特殊功能口的使用方法见光盘文件。

$\overline{\text{RESET}}$:复位输入。超过50ns的低电平将引起系统复位。低于50ns的脉冲不能保证可靠复位。

XTAL1:振荡器放大器的输入端。

XTAL2:振荡器放大器的输出端。

5.3 ATmega8/8L

ATmega8/8L是AVR高档单片机成员之一。它具有AVR高档单片机的性能,且具有低档单片机的价格,深受广大单片机用户的喜爱。尤其AVR单片机不需购买昂贵的仿真器、编程器也可作单片机的开发应用,这对单片机初学者尤为重要。ATmega8的高性能低价格,在产品应用市场上极具强大的竞争力,被很多家用电器厂商、仪器仪表行业看中,从而使ATmega8进入大批量的应用领域。

ATmega8/8L是带8K字节Flash的在线可编程8位微控制器。

5.3.1 特 点

(1) AVR RISC 结构：

—— 高性能、低功耗 RISC 结构；

130 条功能强大的指令，大多数为单指令周期；

· · 32×8 个通用工作寄存器；

——工作在 16 MHz 时具有 16MIPS 的性能；

——片内带有执行时间为 2 个时钟周期的硬件乘法器。

(2) 数据和非易失性程序内存：

——8K 字节在线可编程 Flash(擦写次数为 1 000 次)；

· ··BOOT 区具有独立的加密位；

可通过片内的引导程序实现在系统编程，写操作时真正可读；

——512 字节 EEPROM(擦写次数为 100 000 次)；

——1K 字节内部 SRAM；

——程序加密位。

(3) 外围(Periphera)特点：

——两个带预分频器和一种比较模式的 8 位定时器/计数器；

——一个带预分频器和比较模式、捕获模式的 16 位定时器/计数器；

- ···具有独立振荡器的实时计数器；

——三通道 PWM；

—··TQFP 和 MLF 封装芯片，有 8 通道的 A/D 转换：6 路 10 位和 2 路 8 位转换精度的通道；

——PDIP 封装芯片，有 6 通道的 A/D 转换：4 路 10 位和 2 路 8 位转换精度的通道。

——两线(I^2C)串行接口；

——可编程串行 UART 接口；

—— 主/从 SPI 串行接口；

——带内部振荡器的可编程看门狗定时器；

——片内模拟比较器。

(4) MCU 特点：

——上电复位和可编程的低电压检测；

——内部可校准的 RC 振荡器；

—· 外部和内部中断源；

——五种休眠模式为空闲模式、ADC 噪声抑制模式、省电模式、掉电模式、待命模式。

(5) 在 4MHz、3V、25℃条件下的功耗：

—·—激活模式为 3.6mA；

·——空闲模式为 1.0mA；

· ··—掉电模式为 0.5μA。

(6) I/O 和封装：

—— 23 个可编程 I/O 脚；

28 脚 PDIP 封装、32 脚 TQFP 封装、32 脚 MLF 封装。

(7) 工作电压：

---2.7～5.5V(ATmega8L)；

——4.5～5.5V(ATmega8)。

(8) 速度：

0～8MHz(ATmega8L)；

0～16MHz(ATmega8)。

5.3.2 描 述

图 5.10 为 ATmega8/8L 结构框图。ATmega8 是一款基于 AVR RISC 的、低功耗 CMOS 的 8 位单片机。通过在一个时钟周期内执行一条指令，ATmega8 可以取得 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。

AVR 核将 32 个工作寄存器和丰富的指令集连接在一起。所有的工作寄存器都与 ALU 算术逻辑单元直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

ATmega8 具有以下特点：8K 字节具备写操作时可读的在系统可编程 Flash；512 字节 EEPROM；1K 字节 SRAM；23 个通用 I/O 口；32 个通用工作寄存器；3 个具有比较模式的定时器/计数器；内外中断源；可编程串行 UART；一个两线(I^2C)串行接口；6 通道的 A/D 转换(TQFP 和 MLF 封装芯片为 8 通道)，其中 4(6)通道为 10 位精度、2 通道为 8 位精度；一个带内部振荡器的可编程看门狗定时器；一个 SPI 口，以及五种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而 SRAM、定时器/计数器、SPI 口和中断系统继续工作；掉电模式时，振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留，直到下一个外部中断或硬件复位才退出此状态；省电模式时异步定时器继续工作，以使用户能在芯片的其余部分处于休眠状态时保持定时器基准；ADC 噪声抑制模式除异步定时器和 ADC 继续工作外，停止 CPU 运行和所有的 I/O 单元，以减少 ADC 转换时的开关噪声；待命模式中除晶振工作外，芯片的其余部分处于休眠状态，这就使得在低功耗的情况下能非常快的启动。

ATmega8 芯片是以 ATMEL 的高密度非易失性内存技术生产的。Flash 程序存储器，可以通过 SPI 串行接口，或通用编程器，或运行于 AVR 核上的片内 BOOT 程序，多次编程。BOOT 程序可以用任意的接口下载到应用 Flash 程序存储器中。当应用程序区被更新时，BOOT 区的软件将继续运行，提供写操作时真正可读的功能。ATmega8 通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega8 具有一整套的编程和系统开发工具：C 编译器、宏汇编器、调试/模拟器、在线仿真器和 SL－MEGA8 评估板。

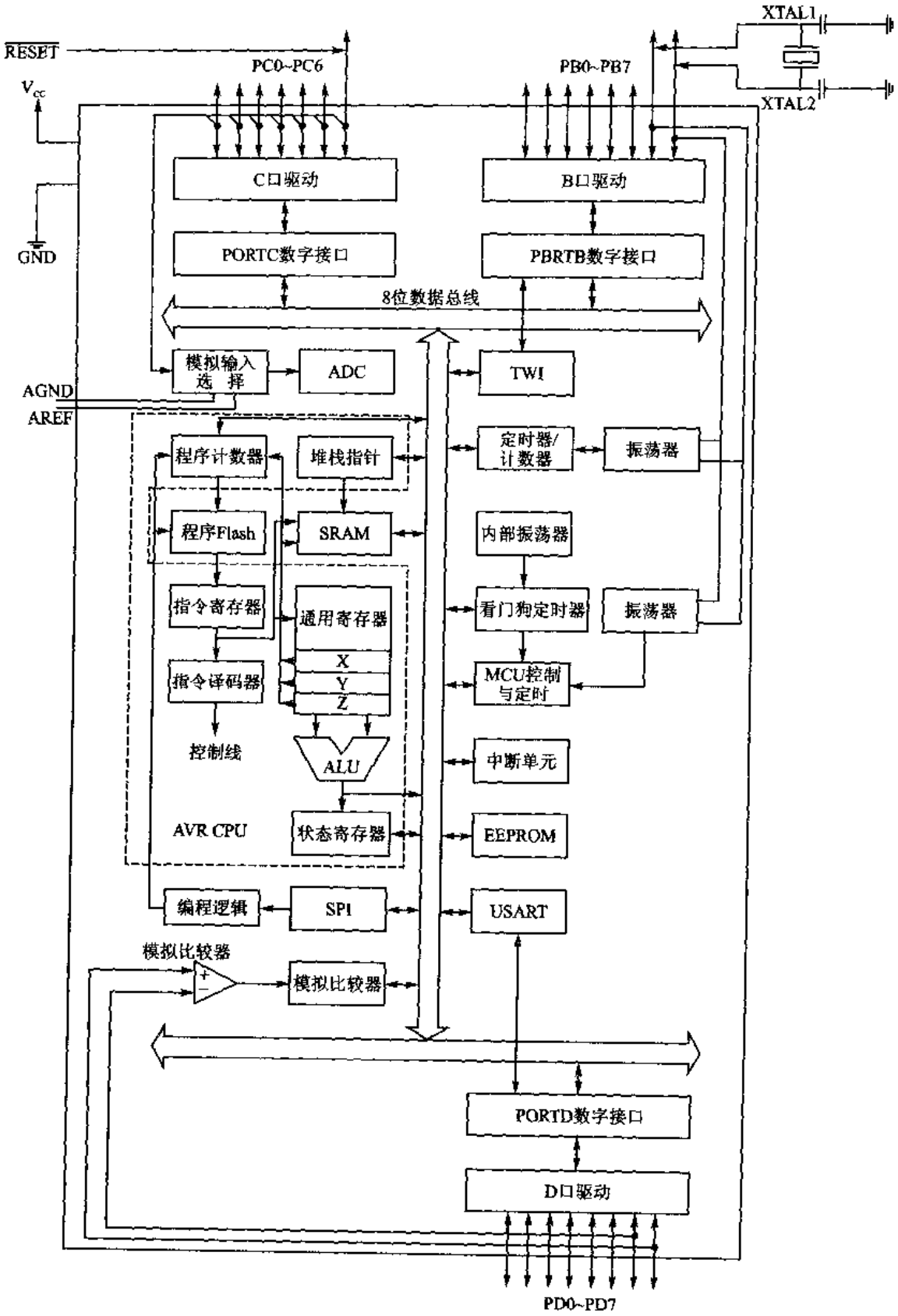


图 5.10 ATmega8/8L 结构框图

5.3.3 引脚配置

ATmega8/8L 引脚配置如图 5.11 所示。

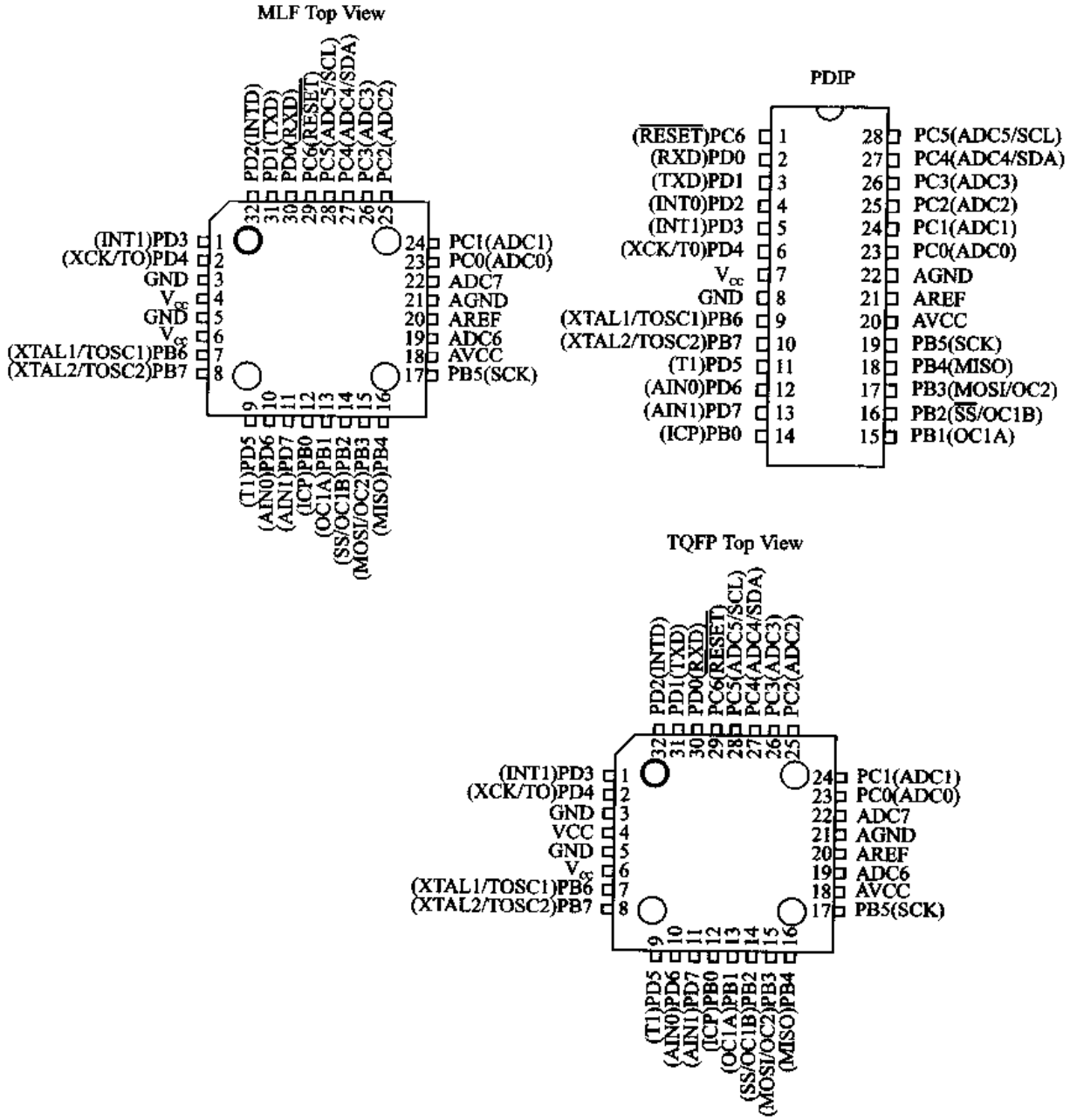


图 5.11 ATmega8/8L 引脚配置

引脚定义

V_{CC}、GND：电源。

B 口(PB7～PB0)：B 口是一个 8 位双向 I/O 口，每一个引脚都有内部可选上拉电阻。B 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。

根据时钟选择熔丝位的设定，PB6 可作为反向振荡放大器的输入和内部时钟工作电路的

输入，PB7 可作为反向振荡放大器的输出。

如果内部可校准的 RC 振荡器用做芯片的时钟源，ASSR 中的 AS2 位被置 1，PB7 和 PB6 就作为异步定时器/计数器 2 的输入 TOSC2 和 TOSC1。

C 口(PC5～PC0)：C 口是一个 7 位双向 I/O 口，每一个引脚都有内部上拉电阻。C 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，C 口为三态，即使此时时钟还未起振。

PC6/$\overline{\text{RESET}}$：如果 RSTDISBL 熔丝位被编程，PC6 就作为一个 I/O 引脚。注意，PC6 的电气特性与 C 口的其它引脚不同。如果 RSTDISBL 熔丝位未被编程，PC6 就作为复位输入脚。即使此时时钟还未起振，超过 50ns 的低电平将引起系统复位，低于 50ns 的脉冲不能保证可靠复位。

D 口(PD7～PD0)：D 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。D 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。

RESET：复位输入。即使此时时钟还未起振，超过 50ns 的低电平将引起系统复位，低于 50ns 的脉冲不能保证可靠复位。

XTAL1：反向振荡放大器的输入和内部时钟工作电路的输入。

XTAL2：反向振荡放大器的输出。

AVCC：AVCC 是 A/D 转换器、C 口(PC3～PC0)和 ADC (7 和 6)的电源端。即使不使用 ADC，也应外接到 V_{CC} 端。如使用 ADC，应该通过一个低通滤波器与 V_{CC} 连接。注意 C 口(PC5、PC4)通过 V_{CC} 供电。

AREF：A/D 转换器的参考电源。

ADC7 和 ADC6：TQFP 和 MLF 封装的芯片中，ADC7 和 ADC6 作为 A/D 转换器的模拟输入。这些引脚由模拟输入供电并作为 10 位 ADC 通道。

5.3.4 开发实验工具

为了使用户深入了解、牢固掌握 ATmega8 的开发与应用，广州双龙电子有限公司迅速开发出 SL－MEGA8 开发实验器(评估系统)。硬件模块充分考虑到 ATmega8 的性能特点及其配套电路接口，软件上也为用户提供相应的软件模块，使用户快速上手，设计出适合自己项目的科研样机。ATmega8 与 ATmega16/32/64/128 仅存在量的差异，主要性能完全兼容；所以，学会 ATmega8 的开发应用，对其它 ATmega 系列单片机的开发应用也就迎刃而解。

SI－MEGA8 开发实验器上配 ATmega8 器件。复位电路 RESET 有 2 种选择：外部按键复位或 PC6 作 I/O 口用。有外接 DC－9V 电源插座，输入整流滤波后，产生＋9V 电源，供 LM358 运放用。电源输入整流稳压滤波后＋5V，供整个电路用。有 RS232 接口，可做 PC 机与 ATmega8 的异步串行 UART 通信；用 ATmega8 BOOT 区创建 ATmega8 自监控后，可作用户程序(Flash/EEPOM)下载、读/写等操作；也可把 PC 机屏幕作为用户显示终端使用，可充分利用 PC 机资源。

晶振有 4 种选择：① 外接 8MHz 无源晶振；② 外接 8MHz 有源晶振；③ 接实时时钟晶振 32.768kHz；④ 用内部 RC 振荡器，可做成无外接器件应用系统。

ATmega8 芯片有 2 个按键作外部中断 INT0，INT1 输入信号。有对应引脚输出插针，供

用户实验用。有 DAC0,DAC1 信号输出接口，DAC 电压也可通过 ADC 采样转换为数字值显示到 PC 屏幕上。有主/从 SPI 同步通信接口,供并口 ISP 下载编程用。有无源音响器,使实验有声有色。有 ADC 输入接口,用多圈电位器调 A/D 模拟电压供测试用,用短路块可连接其它通道做 ADC 实验。有通用 PC 机键盘 PS/2 接口。有 4 位 LED 数码管作显示用。有 8 位 LED 发光二极管,供 ATmega8 的 I/O 口作电平指示实验用。有电源指示,通信忙指示。有 IIC 总线配套外围接口电路 AT24C02,供实验用。有 AREF 基准电压通、断选择。有模拟比较器接口电路。有二路 PWM(D/A)输出接口、三级网络滤波、模拟电压放大电路供用户实验。印制板右侧有大面积布线区,供用户增加驱动接口实验。

SL-MEGA8 仿真开发实验器也可配上组态监控,成为 SL-MEGA8 组态开发实验器系统。这样充分利用 PC 机资源,使 AVR 单片机教学生动、形象、直观,有声有色地了解单片机 I/O 口基本功能、I/O 口的扩展功能、A/D 采样显示及单片机组态开发的复杂应用。SL-MEGA8 组态开发实验器系统,可组态互动演示:如 PC 机屏幕控件开关,可控制单片机开发实验器上的 LED 灯和开关;也可控制 PC 机屏幕上的控件灯,当然 PC 机上控件开关也可控制 PC 机上的控件灯;还可学习工控组态方法,使单片机开发学习与现实社会工程应用接轨。

5.4 AT90S2333/4433

5.4.1 特 点

(1) 高性能、低功耗 AVR RISC 结构:

——118 条指令,大多数为单指令周期;

——32 个 8 位通用(工作)寄存器;

——工作在 8MHz 时具有 8MIPS 的性能。

(2) 数据和非易失性程序内存:

——2K/4K 字节的在线可编程 Flash(擦除次数为 1 000 次);

——128 字节 SRAM;

——128/256 字节在线可编程 EEPROM(寿命为 100 000 次);

——程序、EEPROM 加密位。

3. 外围(Peripheral)特点:

—— 1 个可预分频(Prescale)8 位定时器/计数器;

——1 个可预分频、具有比较、捕捉和 8/9/10 位 PWM 功能的 16 位定时器/计数器;

——片内模拟比较器;

——可编程的看门狗定时器(由片内振荡器生成);

——可编程 UART;

——6 通道 10 位 ADC;

——主/从 SPI 接口。

(4) MCU 的特点:

——电源检测(Brown-Out-Detection)功能;

——增强的上电复位电路;

原书缺页

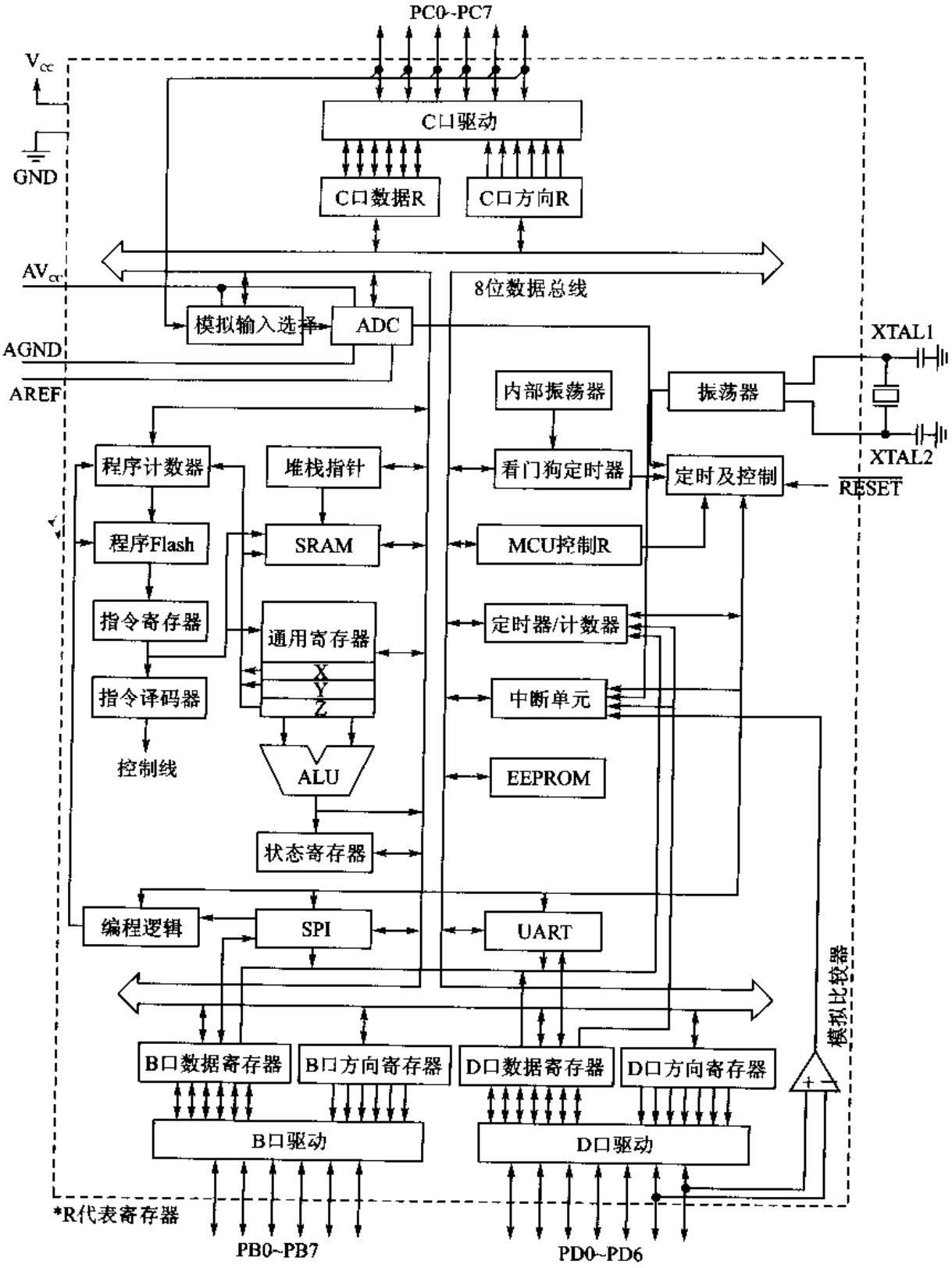


图 5.12 AT90S2333/4433 结构方框图

表 5.8 存储器比较

型 号	Flash	EEPROM	SRAM	电压范围/V	频率/MHz
AT90S2333	2K 字节	128 字节	128 字节	4.0～6.0	0～8
AT90LS2333	2K 字节	128 字节	128 字节	2.7～6.0	0～4
AT90S4433	4K 字节	256 字节	128 字节	4.0～6.0	0～8
AT90LS4433	4K 字节	256 字节	128 字节	2.7～6.0	0～4

5.4.3 引脚配置

AT90S2333/4433 引脚配置如图 5.13 所示。

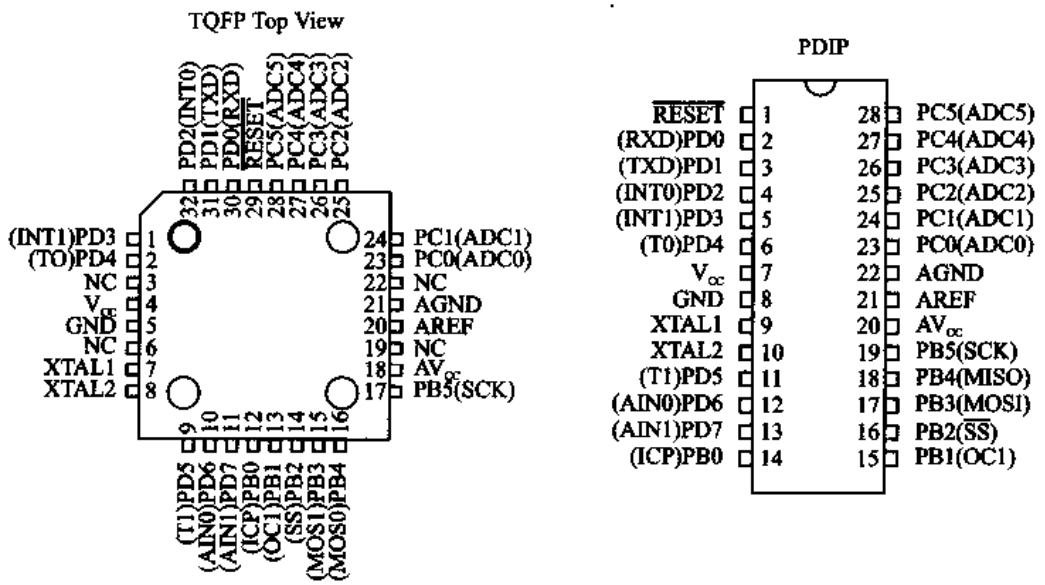


图 5.13 AT90S2333/4433 引脚图

引脚定义

V_{CC}、GND：电源。

B 口(PB5～PB0)：B 口是一个 6 位双向 I/O 口，每一个引脚都有内部上拉电阻。B 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。B 口作为特殊功能口的使用方法见光盘文件。在复位过程中，B 口为三态，即使此时时钟还未起振。

C 口(PC5～PC0)：C 口是一个 6 位双向 I/O 口，每一个引脚都有内部上拉电阻。C 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。C 口还用作 ADC 的模拟输入。在复位过程中，C 口为三态，即使此时时钟还未起振。

D 口(PD7～PD0)：D 口是一个带内部上拉电阻的 8 位双向 I/O 口。输出缓冲器能够吸收 20mA 的电流。当作为输入时，若外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。D 口作为特殊功能口的使用方法见光盘文件。

$\overline{\text{RESET}}$:复位输入。超过 50ns 的低电平将引起系统复位。低于 50ns 的脉冲不能保证可靠复位。

XTAL1/2:振荡器放大器的输入/输出端。

AV_{CC}:A/D 转换器的电源。应该通过一个低通滤波器与 V_{CC} 连接。

AREF:A/D 转换器的参考电源,介于 AGND 与 AV_{CC} 之间。

AGND:模拟地。

5.5 AT90S4414/8515

5.5.1 特 点

(1) AVR RISC 结构:

——高性能、低功耗 RISC 结构;

——118 条指令,大多数为单指令周期;

——32 个 8 位通用(工作)寄存器;

——工作在 8MHz 时具有 8MIPS 的性能。

(2) 数据和非易失性程序内存:

——4K/8K 字节的在线可编程 Flash(擦除次数为 1 000 次);

——256/512 字节 SRAM;

——256/512 字节在线可编程 EEPROM(寿命为 100 000 次);

——程序加密位。

(3) 外围(Peripheral)特点:

——1 个可预分频(Prescale)的 8 位定时器/计数器;

——1 个可预分频、具有比较、捕捉和 8/9/10 位 PWM 功能的 16 位定时器/计数器;

——片内模拟比较器;

——可编程的看门狗定时器(由片内振荡器生成);

——用于下载程序的 SPI 口;

——全双工 UART。

(4) MCU 特点:

——低功耗空闲和掉电模式;

——内外部中断源。

(5) 规范(Specification):

——低功耗、高速 CMOS 工艺;

——全静态工作。

(6) 在 4MHz、3V、25℃条件下的功耗:

——工作模式为 3.0mA;

——空闲模式为 1.0mA;

——掉电模式为$<1\mu A$。

(7) I/O 和封装:

——32 个可编程的 I/O 脚;

——40 脚 PDIP、PLCC 和 TQFP 封装。

(8) 工作电压：

——2.7～6.0V(AT90S4414－4 和 AT90S8515－4)；

— 4.0～6.0V(AT90S4414－8 和 AT90S8515－8)。

(9) 速度：

0～4MHz(AT90S4414－4 和 AT90S8515－4)；

——0～8MHz(AT90S4414－8 和 AT90S8515－8)。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU(算逻单元)直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

AT90S4414/8515 具有以下特点：4K/8K 字节 Flash；256/512 字节 EEPROM；256/512 字节 SRAM；32 个通用 I/O 口；32 个通用工作寄存器；具有比较模式的灵活的定时器/计数器；内外中断源；可编程的 UART；可编程的看门狗定时器；SPI 口以及 2 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而寄存器、定时器/计数器、看门狗和中断系统继续工作；掉电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有外部中断或硬件复位才可以退出此状态。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 可以通过 SPI 接口或通用编程器多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，AT90S4414/8515 为许多嵌入式控制应用提供了灵活而低成本的方案。

AT90S4414/8515 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和 SL－AVR 编程开发实验板。

5.5.2 AT90S4414 和 AT90S8515 的比较

AT90S4414 具有 4K 字节程序 Flash，256 字节 EEPROM，256 字节 SRAM。

AT90S8515 具有 8K 字节程序 Flash，512 字节 EEPROM，512 字节 SRAM。

表 5.9 是两个器件存储器的简单比较。

表 5.9 存储器比较

型 号	Flash	EEPROM	SRAM
AT90S4414	4K 字节	256 字节	256 字节
AT90S8515	8K 字节	512 字节	512 字节

5.5.3 引脚配置

AT90S4414/8515 引脚配置如图 5.14 所示。

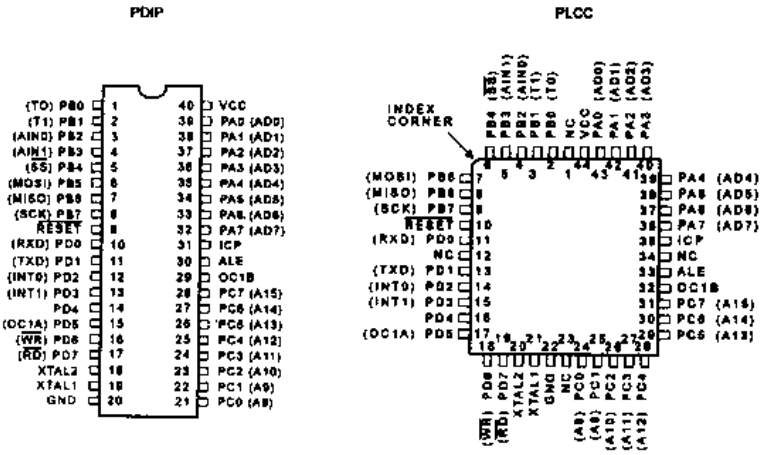


图 5.14 AT90S4414/8515 引脚配置图

5.6 AT90S4434/8535

5.6.1 特 点

(1) AVR RISC 结构：

——高性能、低功耗 RISC 结构；

——118 条指令，大多数为单指令周期；

32 个 8 位通用(工作)寄存器；

——工作在 8MHz 时具有 8MIPS 的性能。

(2) 数据和非易失性程序内存：

——4K/8K 字节的在线可编程 Flash(擦除次数为 1 000 次)；

——256/512 字节 SRAM；

——256/512 字节在线可编程 EEPROM(寿命为 100 000 次)；

——程序加密位。

(3) 外围(Peripheral)特点：

——2 个具有比较模式的可预分频(Prescale)8 位定时器/计数器；

——1 个可预分频、具有比较、捕捉和 2 个 8/9/10 位 PWM 功能的 16 位定时器/计数器；

——片内模拟比较器；

——可编程的看门狗定时器(由片内振荡器生成)；

——8 通道 10 位 ADC；

——全双工 UART。

(4) MCU 特点：

——上电复位电路；

——具有记数功能、有独立振荡器的实时时钟(RTC)；

——低功耗空闲、省电和掉电模式；

——内外部中断源。

(5) 在 4MHz、3V、20℃条件下的功耗：

——工作模式为 6.4mA；
——空闲模式为 1.9mA；
——掉电模式为<1μA。
(6) I/O 和封装：
——32 个可编程的 I/O 脚；
——40 脚 PDIP、PLCC 和 TQFP 封装。
(7) 工作电压：
——2.7～6.0V(AT90LS4434 和 AT90LS8535)；
——4.0～6.0V(AT90S4434 和 AT90S8535)。
(8) 速度：
——0～4MHz(AT90LS4434 和 AT90LS8535)；
——0～8MHz(AT90S4434 和 AT90S8535)。

5.6.2 描 述

AT90S4434/8535 是一款基于 AVR RISC 的、低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令，AT90S4434/8535 可以取得接近 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。其结构如图 5.15 所示。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU(算逻单元)直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

AT90S4434/8535 具有以下特点：4K/8K 字节 Flash；256/512 字节 EEPROM；256/512 字节 SRAM；32 个通用 I/O 口；32 个通用工作寄存器；具有比较模式的、灵活的定时器/计数器；8 通道 10 位 ADC；内外中断源；可编程的 UART；可编程的看门狗定时器；SPI 口以及 3 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而寄存器、定时器/计数器、看门狗和中断系统继续工作；掉电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有外部低电平中断或硬件复位才可以退出此状态。省电模式与掉电模式只有一点差别：省电模式下 T/C2 继续工作以维持时间基准。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 可以通过 SPI 接口或通用编程器多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，AT90S4434/8535 为许多嵌入式控制应用提供了灵活而低成本的方案。

AT90S4434/8535 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和 SL－AVR 编程开发实验器。

5.6.3 AT90S4434 和 AT90S8535 的比较

AT90S4434 具有 4K 字节程序 Flash，256 字节 EEPROM，256 字节 SRAM。
AT90S8535 具有 8K 字节程序 Flash，512 字节 EEPROM，512 字节 SRAM。
表 5.10 列出了 AT90S4434 和 AT90S8535 的比较。

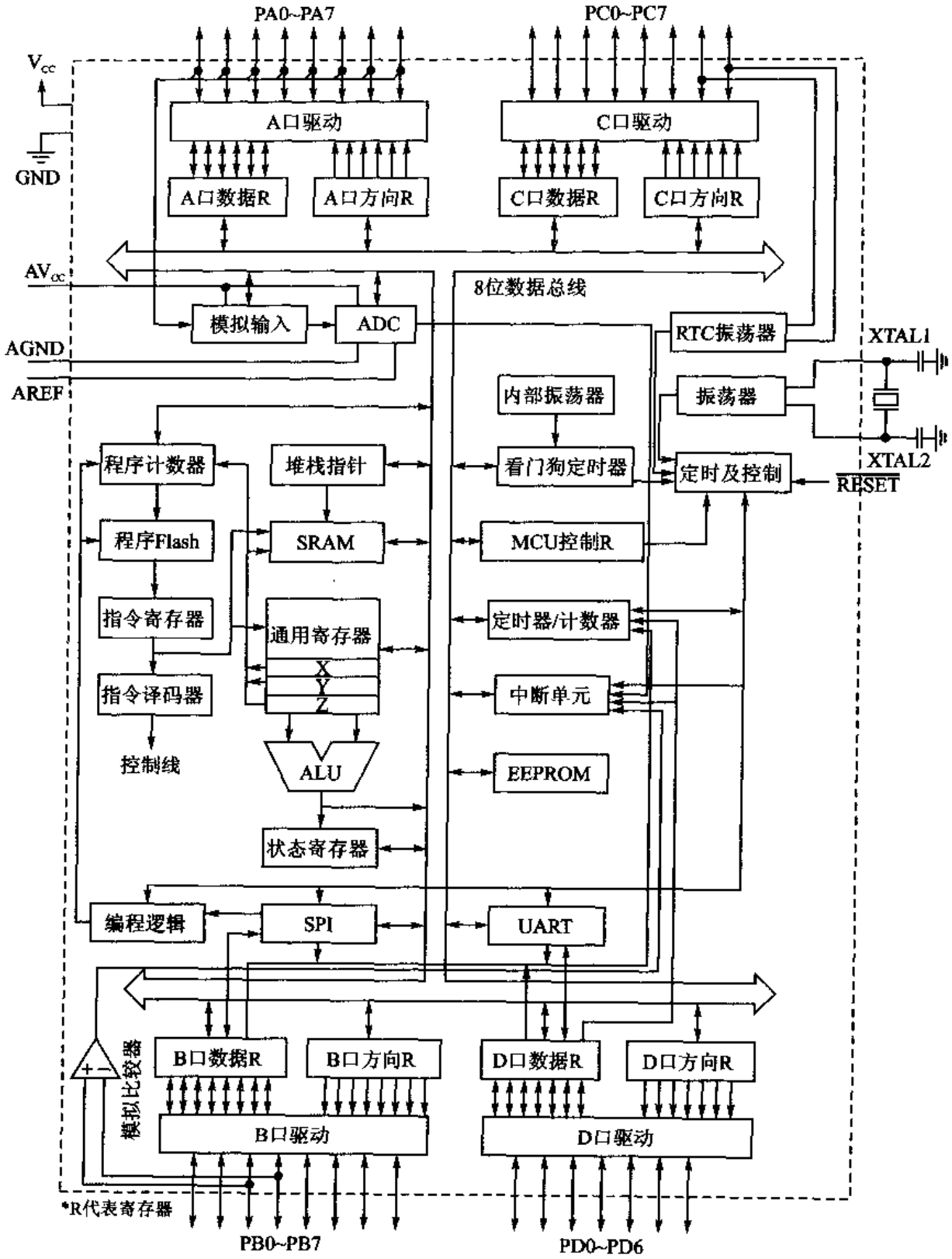


图 5.15 AT90S4434/8535 结构框图

表 5.10 存储器比较

型 号	Flash	EEPROM	SRAM
AT90S4434	4K 字节	256 字节	256 字节
AT90S8535	8K 字节	512 字节	512 字节

5.6.4 引脚配置

AT90S4434/8535 引脚配置如图 5.16 所示。

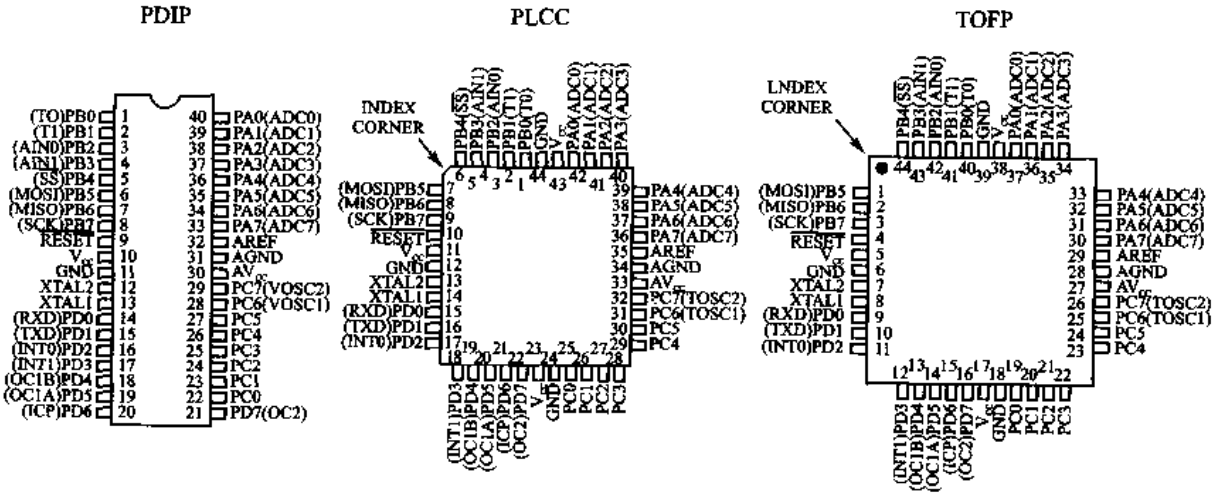


图 5.16 AT90S4434/8535 引脚配置

一、引脚定义

V_{CC}、GND：电源。

A 口(PA7～PA0)：A 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。A 口的输出缓冲器能够吸收 20mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，A 口为三态，即使此时时钟还未起振。A 口还可以用作 ADC 的模拟输入口。

B 口(PB7～PB0)：B 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。B 口的输出缓冲器能够吸收 20mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。B 口作为特殊功能口的使用方法见光盘文件。

C 口(PC7～PC0)：C 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。C 口的 2 个引脚还可以用作 T/C2 的振荡器。在复位过程中，C 口为三态，即使此时时钟还未起振。

D 口(PD7～PD0)：D 口是一个带内部上拉电阻的 8 位双向 I/O 口。输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。D 口作为特殊功能口的使用方法见光盘文件。

$\overline{\text{RESET}}$：复位输入。超过 50ns 的低电平将引起系统复位，低于 50ns 的脉冲不能保证可靠复位。

XTAL1：振荡器放大器的输入端。

XTAL2：振荡器放大器的输出端。

AV_{CC}：A/D 转换器的电源。应该通过一个低通滤波器与 V_{CC} 连接。

AREF：A/D 转换器的参考电源，介于 AGND 与 AV_{CC} 之间。

AGND：模拟地。

二、晶体振荡器

XTAL1 和 XTAL2 分别是片内振荡器的输入、输出端，可使用晶体振荡器或陶瓷振荡器。当使用外部时钟时，XTAL2 应悬空。图 5.17 和图 5.18 分别为振荡器连接图和外部时钟驱动配置图。

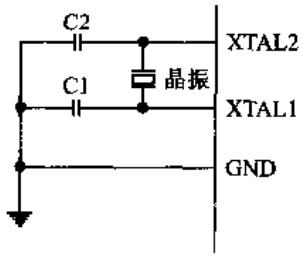


图 5.17　振荡器连接

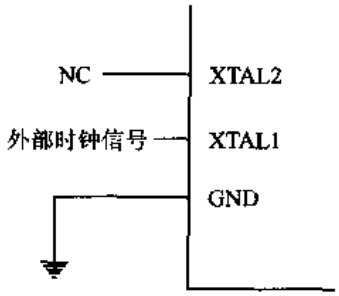


图 5.18　外部时钟驱动配置

三、定时器振荡器

晶振可以直接连接到振荡器的引脚 PC6(TOSC1)和 PC7(TOSC2)上，而无需外部电容。振荡器已经对 32 768Hz 的晶振作了优化。对外加信号的带宽为 256kHz。

5.6.5　AVR RISC 结构

一、AT90S4434/8535 结构

图 5.19 为 AT90S4434/8535 AVR RISC 结构。

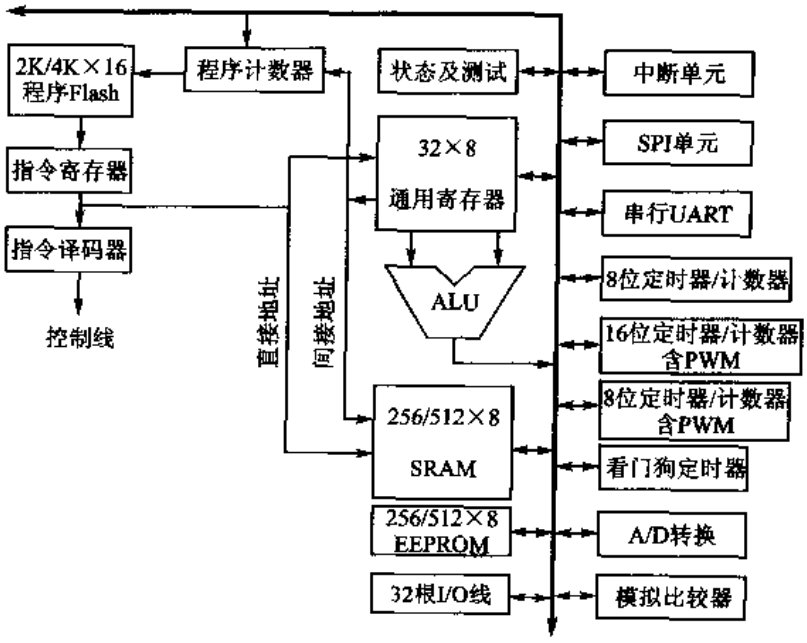


图 5.19　AT90S4434/8535 AVR RISC 结构

快速访问寄存器堆包含 32 个 8 位可单周期访问的通用寄存器。这意味着在一个时钟周期内，ALU 可以完成一次如下操作：读取寄存器堆中的 2 个操作数、执行操作、将结果存回到寄存器堆。

寄存器堆中的 6 个、可以组成 3 个 16 位用于数据寻址的间接寻址寄存器指针，可以提高地址运算能力。其中 Z 指针还用于查表功能。

ALU 支持 2 个寄存器之间、寄存器和常数之间的算术和逻辑操作，以及单寄存器的操作。除寄存器操作模式，通常的内存访问模式也适用于寄存器堆。这是因为 AT90S4434/8535 为寄存器堆分配了 32 个最低的数据空间地址（$00 ～ $1F），允许其像普通内存地址一样访问。

I/O 内存空间包括 64 个地址作为 CPU 外设的控制寄存器，T/C、A/D 转换器以及其他 I/O功能。I/O 内存可以直接访问，也可以作为数据地址（$20～$5F）来访问。

AVR 采用了 HARVARD 结构：程序和数据总线分离。程序内存通过两段式的管道（Pipeline）进行访问：当 CPU 在执行一条指令的同时，就去取下一条指令。这种预取指的概念使得指令可以在一个时钟完成。

相对跳转和相对调用指令可以直接访问 2K/4K 地址空间。多数 AVR 指令都为 16 位长。每个程序内存地址都包含一条 16 位或 32 位的指令。

当执行中断和于程序调用时，返回地址存储于堆栈中。堆栈分布于通用数据 SRAM 之中，堆栈大小只受 SRAM 数量的限制。在复位例程里就应该初始化 SP，SP 为可读写的 16 位堆栈指针。

256/512 个 SRAM 可以通过 5 种不同的寻址方式很容易地进行访问。AVR 结构的内存空间是线性的。

图 5.20 是 AT90S4434/8535 内存映像图。

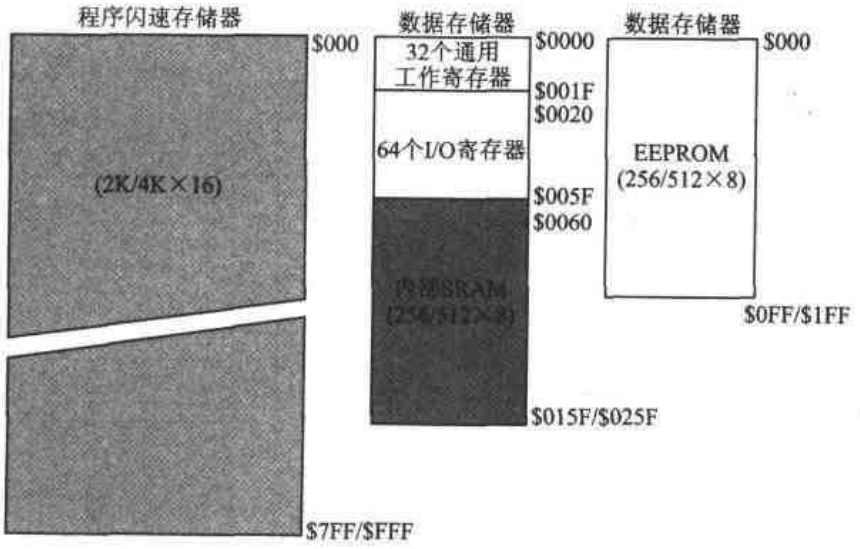


图 5.20 内存映像

中断模块由 I/O 空间中的控制寄存器和状态寄存器中的全局中断触发位组成。每个中断都具有一个中断向量，由中断向量组成的中断向量表位于程序存储区的最前面。中断向量地址低的中断具有高的优先级。

二、通用寄存器

AT90S4434/8535 通用工作寄存器如图 5.22 所示。

所有的寄存器操作指令都可以单指令的形式直接访问所有的寄存器。例外情况为 5 条涉及常数操作的指令：SBCI，SUBI，CPI，ANDI 和 ORI。这些指令只能访问通用寄存器堆的后半部分：R16～R31。

如图 5.21 所示，每个寄存器都有一个数据内存地址，将他们直接映射到数据空间的头 32 个地址。寄存器堆的实现与 SRAM 不同，这种内存组织方式在访问寄存器方面具有极大的灵活性。

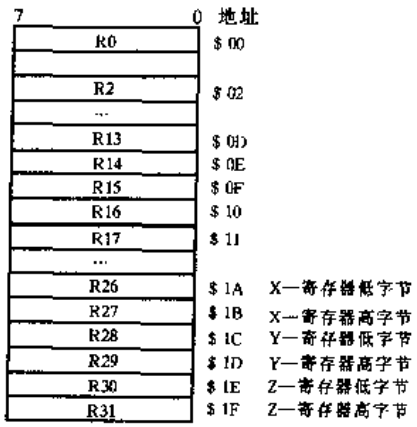


图 5.21　通用寄存器

三、X、Y、Z 寄存器

寄存器 R26～R31 除了用作通用寄存器外，还可以作为数据间接寻址用的地址指针。图 5.22 为 X、Y、Z 寄存器。

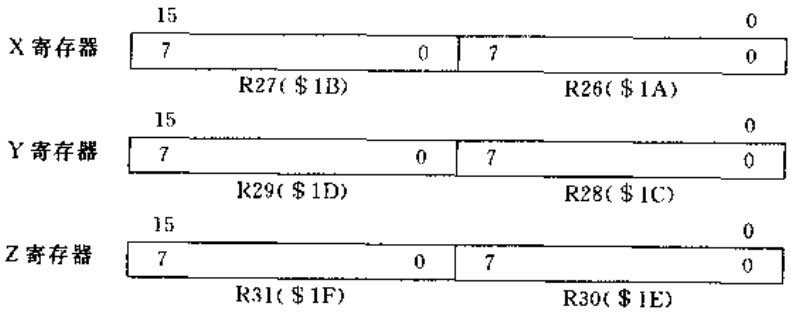


图 5.22　X、Y、Z 寄存器

四、ALU

AVR ALU 与 32 个通用工作寄存器直接相连。ALU 操作分为 3 类：算术、逻辑和位操作。

五、在线可编程 Flash

AT90S4434/8535 具有 4K/8K 字节的 Flash。因为所有的指令为 16 位宽，因此 Flash 结构为 2K×16/4K×16。Flash 的擦除次数至少为 1 000 次。

AT90S4434/8535 的程序计数器(PC)为 11/12 位宽，可以寻址到 2 048/4 096 个字的 Flash 程序区。

六、SRAM

图 5.23 表明了 AT90S4434/8535 的数据组织方式。

352/608 个数据地址用于寻址寄存器堆、I/O 和 SRAM。起始的 96 个地址为寄存器堆 + I/O，其后的 256/512 个地址用于寻址 SRAM。

数据寻址模式分为 5 种：直接、带偏移量的间接、间接、预减的间接和后加的间接。寄存器 R26～R31 为间接寻址的指针寄存器。

直接寻址范围可达整个数据空间；带偏移量的间接寻址模式寻址到 Y、Z 指针给定地址附近的 63 个地址；带预减和后加的间接寻址模式要用到 X、Y、Z 指针。

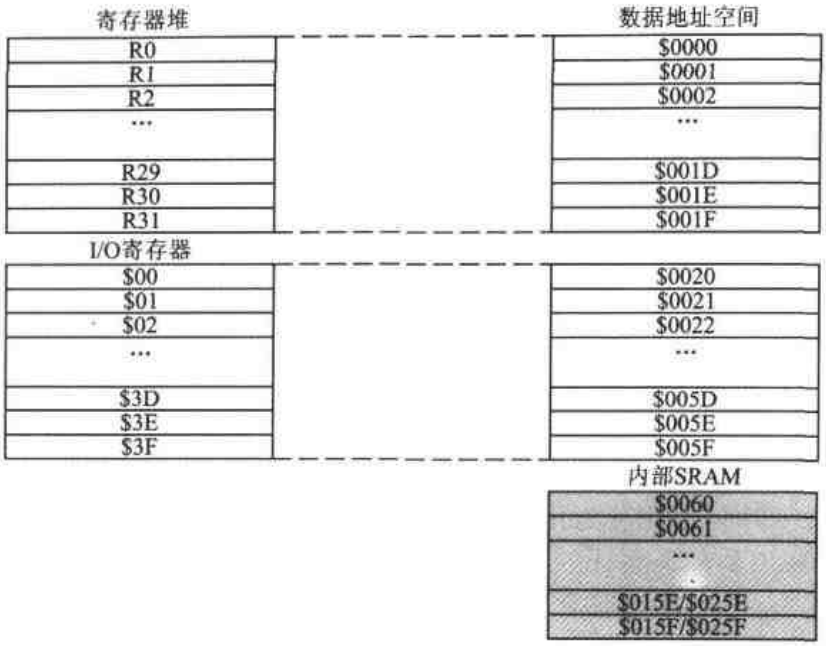


图 5.23 SRAM 分布

32 个通用寄存器、64 个 I/O 寄存器、256/512 字节的 SRAM 和最大可达 64K 的外部 SRAM 可以被所有的寻址模式访问。

七、程序和数据寻址模式

AT90S4434/8535 支持强大而有效的寻址模式：

(1) 单寄存器直接寻址；
(2) 双寄存器直接寻址；
(3) I/O 直接寻址；
(4) 数据直接寻址；
(5) 带偏移的数据间接寻址；
(6) 数据间接寻址；
(7) 带预减的数据间接寻址；
(8) 带后加的数据间接寻址；
(9) 使用 LPM 指令寻址常数；
(10) 程序直接寻址；
(11) 间接程序寻址，IJMP 和 ICALL；
(12) 相对程序寻址，RJMP 和 RCALL。

八、EEPROM

AT90S4434/8535 包含 256/512 字节的 EEPROM。它是作为一个独立的数据空间而存在的，可以按字节读写。EEPROM 的寿命至少为 100 000 次(擦除)。EEPROM 的访问由地址寄存器、数据寄存器和控制寄存器决定。

九、内存访问和指令执行时序

见第二章 2.3.4 存储器访问和指令执行时序。AVR CPU 由系统时钟 Φ 驱动。此时钟由外部晶体直接产生。

十、I/O 内存

表 5.11 所列为 AT90S4434/8535 的 I/O 空间定义。

AT90S4434/8535 的所有 I/O 和外围都被放置在 I/O 空间。IN 和 OUT 指令用来访问不同的 I/O 地址，以及在 32 个通用寄存器之间传输数据。地址为 $00～$1F 的 I/O 寄存器还可用 SBI 和 CBI 指令进行位寻址，而 SIBC 和 SIBS 指令则用来检查单个位置位与否。当使用 IN 和 OUT 指令时地址必须在 $00～$3F 之间。如果要像 SRAM 一样访问 I/O 寄存器，则

相应地址要加上 $20。在本文档里所有 I/O 寄存器的 SRAM 地址写在括号中。

表 5.11 AT90S4434/8535 的 I/O 空间

地 址(十六进制)	名 称	功 能
$3F($5F)	SREG	状态寄存器
$3E($5E)	SPH	堆栈指针高字节
$3D($5D)	SPL	堆栈指针低字节
$3B($5B)	GIMSK	通用中断屏蔽寄存器
$3A($5A)	GIFR	通用中断标志寄存器
$39($59)	TIMSK	T/C 中断屏蔽寄存器
$38($58)	TIFR	T/C 中断标志寄存器
$35($55)	MCUCR	MCU 控制寄存器
$34($54)	MCUSR	MCU 状态寄存器
$33($53)	TCCR0	T/C0 控制寄存器
$32($52)	TCNT0	T/C0(8 位)
$2F($4F)	TCCR1A	T/C1 控制寄存器 A
$2E($4E)	TCCR1B	T/C1 控制寄存器 B
$2D($4D)	TCNT1H	T/C1 高字节
$2C($4C)	TCNT1L	T/C1 低字节
$2B($4B)	OCR1AH	T/C1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	T/C1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	T/C1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	T/C1 输出比较寄存器 B 低字节
$27($47)	ICR1H	T/C1 输入捕捉寄存器高字节
$26($46)	ICR1L	T/C1 输入捕捉寄存器低字节
$25($45)	TCCR2	T/C2 控制寄存器
$24($44)	TCNT2	T/C2(8 位)
$23($43)	OCR2	T/C2 输出比较寄存器
$22($42)	ASSR	异步模式状态寄存器
$21($41)	WDTCR	看门狗控制寄存器
$1F($3F)	EEARH	EEPROM 高地址寄存器
$1E($3E)	EEARL	EEPROM 低地址寄存器
$1D($3D)	EEDR	EEPROM 数据寄存器
$1C($3C)	EECR	EEPROM 控制寄存器

表 5.11(续)

地 址(十六进制)	名 称	功 能
$1B($3B)	PORTA	A 口数据寄存器
$1A($3A)	DDRA	A 口数据方向寄存器
$19($39)	PINA	A 口输入引脚
$18($38)	PORTB	B 口数据寄存器
$17($37)	DDRB	B 口数据方向寄存器
$16($36)	PINB	B 口输入引脚
$15($35)	PORTC	C 口数据寄存器
$14($34)	DDRC	C 口数据方向寄存器
$13($33)	PINC	C 口输入引脚
$12($32)	PORTD	D 口数据寄存器
$11($31)	DDRD	D 口数据方向寄存器
$10($30)	PIND	D 口输入引脚
$0F($2F)	SPDR	SPI 数据寄存器
$0E($2E)	SPSR	SPI 状态寄存器
$0D($2D)	SPCR	SPI 控制寄存器
$0C($2C)	UDR	UART 数据寄存器
$0B($2B)	USR	UART 状态寄存器
$0A($2A)	UCR	UART 控制寄存器
$09($29)	UBRR	UART 波特率寄存器
$08($28)	ACSR	模拟比较器控制及状态寄存器
$07($27)	ADMUX	ADC 多路选择寄存器
$06($26)	ADCSR	ADC 控制和状态寄存器
$05($25)	ADCH	ADC 数据寄存器高字节
$04($24)	ADCL	ADC 数据寄存器低字节

为了与后续产品兼容,保留未用的位应写“0”,而保留的 I/O 寄存器则不应写。

一些状态标志位的清除是通过写“1”来实现的。CBI 和 SBI 指令读取已置位的标志位时,会回写“1”,因此会清除这些标志位。CBI 和 SBI 指令只对 $00～$1F 有效。

I/O 寄存器和外围控制寄存器在后续章节介绍。

● **状态寄存器 SREG(Status Register)**

BIT	7	6	5	4	3	2	1	0
$3F($5F)	I	T	H	S	V	N	Z	C
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

I:全局中断触发。置位时触发全局中断。单独的中断触发由独立控制寄存器控制。如果 I 清零,则不论单独中断标志置位与否,都不会产生中断。I 在复位时清零,RETI 指令执行后

置位。

T：位拷贝存储。位拷贝指令 BLD 和 BST 利用 T 作为目的或源地址。BST 把寄存器的某一位拷贝到 T，而 BLD 把 T 拷贝到寄存器的某一位。

H：半加标志位。

S：符号位。总是 N 与 V 的异或。

V：二进制补码溢出标志位。

N：负数标志位。

Z：零标志位。

C：进位标志位。

状态寄存器在进入中断和退出中断时并不自动进行存储和恢复。这项工作由软件完成。

● **堆栈指针 SP**

BIT	15	14	13	12	11	10	9	8
$3E($5E)	--	-	—		—	—	SP9	SP8
$3D($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
	7	6	5	4	3	2	1	0
读/写	R	R	R	R	R	R	R/W	R/W
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

堆栈指针指向位于 SRAM 的函数及中断堆栈。堆栈空间必须在调用函数或中断触发之前定义。指针必须指向高于 $60 的地址。用 PUSH 指令推数据入栈时，堆栈指针将减 1；而当调用函数或中断时，指针将减 2。使用 POP 指令时，堆栈指针将加 1；而用 RET 或 RETI 指令返回时，指针将加 2。

十一、复位和中断处理

1. 复位源

AT90S4434/8535 有 16 个中断源。每个中断源在程序空间都有一个独立的中断向量。所有的中断事件都有自己的触发位。当触发位置位，且 I 也置位的情况下，中断可以发生。

器件复位后，程序空间的最低位置自动定义为复位及中断向量。完整的中断表见表5.12。在中断向量表中处于低地址的中断具有高的优先级。所以，RESET 具有最高的优先级。

表 5.12　复位与中断向量

向量号	程序地址	来　源	定　义
1	$000	RESET	硬件引脚，上电复位和看门狗复位
2	$001	INT0	外部中断 0
3	$002	INT1	外部中断 1
4	$003	TIMER2 COMP	T/C2 比较匹配
5	$004	TIMER2 OVF	T/C2 溢出
6	$005	TIMER1 CAPT	T/C1 捕捉事件

续表 5.12

向量号	程序地址	来 源	定 义
7	$006	TIMER1 COMPA	T/C1 比较匹配 A
8	$007	TIMER1 COMPB	T/C1 比较匹配 B
9	$008	TIMER1 OVF	T/C1 溢出
10	$009	TIMER0 OVF	T/C0 溢出
11	$00A	ASPI,STC	串行传输结束
12	$00B	UART,RX	UART 接收结束
13	$00C	UART,UDRE	UART 数据寄存器空
14	$00D	UART,TX	UART 发送结束
15	$00E	ADC	ADC 转换结束
16	$00F	EE_RDY	EEPROM 准备好
17	$010	ANA_COMP	模拟比较器

设置中断向量地址最典型的方法如下：

```
地址        标号        代码                                    注释
$000                    RJMP    RESET                           ;复位
$001                    RJMP    EXT_INT0                        ;IRQ0
$002                    RJMP    EXT_INT1                        ;IRQ1
$003                    RJMP    TIM2_COMP                       ;T2 比较匹配
$004                    RJMP    TIM2_OVF                        ;T2 溢出
$005                    RJMP    TIM1_CAPT                       ;T1 捕捉
$006                    RJMP    TIM1_COMPA                      ;T1 比较 A 匹配
$007                    RJMP    TIM1_COMPB                      ;T1 比较 B 匹配
$008                    RJMP    TIM1_OVF                        ;T1 溢出
$009                    RJMP    TIM0_OVF                        ;T0 溢出
$00a                    RJMP    SPI_STC                         ;SPI 传输结束
$00b                    RJMP    UART_RXC                        ;UART 接收结束
$00c                    RJMP    UART_DRE                        ;UART 数据空
$00d                    RJMP    UART_TXC                        ;UART 发送结束
$00e                    RJMP    ADC                             ;AD 转换结束
$00f                    RJMP    EE_RDY                          ;EEP 准备好
$010                    RJMP    ANA_COMP                        ;模拟比较器
;
$011        MAIN:       LDI     R16,HIGH(REMEND)                ;主程序开始
$012                    OUT     SPH, R16
$013                    LDI     R16,LOW(REMEND)
$014                    RJMP
$015                    <指令>  XXX
—           —           —       —                               —
```

AT90S4434/8535 有 3 个复位源：

- 上电复位。当电源电压低于上电门限 V_{POT} 时，MCU 复位。
- 外部复位。当$\overline{\text{RESET}}$引脚上的低电平超过 50ns 时，MCU 复位。
- 看门狗复位。看门狗定时器超时后，MCU 复位。

2. 中断处理

AT90S4434/8535 有 2 个中断屏蔽控制寄存器 GIMSK——通用中断屏蔽寄存器和 TIMSK——T/C 中断屏蔽寄存器。

一个中断产生后，全局中断触发位 I 将被清零，后续中断被屏蔽。可以在中断例程里对 I 置位，从而开放中断。执行 RETI 后，I 重新置位。

当程序计数器指向实际中断向量开始执行相应的中断例程时，硬件清除对应的中断标志。一些中断标志位也可以通过软件写“1”来清除。

当一个符合条件的中断发生后，如果相应的中断触发位为“0”，则中断标志位挂起，并一直保持到中断执行，或者被软件清除。

如果全局中断标志被清零，则所有的中断都不会被执行，直到 I 置位。然后被挂起的各个中断按中断优先级依次中断。

注意：外部电平中断没有中断标志位，因此当电平变为非中断电平后，中断条件即终止。

● 通用中断屏蔽寄存器——GIMSK

BIT	7	6	5	4	3	2	1	0
$3B($5B)	INT1	INT0	—	—	—	—	—	—
读/写	R/W	R/W	R	R	R	R	R	R
初始值	0	0	0	0	0	0	0	0

位 5…0：保留。

INT1：外部中断 1 请求触发。当 INT0 和 I 都为“1”时，外部引脚中断触发。MCU 通用控制寄存器(MCUCR)中的中断检测控制位 1/0(ISC11 和 ISC10)定义中断 1 是上升沿中断还是下降沿中断，或者是低电平中断。即使引脚被定义为输出，中断仍可产生。

INT0：外部中断 0 请求触发。当 INT0 和 I 都为“1”时，外部引脚中断触发。MCU 通用控制寄存器(MCUCR)中的中断检测控制位 1/0(ISC01 和 ISC00)定义中断 0 是上升沿中断还是下降沿中断，或者是低电平中断。即使引脚被定义为输出，中断仍可产生。

● 通用中断标志寄存器——GIFR

BIT	7	6	5	4	3	2	1	0
$3A($5A)	INTF1	INTF0	—	—	—	—	—	—
读/写	R/W	R/W	R	R	R	R	R	R
初始值	0	0	0	0	0	0	0	0

位 5…0：保留。

INTF1：外部中断标志 1。当 INT1 引脚有事件触发中断请求时，INTF1 置位(“1”)。如果 SREG 中的 I 及 GIMSK 中的 INT1 都为“1”，则 MCU 将跳转到中断地址 $002。中断例程执行后，此标志被清除。另外，标志也可以通过对其写“1”来清除。

INTF0：外部中断标志 0。当 INT0 引脚有事件触发中断请求时，INTF0 置位(“1”)。如果 SREG 中的 I 及 GIMSK 中的 INT0 都为“1”，则 MCU 将跳转到中断地址 $001。中断例程执行后，此标志被清除。另外，标志也可以通过对其写“1”来清除。

● **T/C 中断屏蔽寄存器——TIMSK**

BIT	7	6	5	4	3	2	1	0
$39($59)	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	—	TOIE0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
初始值	0	0	0	0	0	0	0	0

位 1：保留。

OCIE2：T/C2 输出比较匹配中断触发。当 TOIE2 和 I 都为“1”时，输出比较匹配中断触发。当 T/C2 的比较匹配发生，或 TIFR 中的 OCF2 置位，中断例程（$003）将执行。

TOIE2：T/C2 溢出中断触发。当 TOIE2 和 I 都为“1”时，T/C2 溢出中断触发。当 T/C2 溢出，或 TIFR 中的 TOV2 位置位时，中断例程（$004）得到执行。

TICIE1：T/C1 输入捕捉中断触发。当 TICIE1 和 I 都为“1”时，输入捕捉中断触发。当 T/C1 的输入捕捉事件发生（ICP），或 TIFR 中的 ICF1 置位，中断例程（$005）将执行。

OCIE1A：T/C1 输出比较 A 匹配中断触发。当 TOIE1A 和 I 都为“1”时，输出比较 A 匹配中断触发。当 T/C1 的比较 A 匹配发生，或 TIFR 中的 OCF1A 置位，中断例程（$006）将执行。

OCIE1B：T/C1 输出比较 B 匹配中断触发。当 TOIE1B 和 I 都为“1”时，输出比较 B 匹配中断触发。当 T/C1 的比较 B 匹配发生，或 TIFR 中的 OCF1B 置位，中断例程（$007）将执行。

TOIE1：T/C1 溢出中断触发。当 TOIE1 和 I 都为“1”时，T/C1 溢出中断触发。当 T/C1 溢出，或 TIFR 中的 TOV1 位置位时，中断例程（$008）得到执行。

TOIE0：T/C0 溢出中断触发。当 TOIE0 和 I 都为“1”时，T/C0 溢出中断触发。当 T/C0 溢出，或 TIFR 中的 TOV0 位置位时，中断例程（$009）得到执行。

● **T/C 中断标志寄存器——TIFR**

BIT	7	6	5	4	3	2	1	0
$38($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
初始值	0	0	0	0	0	0	0	0

位 1：保留。

OCF2：T/C2 输出比较标志。当 T/C2 与 OCR2 的值匹配时，OCF2 置位。此位在中断例程里硬件清零，或者通过对其写“1”来清零。当 SREG 中的位 I、OCIE2 和 OCF2 一同置位时，中断例程得到执行。

TOV2：T/C2 溢出中断标志位。当 T/C2 溢出时，TOV2 置位。执行相应的中断例程后此位硬件清零。此外，TOV2 也可以通过写“1”来清零。当 SREG 中的位 I、TOIE2 和 TOV2 一同置位时，中断例程得到执行。在 PWM 模式中，当 T/C2 在 $0000 改变记数方向时，TOV2 置位。

ICF1：输入捕捉标志位。当输入捕捉事件发生时，ICF1 置位，表明 T/C1 的值已经送到输入捕捉寄存器 ICR1。此位在中断例程里硬件清零，或者通过对其写“1”来清零。当 SREG 中的位 I、TICIE1A 和 ICF1 一同置位时，中断例程得到执行。

OCF1A：输出比较标志 1A。当 T/C1 与 OCR1A 的值匹配时，OCF1A 置位。此位在中断例程里硬件清零，或者通过对其写“1”来清零。当 SREG 中的位 I、OCIE1A 和 OCF1A 一同置

位时，中断例程得到执行。

OCF1B：输出比较标志 1B。当 T/C1 与 OCR1B 的值匹配时，OCF1B 置位。此位在中断例程里硬件清零，或者通过对其写“1”来清零。当 SREG 中的位 I、OCIE1B 和 OCF1B 一同置位时，中断例程得到执行。

TOV1：T/C1 溢出中断标志位。当 T/C1 溢出时，TOV1 置位。执行相应的中断例程后此位硬件清零。此外，TOV1 也可以通过写“1”来清零。当 SREG 中的位 I、TOIE1 和 TOV1 一同置位时，中断例程得到执行。在 PWM 模式中，当 T/C1 在 $0000 改变记数方向时，TOV1 置位。

TOV0：T/C0 溢出中断标志位。当 T/C0 溢出时，TOV0 置位。执行相应的中断例程后此位硬件清零。此外，TOV0 也可以通过写“1”来清零。当 SREG 中的位 I、TOIE0 和 TOV0 一同置位时，中断例程得到执行。

3. 外部中断

外部中断由 INT0 和 INT1 引脚触发。应当注意，如果中断触发，则即使 INT0/INT1 配置为输出，中断照样会被触发。此特点提供了一个产生软件中断的方法。触发方式可以为上升沿、下降沿或低电平。这些设置由 MCU 控制寄存器 MCUCR 决定。当设置为低电平触发时，只要电平为低，中断就一直触发。

4. 中断响应时间

AVR 中断响应时间最少为 4 个时钟周期。在这 4 个时钟期间，PC(2 个字节)自动入栈，而 SP 减 2。在通常情况下，中断向量为一个相对跳转指令，此跳转要花 2 个时钟周期。如果中断在一个多周期指令执行期间发生，则在此一个多周期指令执行完后，MCU 才会执行中断程序。

中断返回亦需 4 个时钟。在此期间，PC 将被弹出栈，SREG 的位 I 被置位。如果在中断期间发生了其他中断，则 AVR 在退出中断程序后，要执行一条主程序指令之后才能再响应被挂起的中断。

要注意 AVR 硬件在中断或子程序中并不操作状态寄存器——SREG。SREG 的存储由软件完成。

对于那些由可以保持为静态的事件(如输出比较寄存器 1 与 T/C1 值相匹配)驱动的中断，事件发生后中断标志将置位。如果中断标志被清除而中断条件仍然存在，则标志只有在新事件发生后才会置位。外部电平中断会一直保持到中断条件结束。

● **MCU 控制寄存器——MCUCR**

BIT	7	6	5	4	3	2	1	0
$35($55)		SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

位 7：保留。

SE：休眠触发。执行 SLEEP 指令时，SE 必须置位才能使 MCU 进入休眠模式。为了防止无意间使 MCU 进入休眠，建议与 SLEEP 指令相连使用。

SM1/0：休眠模式。此位用于选择休眠模式，如表 5.13 所列。

ISC11，ISC10：中断检测控制 1，位 1 和位 0。选择 INT1 中断的边沿或电平，如表 5.14 所列。

ISC01,ISC00:中断检测控制 0,位 1 和位 0。选择 INT0 中断的边沿或电平,如表 5.15 所列。

表 5.13 休眠模式选择

SM1	SM0	休眠模式
0	0	空闲
0	1	保留
1	0	掉电
1	1	省电

表 5.14 中断 1 检测控制

ISC11	ISC10	描 述
0	0	低电平中断
0	1	保留
1	0	下降沿中断
1	1	上升沿中断

注意:改变 ISC11/ISC10 时,首先要禁止 INT1(清除 GIMSK 的 INT1 位),否则可能引发不必要的中断。

表 5.15 中断 0 检测控制

ISC01	ISC00	描 述
0	0	低电平中断
0	1	保留
1	0	下降沿中断
1	1	上升沿中断

注意:改变 ISC01/ISC00 时,首先要禁止 INT0(清除 GIMSK 的 INT0 位),否则可能引发不必要的中断。

INTn 引脚的电平在检测边沿之前采样。如果边沿中断触发,则大于一个 MCU 时钟的脉冲将触发中断。如果选择了低电平触发,则此电平必须保持到当前执行的指令结束。

十二、省电方式

见第二章 2.6 AVR 单片机省电方式。有休眠模式,空闲模式,掉电模式和省电模式。

5.6.6 定时器/计数器

8 位 T/C0 和 16 位 T/C1 见第二章 2.7 AVR 单片机定时器/计数器。

一、8 位 T/C2

图 5.24 为 T/C2 的框图。

T/C2 的时钟可以选择 PCK2 或预分频的 PCK2。另外还可以由 T/C2 控制寄存器 TCCR2来停止它。TIFR 为状态标志寄存器,TCCR2 为控制寄存器,而 TIMSK 控制 T/C2 的中断屏蔽。在低预分频条件下,T/C2 具有高分辨率和高精度的特点;而在高预分频条件下,T/C2 非常适用于低速功能,如计时。

利用输出比较寄存器 OCR2 作为数据源,T/C2 还可以实现输出比较的功能。此功能包括比较匹配发生时清除计数器和比较输出引脚 PD7(OC2)的动作。

T/C2 还可以用作 8 位 PWM 调制器。在此模式下,计数器和 OCR2 寄存器用于无尖峰干扰的、中心对称的 PWM。

● **T/C0 控制寄存器——TCCR2**

BIT	7	6	5	4	3	2	1	0
$25($45)	—	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20
读/写	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

位 7:保留。

PWM2:PWM 触发。

COM21/20:比较匹配模式位 1/0。见表 5.16 所列。

表 5.16 比较模式选择

COM21	COM20	OC2
0	0	不用作 PWM 功能
0	1	OC2 输出变换
1	0	OC2 清零
1	1	OC2 置位

CTC2:比较匹配时清除 T/C2。CTC2 为"1"时,比较匹配事件发生后,T/C2 将复位为 0;若 CTC2 为"0",则 T/C2将连续记数而不受比较匹配的影响。由于比较匹配事件的检测发生在匹配发生之后的一个 CPU 时钟,因此

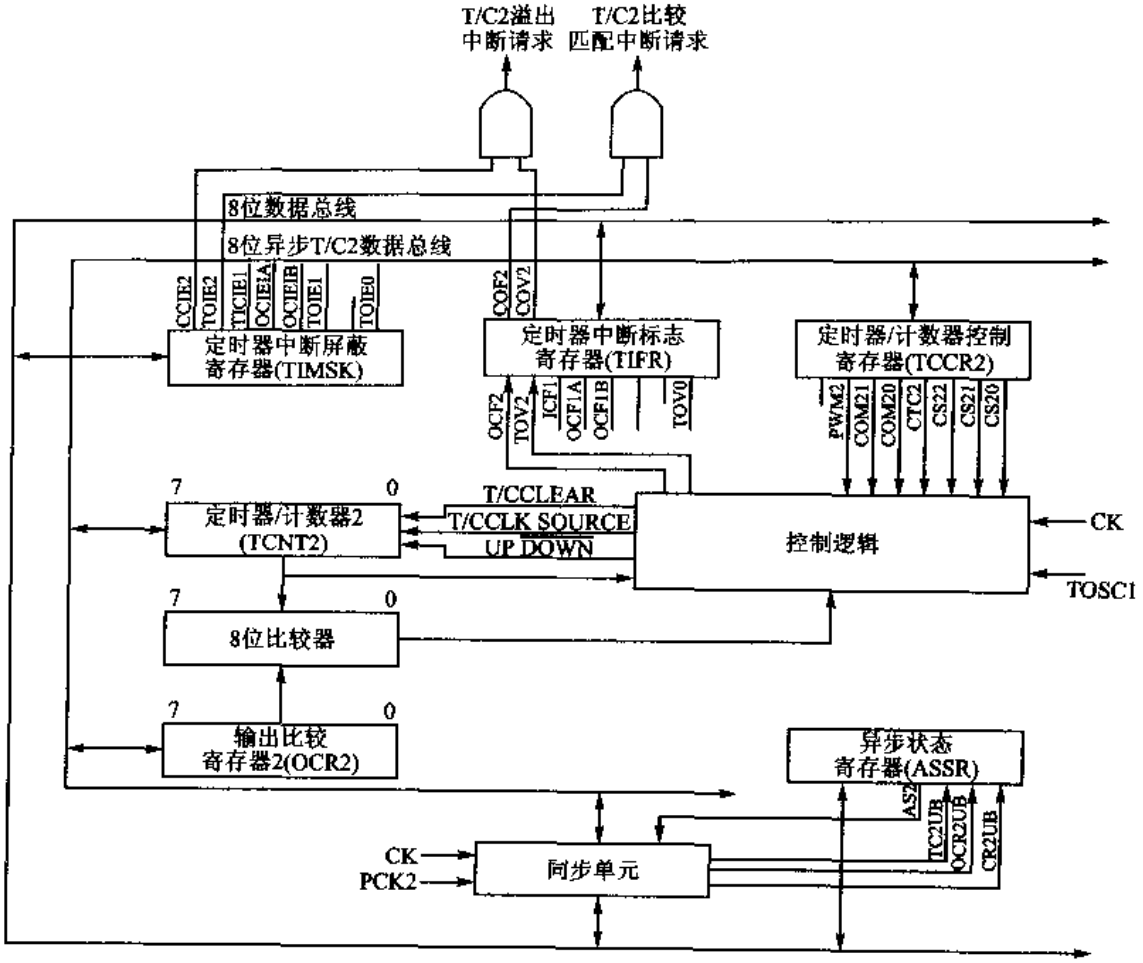


图 5.24 T/C2 工作框图

定时器的预分频比率的不同将引起此功能有不同的表现。当预分频为 1,比较匹配寄存器的值设置为 C 时,定时器的记数方式为:

…| C−2 | C−1 | C | 0 | 1 |…

而当预分频为 8 时,定时器的记数方式则为:

…| C−2, C−2, C−2, C−2, C−2, C−2, C−2, C−2 | C−1, C−1, C−1, C−1, C−1, C−1, C−1, C−1 | C, 0, 0, 0, 0, 0, 0, 0 |…

在 PWM 模式下,这一位没有作用。

CS02、CS01、CS00:时钟选择。表 5.17 为 T/C2 预分频选择。

表 5.17 T/C2 预分频选择

CS02	CS01	CS00	描 述	CS02	CS01	CS00	描 述
0	0	0	停止	1	0	0	PCK2/64
0	0	1	PCK2	1	0	1	PCK2/128
0	1	0	PCK2/8	1	1	0	PCK2/256
0	1	1	PCK2/32	1	1	1	PCK2/1 024

停止条件提供了一个定时器触发/禁止的功能。预分频的 PCK2 直接由时钟振荡器分频而来。

● **T/C2——TCNT2**

BIT	7	6	5	4	3	2	1	0
$24($44)	MSB							LSB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

T/C2 是可以进行读/写访问的向上计数器。只要有时钟输入，T/C2 就会在写入的值基础上向上记数。

● **T/C2 输出比较寄存器——OCR2**

BIT	7	6	5	4	3	2	1	0
$23($43)	MSB							LSB
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

T/C2 输出比较寄存器包含与 T/C2 值连续比较的数据。如果 T/C2 的值与 OCR2 相等，则比较匹配发生，结果由 TCCR2 决定。用软件写操作将 TCNT2 和 OCR2 设置为相等不会引发比较匹配。匹配发生后匹配中断标志置位。

● PWM 模式下的 T/C2：

选择 PWM 模式后，T/C2 和输出比较寄存器 OCR2 共同组成一个 8 位的、无尖峰的、自由运行的 PWM。T/C2 作为上/下计数器，从 0 记数到 TOP，然后反向记数回到 0。当计数器中的数值和 OCR2 的数值一致时，OCR2 引脚按照 COM21/COM20 的设置动作。表 5.18 为 PWM 模式下的比较模式选择。

表 5.18 PWM 模式下的比较模式选择

COM21	COM20	OC2
0	0	不用作 PWM 功能
0	1	不用作 PWM 功能
1	0	向上记数时的匹配清除 OC2；而向下记数时的匹配置位 OC2(正向 PWM)
1	1	向下记数时的匹配清除 OC2；而向上记数时的匹配置位 OC2(反向 PWM)

注意：在 PWM 模式下，OCR2 首先存储在一个临时的位置，等到 T/C2 达到 TOP 时才真正存入 OCR2。这样可以防止在写 OCR2 时，由于失步而出现奇数长度的 PWM 脉冲。图 5.25为失步的 OCR2 锁存。

如果在执行写和锁存操作的时候读取 OCR2，读到的是临时位置的数据。OCR2 的值为 $0000 或 TOP 时 OC1 的输出见表 5.19。

在 PWM 模式下，当计数器达到 $00 时将置位 TOV2。此时发生的中断与正常情况下的中断是完全一样的。

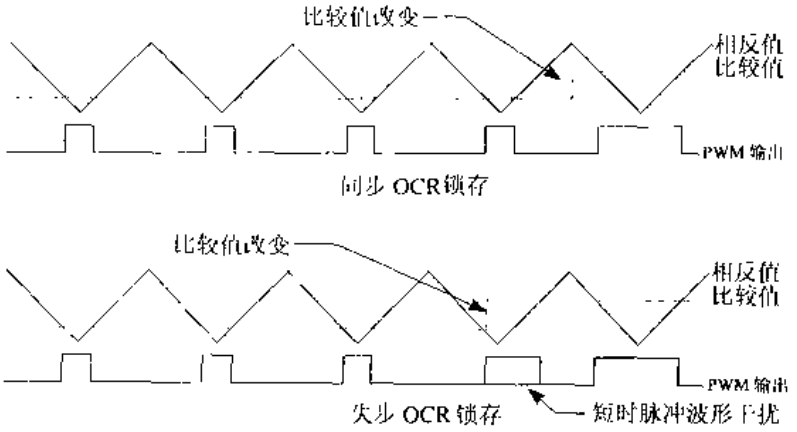


图 5.25　失步的 OCR2 锁存

表 5.19　OCR2= $0000 或 TOP 时的 PWM 输出

COM21	COM20	OCR2	输出
1	0	$0000	L
1	0	TOP	H
1	1	$0000	H
1	1	TOP	L

● **异步模式状态寄存器——ASSR**

BIT	7	6	5	4	3	2	1	0
$22($42)	—	—		-	AS2	TCN2UB	OCR2UB	TCR2UB
读/写	R	R	R	R	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

位 7…4：保留

AS2：异步 T/C2。当 AS2 置位时，T/C2 由 TOSC1 驱动。PC6 和 PC7 连接到晶体振荡器，不能用作普通 I/O。若 AS2 为“0”，则 T/C2 由内部系统时钟驱动。这一位变化时有可能 TCNT2，OCR2 和 TCCR2 的数据破坏。

TCN2UB：T/C2 更新忙。T/C2 工作于异步模式时，写 TCNT2 将引起 TCN2UB 置位。当 TCNT2 从暂存寄存器更新完毕后，TCN2UB 由硬件清零。TCN2UB 为“0”，表明 TCNT2 可以写入新值。

OCR2UB：输出比较寄存器 2 更新忙。T/C2 工作于异步模式时，写 OCR2 将引起 OCR2UB 置位。当 OCR2 从暂存寄存器更新完毕后，OCR2UB 由硬件清零。OCR2UB 为“0”，表明 OCR2 可以写入新值。

TCR2UB：T/C 控制寄存器 2 更新忙。T/C2 工作于异步模式时，写 TCCR2 将引起 TCR2UB 置位。当 TCCR2 从暂存寄存器更新完毕后，TCR2UB 由硬件清零。TCR2UB 为“0”，表明 TCCR2 可以写入新值。

如果在更新忙标志置位的时候写上述任何一个寄存器，都将引起数据的破坏，并引发不必要的中断。

对 TCNT2，OCR2 和 TCCR2 进行读取的机制是不同的。读到的 TCNT2 为实际的值；而 OCR2 和 TCCR2 则是从暂存寄存器中读取的。

二、T/C2 的异步操作

T/C2 异步工作时要考虑到，在同步和异步模式之间的转换有可能造成 TCNT2、OCR2、TCCR2 数据的损毁。安全的步骤应该是：

(1) 关闭 T/C2 的中断 OCIE2 和 TOIE2；

(2) 设置 AS2 以选择合适的时钟源；

(3) TCNT2，OCR2 和 TCCR2 写入新的数值；

(4) 等待 TCN2UB，OCR2UB 和 TCR2UB 清零；

(5) 必要的话，开启中断。

● 振荡器对 32 768Hz 的晶振进行了优化，其对外部输入时钟信号的带宽为 256kHz。因此对外部输入的时钟信号不能高于 256kHz。另外，此信号还不能高于系统主时钟的 1/4。

● 写 TCNT2、OCR2 和 TCCR2 时，数据首先传到暂存寄存器，2 个 TOSC1 正跳变后才锁存。数据从暂存寄存器写入目的寄存器之前不能写入新的数值。3 个寄存器具有各自独立的暂存寄存器，因此写 TCNT2 不会干扰写 OCR2。可以通过 ASSR 检查数据是否已经写入到目的寄存器。

● 如果要用 T/C2 作为 MCU 的唤醒条件，则在 TCNT2、OCR2 和 TCCR2 更新结束之前不能进入省电模式；否则，MCU 可能会在 T/C2 设置生效之前进入休眠模式。这对于用 T/C2 的比较匹配中断唤醒 MCU 尤其重要。因为在更新 OCR2 或 TCNT2 时比较匹配是禁止的。如果在更新过程中 MCU 进入休眠模式，则比较匹配中断永远不会发生。

● 如果要用 T/C2 作为省电模式的唤醒条件，必须注意重新进入省电模式的过程。中断逻辑需要一个 TOSC1 周期进行复位。如果从唤醒到重新进入休眠的时间小于一个 TOSC1 周期，中断将不再发生，器件再也无法唤醒。如果怀疑程序是否满足这一条件，可以采取如下方法：

① 对 TCNT2，OCR2 和 TCCR2 写入一个合适的值；

② 等待更新忙标志变低；

③ 进入省电模式。

● 若选择了异步工作模式，T/C2 的振荡器将一直工作，除非进入掉电模式。应该注意，此振荡器的稳定时间可能长达 1s。因此，建议在器件从掉电模式唤醒或上电时，至少等待 1s 后再使用 T/C2。

● 省电模式唤醒过程：中断条件满足后，在下一个定时器时钟里唤醒过程启动。在 MCU 时钟启动后的 3 个周期，中断标志置位。在此期间，MCU 执行其他指令，但中断条件还不可读，中断例程也不会执行。

● 在异步模式下，中断标志的同步需要 3 个处理器周期加一个定时器周期。输出比较引脚的变化与定时器时钟同步，而不与处理器时钟同步。

5.6.7 看门狗定时器

见第二章2.7.4看门狗定时器。看门狗定时器由片内独立的振荡器驱动。

5.6.8 EEPROM读/写

EEPROM访问寄存器位于I/O空间。

写EEP的时间与电压有关，为2.5～4ms。自定时功能可以监测何时开始写下一字节。操作EEPROM，应当注意如下问题：在电源滤波时间常数比较大的电路中，上电/下电时，V_{cc}上升/下降会比较慢。此时MCU将工作于低于晶振所要求的电源电压。在这种情况下，程序指针有可能跑飞，并执行EEP写指令。为了保证EEP数据的完整性，建议使用电压复位电路。

5.6.9 串行外设接口SPI

串行外设接口SPI允许AT90S4434/8535和外设之间进行高速的同步数据传输。AT90S4434/8535 SPI的特点如下：

- 全双工，3线同步数据传输；
- 主从操作；
- LSB在先或MSB在先；
- 4种可编程的比特率；
- 传输结束中断；
- 写碰撞标志检测；
- 可以从空闲模式唤醒(作为从机工作时)。

5.6.10 通用串行接口UART

AT90S4434/8535具有全双工通用异步收发器。其主要特点为：

- 波特率发生器可以产生大量的波特率(bps)；
- 在低时钟下仍然可以得到高的波特率；
- 8或9位数据；
- 噪声滤波；
- 过速检测；
- 帧错误检测；
- 错误起始位检测；
- 3个独立的中断：发送结束、发送数据寄存器空、接收结束。

5.6.11 模拟比较器

模拟比较器比较正输入端PB2(AIN0)和负输入端PB3(AIN1)的值。如果PB2(AIN0)的电压值高于PB3(AIN1)的电压值，比较器的输出ACO将置位。此输出可用来触发模拟比较器中断(上升沿、下降沿或电平变换)，也可以触发T/C1的输入捕捉功能。

5.6.12 模数转换器

AT90S8535的10位A/D转换应用程序实例见第七章“7.4.1 10位A/D转换”及“7.4.8 AT90S8535的A/D转换”。特点为：

- 10位精度；
- ±2LSB精确度；
- 0.5LSB集成非线性度；
- 65～260μs转换时间；
- 8通道；
- 轨到轨输入范围(Rail-to-Rail Inpvt Range)；
- 自由运行模式和单次转换模式；
- ADC转换结束中断；
- 休眠模式噪声消除。

AT90S4434/8535具有10位精度的逐次逼近型A/D转换器。ADC与一个8通道的模拟多路器相连，这样就允许A口作为ADC的输入引脚。ADC包含一个采保放大器。ADC框图见图5.26。

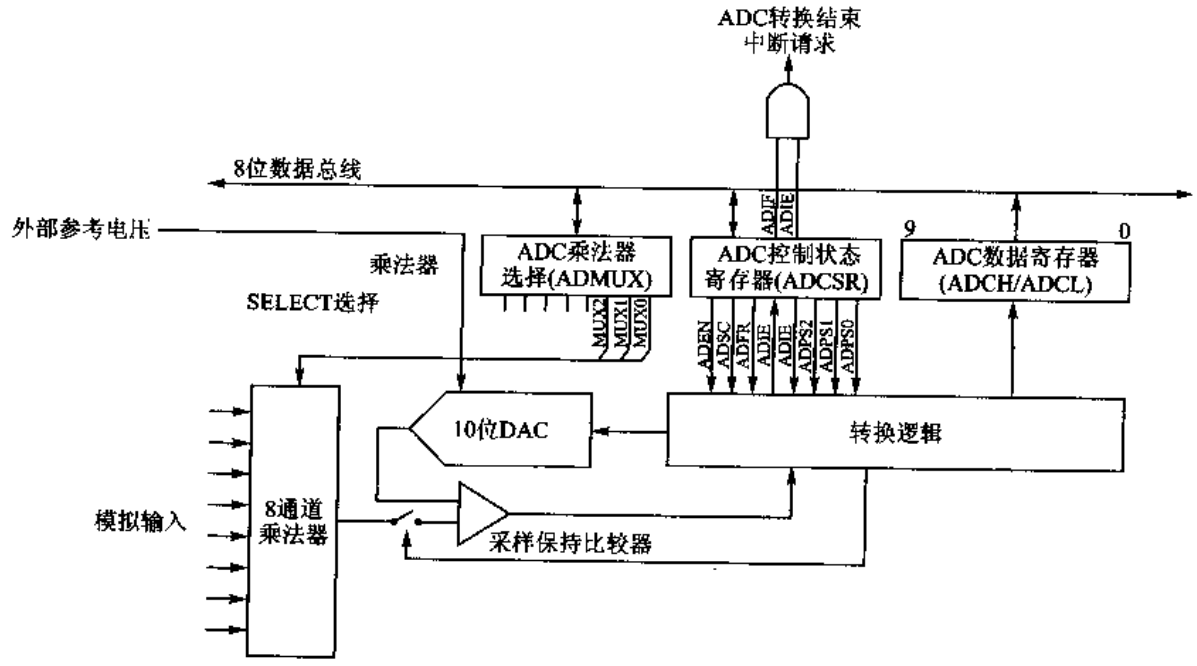


图5.26 ADC框图

ADC具有2个电源引脚AV_{CC}和AGND。AGND必须与GND相连，AV_{CC}与V_{CC}的差别不能大于±0.3V。

AREF为外部参考电压输入端。此电压介于AGND～AV_{CC}。

一、操 作

ADC可以工作于两种模式——单次转换及自由运行。在单次转换模式下，用户必须启动每一次转换；而在自由运行模式下，ADC会连续采样并更新ADC数据寄存器。ADCSR的ADFR位用于选择模式。

ADC 由 ADCSR 的 ADEN 位控制触发。触发 ADC 后，第一次转换将引发一次哑转换过程，以初始化 ADC，然后才真正进行 A/D 转换。对用户而言，此次转换过程比其他转换过程要多 12 个 ADC 时钟周期。

ADSC 置位将启动 A/D 转换。在转换过程中 ADSC 一直保持为高；转换结束后 ADSC 硬件清零。如果在转换过程中通道改变了，ADC 首先要完成当前的转换，然后通道才会改变。

ADC 产生 10 位的结果，ADCH 和 ADCL。为了保证正确读取数据，系统采用了如下保护逻辑：读数据时，首先要读 ADCL。一旦开始读 ADCL，ADC 对数据寄存器的访问就被禁止了。也就是说，如果读取了 ADCL，那么即使在读 ADCH 之前另一次 ADC 结束了，2 个寄存器的值也不会被新的 ADC 结果更新，此次转换的数据将丢失。当读完 ADCH 之后，ADC 才能继续对 ADCH 和 ADCL 进行访问。

ADC 结束后会置位 ADIF。即使发生如上所说的，由于 ADCH 未被读取而丢失转换数据的情况，ADC 结束中断仍将触发。

二、预分频器

ADC 预分频器如图 5.27 所示。

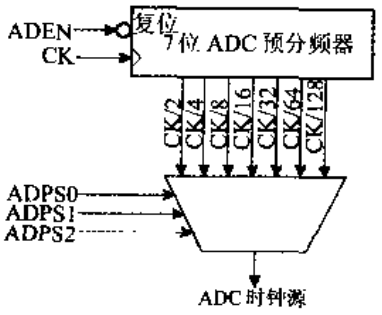


图 5.27 ADC 预分频器

ADC 有一个预分频器，可以将系统时钟调整到可接受的 ADC 时钟(50～200kHz)。过高的频率将导致低的采样精度。

ADCSR 的 ADPS0～ADPS2 用于产生合适的 ADC 时钟。一旦 ADCSR 的 ADEN 置位，预分频器就开始连续不断地记数，直到 ADEN 清零。ADSC 的作用是对 ADC 进行初始化。AD 转换在 ADC 时钟的上升沿启动。采样—保持要花费 1.5 倍 ADC 时钟。在第 13 个时钟 ADC 转换结束，数据进入 ADC 数据寄存器。对于单次转换模式，在进行下一次转换时需要一个额外的 ADC 时钟，如图 5.28 所示，然后转换可继续进行，如果此时 ADSC 为"1"的话；而在自由运行模式下，ADC 结果写入寄存器后立即进行下一次转换。工作于 200kHz 的自由运行模式具有最快的转换速度：65μs，亦即 15.4kSPS(Samplings Per Second)。转换时序见表 5.20。

表 5.20 ADC 时序

条 件	采样周期	得到结果的周期	总的转换周期数	总的转换时间/μs
第一次转换，自由运行	14	25	25	125～500
第一次转换，单次转换	14	25	26	130～520
自由运行模式	2	13	13	65～260
单次转换模式	2	13	14	70～280

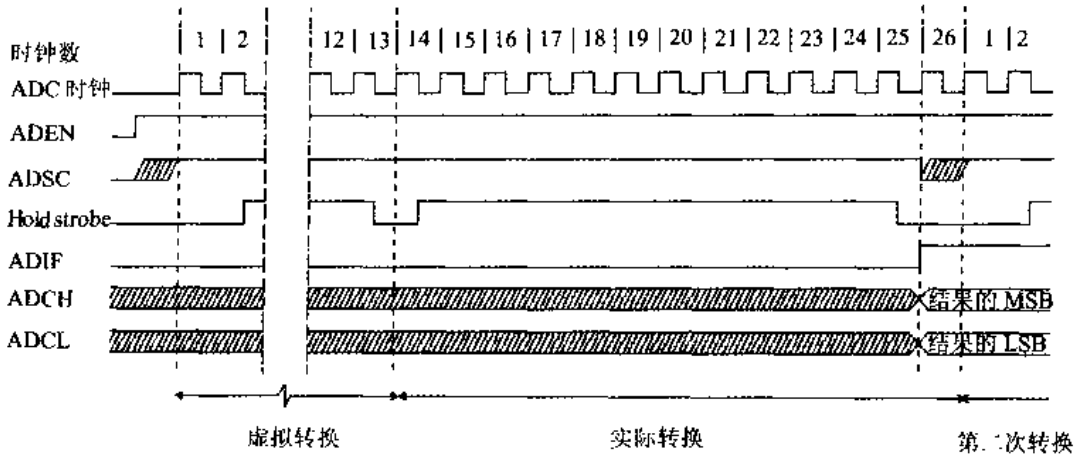


图 5.28 首次转换的时序(单次转换模式)

单次转换的时序和自由运行的时序见图 5.29 和图 5.30。

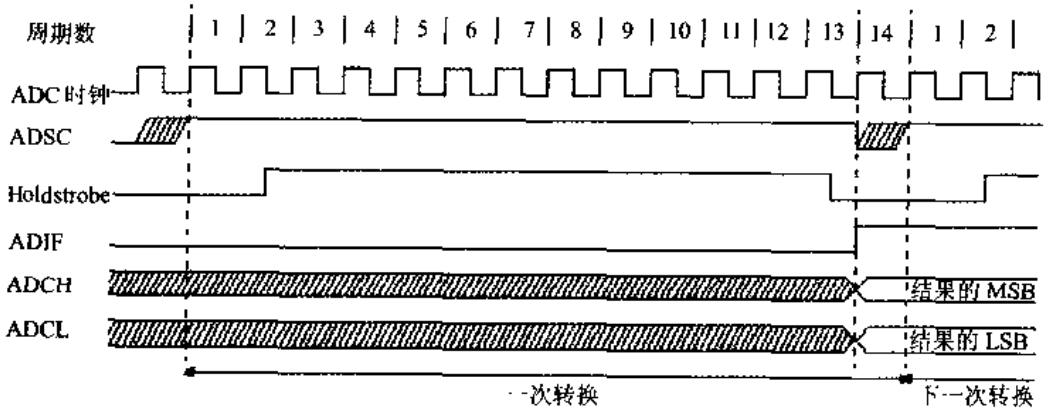


图 5.29 单次转换的时序

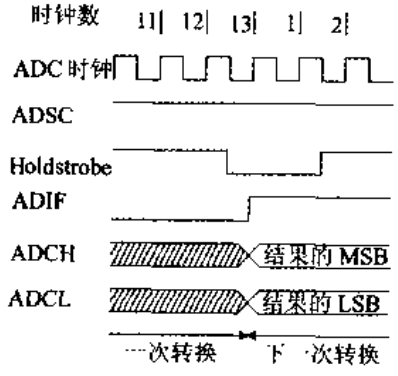


图 5.30 自由运行的时序

三、ADC 噪声抑制功能

ADC 具有消除由 CPU 核引入的噪声的功能。实现过程如下:

(1) 触发 ADC,选择单次转换模式,并触发转换结束中断。

ADEN = 1,ADSC = 0,ADFR = 0,ADIE = 1。

(2) 进入空闲状态。一旦 CPU 停止,ADC 将开始转换。

(3) 如果在 ADC 转换结束中断之前没有发生其他中断,则 ADC 转换结束中断将唤醒 MCU 并执行中断例程。

● **ADC 多路选择寄存器——ADMUX**

BIT	7	6	5	4	3	2	1	0
$07($27)	—	—	—	—	—	MUX2	MUX1	MUX0
读/写	R	R	R	R	R	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

位 7…3：保留。

MUX2～MUX0：模拟通道选择位。用于选择 ADC 的模拟输入通道 0～7。

● **ADC 控制和状态寄存器——ADCSR**

BIT	7	6	5	4	3	2	1	0
$06($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS 2	ADPS 1	ADPS 0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

ADEN：ADC 触发。

ADSC：ADC 开始转换。当 ADC 工作于单次转换模式时，这一位必须设置为“1”，以启动每一次转换；而对于自由运行模式，则只需在第一次转换时设置一次。不论设置动作是在 ADEN置位之后还是一同进行，ADC 都将进行一次哑转换，以初始化 ADC。

转换过程中 ADSC 保持为高。实际转换过程结束后，在转换结果进入 ADC 数据寄存器之前（差一个 ADC 时钟），ADSC 变为低。这样就允许在当前转换完成之前（ADSC 变低之时），对下一次转换进行初始化，一旦当前转换彻底完成，就可以进行新的一次转换过程。在转换过程中，ADSC 保持为高。对 ADSC 写零没有意义。

ADFR：ADC 自由运行模式选择。这一位置位后，ADC 工作于自由运行模式。ADC 将连续不断地进行采样和数据更新。

ADIF：ADC 中断标志。ADC 完成及数据更新完成后，ADIF 置位。如果 I 和 ADIE 置位，则 ADC 结束中断发生。在中断例程里 ADIF 硬件清零，写“1”也可以对其清零。因此要注意对 ADCSR 执行读—修改—写操作时会使挂起的中断。

ADIE：ADC 中断触发。

ADPS2～ADPS0：ADC 预分频器选择。ADC 预分频器选择如表 5.21 所列。

表 5.21　ADC 预分频器选择

ADPS2	ADPS1	ADPS0	分频因子
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

● **ADC 数据寄存器——ADCL 和 ADCH**

BIT	15	14	13	12	11	10	9	8
$05($25)	—	—	—	—	—	—	ADC9	ADC8
$04($24)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
	7	6	5	4	3	2	1	0
读/写	R	R	R	R	R	R	R	R
	R	R	R	R	R	R	R	R
初始值	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

ADCL要先于ADCH读取。

四、扫描多个通道

由于模拟通道的转换总是要延迟到转换结束，因此自由运行模式可以用来扫描多个通道而不中断转换器。一般情况下，ADC转换结束中断用于修改通道，但需注意：中断在转换结果可读时触发。在自由运行模式下，下一次转换在中断触发的同时启动。在ADC中断触发/新一次转换开始后，改变ADMUX将不起作用。

五、ADC噪声消除技术

AT90S4434/8535的内外部数字电路会产生EMI电磁干扰，从而影响模拟测量精度。如果转换精度要求很高，则需要应用如下技术以减少噪声：

(1) AT90S4434/8535的模拟部分及其他模拟器件在PCB上要有独立的地线层。模拟地线与数字地线单点相连。

(2) 应使模拟信号通路尽量短。要使模拟走线在模拟地上通过，并尽量远离高速数字通路。

(3) AV_{CC}要通过一个RC网络连接到V_{CC}。

(4) 利用ADC的噪声消除技术减少CPU引入的噪声。

(5) 如果A口的一些引脚用作数字输出口，则在ADC转换过程中不要改变其状态。

ADC电源连接如图5.31所示。注意：由于AV_{CC}同时也为A口输出驱动提供电源，因此，如果A口有输出引脚，则RC网络不要使用。

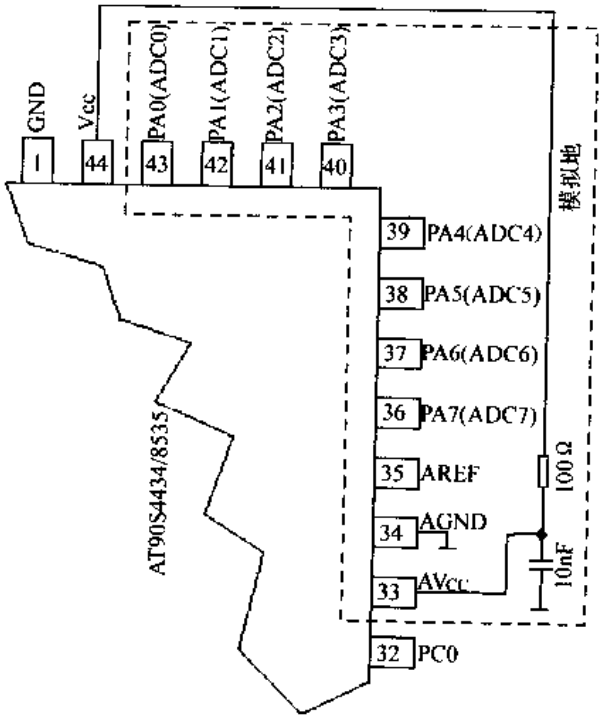


图5.31 ADC电源连接

六、ADC特性

ADC的特性如表5.22所列。

表 5.22　ADC 特性(T_A = −40～85℃)

符　号	参　数	条　件	最小值	典型值	最大值
	精度			10Bit	
	绝对精度	V_{REF} = 4VADC 时钟 = 200kHz		1LSB	
	绝对精度	V_{REF} = 4VADC 时钟 = 1MHz		4LSB	
	绝对精度	V_{REF} = 4VADC 时钟 = 2MHz		16LSB	
	整体非线性	V_{REF} > 2V		0.5LSB	
	差分非线性	V_{REF} > 2V		0.5LSB	
	零误差(偏移)			1LSB	
	转换时间/μs		65		260
	时钟频率/kHz		50		2
AV_{CC}	模拟电源/V	V_{CC} − 3①	V_{CC} − 3①		V_{CC} + 0.3②
V_{REF}	参考电源/V		AGND		AV_{CC}
R_{REF}	参考输入电阻/kΩ		6	10	13
R_{AIN}	模拟输入电阻/MΩ			100	

① AV_{CC} 的最小值为 2.7V；② AV_{CC} 的最大值为 6.0V。

5.6.13　I/O 端口

所有 AVR I/O 端口都具有真正的读—修改—写功能。这意味着，用 SBI 或 CBI 指令改变某些引脚的方向(值、禁止/触发、上拉)时不会无意地改变其他引脚的方向(值、禁止/触发、上拉)。详细内容见第二章 2.11AVR 单片机 I/O 端口。

一、A 口

A 口是 8 位双向 I/O 口。A 口有 3 个 I/O 地址：数据寄存器 PORTA，$1B($3B)；数据方向寄存器 DDRA，$1A($3A)；输入引脚 PINA，$19($39)。PORTA 和 DDRA 可读可写，PINA 只可读。所有的引脚都可以单独选择上拉电阻。引脚缓冲器可以吸收 20mA 的电流，能够直接驱动 LED。当引脚被拉低时，如果上拉电阻已经激活，则引脚会输出电流。A 口的第二功能是 ADC 的模拟输入端。如果 A 口的一些引脚用作数字输出口，则在 ADC 转换过程中不要改变其状态，否则会破坏转换结果。在掉电模式时，数字输入的施密特触发器与引脚相断开。这样接近 V_{CC}/2 的模拟输入就不会造成大的功耗。

1. A 口数据寄存器— PORTA

BIT	7	6	5	4	3	2	1	0
$1B($3B)	PORTA7							PORTA0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

2. A 口数据方向寄存器— DDRA

BIT	7	6	5	4	3	2	1	0
$1A($3A)	DDA7							DDA0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

3. A 口输入引脚 —PINA

BIT	7	6	5	4	3	2	1	0
$19($39)	PINA7							PINA0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	Hi- Z	Hi -Z	Hi—Z	Hi—Z	Hi—Z	Hi—Z	Hi—Z	Hi—Z

PINA 不是一个寄存器，这个地址用来访问 A 口的物理值。读取 PORTA 时，读到的是 A 口锁存的数据；而读取 PINA 时，读到的是施加于引脚上的逻辑数值。

4. A 口作通用数字 I/O

作为通用数字 I/O 时，A 口的 8 个引脚具有相同的功能。

PAn，通用 I/O 引脚；DDRA 中的 DDAn 选择引脚的方向。如果 DDAn 为"1"，则 PAn 为输出脚；如果 DDAn 为"0"，则 PAn 为输入脚。在复位期间，A 口为三态口。A 口的配置如表 5.23 所列。

表 5.23 A 口的配置

DDAn	PORTAn	I/O	上 拉	注 释
0	0	输入	N	三态(高阻)
0	1	输入	Y	外部拉低时会输出电流
1	0	输出	N	推挽 0 输出
1	1	输出	N	推挽 1 输出

n：7，6…0，引脚号。

二、B 口

B 口是 8 位双向 I/O 口。B 口有 3 个 I/O 地址：数据寄存器 PORTB，$18($38)；数据方向寄存器 DDRB，$17($37)；输入引脚 PINB，$16($36)。PORTB 和 DDRB 可读可写，PINB 只可读。所有的引脚都可以单独选择上拉电阻。引脚缓冲器可以吸收 20mA 的电流，能够直接驱动 LED。当引脚被拉低时，如果上拉电阻已经激活，则引脚会输出电流。

1. B 口数据寄存器—PORTB

BIT	7	6	5	4	3	2	1	0
$18($38)	PORTB7							PORTB0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

2. B 口数据方向寄存器— DDRB

BIT	7	6	5	4	3	2	1	0
$17($37)	DDB7							DDB0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

3. B 口输入引脚——PINB

BIT	7	6	5	4	3	2	1	0
$16($36)	PINB7							PINB0
读/写	R	R	R	R	R	R	R	R
初始值	Hi Z	Hi—Z	Hi—Z	Hi—Z	Hi—Z	Hi—Z	Hi -Z	Hi—Z

PINB 不是一个寄存器，这个地址用来访问 B 口的物理值。读取 PORTB 时，读到的是 B 口锁存的数据；而读取 PINB 时，读到的是施加于引脚上的逻辑数值。

4. B 口作通用数字 I/O

作为通用数字 I/O 时，B 口的 8 个引脚具有相同的功能。

PBn，通用 I/O 引脚：DDRB 中的 DDBn 选择引脚的方向。如果 DDBn 为“1”，则 PBn 为输出脚；如果 DDBn 为“0”，则 PBn 为输入脚。在复位期间，B 口为三态口。B 口配置如表 5.24。

表 5.24　B 口的配置

DDBn	PORTBn	I/O	上　拉	注　释
0	0	输入	N	三态(高阻)
0	1	输入	Y	外部拉低时会输出电流
1	0	输出	N	推挽 0 输出
1	1	输出	N	推挽 1 输出

n：7，6…0，引脚号。

5. B 口的第二功能

B 口的第二功能如表 5.25 所列。

当使用 B 口的第二功能时，DDRB 和 PORTB 要设置成对应的值。

● SCK——PB7：SPI 的主机时钟输出，从机时钟输入。配置为从机时，此引脚配置为输入而不管 DDB7 的值；而当 SPI 为主机时，SCK 的配置由 PB7 和 DDB7 控制。

● MISO——PB6：SPI 的主机数据输入，从机数据输出。配置为主机时，此引脚配置为输入而不管 DDB6 的值；而当 SPI 为主机时，MISO 的配置由 PB6 和 DDB6 控制。

● MOSI——PB5：SPI 的主机数据输出，从机数据输入。配置为从机时，此引脚配置为输入而不管 DDB5 的值；而当 SPI 为主机时，MOSI 的配置由 PB5 和 DDB5 控制。

● $\overline{SS}$——PB4：从机选择信号。配置为从机时，此引脚配置为输入而不管 DDB4 的值，$\overline{SS}$为低将激活 SPI；配置为主机时，此引脚的方向由 DDB4 控制。如果 DDB4 为“1”，则上拉仍然可以由 PORTB4 控制。

表 5.25　B 口第二功能

引　脚	第二功能
PB0	T0(T/C0 外部记数输入)
PB1	T1(T/C1 外部记数输入)
PB2	AIN0(模拟比较器正输入端)
PB3	AIN1(模拟比较器负输入端)
PB4	$\overline{SS}$(SPI 从机选择)
PB5	MOSI(程序下载时的数据输入线)
PB6	MISO(程序下载时的数据输出线)
PB7	SCK(串行时钟)

● AIN1——PB3：当配置为输入(DDB2＝3)，无上拉电阻(PB3＝0)时，为模拟比较器的负输入端。

● AIN0——PB2：当配置为输入(DDB2＝0)，无上拉电阻(PB2＝0)时，为模拟比较器的正输入端。

● T1——PB1：T/C1 的外部记数输入。

● T0——PB0：T/C0 的外部记数输入。

三、C　口

C 口是 8 位双向 I/O 口。

C 口有 3 个 I/O 地址：数据寄存器 PORTC，$15($35)；数据方向寄存器 DDRC，$14

($34)；输入引脚 PINC，$13($33)。PORTC 和 DDRC 可读可写，PINC 只可读。

所有的引脚都可以单独选择上拉电阻。引脚缓冲器可以吸收 20mA 的电流，能够直接驱动 LED。当引脚被拉低时，如果上拉电阻已经激活，则引脚会输出电流。

1. C 口数据寄存器 —PORTC

BIT	7	6	5	4	3	2	1	0
$15($35)	PORTC7							PORTC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

2. C 口数据方向寄存器 — DDRC

BIT	7	6	5	4	3	2	1	0
$14($34)	DDC7							DDC0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

3. C 口输入引脚——PINC

BIT	7	6	5	4	3	2	1	0
$13($33)	PINC7							PINC0
读/写	R	R	R	R	R	R	R	R
初始值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z

PINC 不是一个寄存器，这个地址用来访问 C 口的物理值。读取 PORTC 时，读到的是 C 口锁存的数据；而读取 PINC 时，读到的是施加于引脚上的逻辑数值。

4. C 口作通用数字 I/O

作为通用数字 I/O 时，C 口的 8 个引脚具有相同的功能。

PCn，通用 I/O 引脚：DDRC 中的 DDCn 选择引脚的方向。如果 DDCn 为“1”，则 PCn 为输出脚；如果 DDCn 为“0”，则 PCn 为输入脚。在复位期间，C 口为三态口。C 口配置如表 5.26所列。

表 5.26 C 口的配置

DDCn	PORTCn	I/O	上 拉	注 释
0	0	输入	N	三态(高阻)
0	1	输入	Y	外部拉低时会输出电流
1	0	输出	N	推挽 0 输出
1	1	输出	N	推挽 1 输出

n：7，6…0，引脚号。

四、D 口

D 口是 8 位双向 I/O 口。

D 口有 3 个 I/O 地址：数据寄存器 PORTD，$12($32)；数据方向寄存器 DDRD，$11($31)；输入引脚 PIND，$10($30)。PORTD 和 DDRD 可读可写，PIND 只可读。

D 口的引脚缓冲器可以吸收 20mA 的电流，能够直接驱动 LED。当引脚被拉低时，如果上拉电阻已经激活，则引脚会输出电流。

1. D口数据寄存器 —— PORTD

BIT	7	6	5	4	3	2	1	0
$12($32)	PORTD7							PORTD0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

2. D口数据方向寄存器——DDRD

BIT	7	6	5	4	3	2	1	0
$11($31)	DDD7							DDD0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

3. D口输入引脚——PIND

BIT	7	6	5	4	3	2	1	0
$10($30)	PIND7							PIND0
读/写	R	R	R	R	R	R	R	R
初始值	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z

PIND不是一个寄存器，这个地址用来访问D口的物理值。读取PORTD时，读到的是D口锁存的数据；而读取PIND时，读到的是施加于引脚上的逻辑数值。

4. D口作通用数字I/O

PDn，通用I/O引脚；DDRD中的DDDn选择引脚的方向。如果DDDn为"1"，则PDn为输出脚；如果DDDn为"0"，则PDn为输入脚。在复位期间，D口为三态口。D口配置如表5.27所列。

表5.27 D口的配置

DDDn	PORTDn	I/O	上 拉	注 释
0	0	输入	N	三态(高阻)
0	1	输入	Y	外部拉低时会输出电流
1	0	输出	N	推挽0输出
1	1	输出	N	推挽1输出

n：6…0，引脚号。

5. D口的第二功能

D口的第二功能如表5.28所列。

- RD——PD7。
- WR——PD6。
- OC1A——PD5：PD5可以作T/C1比较匹配的外部输出。此时PD5必须配置为输出。OC1A也是PWM的输出引脚。
- INT1——PD3：外部中断源1。
- INT0——PB2：外部中断源0。
- TXD——PD1：发送数据引脚。发送器触发后，引脚自动配置为输出而不管DDR1。

表5.28 D口第二功能

引 脚	第二功能
PD0	RXD(UART接收引脚)
PD1	TXD(UART发送引脚)
PD2	INT0(外部中断0输入)
PD3	INT1(外部中断1输入)
PD5	OC1A(T/C1输出比较A匹配输出)
PD6	$\overline{WR}$
PD7	$\overline{RD}$

● RXD——PD0:接收数据引脚。接收器触发后,引脚自动配置为输入而不管 DDR0。若此时 PORTD0 为"1",则上拉有效。

程序编程见第二章 2.12 AVR 单片机存储器编程。指令系统见附录 3。

5.7 ATmega83/163

5.7.1 特 点

(1) 高性能,低功耗的 AVR RISC 结构:

——ATmega83:128 条指令,大多数为单指令周期;

——Atmega163:130 条指令,大多数为单指令周期;

—32 个 8 位通用(工作)寄存器;

—工作在 8MHz 时具有 8MIPS 的性能;

——2 个周期的硬件乘法器。

(2) 数据和非易失性程序内存:

——ATmega83:8K 字节的在线可编程 Flash(擦除次数为 1 000 次);

— ATmega163:16K 字节的在线可编程 Flash(擦除次数为 1 000 次);

——Boot 代码区具有独立的 Lock Bits,In—System—Programming 可通过 Boot 代码完成(自编程的概念);

——512 字节在线可编程 EEPROM(寿命为 100 000 次);

— —ATmega83:512 字节 SRAM;

——ATmega163:1024 字节 SRAM;

——程序加密位。

(3) 外围(Peripheral)特点:

——2 个具有比较模式的可预分频(Prescale)8 位定时器/计数器;

——1 个可预分频、具有比较、捕捉功能的 16 位定时器/计数器;

— -具有独立振荡器的实时时钟;

——3 个 PWM 通道;

— -8 通道,10 位 ADC;

——8 个单端通道;

— 7 个差分通道;

——2 个具有可编程增益(1,10,200)的差分通道;

— I^2C 接口;

—-可编程的 UART;

——主/从 SPI 接口;

——可编程的看门狗定时器(由片内振荡器生成);

——片内模拟比较器。

(4) MCU 的特点:

——上电复位及可编程的掉电检测电路;

——可标度的内部 RC 振荡器；

— 内外部中断源；

— 4 种休眠模式：空闲、ADC 噪声抑制、省电和掉电模式。

(5) I/O 和封装：

— 32 个可编程的 I/O 脚；

——40 脚 PDIP、44 脚 PLCC 和 TQFP 封装。

(6) 工作电压：

——2.7～5.5V(ATmega83L/163L)；

—4.0～5.5V(ATmega83/163)。

(7) 速度：

——0～4MHz(ATmega83L/163L)；

—0～8MHz(ATmega83/163)。

5.7.2 描 述

图 5.32 为 ATmega83/163 的结构框图。

ATmega83/163 是一款基于 AVR RISC 的、低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令，ATmega83/163 可以取得接近 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU(算逻单元)直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。

ATmega83/163 具有以下特点：8K/16K 字节在线编程/自编程的 Flash；512 字节 EEPROM；512/1024 字节存储器；32 个通用 I/O 口；32 个通用工作寄存器；实时时钟 RTC；3 个具有比较模式的灵活的定时器/计数器；内外中断源；8 位的 I^2C 总线接口；8 通道 10 位 ADC(1 个为差分输入，增益可调)；可编程的看门狗定时器；SPI 口以及 4 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而 SRAM、定时器/计数器、看门狗和中断系统继续工作；掉电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有外部低电平中断或硬件复位才可以退出此状态。

省电模式与掉电模式只有一点差别：省电模式下 T/C2 继续工作以维持时间基准。在 ADC 噪声抑制模式下，CPU 及其他 I/O 模块停止，只有 ADC 和异步定时器 T/C2 工作，以减小 ADC 转换过程中的开关噪声。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 可以通过 SPI 接口、通用编程器及 BOOT 程序进行编程。BOOT 程序可以利用任一接口下载应用程序到应用 Flash 区。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，ATmega83/163 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega83/163 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和评估板。

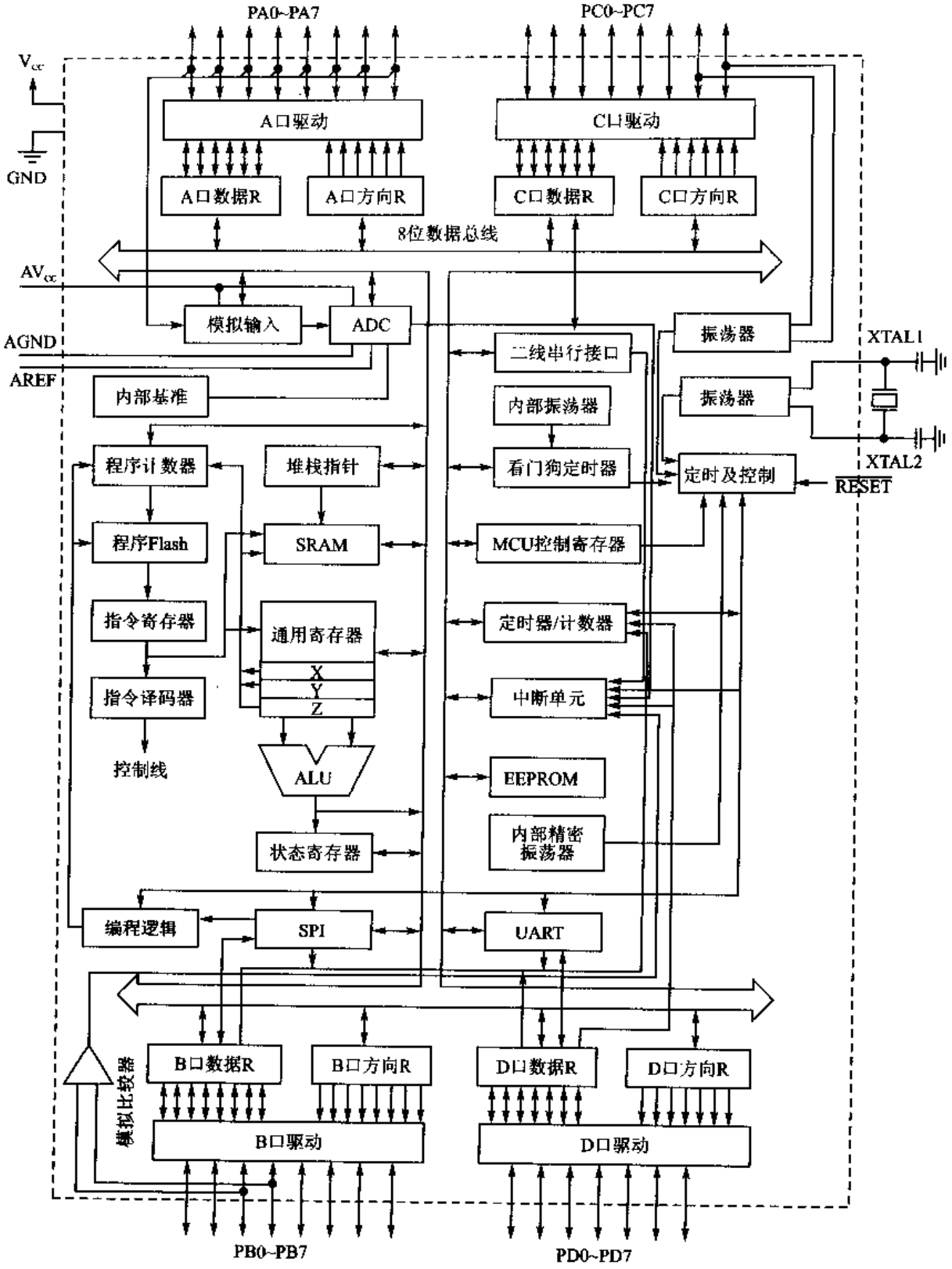


图 5.32 ATmega83/163 结构框图

5.7.3 ATmega83 与 ATmega163 的比较

ATmega83 具有 8K 字节的系统内可编程 Flash 和 512 字节的 SRAM。ATmega163 具有 16K 字节的系统内可编程 Flash 和 1 024 字节的 SRAM。二者都有 512 字节的 EEPROM。由于 ATmega163 的 Flash 大于 8K，因而需要 JMP 和 CALL 指令。由于 JMP 指令占用 2 个字，因此，ATmega163 的中断向量为 2 个字长，而 ATmega83 的中断向量为 1 个字长。表5.29 列出了两者的异同。

表 5.29 存储器大小及特点

工作部件	ATmega83	ATmega163
Flash	8K 字节	16K 字节
EEPROM	512 字节	512 字节
SRAM	512 字节	1 024 字节
JMP, CALL	不支持	支持
中断向量	1 个字	2 个字

5.7.4 引脚配置

ATmega 83/163 引脚配置如图 5.33 所示。

一、引脚定义

V_{CC}、GND：电源引脚。

A 口(PA7～PA0)：A 口为 ADC 的模拟输入。如果 AD 功能禁止，则 A 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。A 口的输出缓冲器能够吸收 20mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，A 口为三态，即使此时时钟还未起振。

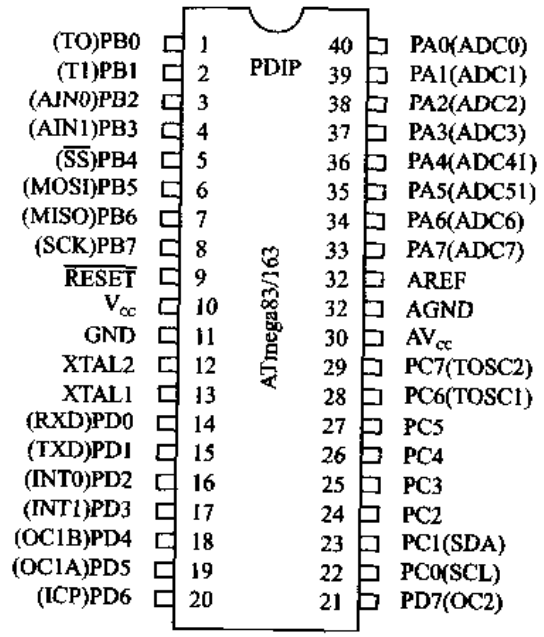


图 5.33 ATmega 83/163 引脚配置图

B 口(PB7～PB0)：B 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。B 口的输出缓冲器能够吸收 20mA 的电流，可直接驱动 LED。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。B 口作为特殊功能口的使用方法见光盘文件。

C 口(PC7～PC0)：C 口是一个 8 位双向 I/O 口，每一个引脚都有内部上拉电阻。C 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。C 口的 2 个引脚还可以作为 T/C2 的振荡器引脚。在复位过程中，C 口为三态，即使此时时钟还未起振。

D 口(PD7～PD0)：D 口是一个带内部上拉电阻的 8 位双向 I/O 口。输出缓冲器能够吸收

20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。D 口作为特殊功能口的使用方法见光盘文件。

$\overline{\text{RESET}}$：复位输入。在振荡器起振的前提下，超过 2 个时钟的低电平将产生复位信号。

XTAL1：振荡器放大器的输入端。

XTAL2：振荡器放大器的输出端。

AV_{CC}：A/D 转换器的电源。应该通过一个低通滤波器与 V_{CC} 连接。

AREF：A/D 转换器的参考电源，介于 AGND 与 AV_{CC} 之间。

AGND：模拟地。

二、晶振器

时钟选择：在熔丝位的控制下，器件可有表 5.30 的时钟选项。

不同的选项决定了不同的启动时间。

表 5.30 时钟选项

时钟选项	CKSEL3…0
外部晶振/陶瓷振荡器	1111～1010
外部低频晶振	1001～1000
外部 RC 振荡器	0111～0101
内部 RC 振荡器	0100～0010
外部时钟	0001～0000

注意："1" 表示未编程，"0" 表示编程。

5.8 ATtiny10/11/12

5.8.1 特 点

(1) AVR RISC 结构：

——高性能、低功耗 RISC 结构；

——90 条指令，大多数为单指令周期；

—— 32 个 8 位通用(工作)寄存器；

——工作在 8MHz 时具有 8MIPS 的性能。

(2) 数据和非易失性程序内存：

——1K 字节的 Flash：QuickFlashTM(ATtiny10)，ISP(ATtiny12)，擦除次数为 1 000 次(ATtiny11/12)；

——64 字节在线可编程 EEPROM(ATtiny12)，寿命为 100 000 次；

——程序加密位。

(3) 外围(Peripheral)特点：

——引脚电平变化中断及唤醒；

——1 个可预分频(Prescale)的 8 位定时器/计数器；

——片内模拟比较器；

——可编程的看门狗定时器(由片内振荡器生成)。

(4) MCU 特点：

——低功耗空闲和掉电模式；

——内外部中断源；

——通过 SPI 口的 ISP(ATtiny12)；

——增强的上电复位电路(ATtiny12)；

——可标度的片内 RC 振荡器。

(5) 规范(Specification)：

— -低功耗、高速 CMOS 工艺；

— -全静态工作。

(6) 在 4MHz、3V、25℃条件下的功耗：

——工作模式为 2.2mA；

——空闲模式为 0.5mA；

- -掉电模式为<1μA。

(7) I/O 和封装：

—--8 脚 PDIP 和 SOIC 封装。

(8) ATtiny10 是 ATtiny11 的 QuickFlash 版本。

(9) 工作电压：

- -1.8～5.5V(ATtiny12V-1)；

-- -2.7～5.5V(ATtiny11L-2 和 ATtiny11L-4)；

-4.0～5.5V(ATtiny11-6 和 ATtiny12-8)。

(10) 速度：

-0～1 MHz(ATtiny12V-1)；

——0～2 MHz(ATtiny11L-2)；

----0～4 MHz(ATtiny12L-4)；

——0～6MHz(ATtiny11-6)；

---0～8MHz(ATtiny12-8)。

5.8.2 描 述

ATtiny10/11/12 是一款基于 AVR RISC 的、低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令，ATtiny10/11/12 可以取得接近 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU(算逻单元)直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。表 5.31 对 ATtiny10/11/12 进行了描述。

表 5.31 器件描述

器 件	Flash/KB	EEPROM/B	寄存器	电压范围/V	频率/MHz
ATtiny10/11L	1	—	32	2.7～5.5	0～2
ATtiny10/11	1	--	32	4.0～5.5	0～6
ATtiny12V	1	64	32	1.8～5.5	0～1
ATtiny12L	1	64	32	2.7～5.5	0～4
ATtiny12	1	64	32	4.0～5.5	0～8

ATtiny10/11 具有以下特点：1K 字节 Flash，多达 5 个通用 I/O 口，1 个输入口；32 个通用工作寄存器；1 个 8 位 T/C；内外中断源；可编程的看门狗定时器以及 2 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而定时器/计数器和中断系统继续工作；掉

电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有低电平中断或硬件复位才可以退出此状态。引脚电平变化中断的特点使得 ATtiny10/11 对外部事件有很高的响应性，同时具有掉电模式的低功耗的优点。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 允许多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，ATtiny10/11 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATtiny10/11 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和评估板。

图 5.34 是 ATtiny10/11 的结构框图。

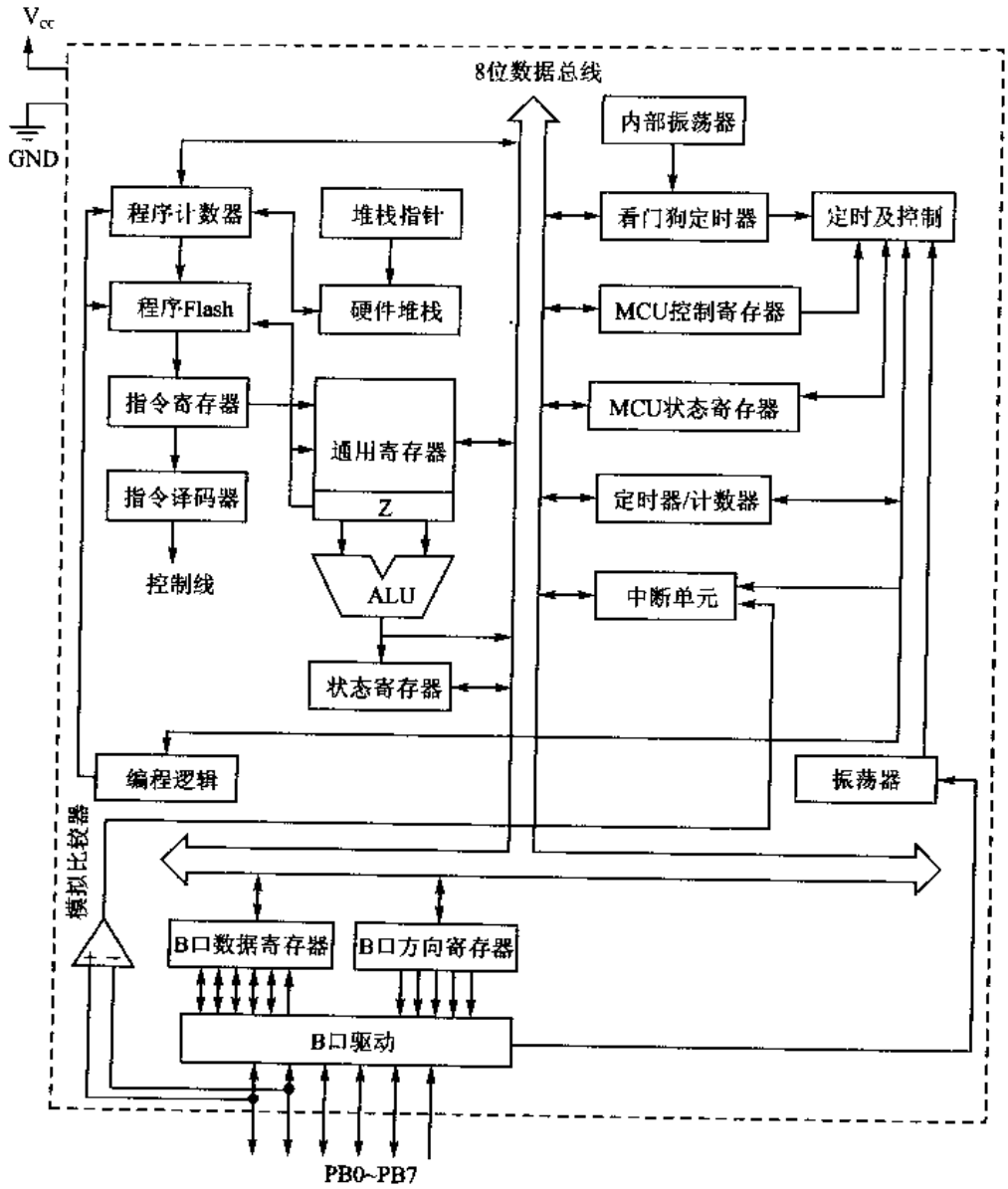


图 5.34 ATtiny10/11 结构框图

图 5.35 是 ATtiny12 结构框图。

ATtiny12 具有以下特点：1K 字节 Flash；64 字节 EEPROM；多达 6 个通用 I/O 口；32 个

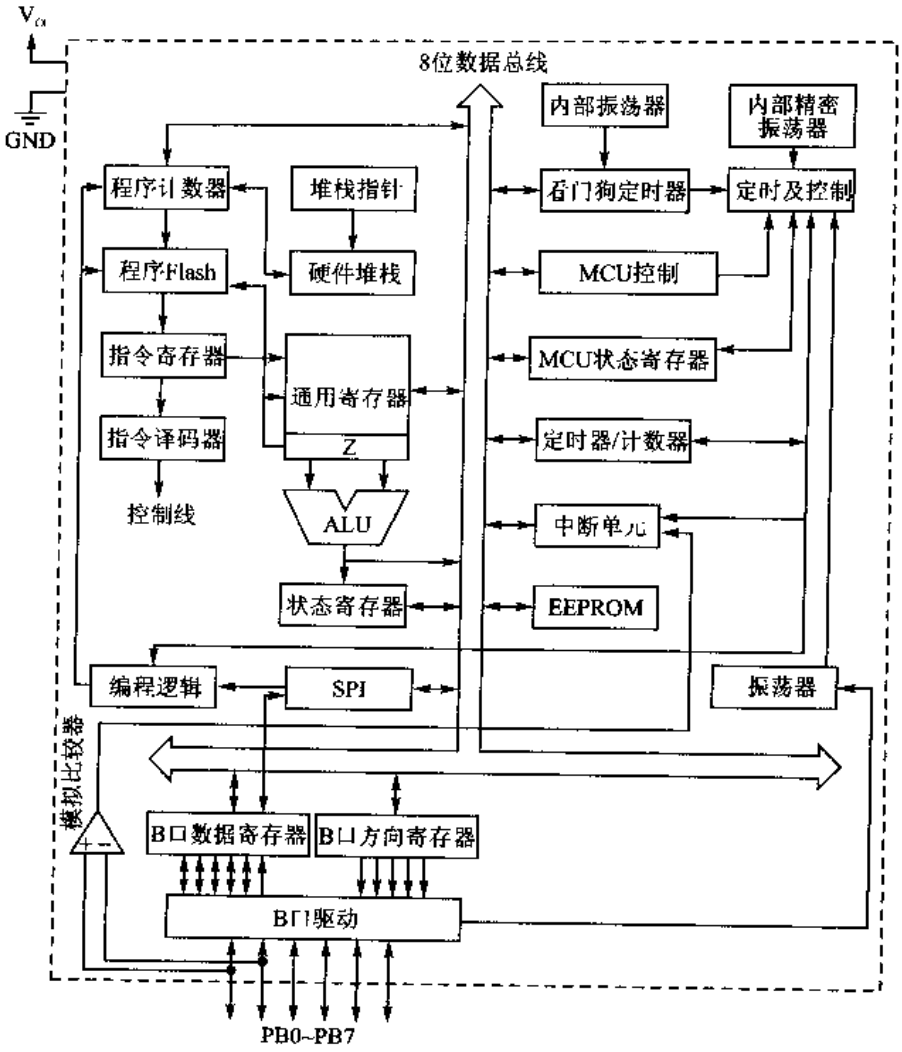


图 5.35　ATtiny12 结构框图

通用工作寄存器；1 个 8 位 T/C；可编程的看门狗定时器以及 2 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而定时器/计数器和中断系统继续工作；掉电模式时振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留。只有外部中断或硬件复位才可以退出此状态。引脚电平变化中断的特点使得 ATtiny12 对外部事件有很高的响应性，同时具有掉电模式的低功耗的优点。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 允许多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，ATtiny12 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATtiny12 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和 SL-AVR 评估板。

5.8.3　引脚配置

一、引脚定义

图 5.36 为 ATtiny10/11/12 引脚配置图。

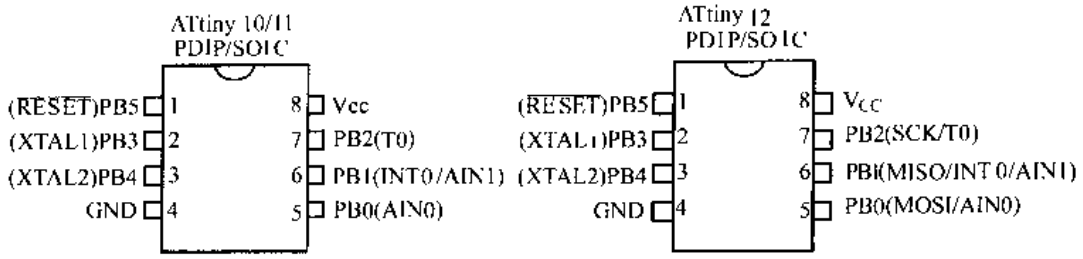


图 5.36 AT10/11/12 引脚配置图

V_{CC}、GND：电源。

B 口(PB5～PB0)：B 口是一个 6 位 I/O 口，PB4～PB0 有内部上拉电阻(可单独选择)。对于 ATtiny10/11，PB5 是输入口；而对于 ATtiny12，PB5 可以是输入口或是开漏输出口。在复位过程中，B 口为三态，即使此时时钟还未起振。PB5～PB3 是用作输入还是 I/O，取决于复位和时钟设置，如表 5.32 所列。

表 5.32 PB5～PB3 功能与时钟设定的关系

时钟选择	PB5	PB4	PB3
外部复位使能	已用		
外部复位禁止	输入/I/O①	—	—
外部晶振	—	已用	已用
外部低频晶振	—	已用	已用
外部陶瓷振荡器	—	已用	已用
外部 RC 振荡器	—	I/O	已用
外部时钟	—	I/O	已用
内部 RC 振荡器		I/O	I/O

① 对于 ATtiny10/11，PB5 是输入口；而对于 ATtiny12，PB5 可以是输入口或是开漏输出口。

XTAL1：振荡器放大器的输入端。

XTAL2：振荡器放大器的输出端。

二、晶振器

晶体振荡器：XTAL1 和 XTAL2 分别是片内振荡器的输入、输出端，可使用晶体振荡器或陶瓷振荡器。当使用外部时钟时，XTAL2 应悬空。

时钟选择：器件具有多种时钟选择，由熔丝位控制。如表 5.33 所列。

内部 RC 振荡器：内部 RC 振荡器的频率固定为 1MHz。器件出厂时已经选择了这项功能。对于 ATtiny10/11，看门狗振荡器用作时钟，而对于 ATtiny12 则使用了单独的可标度振荡器。

表 5.33 器件时钟选择

器件时钟选择	ATtiny10/11 CKSEL2…0	ATtiny12 CKSEL3…0
外部晶振/陶瓷振荡器	111	1111～1010
外部低频晶振	110	1001～1000
外部 RC 振荡器	101	0111～0101
内部 RC 振荡器	100	0100～0010
外部时钟	000	0001～0000
保留	其它选择	—

注："1"代表未编程，"0"代表已编程。

5.9 ATtiny15/L

5.9.1 特 点

(1) 高性能,低功耗,8 位 AVR 结构:
——90 条指令,大多数为单指令周期;
——32 个 8 位通用(工作)寄存器;
—— 全静态工作。
(2) 数据和非易失性程序内存:
——1K 字节的在线可编程 Flash,擦除次数为 1 000 次;
——64 字节在线可编程 EEPROM,寿命为 100 000 次;
——程序加密位
(3) 外围(Peripheral)特点:
——2 个可预分频(Prescale)的 8 位定时器/计数器,1 个高速(100kHz)PWM 输出;
4 通道 10 位 ADC,一个具有可选 20 倍增益的差分通道;
片内模拟比较器;
—— 可编程的看门狗定时器(由片内振荡器生成)。
(4) MCU 特点:
——通过 SPI 口的 ISP;
——内外部中断源;
——低功耗空闲和掉电模式;
——低功耗、减噪和掉电模式;
增强的上电复位电路;
——可编程的 BOD 电路;
——内部 1.6MHz 可调谐振荡器;
——内部 T/C1 25.6MHz 时钟发生器。
(5) I/O 和封装:
——8 脚 PDIP/SOIC:6 个可编程 I/O。
(6) 工作电压:
——2.7~5.5V(ATtiny15/L1L);
——4.0~5.5V(ATtiny15/L)。
(7) 内部系统时钟:0.8~1.6MHz。

5.9.2 描 述

ATtiny15/L 是一款基于 AVR RISC 的、低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令,ATtiny15/L 可以取得接近 1MIPS/MHz 的性能,从而使得设计人员可以在功耗和执行速度之间取得平衡。

AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU（算逻单元）直接相连，允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。ATtiny15/L 具有 4 个单端及一个 20 倍增益的差分 ADC 通道。高速 PWM 输出使得 ATtiny15/L 十分适合于电池充电器应用和电源调省电路。图 5.37 为其结构框图。

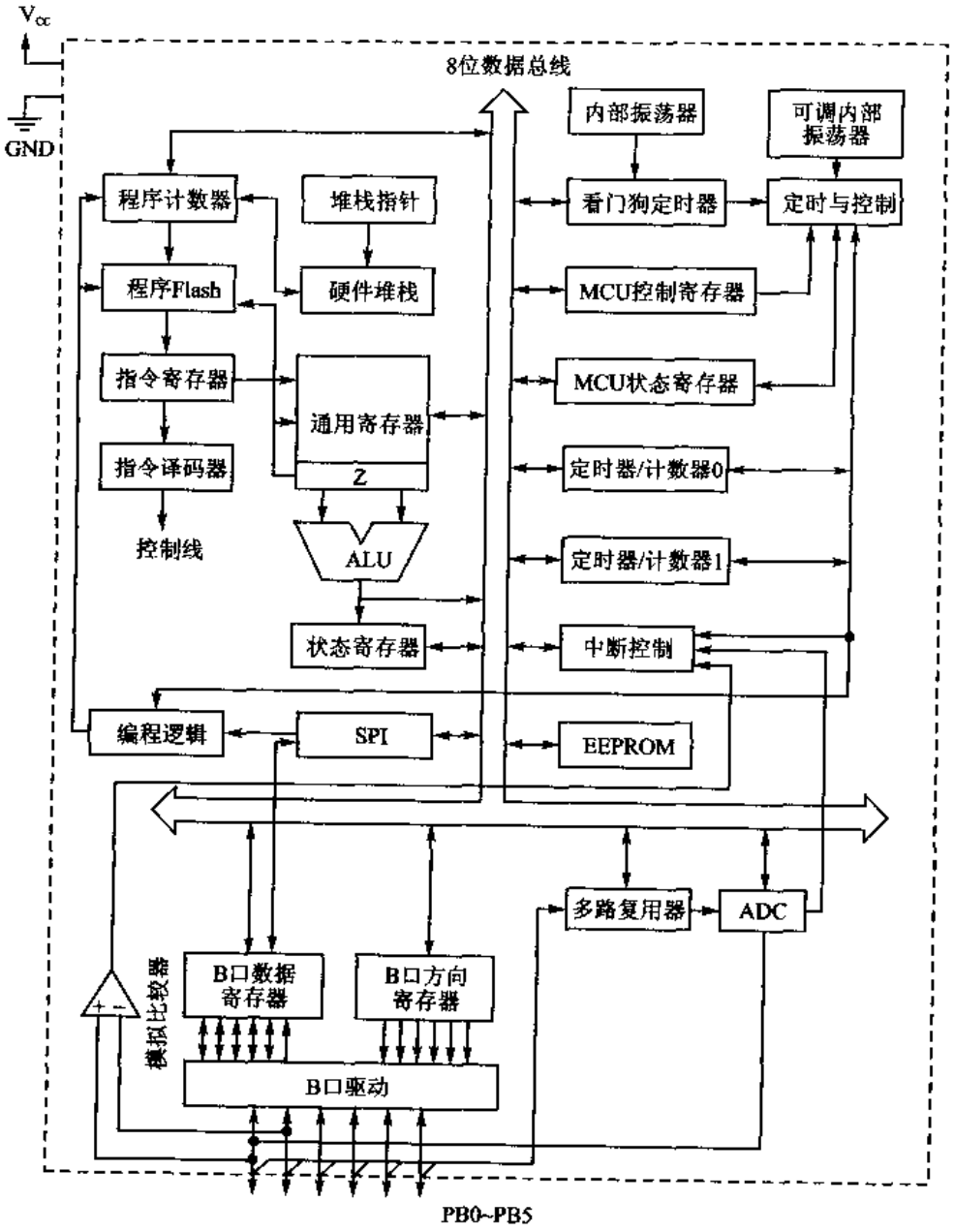


图 5.37 ATtiny15/L 结构框图

ATtiny15/L 具有以下特点：1K 字节 Flash；64 字节 EEPROM；6 个通用 I/O 口；32 个通用工作寄存器；2 个 8 位 T/C（1 个具有 PWM 输出）；内部振荡器；内外中断源；可编程的看门狗定时器；4 通道 10 位 ADC（其中之一为差分通道）以及 3 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而定时器/计数器和中断系统继续工作；掉电模式时振荡

器停止工作，所有功能都被禁止，而寄存器内容得到保留。外部低电平中断或硬件复位可以唤醒此状态。引脚电平变化中断的特点使得 ATtiny15/L 对外部事件有很高的响应性，同时具有掉电模式的低功耗的优点。ATtiny15/L 同时还有一个 ADC 噪声抑制模式以减少 ADC 转换时的噪声。在此模式下，只有 ADC 工作。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 允许多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，ATtiny15/L 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATtiny15/L 具有一整套的编程和系统开发工具：宏汇编、调试/仿真器、在线仿真器和 SL-AVR 编程开发实验器。

5.9.3 引脚配置

ATtiny 15/L 引脚图配置如图 5.38 所示。

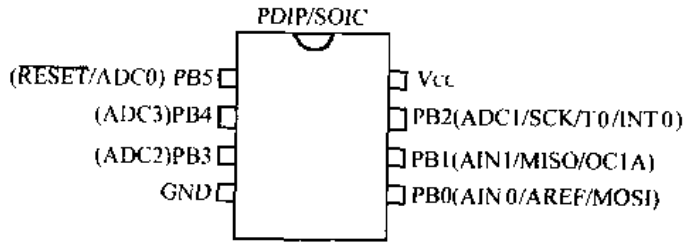


图 5.38 ATtiny 15/L 引脚配置图

一、引脚定义

V_{CC}、GND：电源。

B 口（PB5～PB0）：B 口是一个 6 位 I/O 口，PB4～PB0 有内部上拉电阻（可单独选择）。PB5 是输入口。PB5 的使用由熔丝位定义。此引脚还可以作为外部中断或 ADC 输入通道 0。B 口还是模拟 I/O 引脚。B 口的其它功能见表 5.34。

二、内部振荡器

内部振荡器的标称频率为 1.6MHz。内嵌的调谐功能可以修正偏差（0.8MHz～1.6MHz）。通过对 8 位 OSCCAL 寄存器的控制，可以得到小于 1%的调谐率。内部 PLL 对系统时钟 16 倍频，为 T/C1 提供时钟信号 PCK，最大值为 25.6MHz。

表 5.34 B 口的其他功能

引 脚	功 能
PB0	MOSI(SPI 数据输入) AIN0(模拟比较器输入通道 0) VREF(AIX 电压基准)
PB1	MISO(SPI 数据输出) AIN1(模拟比较器输入通道 1) OCP(T/C1 的 PWM 输出)
PB2	SCK(SPI 时钟输入) INT0(外部中断 0 输入) ADC1(ADC 输入通道 1) T0(T/C0 计数器外部输入)
PB3	ADC2(ADC 输入通道 2)
PB4	ADC3(ADC 输入通道 3)
PB5	RESET(外部复位输入) ADC0(ADC 输入通道 0)

5.10 ATmega128/128L

ATmega128/128L 带 128K 字节 Flash 的在线可编程 8 位微控制器。它是 AVR 系列中

功能最强的单片机。掌握了 ATmega128 的开发应用，对其它 AVR 单片机的开发应用好比杀鸡用牛刀，快极了。

5.10.1 特 点

(1) 先进的 RISC 精简指令集结构：
——高性能，低功耗的 RISC 结构；
——133 条功能强大的指令，大多数为单指令周期；
——32×8 个通用工作寄存器＋外设控制寄存器；
——全静态操作；
——工作在 16 MHz 下，具有 16MIPS 的性能；
——片内带有执行时间为 2 个时钟周期的硬件乘法器。
(2) 数据和非易失性程序内存：
——128K 字节在线可重复编程 Flash(擦写次数为 1 000 次)；
——BOOT 区具有独立的加密位，可通过片内的引导程序实现在系统编程，写操作时真正可读；
——4K 字节 EEPROM(擦写次数为 100 000 次)；
——4K 字节内部 SRAM；
——最大 64K 字节可选外部存储器空间；
——程序加密位；
——在线可编程 SPI 接口。
(3) JTAG (符合 IEEE std. 1149.1 标准) 接口：
——边界扫描能力；
——广泛的片内 Debug 支持；
——通过 JTAG 接口对 Flash、EEPROM、熔丝位和加密位编程。
(4) 外围(Peripheral)特点：
——两个带预分频器和一种比较模式的 8 位定时器/计数器；
——两个扩充的带预分频器和比较模式、捕获模式的 16 位定时器/计数器；
——具有独立振荡器的实时计数器；
——2 通道 8 位 PWM；
——6 通道 2 到 16 位精度 PWM；
——输出比较调节器；
——8 通道 10 位 A/D 转换(8 个单端通道，7 个微分通道，2 个增益为 1x、10x 或 200x 的微分通道)；
——两线(I^2C)串行接口；
——可编程串行 UART 接口；
——主/从 SPI 串行接口；
——带内部振荡器的可编程看门狗定时器；
——片内模拟比较器。
(5) MCU 特点：

- — 上电复位和可编程的低电压检测；
- — 内部可校准的 RC 振荡器；
- —外部和内部中断源；
- ——6 种休眠模式：空闲模式、ADC 噪声抑制模式、省电模式、掉电模式、待命模式和扩展待命模式；
- — 可软件选择时钟频率；
- ——通过一个熔丝选定 ATmega103 兼容模式；
- ——全局上拉禁止。

(6) I/O 和封装：

- — 53 个可编程的 I/O 脚；
- —64 脚 TQFP 封装。

(7) 工作电压：

- ——2.7～5.5V (ATmega128L)；
- ——4.5～5.5V (ATmega128)。

(8) 速度：

- —0～8 MHz (ATmega128L)；
- — 0～16 MHz (ATmega128)。

5.10.2 描 述

Atmega128 是一款基于 AVR RISC 结构的低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令，ATmega128 可以取得 1MIPS/MHz 的性能，从而使得设计人员可以在功耗和执行速度之间取得平衡。

一、结构图

AVR 核将 32 个工作寄存器和丰富的指令集连接在一起。所有的工作寄存器都与 ALU 算术逻单元直接相连，允许在一个时钟周期内执行的单条指令，同时访问两个独立的寄存器。这种结构提高了代码效率，使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。图 5.39 为其结构框图。

ATmega128 具有以下特点：128K 字节具备写操作时可读的在系统可编程 Flash；4K 字节 EEPROM；4K 字节 SRAM；53 个通用 I/O 口；32 个通用工作寄存器；实时计数器(RTC)；4 个具有比较模式和 PWM 的定时器/计数器；2 个 USARTs；一个两线(I^2C)串行接口；一个 8 通道 10 位具有可选增益差分输入的 A/D 转换器；一个带内部振荡器的可编程看门狗定时器；一个 SPI 口；一个符合 IEEE std. 1149.1 标准的 JTAG 测试接口，也可用于访问片内 Debug 系统和编程；6 种可通过软件选择的省电模式。工作于空闲模式时，CPU 将停止运行，而 SRAM、定时器/计数器、SPI 口和中断系统继续工作；掉电模式时，振荡器停止工作，所有功能都被禁止，而寄存器内容得到保留，直到下一个外部中断或硬件复位才退出此状态；省电模式时，异步定时器继续工作，以使用户能在芯片的其余部分处于休眠状态时保持定时器基准；ADC 噪声抑制模式除异步定时器和 ADC 继续工作外，停止 CPU 运行和所有的 I/O 单元，以减少 ADC 转换时的开关噪声。待命模式中除晶振工作外，芯片的其余部分处于休眠状态，这就使得在低功耗的情况下能非常快的启动；在扩展待命模式中，主振荡器和异步定时器继续工作。

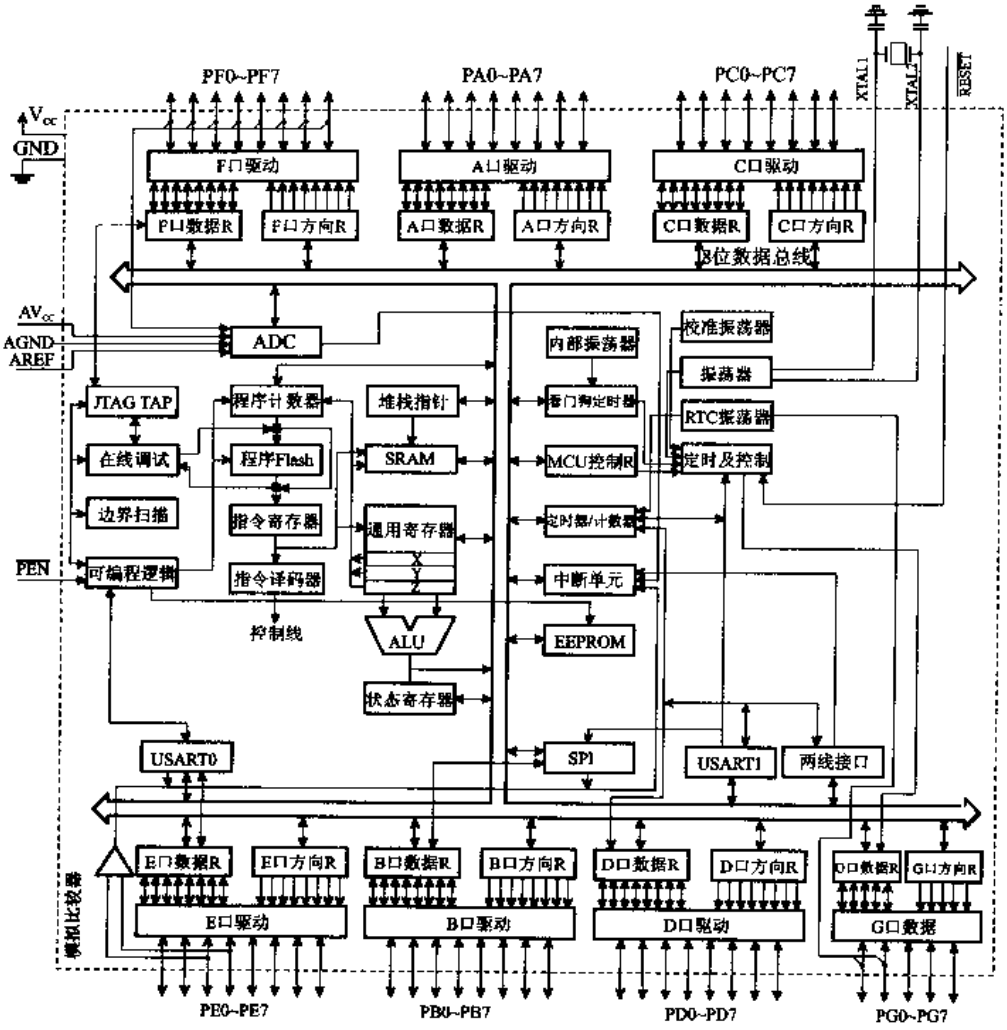


图 5.39 ATmega128 结构框图

芯片是以 ATMEL 的高密度非易失性内存技术生产的。Flash 程序存储器，可以通过 SPI 串行接口，或通用编程器，或运行于 AVR 核上的片内 BOOT 程序，多次编程。BOOT 程序可以用任意的接口下载到应用 Flash 程序存储器中。当应用程序区被更新时，BOOT 区的软件将继续运行，提供写操作时真正可读的功能。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内，ATmega8 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega128 具有一整套的编程和系统开发工具：C 编译器、宏汇编器、调试/模拟器、JTAG ICE 在线仿真器和 SL - MEGA128 评估板。

二、ATmega103 和 ATmega128 的兼容性

ATmega128 是一种很复杂的微控制器，它的 I/O 地址取代了保留在 AVR 指令集中的 64 个 I/O 地址。为确保向后兼容 ATmega103，ATmega103 上所有 I/O 的位置与 ATmega128 上的相同。很多附加的 I/O 地址被加到一个从 $60 到 $FF 的扩展外部 I/O 空间中(例如，在 ATmega103 的内部 RAM 空间中)。

这些地址只能用 LD/LDS/LDD 和 ST/STS/STD 指令访问，而不能用 IN 和 OUT 指令。对于 ATmega103 用户来说，内部 RAM 空间的重定位，可能仍是一个问题。同样，如果代码使

用绝对地址，那么增加的中断向量也是一个问题。要解决这些问题，可以通过编程一个熔丝 M103C 来选择 ATmega103 兼容模式。

在这一模式下，不能使用扩展 I/O 空间中的程序，所以内部 RAM 像 ATmega103 一样定位。同时，扩展中断向量被去除。ATmega128 与 ATmega103 的引脚百分之百兼容，在 PCB 上可以替代 ATmega103。应用笔记的"用 ATmega128 替换 ATmega103" 中说明了用户在用 ATmega128 替换 ATmega103 时应注意的事项。

三、ATmega103 兼容模式

通过编程 M103C 熔丝，ATmega128 将在上述 RAM、I/O 引脚和中断向量等方面与 ATmega103 兼容。然而，在这一兼容模式下，ATmega128 所拥有的如下的新特性就没有了：

- 仅有异步模式的一个 USART，而不是两个。波特率寄存器只有低八位有效。
- 一个带两个比较寄存器的 16 位定时器/计数器，而不是两个带三个比较寄存器的 16 位定时器/计数器。
- 不支持两线串行接口。
- C 口只能作为输出。
- G 口只有替换功能(而不是通用 I/O 口)。
- F 口只能作为数字输入和 ADC 的模拟输入。
- 不支持 BOOT 装载功能。
- 不能调整内部 RC 振荡器的频率。
- 外部存储器接口不能释放任何地址引脚作为通用 I/O，也不能为不同的外部存储器地址区设置不同的等待状态。

另外，还有与 ATmega103 的如下细小差别：

- MCUCSR 中只有 EXTRF 和 PORF。
- 看门狗超时改变，不要求时序。
- 外部中断脚 3～0 只能作为低电平中断。
- USART 没有 FIFO 缓冲，所以数据溢出更早。

ATmega103 中未用到的 I/O 位应置 0，以确保在 ATmega128 中的相同操作。

5.10.3 引脚配置

ATmega128 的引脚配置如图 5.40 所示。

引脚定义

V_{CC}、GND：电源。

A 口(PA7～PA0)：A 口是一个 8 位双向 I/O 口，每一个引脚都有内部可选上拉电阻。A 口的输出缓冲有对称的驱动特性，包括输入和输出电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在引脚将输出电流。在复位过程中，A 口为三态，即使此时时钟还未起振。A 口还可以用做多种特殊用途，请参阅手册。

B 口(PB7～PB0)：B 口是一个 8 位双向 I/O 口，每一个引脚都有内部可选上拉电阻。B 口的输出缓冲有对称的驱动特性，包括输入和输出电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在引脚将输出电流。在复位过程中，B 口为三态，即使此时时钟还未起振。B 口还可以用做多种特殊用途，请参阅手册。

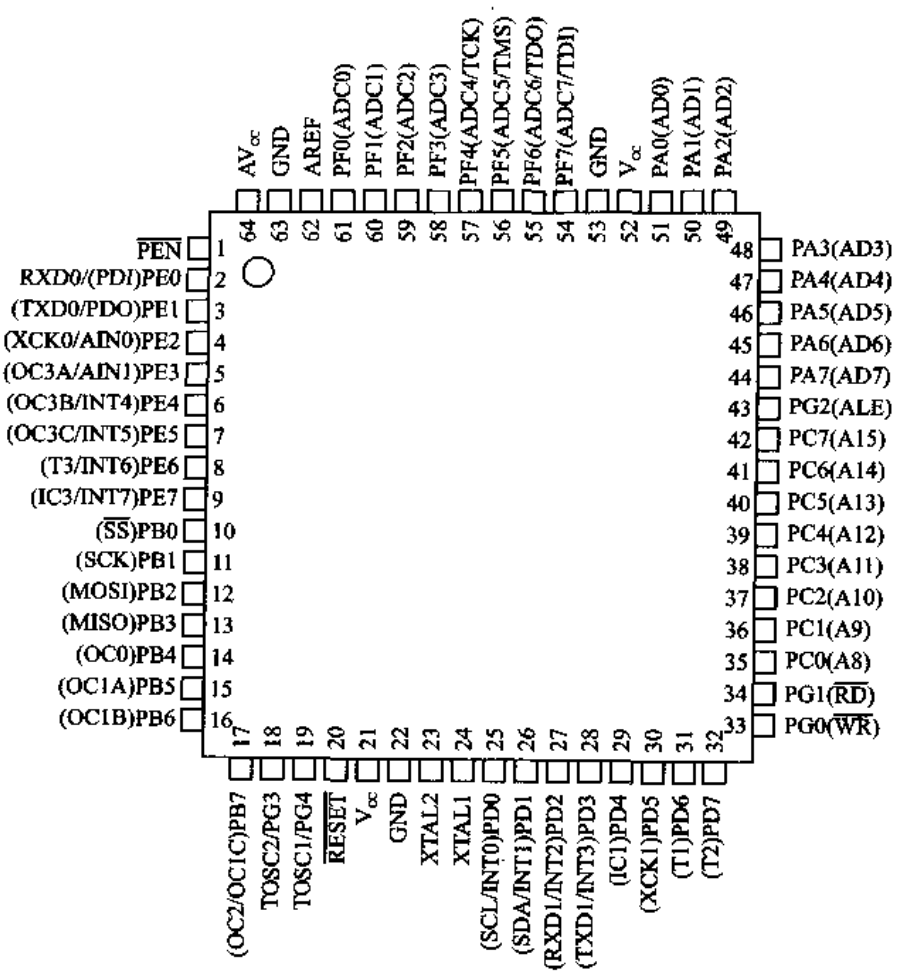


图 5.40 ATmega128 的引脚配置

C 口(PC7～PC0):C 口是一个 8 位双向 I/O 口,每一个引脚都有内部可选上拉电阻。C 口的输出缓冲有对称的驱动特性,包括输入和输出电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在引脚将输出电流。在复位过程中,C 口为三态,即使此时时钟还未起振。C 口还可以用做多种特殊用途,请参阅手册。ATmega103 兼容模式中,C 口只能作为输出,并且在复位过程中,C 口不为三态。

D 口(PD7～PD0):D 口是一个 8 位双向 I/O 口,每一个引脚都有内部可选上拉电阻。D 口的输出缓冲有对称的驱动特性,包括输入和输出电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在引脚将输出电流。在复位过程中,D 口为三态,即使此时时钟还未起振。D 口还可以用做多种特殊用途,请参阅手册。

E 口(PE7～PE0):E 口是一个 8 位双向 I/O 口,每一个引脚都有内部可选上拉电阻。E 口的输出缓冲有对称的驱动特性,包括输入和输出电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在引脚将输出电流。在复位过程中,E 口为三态,即使此时时钟还未起振。E 口还可以用做多种特殊用途,请参阅手册。

F 口(PF7～PF0):F 口作为 A/D 转换器的模拟输入口。如果不使用 A/D 转换器,F 口也可以作为一个 8 位双向 I/O 口。每一个引脚都有内部可选上拉电阻。F 口的输出缓冲有对称的驱动特性,包括输入和输出电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在

引脚将输出电流。在复位过程中，F 口为三态，即使此时时钟还未起振。如果 JTAG 接口使能，即使发生复位，PF7(TDI)、PF5(TMS)和 PF4(TCK)上的上拉电阻将被激活。F 口也具有 JTAG 接口的功能。ATmega103 兼容模式中，F 口只能作为输入。

G 口(PG4～PG0)：G 口是一个 5 位双向 I/O 口，每一个引脚都有内部可选上拉电阻。G 口的输出缓冲有对称的驱动特性，包括输入和输出电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在引脚将输出电流。在复位过程中，G 口为三态，即使此时时钟还未起振。G 口还可以用做多种特殊用途，请参阅手册。ATmega103 兼容模式中，这些引脚只作为外部存储器的频闪信号和 32 kHz 振荡器的输入，即使此时时钟还未起振。在复位过程中，引脚被异步地初始化为 PG0 = 1，PG1 = 1，PG2 = 0，PG3 和 PG4 是晶振脚。

RESET：复位输入。即使此时时钟还未起振，超过最小脉冲宽度的低电平将引起系统复位，最小脉冲宽度请参阅资料手册。低于最小脉冲宽度的脉冲不能保证可靠复位。

XTAL1：反向振荡放大器的输入和内部时钟工作电路的输入。

XTAL2：反向振荡放大器的输出。

AV_{CC}：AV_{CC}是 F 口和 A/D 转换器的电源端。即使不使用 ADC，也应外接到 V_{CC}端；如使用 ADC，应该通过一个低通滤波器与 V_{CC}连接。

AREF：A/D 转换器的参考电源。

PEN：PEN 是串行下载的编程使能信号。上电复位时通过保持 PEN 为低电平，芯片会进入 SPI 串行编程模式。正常操作时无任何功能。

5.10.4 开发实验工具

SL-MEGA 高档仿真开发实验器，可对高档 AVR 单片机 ATmega128 进行仿真、ISP/JTAG 下载编程、开发实验，本机也可做科研样机使用。配有主芯片 ATmega128(也可选配 ATmega103)芯片；主芯片的 I/O 口均可外引，供用户做 I/O 口输入/输出实验；有 8 路开关可作输入信号；8 只 LED 发光二极管作输出显示用；有 4 位 LED 数码管；OCMJ4X8 汉字 LCD 液晶显示器，自带 GB2312 16×16 点阵国标一级简体汉字和 ASCII8X8(16×16)英文字库，用户输入区位码或 ASCII 码即可实现文本显示(另文详解 OCMJ4X8 汉字 LCD 液晶显示器)；有 20 键的键盘，也可接 PC 机通用键盘(购机附通用键盘操作原代码)；硬件电路采用模块式结构，用短路块、接插件、专用接线连接；64 脚 TQFP 封装的主芯片(AT mega103/128 单片机)，通过转换接口板与主机相连；有双路 RS232 接口，一路为串行 ISP 下载通信接口，可对单片机实现 ISP 下载编程，另一路为标准 RS232 接口，供用户做通信实验；ATmega128 配 JTAG 接口(JTAG ICE 仿真器为选购件)，解决器件仿真调试、ISP 编程工作；有 2 路 A/D 模拟电压输入接口电路(通过跳线可做全部 8 路的 A/D 实验)；2 路 PWM 输出接口电路(通过跳线可做全部 PWM 实验)；有音响器；电源模块、电源开关、电源指示、复位电源管理器件，保障复位的可靠性。

SL-MEGA128 仿真开发实验器是功能齐全的 AVR 单片机仿真(软件模拟仿真或 JTAG ICE 在线实时仿真)、开发、编程(ISP 下载)、实验工具，使 AVR 单片机开发应用提高到一个崭新水平！

5.11 ATmega161

5.11.1 特 点

(1) AVR RISC 结构：
——高性能、低功耗 RISC 结构；
——130 条指令，大多数为单指令周期；
——32 个 8 位通用(工作)寄存器；
——工作在 8MHz 时具有 8MIPS 的性能；
——只需 2 个时钟的乘法器。
(2) 数据和非易失性程序内存：
——16K 字节的在线可编程 Flash(擦除次数为 1 000 次)；
——1K 字节存储器；
——512 字节在线可编程 EEPROM(寿命为 100 000 次)；
——程序加密位；
——具有独立加密位的 BOOT 代码区。
(3) 外围(Peripheral)特点：
——2 个具有比较模式的可预分频(Prescale)8 位定时器/计数器；
——1 个可预分频、具有比较、捕捉和 2 个 8/9/10 位 PWM 功能的 16 位定时器/计数器；
——双串口；
——主/从 SPI 接口；
—— 自具振荡器的实时时钟 RTC；
——可编程的看门狗定时器(由片内振荡器生成)；
——片内模拟比较器。
(4) MCU 特点：
——上电复位和可编程的电源检测；
——低功耗空闲、省电和掉电模式；
——内外部中断源。
(5) 在 4MHz、3V、25℃条件下的功耗：
——工作模式为 3.0mA；
——空闲模式为 1.2mA；
——掉电模式为 15μA。
(6) I/O 和封装：
——35 个可编程的 I/O 口线；
——40 脚 PDIP、44 脚 PLCC 及 QFP 封装。
(7) 工作电压：
—— 2.7～5.5V(ATmega161L)；
——4.0～5.5V(ATmega161)。

(8) 速度:

——0～4MHz(ATmega161L);

—— 0～8MHz(ATmega161)。

5.11.2 描 述

ATmega161 是一款基于 AVR RISC 的、低功耗 CMOS 8 位单片机。通过在一个时钟周期内执行一条指令,ATmega161 可以取得接近 1MIPS/MHz 的性能,从而使得设计人员可以在功耗和执行速度之间取得平衡。AVR 核将 32 个工作寄存器和丰富的指令集联结在一起。所有的工作寄存器都与 ALU(算逻单元)直接相连,允许在 1 个时钟周期内执行的单条指令同时访问 2 个独立的寄存器。这种结构提高了代码效率,使 AVR 得到了比普通 CISC 单片机高将近 10 倍的性能。其结构框图如图 5.41。

ATmega161 具有以下特点:16K 字节在线编程/自编程的 Flash;512 字节 EEPROM; 1K 字节 SRAM 存储器;35 个通用 I/O 口; 32 个通用工作寄存器;实时时钟 RTC;3 个具有比较模式的、灵活的定时器/计数器;内外中断源;2 个可编程的 UART;可编程的看门狗定时器;SPI 口以及 3 种可通过软件选择的省电模式。工作于空闲模式时,CPU 将停止运行,而寄存器、定时器/计数器、看门狗和中断系统继续工作;掉电模式时振荡器停止工作,所有功能都被禁止,而寄存器内容得到保留。只有外部低电平中断或硬件复位才可以退出此状态。省电模式与掉电模式只有一点差别:省电模式下 T/C2 继续工作以维持时间基准。

器件是以 ATMEL 的高密度、非易失性内存技术生产的。片内 Flash 可以通过 SPI 接口或通用编程器多次编程。通过将增强的 RISC 8 位 CPU 与 Flash 集成在一个芯片内,ATmega161 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega161 具有一整套的编程和系统开发工具:宏汇编、调试/仿真器、在线仿真器和评估板。

5.11.3 引脚配置

ATmega 161 引脚配置如图 5.42。

一、引脚定义

V_{CC}、GND:电源。

A 口(PA7～PA0):A 口是一个 8 位双向 I/O 口,每一个引脚都有内部上拉电阻。A 口的输出缓冲器能够吸收 20mA 的电流,可直接驱动 LED。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,A 口为三态,即使此时时钟还未起振。在访问外部存储器时,A 口作为地址/数据复用口。

B 口(PB7～PB0):B 口是一个 8 位双向 I/O 口,每一个引脚都有内部上拉电阻。B 口的输出缓冲器能够吸收 20mA 的电流,可直接驱动 LED。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,B 口为三态,即使此时时钟还未起振。B 口作为特殊功能口的使用方法见光盘文件。

C 口(PC7～PC0):C 口是一个 8 位双向 I/O 口,每一个引脚都有内部上拉电阻。C 口的输出缓冲器能够吸收 20mA 的电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,C 口为三态,即使此时时钟还未起振。在访问外部存储器时

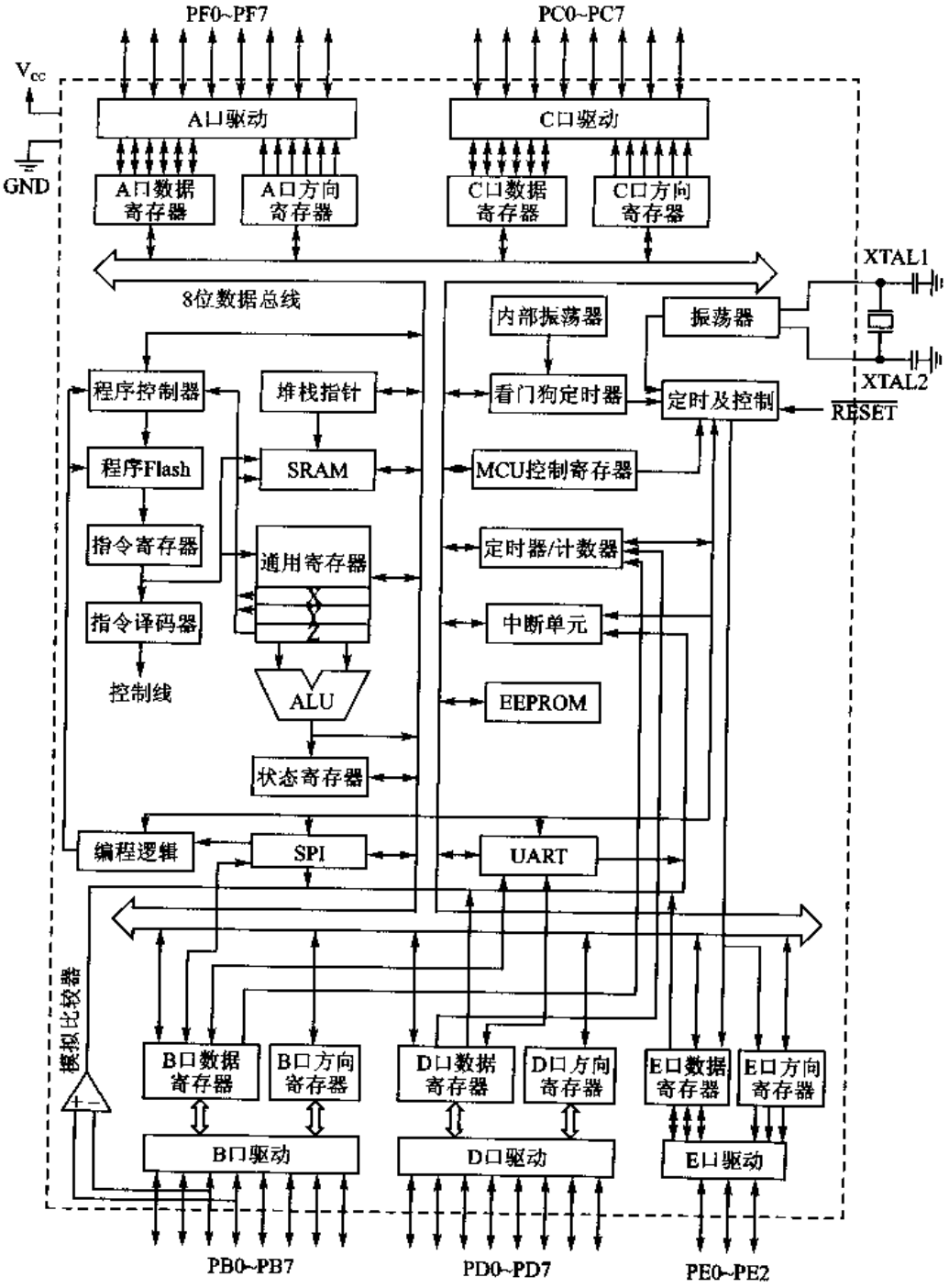


图 5.41 ATmega161 结构框图

C 口作为高 8 位地址线。

D 口(PD7～PD0):D 口是一个带内部上拉电阻的 8 位双向 I/O 口。输出缓冲器能够吸收 20mA 的电流。当作为输入时,如果外部被拉低,由于上拉电阻的存在,引脚将输出电流。在复位过程中,D 口为三态,即使此时时钟还未起振。D 口作为特殊功能口的使用方法见光盘文件。

E 口(PE2～PE0):E 口是一个带内部上拉电阻的 3 位双向 I/O 口。输出缓冲器能够吸收

20mA 的电流。当作为输入时，如果外部被拉低，由于上拉电阻的存在，引脚将输出电流。在复位过程中，D 口为三态，即使此时时钟还未起振。D 口作为特殊功能口的使用方法见光盘文件。

$\overline{\text{RESET}}$：复位输入。超过 500ns 的低电平将产生复位信号，即使此时时钟还未起振。低于 500ns 的脉冲不能保证可靠复位。

XTAL1：振荡器放大器的输入端。

XTAL2：振荡器放大器的输出端。

二、晶振器

晶体振荡器：XTAL1 和 XTAL2 分别是片内振荡器的输入、输出端，可使用晶体振荡器或陶瓷振荡器。当使用外部时钟时，XTAL2 应悬空。

定时器振荡器：晶振可以直接连接到振荡器的引脚 TOSC1 和 TOSC2 上，而无需外部电容。振荡器已经对 32 768Hz 的晶振作了优化。对外加信号的带宽为 256kHz。

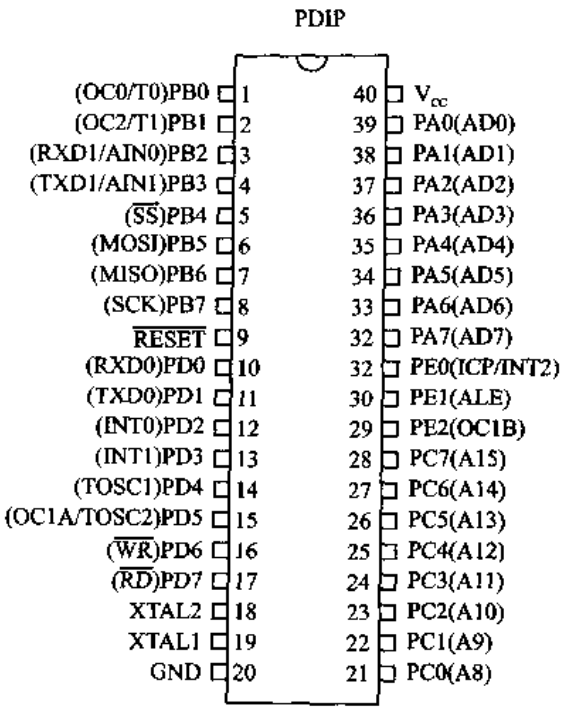


图 5.42　ATmega161 引脚配置图

5.12　AVR 单片机替代 MCS－51 单片机

高速嵌入式 AVR AT90S 系列单片机的独特性能，得到我国科技界认可，广泛应用到高科技含量的产品中去。尤其 AVR 单片机有些器件硬件设计与 MCS－51 系列引脚兼容，仅复位电平要求不同：MCS－51 高电平复位，AVR 低电平复位。我们可以很容易把 MCS－51 单片机的硬件电路用 AVR 单片机替代，并增加了很多新功能，如：多累加器型、高速数据处理、I/O 口驱动能力强、看门狗、实时时钟、A/D、PWM、模拟比较器、低功耗和在线下载等等功能。表 5.35 为 MCS－51 单片机与 AVR 单片机替换表。

表 5.35　MCS－51 单片机与 AVR 单片机替换表

MCS－51 系列单片机	AVR 单片机	程序存储器/KB
89C1051	AT90S1200	1
89C2051	AT90S2313	2
8751/89C51	AT90S4414	4
8752/89C52/8XC51FA/FB	AT90S8515	8
8XC504/8XC524	ATmega161	16

第六章　实用程序设计

6.1　程序设计方法

程序设计就是用计算机所能接受的语言，把解决问题的步骤描述出来，也就是编制计算机的程序。AVR 单片机程序设计语言有：C 编译高级语言、BASCOM－AVR 高级语言和宏汇编汇编语言。

在设计应用系统时，软件的编制是重要环节，软件的质量直接影响整个系统功能的实现。本章从应用的角度出发，介绍一些实用子程序。读者既可按需要改编调用，又可以吸收其设计方法，以便更好地设计出适合于自己系统的实用软件。

6.1.1　程序设计步骤

应用程序的设计因系统而异，因人而异。尽管如此，程序设计还是有共同特点及其规律的。在编写程序时，设计人员可以采取如下几个步骤：

(1)分析问题，明确所要解决问题的要求。将软件分成若干个相对独立的部分，根据功能关系和时序关系，设计出合理的软件总体结构。

(2)建立正确的数学模型。根据功能要求，描述出各个输入和输出变量之间的数学关系，并确定采用的计算公式和计算方法。

(3)制定程序框图。根据所选择的计算方法，制定出运算的步骤和顺序，并画出程序框图。这不仅是程序设计的一个重要组成部分，而且是决定成败的关键部分。

(4)合理分配系统资源，包括程序 Flash、EEPROM、SRAM、定时器/计数器、中断、堆栈等。确定数据格式，分配好工作单元，进一步将程序框图画成详细的操作流程。

(5)根据程序的流程图和指令系统，编写出程序。注意在程序的有关位置处写上功能注释，提高程序的可读性。

(6)程序调试。通过编辑软件编辑出的源程序，必须用编译程序汇编后生成目标代码。如果源程序有语法错误，需修改源文件后继续编译，直到无语法错误为止。这之后利用目标码通过仿真器进行程序调试，排除设计和编程中的错误，直到成功。

(7)程序优化。使各功能程序实行模块化、子程序化，缩短程序的长度，加快运算速度和节省数据存储空间，缩短程序执行的时间。

6.1.2　程序设计技术

1. 模块化程序设计

模块化程序设计是单片机应用中常用的一种程序设计技术。它是把功能完整的、较长的程序，分解为若干个功能相对独立的、较小的程序模块，对各个程序模块分别进行设计、编程和调试，最后把各功能模块集成为所需的程序。

模块化程序设计的优点是，单个功能明确的程序模块的设计和调试比较方便、容易完成。一个模块可以为多个程序所共享，也可利用现成的程序模块。

2. 自上而下的程序设计

自上而下的程序设计时，先从主程序开始设计，从属的程序和子程序用符号来代替。主程序编好后，再编制各个从属程序和子程序，最后完成整个系统软件的设计。调试也按这个次序进行。

自上而下程序设计的优点是，比较习惯人们的思维，设计、调试和连接同时按一个线索进行，程序错误可以较早发现。缺点是修改比较麻烦。

3. 软件抗干扰设计

用于生产现场的单片机应用系统，易受各种干扰侵袭，直接影响到系统的可靠性。因此，应用系统的抗干扰设计是非常重要的。

在实际情况中，针对不同的干扰后果，采用不同的软件对策。在实时数据采集系统中，为了消除传感器通道中的干扰信号，可采用软件数据滤波，如算术平均法、比较舍取法、中值法、一阶递推数字滤波法等。在开关量控制系统中，为防止干扰进人系统，造成各种控制条件超差、输出失控，可采取软件冗余、程序自检等措施。为防止程序计数器失控，造成程序盲目运行或“死机”，可设置软件“看门狗”以监视程序运行状态；也可在非程序区设置软件陷阱，强行使程序拉回复位状态，重新启动。

6.2　应用程序举例

应用程序中包括一些算术运算、代码转换、数据传送、滤波算法、串行通信、A/D 转换子程序。根据需要，可以应用其中的一些子程序。

6.2.1　内部寄存器和位定义文件

AVR 单片机内部寄存器和位定义文件，用于定义器件内部的寄存器名和寄存器的位名。在汇编程序文件中如包括了定义文件，则所有数据块中 I/O 寄存器名和 I/O 寄存器位名都能使用。寄存器名用十六进制地址表示；寄存器位名用 0～7 位表示。

另外，被配置命名的 XL～ZH 6 个寄存器形成 3 个数据指针 X、Y 和 Z，作为内部 SRAM 高端 RAM 地址同样被定义。

注意，在指令中使用的位名意义是不同的。如"sbr"/"cbr"指令表示，若位置位/清零则跳行执行。

一、AT90S1200 单片机内部寄存器和位定义文件

详见光盘 AT90S1200 器件配置文件 ＊:\AVR 组合软件\asmpack\appnotes\1200def.inc

在源文件编译时，源文件所在的文件夹中一定要有相应器件配置文件存在，不然将造成编译出错提示！图 6.1 为该示意图。

```
;***************************************************************************
;* 编号 :AVR000                          ;* 日期 :99.01.28
;* 文件名:"1200def.inc"                  ;* 版本 :1.30
;* 标题: AT90S1200 的寄存器/位定义       ;* 技术支持热线:+47 72 88 43 88 (ATMEL Norway)
```

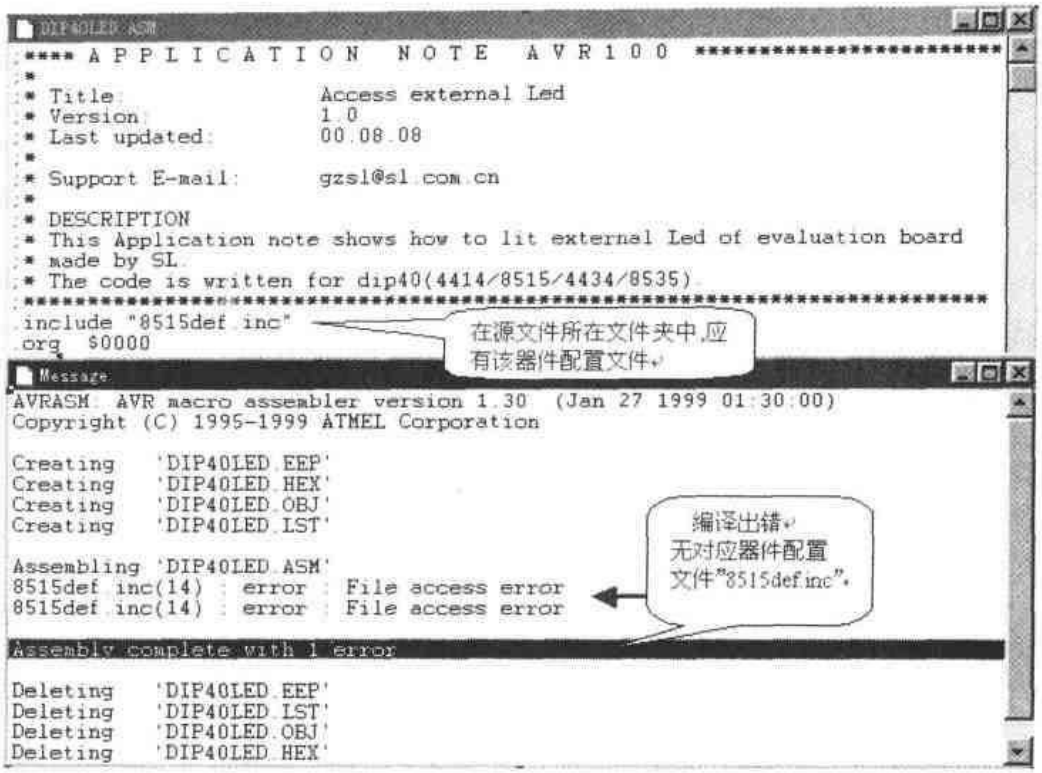


图 6.1 编译出错提示示意图

```
;*  技术支持传真:+47 72 88 43 99 (ATMEL Norway)
;*  技术支持 E-mail:avr@atmel.com
;*  目标 MCU:AT90S1200
;*  说明:
;*  当此文件包含在汇编文件中时,所有 I/O 寄存器
    名和 I/O 寄存器位名出现在数据书中以供使用。
    寄存器名用它们的十六进制地址代表。寄存器
    位名用它们的位编号(0～7)代表。请观察在指
    令"sbr"/"cbr"(置/清除寄存器中的位)和
    "sbrs"/"sbrc"(如寄存器中的位被置 1/清除则
    跳过)等中使用位名的差别。
;*  以下示例说明了这一点:
;*  in r16,PORTB        ;读取 PORTB latch
;*  sbr     r16,(1<<PB6)+(1<<PB5)
                        ;PB6 和 PB5 置 1(使用
                         屏蔽,而不是 bit#)
;*  out   PORTB,r16     ;输出到 PORTB
;*
;*  in     r16,TIFR     ;读取定时器中断标志寄
                         存器
;*  sbrc     r16,TOV0   ;检查溢出标志(用 bit#)
;*  rjmp     TOV0_is_set ;如为 1 则跳转
;*  ...                 ;否则...
;*********************
;*****  指定器件
.device AT90S1200
;*****  I/O 寄存器定义
.equ    SREG    = $3f
.equ    GIMSK   = $3b
.equ    TIMSK   = $39
.equ    TIFR    = $38
.equ    MCUCR   = $35
.equ    TCCR0   = $33
.equ    TCNT0   = $32
.equ    WDTCR   = $21
.equ    EEAR    = $1e
.equ    EEDR    = $1d
.equ    EECR    = $1c
.equ    PORTB   = $18
.equ    DDRB    = $17
.equ    PINB    = $16
.equ    PORTD   = $12
.equ    DDRD    = $11
.equ    PIND    = $10
```

```
.equ    ACSR        =$08
;*****  位定义
.equ    INT0        =6
.equ    TOIE0       =1
.equ    TOV0        =1
.equ    SE          =5
.equ    SM          =4
.equ    ISC01       =1
.equ    ISC00       =0
.equ    CS02        =2
.equ    CS01        =1
.equ    CS00        =0
.equ    WDE         =3
.equ    WDP2        =2
.equ    WDP1        =1
.equ    WDP0        =0
.equ    EEWE        =1
.equ    EERE        =0
.equ    PB7         =7
.equ    PB6         =6
.equ    PB5         =5
.equ    PB4         =4
.equ    PB3         =3
.equ    PB2         =2
.equ    PB1         =1
.equ    PB0         =0
.equ    DDB7        =7
.equ    DDB6        =6
.equ    DDB5        =5
.equ    DDB4        =4
.equ    DDB3        =3
.equ    DDB2        =2
.equ    DDB1        =1
.equ    DDB0        =0
.equ    PINB7       =7
.equ    PINB6       =6
.equ    PINB5       =5
.equ    PINB4       =4
.equ    PINB3       =3
.equ    PINB2       =2
.equ    PINB1       =1
.equ    PINB0       =0
.equ    PD6         =6
.equ    PD5         =5
.equ    PD4         =4
.equ    PD3         =3
.equ    PD2         =2
.equ    PD1         =1
.equ    PD0         =0
.equ    DDD6        =6
.equ    DDD5        =5
.equ    DDD4        =4
.equ    DDD3        =3
.equ    DDD2        =2
.equ    DDD1        =1
.equ    DDD0        =0
.equ    PIND6       =6
.equ    PIND5       =5
.equ    PIND4       =4
.equ    PIND3       =3
.equ    PIND2       =2
.equ    PIND1       =1
.equ    PIND0       =0
.equ    ACD         =7
.equ    ACO         =5
.equ    ACI         =4
.equ    ACIE        =3
.equ    ACIS1       =1
.equ    ACIS0       =0
.equ    XRAMEND     =0
.equ    E2END       =3F
.equ    FLASHEND    =1FF
.equ    INT0addr    =$001   ;外部中断0向量
                            地址
.equ    OVF0addr    =$002   ;溢出0中断向量
                            地址
.equ    ACIaddr     =$003   ;模拟比较器中断
                            向量地址
.def    ZL          =r30
```

二、AT90S2313 单片机内部寄存器和位定义文件

详见光盘 AT90S2313 器件配置文件 *:\AVR 组合软件\asmpack\appnotes\2313def.inc

三、AT90S4414 单片机内部寄存器和位定义文件

详见光盘 AT90S4414 器件配置文件 *:\AVR 组合软件\asmpack\appnotes\4414def.inc

四、AT90S8515 单片机内部寄存器和位定义文件

详见光盘 AT90S8515 器件配置文件 *:\AVR 组合软件\asmpack\appnotes\8515def.inc

五、AT90S8535 单片机内部寄存器和位定义文件

详见光盘 AT90S8535 器件配置文件 *:\AVR 组合软件\asmpack\appnotes\8535def.inc

图 6.2 为 AVR 器件配置文件界面图。

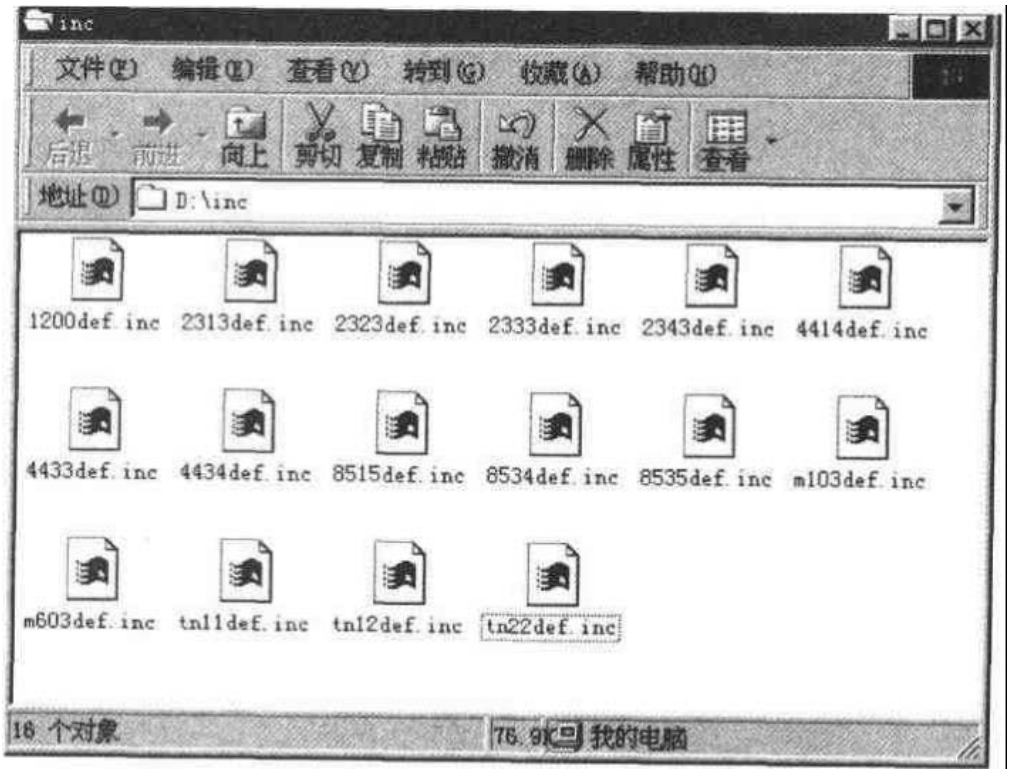


图 6.2 AVR 器件配置文件

6.2.2 访问内部 EEPROM

见光盘文件 *:\AVR 组合软件\asmpack\appnotes\AVR100.ASM

该应用程序说明,如何读 EEPROM 数据和写数据到 EEPROM。其中包括:写 EEPROM 子程序"EEWrite",读 EEPROM 子程序"EERead",按序写 EEPROM 子程序"EEWrite_seq";按序读 EEPROM 子程序"EERead_seq"和一个测试程序。

6.2.3 数据块传送

见光盘文件 *:\AVR 组合软件\asmpack\appnotes\AVR102.ASM

该应用程序说明,如何传送程序存储器的一个数据块到 SRAM 和把 SRAM 中的一个数据块传送到另一个 SRAM。其中包括:一个程序存储器块传送子程序"flash 2ram";SRAM 块

传送子程序“ram 2ram”和一个测试程序。

6.2.4　乘法和除法运算应用一

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR200. ASM

该应用程序列出了 8×8 位无符号乘法子程序;16×16 位无符号乘法子程序;8/8 位无符号除法子程序;16/16 位无符号除法子程序和一个测试程序。

6.2.5　乘法和除法运算应用二

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR200B. ASM

该应用程序列出了 8×8 位无符号乘法子程序,8×8 位带符号乘;16×16 位无符号乘法子程序,16×16 位带符号乘;8/8 位无符号除法子程序,8/8 位带符号除;16/16 位无符号除法子程序,16/16 位带符号除和一个测试程序。

6.2.6　16 位运算

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR202. ASM

该应用程序列出了 16 位加法、16 位带立即数加法;16 位减法、16 位带立即数减法;16 位比较、16 位带立即数比较;16 位取反子程序和一个测试程序。

6.2.7　BCD 运算

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR204. ASM

该应用程序列出了 16 位二进制转换成 BCD、8 位二进制转换成 BCD;BCD 转换成 16 位二进制、BCD 转换成 8 位二进制;二位数压缩 BCD 加法、二位数压缩 BCD 减法和一个测试程序。

6.2.8　冒泡分类算法

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR220. ASM

该应用程序列出了如何用代码有效冒泡分类算法分类 SRAM 的数据块子程序和一个测试程序。

6.2.9　设置和使用模拟比较器

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR400. ASM

该应用程序列出设置和使用模拟比较器的子程序。

6.2.10　半双工中断方式 UART 应用一

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR304. ASM

该应用程序包含了一个非常有效的 UART 软件,并给出一个接收字符,然后返回一个应答的子程序。

6.2.11 半双工中断方式 UART 应用二

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR305. ASM

该应用程序给出一个用 8 位定时器/计数器 0 和外部中断实现半双工 UART 的软件。

6.2.12 8 位精度 A/D 转换器

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR401. ASM

该程序显示如何使用单片机片内模拟比较器和几个外部元件实现双斜率 A/D 转换器，包括一个测试程序，完成外循环的转换并输出到 8 位 LED 显示。

6.2.13 装载程序存储器

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR108. ASM

```
; *  标题:装载程序存储器
; *  版本:1.0
; *  最后更新:98.02.27
; *  目标器件:AT90Sxx1x 及更高档的器件(带 SRAM 的器件)
; *  技术支持 E mail:avr@atmel.com
; *  说明:
; *  此应用程序说明怎样使用装载程序存储器(LPM)指令。此应用程序从程序存储器中一字节一字节地装入字符串 "Hello World",并输出到 B 口。
```

6.2.14 安装和使用相同模拟比较器

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR128. ASM

```
; *  标题:安装和使用模拟比较器
; *  版本:1.1
; *  最后更新:97.07.04
; *  目标器件:AT90Sxxxx(带模拟比较器的器件)
; *  技术支持 E-mail:avr@atmel.com
; *  说明:
; *  此应用程序说明怎样激活和使用 AVR 自带的精密模拟比较器。
; *  此程序执行以下操作:
; *  轮流检测模拟比较器的输出以等待正输出边沿;轮流检测中断标志以等待正输出边沿;比较器输出模式下中断使能。每次中断程序执行时将一个 16 位计数寄存器加 1。
```

6.2.15 CRC 程序存储的检查

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR235. ASM

```
; *  标题:CRC 程序存储器校验
; *  版本:1.0
; *  最后更新:98.06.15
; *  目标器件:AT90Sxxxx(所有支持 LPM 指令的 AVR 器件,除了 AT90S1200)
; *  技术支持 E-mail:avr@atmel.com
; *  注意:经常登录 Atmel 的网站, www.atmel.com 以获取软件的最新版本。
; *  说明:
; *  此程序说明怎样利用一个简单的算法来实现代码存储器内容的 CRC 计算。要产生 CRC 校验,则寄存器"状态"装入 00,并调用"crc_gen"程序。校验的结果存于寄存器的第二字节(低字节)和第三字节(高字节)。要检查 CRC 校验,寄存器"状态"装入 FF,并调用"crc_gen"程序。校验的结果存于寄存器的第二字节(低字节)和第三字节(高字节)。如果校验的结果为 00,则程序代码是正确的;如果校验的结果不是 00,则程序代码有错误。
```

6.2.16　4×4 键区休眠触发方式

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR240. ASM

```
; *  标题:4×4 键盘,按键唤醒
; *  版本:1.1
; *  最后更新:98.10.22
; *  目标器件:所有 AVR 器件
; *  技术支持 E-mail:avr@atmel.com
; *  说明:
; *  此程序用来扫描一个 4×4 键盘,并用休眠模式使 AVR 由按键唤醒。此设计使用最少的外部器件。测试程序唤醒 AVR 并扫描,当有键按下时,根据所按键的键号点亮 2 个 LED 中的 1 个。外部中断线作唤醒之用。
; *  此例应用于 AT90S1200,但在向量、EEPROM 和堆栈指针上作一些适当的改动即适用于任何 AVR 器件。
; *  定时采用一个 4 MHz 的时钟。
; *  一个查询表用于 EEPROM 中,以使同样的结构被用于更高级的程序,例如 ASCII 输出到显示。
```

6.2.17　多工法驱动 LED 和 4×4 键区扫描

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR242. ASM

```
; *  标题:多路 LED 驱动和 4×4 键盘采样
; *  版本:1.0
; *  最后更新:98.07.24
; *  目标器件:所有 AVR 器件
; *  技术支持 E-mail:avr@atmel.com
; *  说明:
; *  此程序包括一个提供 24 小时工业定时器或以 I/O 引脚作双重作用的实时时钟。通过 4×4 矩阵键盘输入,输出到多元四位 LED 显示并通过附加接口电路由两开/关输出驱动负载。此例中驱动的是 LED 负载,但附加适当的器件就可驱动任何负载。
; *  触觉反馈是通过按键时,一个压电音响器发出嘟嘟响声来实现的。
; *  主程序允许通过键盘设定时钟,每 24 小时为每个负载进行开/关时间设定。
; *  实时时钟功能、键扫描功能和调整程序。
; *  此例以 AT90S1200 来显示怎样克服有限的 I/O,此例同样适用于任何 AVR 器件,仅需对矢量、EEPROM 和堆栈指针作适当的修改。
; *  定时采用一个 4.096 MHz 晶振(如果 178 代替 176 用在定时器装载次序中,4 MHz 晶振会产生 0.16%的误差,但这可以在软件中以规则的时间间隔来调整)。
; *  查询表格用在 EEPROM 中解码显示数据,附加的字符用于定时和开/关设定显示及键盘转换表。
; *  如果应用到 EEPROM,则表格可移到较大的 AVR 器件的 ROM 中。
```

6.2.18　I^2C 总线

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR300. ASM

```
; *  标题 :I²C (单)主器件执行
; *  版本 : 1.0 (BETA)
; *  最后更新: 97.08.27
; *  目标器件: AT90Sxxxx (所有 AVR 器件)
; *  技术支持 E-mail: avr@atmel.com
; *  说明:
; *  与 I²C 从器件通信的基本程序。在 I²C 总线上,单一主器件执行受限制于单一总线控制器。绝大部分应用并不需要 I²C 总线提供的多主器件功能。到目前为止,单一主器件执行使用较少的资源并有较少的晶体频率依靠。
; *  特点:
; *  全部中断可用。可用于其他用途。支持正常和快速模式;支持 7bit 和 10bit 寻址;支持 AVR 全系列微控制器。
; *  I²C 主程序 :
```

;* 'i²c_start': 发出起始条件并发送地址和传递方向。
;* 'i²c_rep_start': 发出重复的起始条件并发送地址和传递方向。
;* 'i²c_do_transfer': 根据 address/dir 字节中给出的方向发送或接收数据。
;* 'i²c_stop': 通过发出停止条件终止数据传递。
;* 用法:
;* 传递格式如 AVR300 文件中所述。(主代码中有一例)
;* 注意:
;* I²C 程序可被非中断或中断程序的任意一个调用,非两者同时调用。
;* 统计:
;* 代码量 : 81 个字 (最多)
;* 寄存器占用 : 4 高, 0 低
;* 中断占用:无
;* 其他占用: D 口的两个 I/O 引脚
;* XTAL 范围: N/A

6.2.19 I²C 工作

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR302. ASM

;* 标题: I²C 从器件执行
;* 版本: 1.2
;* 最后更新: 97.07.17
;* 目标器件: AT90Sxxxx (所有 AVR 器件)
;* 快速模式:仅 AT90S1200
;* 技术支持 E-mail: avr@atmel. com
;* 代码量: 160 个字
;* 低寄存器占用: 0
;* 高寄存器占用: 5
;* 中断占用:外部中断 0,定时器/计数器 0 溢出中断
;* 说明:
;* 此程序说明,怎样实现 AVR AT90S1200 作为一个 I²C 从外部设备的应用。如使用其他 AT90S AVR 器件,则可能不得不用到其他的 I/O 引脚,堆栈指针也必须初始化。
;* 接收的数据存储在 R0 中,R0 中的数据在主器件读传递中被传输。
;* 这个简单的协议仅用于演示的目的。
;* 特点 :
;* 基于使用 INT0 和 TIM0 的中断;器件可接收任何的 7 位地址(可扩展到 10 位);支持标准和快速模式;易插入"等待状态";容量和速度优化;支持从闲置模式唤醒。
;* 参考应用程序 AVR302 文件以获得更详细的说明。
;* 用法:在两个标记"插入用户代码"的地方插入用户代码。
;* 注意:每一个中断之间最少会有一条指令被执行。
;* 统计:
;* 代码量: 160
;* 寄存器占用:5 高, 0 低
;* 中断占用: EXT_INT0 和 TIM0_OVF
;* 端口占用: PD4(T0)和 PD2(INT0)
;* XTAL:I²C 快速模式:最低 16.0MHz
;* I²C 标准模式 :最低 3.0MHz

6.2.20 SPI 软件

见光盘文件 * :\AVR 组合软件\asmpack\appnotes\AVR320. ASM

;* 标题:软件 SPI 主器件
;* 版本 : 1.0
;* 最后更新: 98.04.21
;* 目标器件: AT90S1200
;* Easily modified for: Any AVR microcontroller
;* 技术支持 E-mail:avr@atmel. com
;* 说明:
;* 这是一个集成了 8/16 位字、模式 0、主器件 SPI 的程序。它同时传输和接收 8/16 位字格式的 SPI 数据。数据首先发送和接收 MSB。一个寄存器组用来发送和接收。比如,当一位移出(被传输),空余的位就用来存储新的位。这些程序是低电平接口程序,因而不包含一个命令结构,那是由连接的 SPI 外设决定的。因为有分离的

允许/禁止和读/写字程序，大块的数据只能在禁止$\overline{SS}$之前通过多次调用 RW_SPI 程序来发送。

;* 主程序：

;* init_spi：初始化口线用作 SPI。无需调用，返回。

;* ena_spi：迫使 SCK 为低，激活$\overline{SS}$信号。无需调用，返回。

;* disa_spi：令$\overline{SS}$信号为高（去激活）。无需调用，返回。

;* rw_spi：发送/接收一个 8 位或 16 位数据字。在调用之前必须在(spi_hi，spi_lo)中设置数据发送；返回接收的数据到同一寄存器组中(如是 8 位，仅使用 spi_lo 寄存器)。

;* 变量：

;* spi_hi 和 spi_lo 变量是高和低数据字节。它们可在寄存器文件中的任何地方。临时变量用来位计数，也用作高/低最小脉冲宽度定时。这必须位于上部寄存器，因为使用了一条 IMMEDIATE 模式指令。

;* 备忘 V1.0　　98.04.21 (rgf)发表

6.2.21　验证 SL－AVR 实验器及 AT90S1200 的口功能 1

见光盘文件 *：\AVR 组合软件\asmpack\appnotes\AVRLED. ASM

该程序验证 OK－AVR 万用串行下载开发实验器及 AT90S1200 的 A 口、B 口 LED 灯亮灭程序，也同时验证实验器通讯接口连机正常与否，avrled2. asm 和 avrled3. asm 仅延时常数不同，所以 LED 闪动快慢不同。

6.2.22　验证 SL－AVR 实验器及 AT90S1200 的口功能 2

见光盘文件 *：\AVR 组合软件\asmpack\appnotes\AVRSTEP. ASM

该程序用模拟调试单步验证 AT90S1200 B 口、D 口的输出状态。

6.2.23　验证 SL－AVR 实验器及具有 DIP40 封装的口功能

见光盘文件 *：\AVR 组合软件\asmpack\appnotes\DIP40LED. ASM

该程序验证 OK－AVR 万用串行下载开发实验器及具有 DIP40 封装的 AT90S4414/AT90S8515/AT90S8535 等器件的 A 口、B 口、C 口、D 口 LED 灯亮灭程序，可修改延时常数。

更多的实用实验程序见第七章 AVR 单片机的应用；源程序见\双龙 SL－AVR 电子书\SLAVR 文件夹中。

* 对应第七章的高级语言在双龙电子光盘中作为第十章、第十一章提供给读者及用户。

第七章　AVR 单片机的应用

ATMEL 公司的 AVR 单片机，是增强型 RISC 内载 Flash 的单片机。芯片上的 Flash 存储器附在用户的产品中，可随时编程、再编程，使产品设计容易，更新换代方便。AVR 单片机采用增强的 RISC 结构，使其具有高速处理能力，在一个时钟周期内可执行复杂的指令，1 MHz可实现 1MIPS 的处理能力。AVR 单片机工作电压为 2.7～6.0V，可实现耗电最优化。AVR 单片机广泛应用于计算机外部设备、工业实时控制、仪器仪表、通讯设备、家用电器和宇航设备等许多领域。

本书所提供的实用、实验程序均是在 SL－AVR 开发下载实验器上汇编、调试、下载验证通过的，用户可以放心地学习、修改、移植。今后我们还将在网上或以光盘形式不断提供实用、实验程序的软、硬件资料。

应用实验源程序见文件夹《SLAVR》；应用例子 *.ASM，必须编译生成 *.OBJ 文件才可调试；如要修改 *.ASM，必须修改文件属性，去掉 *.ASM 只读文件属性。

7.1　通用延时子程序

```
; * * * * * * * * * * * * * * * * * * * * * * * * AVR 单片机实用程序 * * * * * * * * * * * * *
;*  标题：通用延时子程序，文件名：DELAY.ASM
;*  版本：1.0
;*  最后更新日期：2000.09.10
;*  支援 E-mail：gzsl@sl.com.cn
;*  描述：
;*  利用寄存器内容减 1 不为 0 转的多级嵌套，只需改变一个寄存器延时常数，就可改变延时时间
;*  作者：SL. Z
;*  程序适用于有 SRAM 的 AVR 单片机
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "8515def.inc"                    ;器件配置文件，绝不可少，不然汇编通不过
.DEF    TEMP1     =R20
.DEF    CON       =R21
.org    $0000
rjmp        RESET                         ;复位
.ORG      $0010                           ;跳过中断区
RESET:    ldi     r16,high(RAMEND)        ;设 AT90S8515 堆栈为 $025F
          OUT       SPH,r16               ;见器件配置文件"8515def.inc"
          ldi     r16,low(RAMEND)
          OUT       SPL,R16
          ser     temp1
          SER       CON                   ;temp1 直接置数 $FF，A 口
```

```
          out     DDRA,templ           ;方向寄存器设定 A 口为输出
          LDI     R16,0X70
LOOP:     OUT     PORTA,TEMP1
          RCALL   DELAY                ;调用通用延时子程序
          EOR     TEMP1,CON            ;异或
          RJMP    LOOP
;注意:以后程序中的通用延时子程序从略!
delay:                          ;通用延时子程序
          push r16              ;进栈需 2t
      L0: push r16              ;进栈需 2t
      L1: push r16              ;进栈需 2t
      L2: push r16              ;进栈需 2t
      L3: dec  r 16             ;-1 需 1t
          brne L3               ;不为 0 转,为 0 顺执,需 1t/2t
          pop  r16              ;出栈需 2t
          dec  r16              ; 1 需 1t
          brne L2               ;不为 0 转,为 0 顺执,需 1t/2t
          pop  r16              ;出栈需 2t
          dec  r16              ;-1 需 1t
          brne L1               ;不为 0 转,为 0 顺执,需 1t/2t
          pop  r16              ;出栈需 2t
          dec  r16              ;-1 需 1t
          brne L0               ;不为 0 转,为 0 顺执,需 1T/2T
          pop  r16              ;出栈需 2t
          ret                   ;子程序返回需 4t
```

一次嵌套循环公式:

$$T = 7X - 1 + \sum_{i=1}^{x}(3x - 1)$$

二次嵌套循环公式:

$$T = 7X - 1 + \sum_{i=1}^{x}(7x - 1) + \sum_{i=1}^{x}\sum_{i=1}^{x}(3x - 1)$$

三次嵌套循环公式:

$$T = 7X - 1 + \sum_{i=1}^{x}(7x - 1) + \sum_{i=1}^{x}\sum_{i=1}^{x}(7x - 1) + \sum_{i=1}^{x}\sum_{i=1}^{x}\sum_{i=1}^{x}(3x - 1)$$

式中:T 为机器周期数,t 为延时时间。AVR 8MHz 晶振时,每个机器周期为 0.125 μs。

$$t = T \times (1\text{ 个机器周期时间})$$

```
;*****************************************************************
;*   二次嵌套通用延时程序
;* if fos=8MHz        time (3.5μs~1s)        ;*    29        2ms
;*      dt            time                   ;*    40        5ms
;*      22            1ms                    ;*    51        10ms
```

```
;*      65          20ms          ;*      144         200ms
;*      90          50ms          ;*      197         500ms
;*      114         100ms         ;*      249         1s

;*********************************************************************
delay:  push    dt                      brne    del2
del1:   push    dt                      pop     dt
del2:   push    dt                      dec     dt
del3:   dec     dt                      brne    del1
        brne    del3                    pop     dt
        pop     dt                      ret
        dec     dt

;*********************************************************************
;*   一次循环通用延时程序
;* if fos=8MHz       time (3.5μs~10ms)
;*      dt           time
;*      71           1ms
;*      101          2ms
;*      161          5ms
;*      228          10ms
;*********************************************************************
delay:  push    dt                      dec     dt
del2:   push    dt                      brne    del2
del3:   dec     dt                      pop     dt
        brne    del3                    ret
        pop     dt
```

三次嵌套通用延时程序，在 8MHz 晶振下测试数据，H 为十六进制，D 为十进制。

通用延时子程序时间常数所对应的延时周期数及时间见表 7.1。

表 7.1 通用延时子程序时间常数所对应的延时周期及时间

H	D	T 周期数	t	H	D	T 周期数	t
1	1	28	3.50 μs	9	9	2 806	350.75 μs
2	2	76	9.50 μs	A	10	3 862	482.75 μs
3	3	166	20.75 μs	B	11	5 200	0.65 ms
4	4	316	39.50 μs	C	12	6 800	0.85 ms
5	5	547	68.38 μs	D	13	8 800	1.10 ms
6	6	883	110.38 μs	E	14	11 221	1.4 ms
7	7	1 351	168.88 μs	F	15	14 077	1.76 ms
8	8	1 981	247.63 μs	10	16	17 443	2.18 ms

表 7.1(续)

H	D	T周期数	t	H	D	T周期数	t
11	17	21 376	2.67 ms	35	53	1 268 191	158.523 9 ms
12	18	25 936	3.24 ms	36	54	1 360 591	170.073 9 ms
13	19	31 186	3.90 ms	37	55	1 457 947	182.243 4 ms
14	20	37 192	4.65 ms	38	56	1 560 433	195.054 1 ms
15	21	41 023	5.50 ms	39	57	1 668 226	208.528 3 ms
16	22	51 751	6.47 ms	3A	58	1 791 506	222.688 3 ms
17	23	60 451	7.56 ms	3B	59	1 900 456	237.557 ms
18	24	70 201	8.78 ms	3C	60	2 025 262	253.157 8 ms
19	25	81 082	10.14 ms	3D	61	2 156 113	269.514 1 ms
1A	26	93 178	11.65 ms	3E	62	2 293 201	286.650 1 ms
1B	27	106 576	13.32 ms	3F	63	2 436 721	304.590 1 ms
1C	28	121 366	15.17 ms	40	64	2 586 871	323.36 ms
1D	29	137 641	17.21 ms	41	65	2 743 852	0.342 981 5 s
1E	30	155 497	19.44 ms	42	66	2 907 868	0.363 483 5 s
1F	31	175 033	21.88 ms	43	67	3 079 126	0.384 890 8
20	32	196 351	24.54 ms	44	68	3 257 836	0.407 229 5 s
21	33	219 556	27.444 5 ms	45	69	3 444 211	0.430 526 4 s
22	34	244 756	30.594 5 ms	46	70	3 638 467	0.454 808 4 s
23	35	272 062	34.007 75 ms	47	71	3 840 823	0.480 102 9 s
24	36	301 588	37.698 5 ms	48	72	4 051 501	0.506 437 6 s
25	37	333 451	41.681 38 ms	49	73	4 270 726	0.533 840 8 s
26	38	367 771	45.971 38 ms	4A	74	4 498 726	0.562 340 8 s
27	39	404 671	50.583 88 ms	4B	75	4 735 732	0.591 966 5 s
28	40	444 277	55.534 63 ms	4C	76	4 981 978	0.622 747 3 s
29	41	486 718	60.839 75 ms	4D	77	5 237 701	0.654 712 6 s
2A	42	532 126	66.515 75 ms	4E	78	5 503 141	0.687 892 6 s
2B	43	580 636	72.579 5 ms	4F	79	5 778 541	0.722 317 6 s
2C	44	632 386	79.048 25 ms	50	80	6 064 147	0.758 018 4 s
2D	45	697 517	85.939 63 ms	51	81	6 360 208	0.795 026 s
2E	46	746 173	93.271 63 ms	52	82	6 666 976	0.833 372 s
2F	47	808 501	101.062 6 ms	53	83	6 984 706	0.873 088 2 s
30	48	874 651	109.331 4 ms	54	84	7 313 656	0.914 207 s
31	49	944 776	118.097 ms	55	85	7 654 087	0.956 760 9 s
32	50	1 019 032	127.379 ms	56	86	8 006 263	1.000 78 s
33	51	1 097 578	137.197 3 ms	57	87	8 370 451	1.046 31 s
34	52	1 180 576	147.572 ms	58	88	8 746 921	1.093 37 s

表 7.1(续)

H	D	T 周期数	t/s	H	D	T 周期数	t/s
59	89	9 135 946	1.141 993	7D	125	31 052 260	4.256 532
5A	90	9 537 802	1.192 225	7E	126	35 125 150	4.390 644
5B	91	9 952 768	1.244 096	7F	127	36 223 200	4.527 9
5C	92	10 381 130	1.297 641	80	128	37 346 790	4.668 349
5D	93	10 823 160	1.352 895	81	129	38 496 320	4.812 039
5E	94	11 279 160	1.409 895	82	130	39 672 170	4.959 022
5F	95	11 749 420	1.468 677	83	131	40 874 760	5.109 345
60	96	12 234 220	1.529 28	84	132	42 104 480	5.263 06
61	97	12 733 880	1.591 735	85	133	43 361 730	5.420 217
62	98	13 248 680	1.656 085	86	134	44 646 930	5.580 866
63	99	13 778 930	1.722 366	87	135	45 960 490	5.745 061
64	100	14 324 930	1.790 617	88	136	47 302 810	5.912 851
65	101	14 887 000	1.860 875	89	137	48 674 330	6.084 291
66	102	15 465 450	1.933 181	8A	138	50 075 450	6.259 431
67	103	16 060 590	2.007 574	8B	139	51 506 600	6.438 324
68	104	16 672 740	2.084 093	8C	140	52 968 200	6.621 025
69	105	17 302 220	2.162 778	8D	141	54 460 690	6.807 587
6A	106	17 949 360	2.243 67	8E	142	55 984 500	6.998 063
6B	107	18 614 480	2.326 809	8F	143	57 540 060	7.192 507
6C	108	19 297 910	2.412 238	90	144	59 127 810	7.390 976
6D	109	19 999 980	2.499 998	91	145	60 748 190	7.593 524
6E	110	20 721 040	2.590 13	92	146	62 401 650	7.800 206
6F	111	21 461 410	2.682 677	93	147	64 088 620	8.011 078
70	112	22 221 450	2.777 682	94	148	65 809 580	8.226 197
71	113	23 001 500	2.875 187	95	149	67 564 950	8.445 619
72	114	23 801 900	2.975 237	96	150	69 355 210	8.669 401
73	115	24 623 000	3.077 875	97	151	71 180 800	8.897 6
74	116	25 465 170	3.183 146	98	152	73 042 200	9.130 275
75	117	26 328 750	3.291 094	99	153	74 939 860	9.367 483
76	118	27 214 110	3.401 764	9A	154	76 874 260	9.609 283
77	119	28 121 610	3.515 202	9B	155	78 845 870	9.855 734
78	120	29 051 620	3.631 452	9C	156	80 855 160	10.106 9
79	121	30 004 500	3.750 562	9D	157	82 902 600	10.362 83
7A	122	30 980 630	3.872 578	9E	158	8 498 680	10.623 58
7B	123	31 980 380	3.997 547	9F	159	87 113 880	10.889 23
7C	124	33 004 130	4.125 516	A0	160	89 278 690	11.1598 4

表 7.1(续)

H	D	T 周期数	t/s	H	D	T 周期数	t/s
A1	161	91 483 580	11.435 45	C5	197	20 193 528	25.241 91
A2	162	93 729 070	11.716 13	C6	198	20 599 488	25.749 36
A3	163	96 015 650	12.001 96	C7	199	21 011 536	26.264 42
A4	164	98 313 790	12.292 97	C8	200	21 429 736	26.787 17
A5	165	10 071 400	12.589 25	C9	201	21 854 144	27.317 68
A6	166	10 312 680	12.890 35	CA	202	22 284 832	27.856 04
A7	167	10 558 280	13.197 84	CB	203	22 721 848	28.402 31
A8	168	10 808 230	13.510 28	CC	204	23 165 264	28.956 58
A9	169	11 062 590	13.828 21	CD	205	23 615 136	29.518 92
AA	170	11 321 110	14.151 77	CE	206	24 071 536	30.089 42
AB	171	11 584 760	14.480 91	CF	207	24 534 512	30.668 14
AC	172	11 852 660	11.815 83	D0	208	25 004 136	31.255 17
AD	173	12 125 190	15.156 49	D1	209	25 480 461	31.850 58
AE	174	12 402 390	15.502 99	D2	210	25 963 568	32.454 46
AF	175	12 684 320	15.855 4	D3	211	26 453 512	33.066 89
B0	176	12 971 020	16.213 78	D4	212	26 950 360	33.687 95
B1	177	13 262 560	16.578 2	D5	213	27 454 168	34.317 71
B2	178	13 558 980	16.948 73	D6	214	27 965 008	34.956 26
B3	179	13 860 350	17.325 13	D7	215	28 482 944	35.603 68
B4	180	14 166 710	17.708 38	D8	216	29 008 040	36.260 05
B5	181	14 478 120	18.097 65	D9	217	29 540 360	36.825 45
B6	182	14 794 632	18.193 29	DA	218	30 079 976	37.599 97
B7	183	15 116 312	18.895 39	DB	219	31 181 352	38.283 69
B8	184	15 443 208	19.304 01	DC	220	31 181 352	38.976 69
B9	185	15 775 376	19.719 22	DD	221	31 743 218	39.679 06
BA	186	16 112 872	20.141 09	DE	222	32 312 704	40.390 88
BB	187	16 455 760	20.569 7	DF	223	32 889 776	41.112 22
BC	188	16 804 088	21.005 11	E0	224	33 474 552	41.843 19
BD	189	17 157 912	21.447 39	E1	225	34 067 096	12.583 87
BE	190	17 517 296	21.896 62	E2	226	34 667 464	43.334 33
BF	191	17 882 296	22.352 87	E3	227	35 275 736	44.094 67
C0	192	18 252 976	22.816 22	E4	228	35 891 968	44.864 96
C1	193	18 629 384	23.286 73	E5	229	36 516 248	15.645 31
C2	194	19 011 584	23.764 18	E6	230	37 148 632	46.435 79
C3	195	19 399 632	24.249 54	E7	231	37 789 200	47.236 5
C4	196	19 793 592	24.741 99	E8	232	38 438 008	48.047 51

表 7.1(续)

H	D	T 周期数	t/s	H	D	T 周期数	t/s
E9	233	39 095 136	48.868 92	F5	245	47 655 640	59.569 55
EA	234	39 760 656	49.700 82	F6	246	48 427 464	60.534 33
EB	235	40 434 640	50.543 3	F7	247	49 208 624	61.510 78
EC	236	41 117 152	51.396 44	F8	248	49 999 200	62.499
ED	237	41 808 272	52.260 34	F9	249	50 799 264	63.499 08
EE	238	42 508 056	53.135 07	FA	250	51 608 888	64.511 11
EF	239	43 216 600	54.020 75	FB	251	52 428 152	65.535 19
F0	240	43 933 960	54.917 45	FC	252	53 257 136	66.571 42
F1	241	44 660 216	55.825 27	FD	253	54 095 904	67.619 88
F2	242	45 395 440	56.744 3	FE	254	54 944 544	68.680 68
F3	243	46 139 704	57.671 63	FF	255	55 803 128	69.753 91
F4	244	46 893 072	58.616 34	00	256	56 671 736	70.839 67
				$00 为延时最长，因为 $00－1＝$FF			

7.2 简单 I/O 口输出实验

7.2.1 SLAVR721.ASM

测试验证 DIP20 AVR 单片机 B 口、D 口引脚输出和 SL－AVR 并发下载实验器功能。LED 逐位移位，移位速度会变化。

```
; AT90S1200 引脚图，"*"表示引脚接 LED 发光二极管，"↓ ↑"表示灯亮移位方向。
;RST        。1    20  。Vcc
;PD0  ↑ *   。          。 * ↓      PB7
;PD1  ↑ *   。          。 * ↓      PB6
;XTAL2      。          。 * ↓      PB5
;XTAL1      。          。 * ↓      PB4
;PD2  ↑ *   。          。 * ↓      PB3
;PD3  ↑ *   。          。 * ↓      PB2
;PD4  ↑ *   。          。 * ↓      PB1
;PD5  ↑ *   。          。 * ↓      PB0
;GND        。10   11   。 * ↓      PD6
.include "1200def.inc"              ;必须写器件配置文件
 rjmp RESET                         ;Reset Handle
.org $005
RESET:
        LDI r16,0XFF                ;设 B 口、D 口为输出
        OUT ddrb,R16                ;设 b 口方向寄存器为输出
        OUT DDRD,R16                ;设 D 口方向寄存器为输出
```

```
        out portd,r16          ;关 D 口 LED,SL - AVR 实验器硬件设定高电平 LED 灯灭
        out portb,r16          ;关 B 口 LED
start:  ldi R17,0x08           ;循环次数
        ldi r18,0x7f           ;0b0111 1111,SL - AVR 实验器硬件设定低电平 LED 灯亮
loop:   out portb,r18          ;B 口 7 位 LED 灯亮
        sec                    ;c=1
        ror r18                ;通过进位右循环
        rcall delay            ;调用延时子程序
        dec r17                ;-1
        brne loop              ;检测 R17 循环不为 0 转移,为 0 按顺序执行
        out portb,r16          ;关 B 口
        ldi r18,0xbf           ; 0b1011 1111
        out portd,r18          ;D 口 6 位 LED 灯亮
        rcall delay            ;延时
        ldi r18,0xff
        out portd,r18          ;关 D 口
        rjmp start             ;循环
delay:  ldi r29,0x0a           ;延时子程序
delay1: dec r30                ;复位后 R30=0X00
        brne delay1            ;R30 不为 0 转,为 0 按顺序执行
        dec r31                ;复位后 R31=0X00
        brne delay1            ;R30 不为 0 转,为 0 按顺序执行
        dec r29                ;复位后 R29=0X00
        brne delay1            ;R29 不为 0 转,为 0 按顺序执行
        ret                    ;子程序返回
```

7.2.2　SLAVR722.ASM

测试验证 AVRDIP40 引脚输出和 SL - AVR 开发下载实验器功能。

```
.include "8515def.inc"         ;必须写器件配置文件
.org   $0000
    rjmp RESET                 ;Reset Handle
.def temp=r20
.def zh     =r31
.org $0010
RESET:
    ldi  r16,high(RAMEND)      ;设堆栈
    out  SPH,r16               ;
    ldi  r16,low(RAMEND)
    out  SPL,r16
    ser  temp                  ;直接装入 $FF,
    out  DDRA,  temp           ;口的方向寄存器设定,为输出
    out  DDRB,  temp
```

```
    out  DDRC,   temp
    out  DDRD,   temp
forever:
    clr  temp                    ;硬件设低电平 LED 灯亮
    out  PORTA,  temp            ;PORTA 口 LED 灯亮
    out  PORTB,  temp            ;B 口 LED 灯亮
    out  PORTC,  temp            ;C 口 LED 灯亮
    out  PORTD,  temp            ;D 口 LED 灯亮
    ldi  R16,0X56                ;装延时常数,灯亮延时 1 s,可修改该参数,应另存一个文件名
    rcall delay                  ;调用延时子程序
    ser  temp                    ;硬件设高电平 LED 灯灭
    out  PORTA,  temp            ;PORTA 口 LED 灯灭
    out  PORTB,  temp            ;B 口 LED 灯灭
    out  PORTC,  temp            ;C 口 LED 灯灭
    out  PORTD,  temp            ;D 口 LED 灯灭
    ldi  R16,0X48                ;装延时常数,灯灭延时 0.5 s,可修改该参数
    rcall delay                  ;调用延时子程序
    rjmp forever                 ;无限循环
delay:                           ;通用延时子程序略,R16=$56,延时 1 s,$67 延时 2 s
......
```

7.2.3 SLAVR723.ASM

测试验证 AVRDIP40 引脚输出和 SL-AVR 开发下载实验器功能;测试 A 口、B 口、C 口、D 口 LED 灯亮循环变速移位。

```
;    DIP40 AT90S8515 引脚排列图,"*"表示引脚上接 LED 灯,"↓↑"表示 LED 亮灯移动方向
;
;    PB0  ↓*。1   40  。VCC
;    PB1  ↓*。        。*↑      PA0
;    PB2  ↓*。        。*↑      PA1
;    PB3  ↓*。        。*↑      PA2
;    PB4  ↓*。        。*↑      PA3
;    PB5  ↓*。        。*↑      PA4
;    PB6  ↓*。        。*↑      PA5
;    PB7  ↓*。        。*↑      PA6
;    RESET  。        。*↑      PA7
;    PD0  ↓*。        。        ICP
;    PD1  ↓*。        。        ALE
;    PD2  ↓*。        。        OC1B
;    PD3  ↓*。        。*↑      PC7
;    PD4  ↓*。        。*↑      PC6
;    PD5  ↓*。        。*↑      PC5
;    PD6  ↓*。        。*↑      PC4
```

```
;      PD7    ┼ * 。          。 *  ↑        PC3
;      XTAL2    。          。 *  ↑        PC2
;      XTAL1    。          。 *  ↑        PC1
;      GND      。 20  21   。 *  ↑        PC0
. include "8515def. inc"                ;器件配置文件
rjmp     RESET                          ;Reset Handle
. org    $ 00d                          ;跳过中断区
RESET:        LDI R16, $ 5F             ;必须先设堆栈,因为复位后 SPL=0X00,SPH=0X00
              OUT SPL,R16               ;AVR 进堆栈时-1,出栈时+1,与 MCS-51 进出栈方向相反
              LDI R16, $ 02             ;
              OUT SPH,R16               ;设堆栈底为 $ 025F,为 AVR AT90S8515
                                        ;内部 SRAM( $ 0060- $ 025F)底
        LDI r16,0XFF                    ;
        OUT DDRB,R16                    ;设方向寄存器为输出
              OUT DDRD,R16
              out ddra,r16
              out ddrc,r16
              out portd,r16             ;关 D 口,硬件设定高电平 LED 关
              out portb,r16             ;关 B 口,硬件设定高电平 LED 关
              out porta,r16             ;关 A 口,硬件设定高电平 LED 关
              out portc,r16             ;关 C 口,硬件设定高电平 LED 关
st:           ldi r28,0x08              ;循环次数
startb:       ldi R17,0x08
              ldi r18,0xfe              ;0b1111 1110
loopb:        out portb,r18             ;开 b 口,0 位 LED 灯亮,如何修改使 2 个、
                                        ;3 个或 1 隔 1 等,LED 灯亮移位
              sec                       ;置进位标志 C=1
              rol r18                   ;通过进位左循环
              mov r29,r28               ;移位(延时)次数
              rcall delay               ;调用延时子程序
              dec r17                   ;
              brne loopb                ;R17 不为 0 转,为 0 顺执
              out portb,r16             ;关 B 口
startd:       ldi R17,0x08
              ldi r18,0xfe              ;0b1111 1110
loopd:        out portd,r18             ;开 d 口,0 位 LED 灯亮
              sec                       ;C=1
              rol r18                   ;通过进位左循环
              mov r29,r28
              rcall  delay
              dec r17
              brne loopd
              out portd,r16
```

```
startc:     ldi R17,0x08
            ldi r18,0xfe          ;0b1111 1110
loopc:      out portc,r18         ;开 c 口,0 位 LED 灯亮
            sec
            rol r18               ;通过进位左循环
            mov r29,r28
            rcall  delay
            dec r17
            brne loopc
            out portc,r16
starta:     ldi R17,0x08
            ldi r18,0x7f          ;0b0111 1111
loopa:      out porta,r18         ;开 a 口,7 位 LED 灯亮
            sec
            ror r18               ;通过进位右循环
            mov r29,r28
            rcall  delay
            dec r17
            brne loopa
            out porta,r16         ;关 a 口
            dec r28
            breq  st              ;r28 为 0 转
            rjmp  startb          ;循环
delay:      ldi r31,0x23          ;延时子程序,可修改时间常数
delay1:     dec r30
            brne delay1
            dec r31
            brne delay1
            dec r29               ;移位速度次数
            brne delay
            ret                   ;子程序返回
```

7.2.4 SLAVR724.ASM

调节延时时间,就可用 AVR 的 I/O 口发出 1234567 音符声。

```
.include "8515def.inc"            ;器件配置文件
.org   $0000
    rjmp    RESET
.org $0010
RESET:      ldi    r16,0x02
            out    sph,r16
            ldi    r16,0x5f
            out    spl,r16        ;设堆栈为 0X025F
```

```
        ldi     r16,0xff        ;设口为输出状态
        out     ddra,r16
        out     ddrb,r16
        out     ddrc,r16
        out     ddrd,r16
        out     porta,r16       ;关口,灭 LED灯
        out     portb,r16
        out     portc,r16
        out     portd,r16
        ldi     r18,0x20          ;设延时常数
        ldi     r17,0x01
        ldi     r19,0x60
loop:   mov     r16,r19
        rcall   delay           ;调用延时子程序
        eor     r18,r17         ;异或
        out     portc,r18       ;输出 AT90S8515 的 C 口引脚
        dec     r20             ;-1
        brne    loop            ;R20 不为 0 转,为 0 顺执
        subi    r19,0x05        ;R19 减立即数
        cpi     r19,0x1f        ;R19 与立即数比
        brne    loop            ;R19 不为 0 转
        RJMP RESET              ;循环
delay:  push    r16             ;2t  延时子程序
delay1: dec     r16             ;1t
        brne    delay1          ;1t/2t
        pop     r16             ;2t
        dec     r16             ;1t
        brne    delay           ;1t/2t
        ret                     ;4t
```

7.2.5　SLAVR725.ASM

利用延时程序 I/O 口输出报警声。

```
.include"8515def.inc";          器件配置文件
.org    $0000
reset:ldi r16,$5f               ;设堆栈
      out spl,r16
      ldi r16,$02
      out sph,r16
      ldi r18,0xff              ;设口为输出
      out ddrc,r18
      ldi r19,0xf0              ;报警参数
lp: sbi portc,$00               ;开 pc 口
```

```
        rcall delay                ;延时
        cbi portc,$00              ;关 pc 口
        rcall delay
        dec r19                    ;一1
        brne lp                    ;r19 不为 0 转,为 0 顺执
        rcall delay1               ;较长延时,不发声
        rjmp lp                    ;循环报警
delay1:ldi r17,$40                 ;延时子程序,报警声快慢调节 $30~ $60
        rcall delay0
        ret
delay:ldi  r17,$9                  ;延时子程序,报警声频率可调 $a~ $7
        rcall delay0
        ret
delay0:                            ;通用延时子程序略
      .....
```

7.2.6 SLAVR726.ASM

AT90S8515 的 PA 口使用建表方式的 LED 广告灯演示程序。

```
.include"8515def.inc"              ;器件配置文件
.org    $0000                      ;设置起始地址
.equ    leddata=0x0250
  rjmp   reset
.cseg
.org    $0010
RESET: ldi  r16,$5f                ;设置堆栈
        out  spl,r16
        ldi  r16,$02
        out  sph,r16
        ldi  r16,$90
        mov r15,r16
        ser  r16                   ;设置 A 口为输出口
        out  ddra,r16              ;设置 A 口方向寄存器
   L0: ldi  zl,low(leddata*2)
        ldi  zh,high(leddata*2)
   L1: lpm
        mov r16,r0
        cpi  r16,$0a
        breq L0
        out  porta,r16
        rcall delay                ;调用延时子程序
        ld   r0,z+
        rjmpL1
```

```
DELAY:                          ;通用延时子程序从略
.cseg                           ;设置 LED 广告灯数据表
.org   leddata
.db 0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f,0xbf,0xdf,0xef,0xf7,0xfb,0xfd
.db 0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f,0xbf,0xdf,0xef,0xf7,0xfb,0xfd
.db 0x00,0x18,0x3c,0x7e,0xff,0x7e,0x3c,0x18
.db 0x00,0x18,0x3c,0x7e,0xff,0x7e,0x3c,0x18
.db 0xf8,0xf1,0xe3,0xc7,0x8f,0x1f,0x8f,0xc7,0xe3,0xf1
.db 0xf8,0xf1,0xe3,0xc7,0x8f,0x1f,0x8f,0xc7,0xe3,0xf1
.db 0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0x80,0xc0,0xe0,0xf0,0xf8,0xfc
.db 0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0x80,0xc0,0xe0,0xf0,0xf8,0xfc
.db 0xff,0xe7,0xdb,0xbd,0x7e,0xbd,0xdb,0xe7
.db 0xff,0xe7,0xdb,0xbd,0x7e,0xbd,0xdb,0xe7
.db 0xff,0x00
.db 0xff,0x00
.db 0xff,0x00
.db 0xff,0x00
.db 0x0a,0x0a
```

7.2.7　SLAVR727.ASM

LED 发光二极管加 1 计数程序。

```
;AT90S8515 的 PB、PD 口设计成十六位二进制加 1 计数程序,用 LED 发光二极管显示
.include"8515def.inc"
.org    $0000                   ;设置起始地址
   AB: ldi   r16,$5f            ;设置堆栈
       out   spl,r16
       ldi   r16,$02
       out   sph,r16
RESET: ldi   r18,0xff           ;设置 B 口,D 口为输出口
       out   ddrb,r18           ;设置 B 口,D 口方向寄存器
       out   ddrd,r18
       clr   r0
       clr   r1
    L0: mov  r2,r0
       mov   r3,r1
       com   r2                 ;R2,R3 取反
       com   r3
       out   portb,r2           ;R2,R3 数据送 B 口,D 口
       out   portd,r3
       rcall delay              ;调用延时子程序
       inc   r0                 ;R0 加 1,不为 0 跳转,为 0 顺执
       brne  L0
```

```
        inc   r1
        brne  L0
        dec   r0
        dec   r1
   L1:  mov   r2,r0
        mov   r3,r1
        com   r2              ;R2,R3 取反
        com   r3
        out   portb,r2        ;R2,R3 数据送 B 口,D 口
        out   portd,r3
        rcall delay           ;调用延时子程序
        dec   r0              ;R0 减 1,不为 0 跳转,为 0 顺执
        brne  L1
        dec   r1
        brne  L1
        rjmp  reset
DELAY:  ldi   r18,$01         ;延时子程序
    L2: dec   r16
        brne  L2
        dec   r17
        brne  L2
        dec   r18
        brne  L2
        ret
```

7.3 综合程序

7.3.1 LED/LCD/键盘扫描综合程序

源程序：SLAVR73A. ASM。

综合程序功能如下：

(1) LED 及 LCD 显示程序，有自动识别 LED 或 LCD 功能，设 LCD 优先级高。

(2) 键盘扫描输入程序，0～F 为 16 个数字键，还有上档命令键。EXEC——执行键；EEPROM——读键；SRAM——读写键；MON——返回初始状态键；LAST——上一单元地址键；NEXT——下一单元地址键；SHIFT——转换上档命令键，先按 SHIFT 键，再按命令键，就执行上档键的命令；$\overline{\text{RST}}$——复位键，执行程序后，要机器回到初始化状态，必须按复位键。

(3) 按数字键显示对应数字，并有小数点作为光标，提示下一步工作位置；按命令键(先按 SHIFT)执行相应命令。

(4) 对应功能入口地址(地址数字后零可省)：

0070H～01FFH——读、写内部 SRAM(监控规定 SRAM 读写范围)；

0000H～01FFH——读片内 EEPROM 数据；

0200H——歌曲："祝你生日快乐"，"万水千山总是情"；

0300H——LED上8字循环显示；

0320H——LED上0～F字符循环显示；

0400H——逐次逼近法A/D转换(需接网络电阻，另见说明)；

0500H——LCD初始化程序；

0700H——LCD上尖头字符左右移位程序；

0740H——LCD上0～F字符循环显示；

0800H——LCD显示LCD所有字符；

程序清单见光盘文件SLAVR73A.ASM。

7.3.2 LED键盘扫描综合程序

源程序：SLAVR73B.ASM。

综合程序功能如下：

(1) 键盘扫描输入程序，0～F为16个数字键，还有上档命令键。EXEC——执行键；EEPROM——读键；SRAM——读写键；MON——返回初始状态键；LAST——上一单元地址键；NEXT——下一单元地址键；SHIFT——转换上档命令键，先按SHIFT键，再按命令键，就执行上档键的命令；$\overline{RST}$——复位键，执行程序后，要机器回到初始化状态，必须按复位键。

(2) 按数字键显示对应数字，并有小数点作为光标，提示下一步工作位置；按命令键(先按SHIFT)执行相应命令。

(3) 对应功能入口地址(地址数字后零可省)。

0070H～01FFH——读、写内部SRAM(监控规定SRAM读写范围)；

0000H～01FFH——读片内EEPROM数据；

0200H——歌曲："祝你生日快乐"，"万水千山总是情"；

0300H——LED上8字循环显示；

0320H——LED上0～F字符循环显示；

程序清单见光盘文件SLAVR73B.ASM。

7.3.3 在LED上实现字符8的循环移位显示程序

源程序：SLAVR731.ASM；本程序在SL-AVR开发实验器上通过。

请修改程序：

(1) 修改字形；

(2) 改变字符个数，二位、三位或一隔一显示；

(3) 改变字形移动方向；

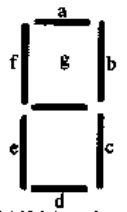
(4) 改变字符移位速度。

```
.include"8515def.inc"          ;器件配置文件
.def temp=r16                  ;数据暂存器
.def scndp=r22                 ;LED显示位置暂存器(如表7.2所列)
.org $0000
rjmp reset
.org $030
```

```
reset:    ldi temp,low(ramend)    ;设置堆栈指针
out spl,temp
ldi temp,high(ramend)
out sph,temp
ldi temp,$ff                      ;设置B、D口输出
```

表7.2 LED字形表

h	g	f	e	d	c	b	a	十六进制码	字形
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0		
0	0	1	1	1	1	1	1	3F	0
0	0	0	0	0	1	1	0	06	1
0	1	0	1	1	0	1	1	5B	2
0	1	0	0	1	1	1	1	4F	3
0	1	1	0	0	1	1	0	66	4
0	1	1	0	1	1	0	1	6D	5
0	1	1	1	1	1	0	1	7D	6
0	0	0	0	0	1	1	1	07	7
0	1	1	1	1	1	1	1	7F	8
0	1	1	0	1	1	1	1	6F	9
0	1	1	1	0	1	1	1	77	A
0	1	1	1	1	1	0	0	7C	B
0	0	1	1	1	0	0	1	39	C
0	1	0	1	1	1	1	0	5E	D
0	1	1	1	1	0	0	1	79	E
0	1	1	1	0	0	0	1	71	F
1	1	1	1	0	0	1	1	F3	P.
0	1	1	1	0	1	1	0	76	H



硬件设定高电平
LED笔划点亮
低电平LED管选中

```
out ddrb,temp
out ddrd,temp
out portd,temp
ldi temp,$7f                      ;字形8的代码为$7F(可修改)
out portb,temp
again:    sec                     ;置进位位为1(低电平LED亮,高电平LED灭)
ldi scndp,0b11011111              ;扫描显示SCANDP(可修改)
route:    out portd,scndp         ;从LED最左一位(D5)右移(可修改)
ldi temp,$40                      ;设置延时常数(可修改)
rcall delay                       ;调用延时
ror scndp                         ;右循环(可修改)
brcc again                        ;显示下一位
rjmp route                        ;循环显示
delay:                            ;通用延时子程序略
```

7.3.4　电脑放音机

源程序：SLAVR732.ASM。

AVR单片机在儿童智能玩具中的应用——音乐玩具。

利用单片机智能玩具开发儿童智力大有作为。因为单片机扩展存储器方便，而大容量存储器价格也很低。64KB的EPROM可存放300多首歌曲，8M位EPROM可存放5000多首歌曲，几个芯片就可组成一个音乐库。

利用AVR单片机产生乐曲音符，再把这些音符翻译成计算机音乐语言，由单片机进行信息处理，再经过信号放大，由耳机或喇叭放出乐曲声。由于音符和节拍是由计算机产生的，所以音符和节拍准确，对儿童的音乐教育很有帮助。利用单片机的中断、I/O口控制功能，可以做到电脑放音机自动连续放音，乐曲全部放完，自动从头开始放音，循环不断。

一、如何产生音乐频率

(1) 要产生音频脉冲，需算出某一音频的周期(1/f)，然后将此周期除以2，即为半周期的时间；再利用计时器计时此半周期时间，当计时到后，就将输出脉冲对I/O口反相。重复计时此半周期时间，再对I/O口反相，如此就可在I/O口引脚上得到此频率的脉冲(程序驱动I/O口反相，即正、负各半周期为一个周期，才能使喇叭"吸、放"发声)。

(2) 利用AVR单片机的内部计时器，让其工作在计数模式MODE1(16位计数器)下，改变计数值TCNT1H及TCNT1L以产生不同的频率。

(3) 以6MHz晶振为例：要产生频率523Hz，其周期T＝1/523＝1 912μs，其半周期为1 912/2＝956μs，因此只要令计数器计时956μs/1μs＝956(为半周期)。所以在每计数956次时将I/O反相，就可得到中音D0(523Hz)。

为计数脉冲值与频率的关系公式如下：

$$N=f_i(\text{6MHz 晶振，CPU 产生的频率})\div 2(\text{半周期})\div f_r$$

式中，N为计数值；f_r为要产生的频率；f_i为以6MHz晶振为例，内部计时(数)一次需2μs，频率单位为1周期/秒，即Hz。

$$f=1/2\mu s=1/2\times 10^{-6}s=500\ 000\ 次/秒=500\ 000Hz=500\ kHz$$

(4) 其计数值的求法如下：

T(16位计数器计多少后溢出)＝65 536(16位二进制计数器，计满数溢出时的计数值为

$$2^{16}-N=65\ 536-f_i/2)/f_r$$

例如：求低音D0(262Hz)，中音D0(523Hz)，高音D0(1046Hz)的计数值：

设K＝65 535，f_i＝500 000＝f_i＝0.5MHz。

T＝65 536－N＝65 536－(f_i/2)/f_r＝65 536－(500 000/2)/f_r＝65 536－250 000/f_r

低音D0的T＝65 535－1 908＝63 627(十进制数)；

中音D0的T＝65 535－0 956＝64 579(十进制数)；

高音D0的T＝65 535－0 478＝65 057(十进制数)。

(5) C调各音符频率与计数值T的对照，如表7.3。

表 7.3 C 调各音符与计器值 T 的对照

音符	频率/Hz	半周期/μs	TCNT 值	音符	频率/Hz	半周期/μs	TCNT 值
低 1D0	262	1 908	63 627	#4FA#	740	0 676	64 859
#1D0#	277	1 805	63 730	中 5S0	784	0 638	64 897
低 2RE	294	1 700	63 835	#5S0#	831	0 602	64 933
#2RE#	311	1 608	63 927	中 6LA	880	0 568	64 967
低 3M	330	1 516	64 020	#6LA#	932	0 536	64 999
低 4FA	349	1 433	64 012	中 7SI	988	0 506	65 029
#4FA#	370	1 350	64 185	高 1D0	1 046	0 478	65 057
低 5S0	392	1 276	64 259	#D0#	1 109	0 451	65 084
#5S0	415	1 205	64 330	高 2RE	1 175	0 426	65 109
低 6LA	440	1 136	64 399	#2RE#	1 245	0 402	65 133
#6LA#	466	1 072	64 463	高 3M	1 318	0 372	65 156
低 7SI	494	1 012	64 523	高 4FA	1 397	0 358	65 177
中 1D0	523	0 956	64 579	#4FA#	1 480	0 338	65 197
#1D0#	554	0 903	64 632	高 5S0	1 568	0 319	65 216
中 2RE	578	0 842	64 683	#5S0#	1 661	0 292	65 243
#2RE#	622	0 804	64 731	高 6LA	1 760	0 284	65 251
中 3M	659	0 759	64 776	#6LA#	1 865	0 268	65 267
中 4FA	698	0 716	64 819	高 7SI	1 976	0 253	65 282

"#"表示半音,用于上升或下降半个音。

二、如何产生节拍

每个音符使用 1 个字节,每个节拍使用 1 个字节,AVR 程序存储器可以设为 16 位,即 1 个字,或称双字节。所以一个字的高 8 位存放音符码,低 8 位存放节拍码。如果 1 节拍为 0.4s 则 1/4 拍是 0.1s,只要设定延迟时间就可求得节拍的时间。假设 1/4 拍为 1 DELY 单位,则 1 拍应为 4 个 DELY,以此类推,只要求得 1/4 拍的 DELY 单位时间,其余的节拍就是它的倍数。1/4 拍的延迟时间=187ms。

节拍与节拍码对照如表 7.4。

表 7.4 节拍与节拍码对照表

节拍码	节拍数(拍)	节拍码	节拍数(拍)
1	1/4	1	1/8
2	2/4	2	1/4
3	3/4	3	3/8
4	1	4	1/2
5	$1\frac{1}{4}$	5	5/8
6	$1\frac{1}{2}$	6	3/4
8	2	8	1
10	$2\frac{1}{2}$	10	$1\frac{1}{4}$
12	3	12	$1\frac{1}{2}$
16	$1\frac{3}{4}$		

三、建立音乐的步骤

找出乐曲，然后对照音符表，翻译出乐曲码，用程序伪指令DB输入曲码和节拍码；也可直接在调试窗口的程序存储器窗口＄0100地址输入曲码和节拍码（只适用于在线实时仿真器）。

例如音符表练习：

（1）把简谱翻译成曲码代码：

以下音符均设为一拍，代码为4。

1 2 3 4 5 6 7(低八度音)1 2 3 4 5 6 7(中音)1(高音)1(高音)7 6 5 4 3 2 1(中音)7 6 5 4 3 2 1(低八度音)

曲码	1	3	5	6	8	10	12	13	15	17	18	20	22	24	25
简码	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1
	低八度音							中　音							高音

曲码	36	34	32	30	29	27	25	24	22	20	18	17	15	13	12
简码	7	6	5	4	3	2	1	7	6	5	4	3	2	1	7
	高八度音							中　音							低音

（2）翻译成乐曲加节拍代码：

01,04,03,04,05,04,06,04,08,04.10,04,12,04,13,04,13,04,15,04,17,04,18,04,20,04,22,04,24,04,25,04

25,04,36,04,34,04,32,04,30,04,30,04,29,04,27,04,25,04,24,04,22,04,20,04,18,04,17,04,15,04,13,04,12,04

以上乐曲数据用伪指令DB方式输入"乐曲.ASM"的＄0100地址，再汇编一次就可下载试听。

注意：音符节拍间用逗号隔开，不要不小心键入小数点。因为逗号键右边是小数点键，键入小数点，程序汇编时将造成计算机死机！00 00为所有曲结束标志。

（3）把乐曲代码输入计算机。

把SL-AVR实验器与PC机连机，U4插上AT90S8515芯片，插上音响器短路块，开机通电。进入AVR下载窗口，进行下载操作，下载结束应能听到乐曲声。

```
;* * * * * * * * * * * * * * 乐曲程序 SLAVR732.ASM * * * * * * * * * * * * * * * * * * *
;*  标题:AT90S8515 C口输出乐曲声——电脑放音机
;*  版本:1.0
;*  最后更新日期:2000.08.08
;*  支援 E-mail:gzsl@sl.com.cn
;*  描述:
;*  用SL-AVR万用下载开发实验器做样机,在AT90S8515的C口接喇叭发出乐曲声,请把最喜爱的乐
    曲送入单片机!起始地址为$0100,也可把曲码、节拍码在调试窗口中的程序存储器窗口(Program -
    Memory)内,从$0100地址,用键盘直接输入乐曲(仅适合ICE-200实时仿真器)。
;*  作者:SL
;*  程序适用于所有单片机
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include"8515def.inc"            ;文件头AT90S8515器件配置文件,不同的器件有不同的器件配置文件
```

```
    rjmp RESET                    ;AVR 重新定位
.def    TEMPDH      =r2           ;寄存器定义
.def    TEMPDL      =r3
.def    CNT         =r10
.def    SCNN        =r11
.def    KEYN        =r12
.def    SCNK        =r13
.def    SCNDP       =r14
.def    KSNI        =r15
.def    TEMP        =r16          ;数据暂存器
.def    TEMP1       =r17
.def    TEMP2       =r18
.def    TEMP3       =r19
.def    SCNTT       =r26;
.def    MUSN        =r22          ;输出乐曲声暂存器
.def    TONL        =r21          ;节拍码低位
.def    TONH        =r20          ;节拍码高位
.def    PLYTON      =r25          ;存乐曲码
.def    TONSET      =r24
.def    TONLNG      =r23          ;存节拍码
.cseg
.org    0x06                      ;TIM1_OVF 定时器 1 溢出中断处理入口地址
inttl:   RJMP    OUTPM            ;转定时器 1 溢出中断处理,发音周期到,则跳转到发音输出态
.cseg
.org    0x010                     ;定时器 1 溢出中断处理程序,发音起始地址
                                  ;发音周期到,重新装入计时值,并将输出到 PORTC 口
OUTPM:   OUT    TCNT1H,TONH       ;重新将 TONH 新计时值载入 TCNT1H 内
    OUT     TCNT1L,TONL           ;重新将 TONL 新计时值载入 TCNT1L 内
    SBIS    PORTC,00              ;先检测 PORTC 口是否为 1 而跳转
    RJMP    SETOP1                ;若是 PORTC 口为 0,则跳到 SETOP1,令 PORTC 口转为 1
SETOP0:    CBI    PORTC,00        ;若 PORTC 为 1,则令 PORTC 转为 0
    LDI     MUSN,$00              ;同时令 MUSN 为 00 值
    RETI                          ;回中断前主程序,并令可再次中断返回
SETOP1:    SBI    PORTC,00        ;若 PORTC 为 0,则令 PORTC 转为 1
    LDI     MUSN,$01              ;同时令 MUSN 为 01 值
RETI                              ;回中断前主程序,并令可再次中断返回
.cseg
.org    0x020                     ;主程序起始地址,必须跳过中断区
RESET:
    ldi     temp,low(RAMEND)      ;RAMEND 为 8515def.inc,内建值为 025FH
    out     SPL,temp              ;起始堆栈指针低位将 TEMP=02H 放入 SP=3DH
;若硬件堆栈或者 AVR 片内含 SRAM 小于 256B 时,下列二行程序可省略
```

```
    ldi     temp,high(RAMEND)      ;以 TEMP=R16<5F 为数据装入缓冲暂存器
    out     SPL+1,temp             ;堆栈指针高位将 R16=TEMP=5FH 放入 SPL+1=3EH
    wdr                            ;在使用看门狗计时器前需重设看门狗计时器,为避免在接下来程
                                   ;序前就因 WDT 已快计时溢出而重设
    ldi     temp,$0F               ;WDTCR 地址 $21 设定以 TEMP 缓冲,令 WDE=D3=1
    out     WDTCR,temp             ;并令预除为 2 048ms,设定 WDE=1=D3,输出到 WDTCR 内
    LDI     MUSN,$00               ;令 MUSN 为 00 值
    ldi     temp,$00               ;令 TEMP 暂存器放入 00
    OUT     TCCR1A,TEMP            ;TEMP=00 内含输出到 TCCR1A 内,禁止比较器及 PWM 动作
    OUT     TCCR1B,TEMP            ;将 TEMP=00 内含输出到 TCCR1B 内,停止 TC1 计时及捕抓
    LDI     TEMP,$02               ;将 02 值预存入 SRAM 的 0100H 内,作 TC1 的 TCCR1B 控制内含
                                   ;令 TC1 为计时预除 8
    STS     $0100,TEMP
    LEDA;   CLI                    ;中断总开关 sreg=d7=i=1
    Ldi     r16,0b10000000         ;令 toiel=1 触发中断
    out     timsk,r16              ;R16 的 D7=1,令 TOIE1=1 触发中断
    LDI     TEMP,$FF               ;设 AVR 单片机 I/O 口方向寄存器为输出
    OUT     DDRA,TEMP              ;A 口为输出
    OUT     DDRB,TEMP              ;B 口为输出
    OUT     DDRC,TEMP              ;C 口为输出
;   OUT     DDRD,TEMP              ;D 口为输出
    LDI     TEMP,0B11111111        ;关灭 I/O 口的 LED 发光二极管
    OUT     PORTC,TEMP             ;C 口输出乐曲声
    OUT     PORTA,TEMP             ;关 A 口 LED 灯,硬件设定高电平 LED 暗
    OUT     PORTB,TEMP             ;关 B 口 LED 灯,硬件设定高电平 LED 暗
    OUT     PORTD,TEMP             ;关 D 口 LED 灯,硬件设定高电平 LED 暗
    CLR     TEMP2                  ;暂存器清零
    CLR     TEMP1                  ;暂存器清零
    CLR     KSNI                   ;暂存器清零
    LDI     SCNTT,$02;
    CLR     TONLNG                 ;暂存器清零
STARTP;     WDR                    ;关看门狗
    LDI     ZH,HIGH(PLYTAB*2)      ;启动演奏则令数据装入 Z 地址
    LDI     ZL,LOW(PLYTAB*2)       ;音乐演奏乐谱存放在 PLYTAB*2 起始地址
NEXMUT;     LPM                    ;将 Z 所指程序存储器乐曲,依次取音符码及节拍码置于 R0
    MOV     PLYTON,R0              ;将取出的第一个音符码装入 PLYTON 作周期控制
    LD      R0,Z+                  ;以 LD R0,Z+指令使得 Z 间接寻址加 1
    LPM                            ;将 Z 所指程序存储器乐曲,依次取节指码置于 R0
    MOV     TONLNG,R0              ;将取出的第一个节拍码装入 PLYTON 作节拍控制
    OR      R0,PLYTON              ;将此 R0 节拍码与音符码 PLYTON 作 OR 运算
    LD      R0,Z+                  ;以 LD R0,Z+指令使得 Z 间接寻址加 1
    BRNE    PLAYM                  ;若音符码及节拍码非全为 00 值,则跳到 PLAYM 演奏
    LDI     TEMP,$00               ;若音符码及节拍码全为零(0000),则为乐曲结束标记
```

```
    OUT     TCCR1B,TEMP          ;将 TEMP=00 内含输出到 TCCR1B 内,停止 TC1 计时及捕抓
    CLI                          ;令中断总开关 SREG 的 I 标志清零,而禁止中断
    SBI     PORTD,00             ;令 PINC=1,将喇叭输出 OFF
    RJMP    STARTP               ;循环演奏
PLAYM:  PUSH    ZH               ;进栈保存数据
    PUSH    ZL                   ;进栈保存数据
    TST     PLYTON               ;检测 PLYTON 是否为 0
    BREQ    MUSTD                ;若为 0 则跳至 MUSTD 作节拍等待
    LDI     ZH,HIGH(MUSTAB*2)    ;乐曲码装入 Z 地址
    LDI     ZL,LOW(MUSTAB*2)     ;计时器值存放于 MUSTAB*2 起始地址
    MOV     TEMP,PLYTON          ;将乐曲码 PLYTON 装入 TEMP 寄存器内
    DEC     TEMP                 ;寄存器 TEMP 减 1
    LSL     TEMP                 ;寄存器 TEMP 左移即 X2
    ADD     ZL,TEMP              ;将正确的计时器控制乐曲码的存表位移,且使 TEMP 加入 ZL
    LDI     TEMP,$00             ;令 TEMP=00,以便让 ZH 与进行标志位 C 相加
    ADC     ZH,TEMP              ;将 ZH 与 TEMP=00 以及进行标志位 C 相加,得到真正的 Z 地址值
    LPM                          ;将 Z 所指 PROM 的预存乐曲码计时长度低位值装入 R0
    MOV     TONL,R0              ;将乐曲码计时长度低位值 R0 装入 TONL 内
    OUT     TCNT1L,R0            ;将乐曲码计时长度低位值 R0 也装入 TCNT1L 内
    LD      R0,Z+                ;以 LD R0,Z+指令使 Z 间接寻址加 1
    LPM                          ;将 Z 所指 PROM 的预存乐曲码的计时节拍码置于 R0
    MOV     TONH,R0              ;将节拍高位值 R0 装入 TONH 内
    OUT     TCNT1H,R0            ;将节拍高位值 R0 装入 TCNT1H 内
    POP     ZL                   ;出栈将 ZL,ZH 由堆栈指针依次取回
    POP     ZH                   ;出栈
    LDS     TEMP,$0100           ;将 SRAM 地址 0100H 的内容装入 TEMP 内
    OUT     TCCR1B,TEMP          ;将原 0100H 的 SRAM 内容输出到 TCCR1B,控制 TC1
    SEI                          ;内容 02 令 TC1 为计时预除 8,令中断总开关 I=1 触发
MUSTD:  RCALL   PLYDEL           ;调用延时子程序 0.2s
    DEC     TONLNG               ;将节拍码 TONLNG 减 1
    BRNE    MUSTD                ;若节拍码 TONLNG 不为 0 则转回,再发音;为 0 则顺执
    RJMP    NEXMUT               ;继续到 NEXMUT 取乐曲码和节拍码
PLYDEL:LDI  TEMP,185             ;延时子程序,185×1ms=185ms,
                                 ;即 PLYDEL=185ms 为十进制时间常数
DT3:    LDI     TEMP1,04         ;送时间常数 4×250μs=1ms,DT3 约为 1ms
DT2:    LDI     TEMP2,250        ;250 为十进制时间常数,250×8×125ns=250ms,dt2=250μs
DT1:    WDR                      ;1T
        WDR                      ;2T
        WDR                      ;3T
        WDR                      ;4T
        WDR                      ;5T
        DEC     TEMP2            ;6T,TEMP-1
        BRNE    DT1              ;8T,TEMP2 不为 0(则共执行 250×8×T=250μs)转为 0
```

```
                                ;按顺序执行
DEC     TEMP1                   ;TEMP1－1
BRNE    DT2                     ;TEMP1 不为 0(则共执行 250×4μs=1ms)转,为 0 按顺序执行
DEC     TEMP                    ;TEMP－1
BRNE    DT3                     ;TEMP 不为 0(则共执行 185×1ms=185ms)转,为 0 按顺序执行
RET                             ;子程序返回
;约定:因为计算机不能表示简谱乐曲,低音用数字后一点表示;高音用数字前一点表示;半音用#号,'为隔开音符
;乐曲节拍应对照简谱查看,音长为节拍,一拍为 04,3/4 拍为 03,1/2(2/4)拍为 02,1/4 拍为 01
;00 表示休止符,00 00 连续 2 个字节为零,表示乐曲结束
;                       乐曲低八度音
```

曲码号	1	2	3	4	5	6	7	8	9	10	11	12
音符号	1	#1	2	#2	3	4	#4	5	#5	6	#6	7

;　　乐曲中音

曲码号	13	14	15	16	17	18	19	20	21	22	23	24
音符号	1	#1	2	#2	3	4	#4	5	#5	6	#6	7

;　　乐曲高八度音

曲码号	25	26	27	28	29	30	31	32	33	34	35	36
音符号	1	#1	2	#2	3	4	#4	5	#5	6	#6	7

```
.EQU    PLYTAB=0X0100           ;乐曲存放首地址
.EQU    MUSTAB=0X00A0           ;乐曲音符表存放首地址
.cseg                           ;实例
.org    PLYTAB                  ;"祝你生日快乐"乐曲,1=C 3/4 乐曲存放起始地址,请查看对照简谱乐曲
```

例：生日快乐歌　1=C 3/4

简码　| 5.5　6　5 | 1̇　7 – | 5.5　6　5 | 2̇　1̇ – |

简码　| 5.5　5̇　3̇ | 1̇　7　6 | 4̇. 4̇　3̇　1 | 2̇　1̇ – ‖

```
;注意:曲码中不能用小数点,只能用逗号隔开,否则汇编时造成死机!

;| 5          5,    6     5|;| .1     7     –|
.DB 20,02,00,01,20,01,22,04,20,04;.DB 25,04,24,04,00,04
;| 5          5,    6     5|;| .2     .1    –|
.DB 20,02,00,01,20,01,22,04,20,04;DB 27,04,25,04,00,04
```

```
;|  5              5,   .5     .3|;|.1      7      6|
.DB 20,02,00,01,20,01,32,04,29,04;DB 25,04,24,04,22,04
;|.4             .4     .3     .1|;|.2     .1      —|
.DB 30,02,00,01,30,01,29,04,25,04;DB 27,04,25,04.00,04
;REAGAIN
;|  5              5,    6      5|;|.1      7      —|
.DB 20,02,00,01,20,01,22,04,20,04;DB 25,04,24,04,00,04
;|  5              5,    6      5|;|.2     .1      —|
.DB 20,02,00,01,20,01,22,04,20,04;DB 27,04,25,04,00,04
;|  5              5,   .5     .3|;|.1      7      6|
.DB 20,02,00,01,20,01,32,04,29,04;DB 25,04,24,04,22,04
;|.4             .4     .3     .1|;|.2     .1      —|
.DB 30,02,00,01,30,01,29,04,25,04;DB 27,04,25,04,00,04
;              万水千山总是情
.db 17,04,18,04,20,06,20,02,22,04,20,04,17,12,15,04    ;
.db 13,06,17,02,15,04,13,04,10,12,10,04,8,8,13,04       ;注意:08 应写成 8 才能编译通过
.db 15,04,17,04,20,04,22,04,17,04,15,15,15,04,00,04    ;
.db 17,04,18,04,20,06,20,02,22,04,20,04,17,12,15,04    ;
.db 13,06,17,02,15,04,13,04,10,12,10,04,8,8,13,06       ;
.db 17,02,15,06,13,02,13,04,10,04,13,15,13,8,17,04     ;
.db 20,04,22,12,25,10,22,04,18,04,20,06,22,02,20,12    ;
.db 17,04,20,8,17,04,20,04,22,12,25,04,25,04,22,04     ;
.db 20,04,17,04,15,15,15, 8,17,04,18,04,20,06,20,02    ;
.db 22,04,20,04,17,12,15,04,13,06,17,02,15,04,13,04    ;
.db 10,12,10,04,8,8,13,04,17,04,15,06,13,02,10,04
.db 12,04,13,15,13,15                                   ;
    .DB 00,00                                           ;END
.cseg
.org    MUSTAB                                          ;音符表地址标号
;1     2    3     4    5     6    7     8    9
;1.   '#1.   2.   '#2.   3.    4.   '#4.   5.   '#5.
;10   11   12   13    14   15   16   17   18
;6.   '#6.   7.   1    '#1   2   '#2    3   '1
.DW 63627,63730,63835,63927
.DW 64020,64102,64185,64259
.DW 64330,64399,64463,64523
.DW 64579,64632,64683,64731
.DW 64776,64819
;19   20   21   22   23   24   25    26    27
;#4   5   '#5   6   '#6   7    '1    '#.1   '2
;  28    29   30    31   32    33   34    35   36
;  '#.2   '3   '4   '#.4   '5   '#.5   '6   '#.6   '7
```

```
.DW 64859,64897
.DW 64933,64967,64999,65029
.DW 65057,65084,65109,65133
.DW 65156,65177,65197,65216
.DW 65243,65251,65267,65282
```

7.3.5　键盘扫描程序

源程序见SLAVR73A(73B).ASM(其中部分程序、键盘扫描程序,可供调用)。

```
SCAN1:  PUSH XH                 ;键扫显示子程序
        PUSH XL                 ;将 xl 压入堆栈
        PUSH TEMP3
        PUSH TEMP2
        PUSH TEMP1
        PUSH TEMP
        LDI XL,$60
        SET                     ;T 标志为 1 表示未按键
        LDI  SCNN,$00           ;按键起始扫描码 SCNN 为 00
        LDI  SCNDP,0B11011111   ;令 6 位 7 段 LED 扫描显示码初始为 11011111
        LDI  CNT,$06            ;7 段 LED 共 6 位,故 CNT=6 为位数计数
        LDI  KSNI,0B11110111    ;4×4 键盘扫描码 KSNI 初始为 11110111
COL1:   LDI  TEMP,$FF           ;PORTB 设定为输出
        OUT  DDRb,TEMP
        OUT  DDRC,TEMP          ;PORTC 设定为输出
        OUT  PORTC,TEMP
        OUT  DDRd,TEMP          ;PORTD 设定为输出
        OUT  PORTd,SCNDP        ;6 位 7 段 LED 扫描显示码输出到 PORTD
        LD   R1,X+              ;要显示于 7 段 LED 的间接寄存器 X 中的内容送入 R1,并令 X
                                 加 1
        OUT  PORTb,R1           ;显示内容输出到 PORTB,以驱动 LED 显示
        RCALL  DELAY            ;调用延时,以显示此位数一段时间
        MOV  TEMP,CNT           ;LED 位数为 6,而按键码行数为 4,故需作 CNT 值检测
        SUBI  TEMP,$03          ;CNT=TEMP 与 3 相减比较
        BRCS  NOSK              ;位数扫描 CNT 超过 3,则 C 为 1,跳到 NOSK 不作按键处理
        LDI  TEMP1,$04          ;一共要检查 4 个按键
        LDI  TEMP,0B00001111    ;设定 PC0～PC3 为输出,PC4～PC7 为输入
        OUT  DDRc,TEMP
        OUT  PORTc,KSNI         ;KSNI 输出到 PORTC,并令 PC7～PC4 为上拉电阻输入态
        RCALL  DELYT            ;调用延时,以稳定读取键盘 I/O 输入端
        IN   TEMP,PINc          ;读取 C 口检测 PC7～PC4,看是否有按键低电位输入
        ANDI  TEMP,0B11110000   ;取 TEMP 的高 4 位
        SWAP  TEMP              ;键码顺序为 PC4～PC7,故将 TEMP 的高低 4 位互换成 D0～
                                 D3
```

```
KROW:   SEC                     ;令 C 标志为 1,以便将键盘码 D0~D3 移到 C 标志位检测
        ROR   TEMP              ;TEMP 的内容右移 1 位,将第一个键码 D0=PC4 移到 C 标志
;                               ;位检测
        BRCS  NOKEY             ;若有键按下则测到 PC4=D0=0,若 C=1 无按键则转到 NOK
                                ;EY
        CLT                     ;若 PC4=D0=CF=0,表示有按键,令 T=0 表示有按键
        MOV   KEYN,SCNN         ;把按键扫描码 SCNN 送键码 KEYN 中保存
        SBIS PINd,$07
        ADIW KEYN,$10           ;判定 SHIFT 键是否按下,按下则键值加 10
NOKEY:  INC   SCNN              ;按键扫描码 SCNN 加 1
        DEC   TEMP1             ;扫描读取键数 TEMP1 减 1
        BRNE  KROW              ;每行有 4 个按键,如 TEMP1 不为 0,则跳到 KROW 再检测 PC5
                                ;~PC7
        SEC                     ;此行 4 个键码检测完后,令 C 为 1,以方便键盘扫描码 KSNI 内
                                ;容的移位
        ROR   KSNI              ;键盘扫描码 KSNI=CF=1>11110111,移位以进行下一行按键
                                ;扫描
NOSK:   SEC                     ;令进位标志 CF=1
        ROR   SCNDP             ;将扫描显示码 SCNDP 左移,作下一位扫描
        DEC   CNT               ;共需作 6 位数扫描显示故 CNT 减 1
        BRNE  COL1              ;CNT 减 1 不为 0,则跳回 COL1,再作扫描显示及读取键盘输入
        LDI   TEMP,$FF          ;若已完成全部扫描显示和读取,则按键令 TEMP=0ff
        OUT   DDRC,TEMP         ;TEMP 输出到 DDRC,设定 PORTC 为输出,驱动 LED
        OUT   PORTC,TEMP
        POP TEMP
        POP TEMP1
        POP TEMP2
        POP TEMP3
        POP XL
        POP Xh
        RET
```

7.3.6 十进制计数显示

源程序:SLAVR734.ASM。

应用例子*.ASM,必须编译生成*.OBJ 文件才可调试。如要修改*.ASM,必须修改文件属性,去掉*.ASM 只读文件属性。

```
;* * * * * * * * * * * * * * * * * * * * * * 十进制计数程序 * * * * * * * * * * * * * * * * * * *
;*  标题:用二位 LED 显示十进制计数
;*  版本:1.0
;*最后更新日期:2000.08.08
;*  支援 E-mail:gzsl@sl.com.cn
;*  描述:
```

;* 用 AVR-AT90S1200 的 D 口接 2 只 LED 数码管,PB7,PB6 作片选,硬件设定 D 口高电平 LED 灯
;* 亮,B 口低电平选中 LED,即选用共阴极数码管,最大显示十进制 99。硬件接线原理图如图 7.1。
;* 作者:SL、Z
;* 程序适用于所有单片机
;* *

必须按图 7.1 接线才能正常工作!

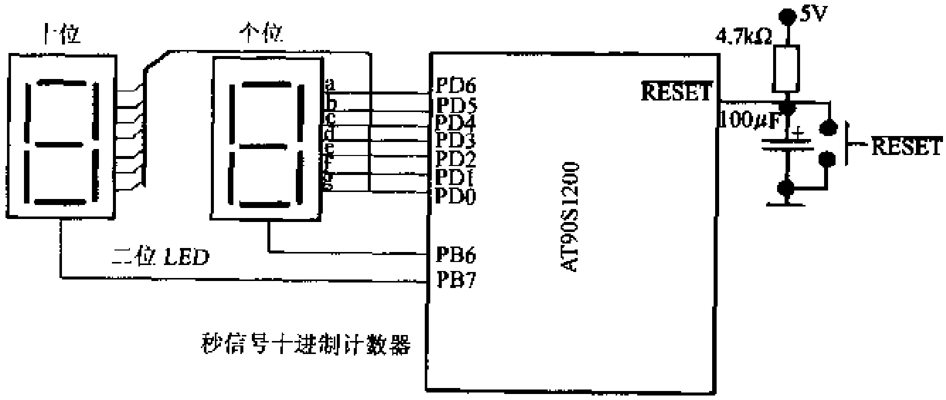
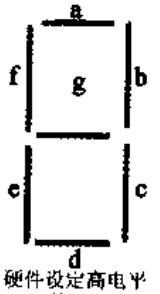


图 7.1 硬件接线原理图

```
.include"1200def.inc"        ;AT90S1200 配置文件
.org $0000
    rjmp reset
.org $0010
reset:ldi r20,$ff            ;设 B 口、D 口为输出
        out ddrb,r20         ;B 口方向寄存器
        out ddrd,r20         ;D 口方向寄存器
        sbi $18,7            ;硬件设定 B 口低电平选中 LED,PB7 关高位 LED
        sbi $18,6            ;PB6 关低位 LED
        ldi r20,$fe          ;建字形表如表 7.5
    mov r0,r20               ;0
        ldi r20,$b0
    mov r1,r20               ;1
        ldi r20,$ed
    mov r2,r20               ;2
        ldi r20,$f9
    mov r3,r20               ;3
        ldi r20,$b3
    mov r4,r20               ;4
        ldi r20,$db
    mov r5,r20               ;5
```

表 7.5 LED 字形表

h	a	b	c	d	e	f	g	十六进制码	字形	R
1	1	1	1	1	1	1	0	FEH	0	R0
1	0	1	1	0	0	0	0	B0H	1	R1
1	1	1	0	1	1	0	1	EDH	2	R2
1	1	1	1	1	0	0	1	F9H	3	R3
1	0	1	1	1	0	1	1	B3H	4	R4
1	1	0	1	1	0	1	1	DBH	5	R5
1	1	0	1	1	1	1	1	DFH	6	R6
1	1	1	1	0	0	0	0	F0H	7	R7
1	1	1	1	1	1	1	1	FFH	8	R8
1	1	1	1	0	0	1	1	F3H	9	R9
1	1	1	1	0	1	1	1	F7H	A	R10
1	0	0	1	1	1	1	1	9FH	B	R11
1	1	0	0	1	1	1	0	CEH	C	R12
1	0	1	1	1	1	0	1	BDH	D	R13
1	1	0	0	1	1	1	1	CFH	E	R14
1	1	0	0	0	1	1	1	C7H	F	R15



硬件设定高电平
LED数码管点亮

```
        ldi r20, $ df
    mov r6,r20          ;6
        ldi r20, $ f0
    mov r7,r20          ;7
        ldi r20, $ ff
    mov r8,r20          ;8
        ldi r20, $ f3
    mov r9,r20          ;9
        ldi r20, $ f7
    mov r10,r20         ;A
        ldi r20, $ 9f
    mov r11,r20         ;B
        ldi r20, $ ce
    mov r12,r20         ;C
        ldi r20, $ bd
    mov r13,r20         ;D
        ldi r20, $ cf
    mov r14,r20         ;E
        ldi r20, $ c7
    mov r15,r20         ;F
    bclr 7              ;清 I 标志,关中断
    clr r28             ;R28= $ 00
main:   ldi r20, $ 28   ;扫描次数
start:  mov r30,r28     ;十位显示字符值,第一次显示 0
display:andi r30, $ f0  ;取十位
```

```
        swap r30            ;半字节交换,获取Z地址
ledh:   ld r25,z            ;LED高位,复位后(R31)=$00
        out portd,r25       ;送D口显示
        sbi $18,6           ;关个位,硬件设定高电平不亮
        cbi $18,7           ;选通十位,硬件设定低电平亮
        ldi r27,$10         ;延时常数
delay1: dec r26             ;-1,延时
        brne delay1         ;不为0转
        dec r27             ;-1,延时
        brne delay1         ;不为0转
        sbi $18,7           ;关十位
        nop
        mov r30,r28         ;个位显示字符值,第一次显示0
        andi r30,$0f        ;取个位
ledl:   ld r25,z            ;获显示字符
        out portd,r25       ;送D口显示
        sbi $18,7           ;关十位,硬件设定高电平不亮
        cbi $18,6           ;选通个位,硬件设定低电平亮
        ldi r27,$10         ;延时常数
delay2: dec r26             ;-1,延时
        brne delay2         ;不为0转
        dec r27             ;-1.延时
        brne delay2         ;不为0转
        sbi $18,6           ;关个位
        nop
        dec r20             ;显示数-1
        brne start          ;(R20)不为0转
        inc r28             ;+1
        rjmp main           ;返回主程序
```

7.3.7 廉价的A/D转换器

源程序:SLAVR735.ASM。

必须按图7.2接线才能正常工作!

AVR单片机的AT90系列,片内置有模拟比较器。这一节介绍用AT90S1200单片机实现廉价A/D转换器。

一、硬件设计

使用AVR单片机、一个外部电阻和一个外部电容器设计成一个A/D转换器,并使用片内的定时器/计数器中断和模拟比较器中断。采用RC模拟转换原理。这种转换方法精确度较低,转换时间较短,适用一般要求不高的场合。硬件连接如图7.2所示。必须按此接线才能正常工作!

二、软件编程

源程序:模拟比较AD.ASM。

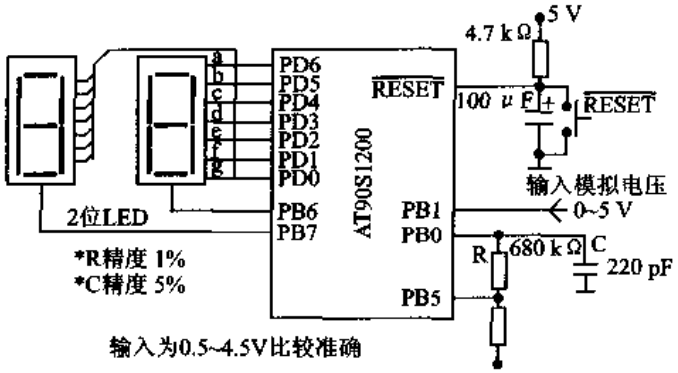
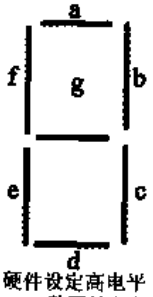


图 7.2 硬件连接图

```
;* * * * * * * * * * * * * * * * * * * * AVR 单片机实用实验程序 * * * * * * * * * * * * * * *
;* 标题:廉价的 A/D 转换器
;* 版本:1.0
;* 最后更新日期:2000.08.08
;* 支援 E-mail: gzsl@sl.com.cn
;* 描述:
;* 用 AVR Studio 调试软件窗口,观察指令执行变化情况
;* 作者: SL.Z
;* 程序适用于所有 AT90S 系列单片机
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include"1200def.inc"          ;应用器件配置文件,*.ASM 所在文件夹中不能缺,不然汇编出错提示
.org $0000
    rjmp reset                 ;复位处理
.org $0002                     ;EXT_INT0 外部中断入口地址
    rjmp inter                 ;转 T0 溢出中断服务程序
.org $0003                     ;TIMI_CAPT 定时器外部中断入口地址
    rjmp inter                 ;转模拟比较器处理中断服务程序
.org $0010                     ;主程序
reset: ldi r20,$ff             ;设置 D 口为输出
       out ddrd,r20            ;送 D 口方向寄存器
       sbi $18,7               ;置 I/O 寄存器 PORTB 的 7 位,LED 数目管 10 位片选
       sbi $18,6               ;置 I/O 寄存器 PORTB 的 6 位,数目管个位片选
       ldi r20,$fe             ;R0~R15 存放显示字符.见表 7.6
       mov r0,r20              ;0
       ldi r20,$b0
       mov r1,r20              ;1
       ldi r20,$ed
       mov r2,r20              ;2
       ldi r20,$f9
       mov r3,r20              ;3
       ldi r20,$b3
       mov r4,r20              ;4
```

表 7.6　LED 字形表

h	a	b	c	d	e	f	g	十六进制码	字形	R
1	1	1	1	1	1	1	0	FEH	0	R0
1	0	1	1	0	0	0	0	B0H	1	R1
1	1	1	0	1	1	0	1	EDH	2	R2
1	1	1	1	1	0	0	1	F9H	3	R3
1	0	1	1	1	0	1	1	B3H	4	R4
1	1	0	1	1	0	1	1	DBH	5	R5
1	1	0	1	1	1	1	1	DFH	6	R6
1	1	1	1	0	0	0	0	F0H	7	R7
1	1	1	1	1	1	1	1	FFH	8	R8
1	1	1	1	0	0	1	1	F3H	9	R9
1	1	1	1	0	1	1	1	F7H	A	R10
1	0	0	1	1	1	1	1	9FH	B	R11
1	1	0	0	1	1	1	0	CEH	C	R12
1	0	1	1	1	1	0	1	BDH	D	R13
1	1	0	0	1	1	1	1	CFH	E	R14
1	1	0	0	0	1	1	1	C7H	F	R15



硬件设定高电平
LED数码管点亮

```
        ldi r20, $ db
        mov r5,r20          ;5
        ldi r20, $ df
        mov r6,r20          ;6
        ldi r20, $ f0
        mov r7,r20          ;7
        ldi r20, $ ff
        mov r8,r20          ;8
        ldi r20, $ f3
        mov r9,r20          ;9
        ldi r20, $ f7
        mov r10,r20         ;A
        ldi r20, $ 9f
        mov r11,r20         ;B
        ldi r20, $ ce
        mov r12,r20         ;C
        ldi r20, $ bd
        mov r13,r20         ;D
        ldi r20, $ cf
        mov r14,r20         ;E
        ldi r20, $ c7
        mov r15,r20         ;F
        main:rcall conini   ;初始化 A/D 转换器
        sei                 ;开中断
        ldi r16, $ fc       ;置 B 口为输出,PB0,PB1 为输入,0B1111 1100
        out ddrb,r16
ddelay: clr r16             ;延时,清暂存计数器 1
        ldi r17, $ f1       ;复位暂存计数器 2
loop:   inc r16             ;清暂存计数器 1,加 1 计数
```

```
        brne loop           ;检查暂存计数器1,不为0转,为0顺执
        inc r17             ;清暂存计数器2,加1计数
        brne loop           ;检查暂存计数器2,不为0转,为0顺执
        rcall adconv        ;调用启动转换器
wait:   brtc wait           ;等待中断,T标志被清零转移,为1顺执
        clt                 ;清T标志
        rcall fetch         ;调用取显示
        ldi r20,$38         ;反复显示次数
start:  mov r30,r28         ;R28送Z寄存器低位
display:andi r30,$f0        ;显示高位,与,即屏蔽高位,
        swap r30            ;交换半字节,取得高位数据地址
ledh:   ld r25,z            ;取LED高位数据
        out portd,r25       ;高位显示送D口
        sbi $18,6           ;置位,关低位LED显示,
        cbi $18,7           ;清零,硬件设定低电平选中高位LED
        ldi r27,$10         ;延时常数$10
delay1: dec r26             ;延时
        brne delay1
        dec r27
        brne delay1
        sbi $18,7           ;置位,关高位LED
        nop
        mov r30,r28         ;显示低位
        andi r30,$0f
ledl:   ld r25,z
        out portd,r25
        sbi $18,7           ;置位,关高位LED显示,
        cbi $18,6           ;清零,硬件设定低电平LED亮
        ldi r27,$10         ;延时常数$10
delay2: dec r26             ;延时
        brne delay2
        dec r27
        brne delay2
        sbi $18,6           ;关低位LED
        nop
        dec r20             ;-1,
        brne start          ;不为0转,原(R20)=$38即扫描显示38次
        rjmp main           ;返回主程序
inter:  in r28,tcnt0        ;中断服务程序
        clr r16             ;关闭T0
        cli
        out tccr0,r16
        cbi portb,5
```

```
        set
        reti
conini: ldi r16,$0b
        out acsr,r16
        ldi r16,$02
        out timsk,r16
        cbi portb,5
        ret
adconv: ldi r16,$40
        out tcnt0,r16
        clt
        ldi r16,$02
        out tccr0,r16
        sbi portb,5
        ret
fetch:  mov r18,r28          ;取显示
        swap r18             ;半字节交换
        andi r18,$0f         ;与,屏蔽低位
        dec r18              ;-1
        dec r18
        dec r18
        dec r18
        brne l50             ;R18不为0转
        mov r18,r28
        andi r18,$0f         ;与,屏蔽低位
        rjmp fee
  l50:  dec r18
        brne l60
        mov r18,r28
        andi r18,$0f
        ori r18,$10
        rjmp fee
  l60:  dec r18
        brne l70
        mov r18,r28
        andi r18,$0f
        ori r18,$20
        rjmp fee
  l70:  dec r18              ;-1
        brne l80             ;不为0转,为0顺执
        mov r18,r28
        andi r18,$0f         ;与
        ori r18,$30          ;或
```

```
        rjmp fee
    l80: ldi r28, $ ff          ;(R28)= $ FF
        ret
    fee: cbi eecr,0             ;清 I/O 寄存器、EEPROM 控制寄存器 EECR 的 0 位,设为读操作
        out eear,r18            ;R18 送 EEPROM 地址口,输出地址
        sbi eecr,0              ;置位 I/O、寄存器 EEPROM 控制寄存器的 0 位,读选通
        in r28,eedr             ;获得数据、EEPROM 数据寄存器内容送 R28
        ret
.eseg
.org $ 0000                     ; EEPROM 数据地址首址在调试时应把数据写入 EEPROM 中
.db 0x00,0x00,0x03,0x14,0x21,0x25,0x2f,0x33
.db 0x38,0x3f,0x47,0x52,0x59,0x6a,0x78,0x7d
.db 0x81,0x86,0x8b,0x90,0x9a,0x9f,0xa4,0xa7
.db 0xac,0xaf,0xbe,0xc5,0xc9,0xca,0xcc,0xd0
.db 0xd3,0xd5,0xd7,0xd9,0xda,0xdf,0xe0,0xe1
.db 0xe2,0xe3,0xe4,0xe5,0xe6,0xe8,0xe9,0xec
.db 0xed,0xee,0xef,0xf0,0xf3,0xf4,0xf5,0xf6
.db 0xf8,0xf9,0xfa,0xfc,0xff,0xff,0xff,0xff
```

7.3.8 高精度廉价的 A/D 转换器

源程序:SLAVR736.ASM。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * AVR 单片机实用程序 * * * * * * * *
;*  标题:高精度廉价的 A/D 转换器
;*  版本:1.0
;*  最后更新日期:2000.08.08
;*  支援 E mail:gzsl@sl.com.cn
;*  描述:
;*  用网络电阻实现高精度廉价的 A/D 转换,本程序实测调试通过
;*  作者: SL.Z
;*  程序适用于所有单片机,硬件接线图如图 7.3。必须按此接线才能正常工作!
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include"8515def.inc"
.org $ 0000
    rjmp reset
.def temp=r16
.def temp1=r17
.equ label= $ 0100              ;字形表首址
.org $ 0010
reset:  ldi r20, $ 02           ;设堆栈指针
    out sph,r20
    out spl,r20
    ldi r20, $ ff               ;设置 D 口、C 口为输出
```

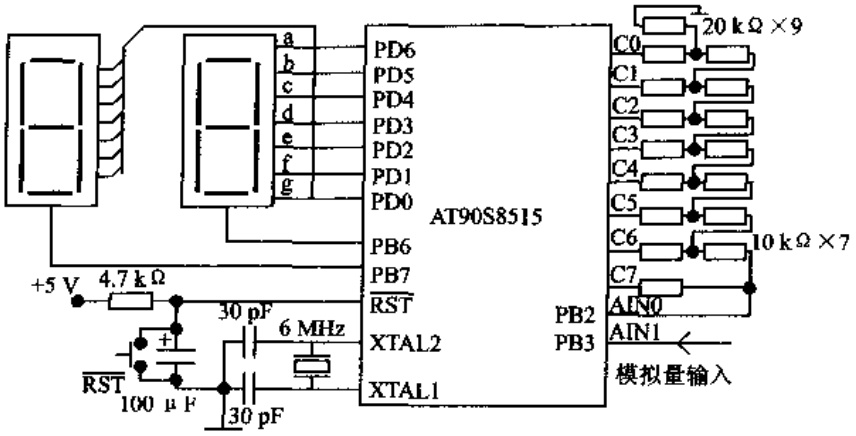


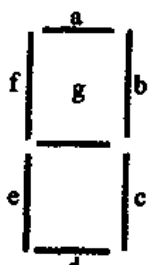
图 7.3　硬件接线图

```
        out ddrd,r20
        out ddrc,r20
        ldi r20,$f0                ;设 PB7～PB4 为输出,PB3～PB0 为输入
        out ddrb,r20
        out portb,r20
        clr r20                    ;清 C 口
        out portc,r20
        sbi $18,7                  ;关显示,硬件设定片选 LED 数码管低电平亮,高电平关
        sbi $18,6
        cli                        ;关中断
        ldi zh,high(label*2);
main;   ldi temp,$00
        nop
loop1;  out portc,temp
        nop
        nop
        nop
        in temp1,acsr              ;读模拟比较器控制和状态寄存器
        sbrs temp1,5
        rjmp naco                  ;模拟比较器输出为 0 转
        rjmp haco                  ;模拟比较器输出为 1 转
naco;   inc temp ;+1
        brne loop1                 ;不为 0 转
        ldi temp,$ff
haco;   mov r28,temp               ;暂存
        ldi r20,$38
display;mov temp,r28               ;显示十位字形
        andi temp,$f0
        swap temp
        clr zl
```

```
    add zl,temp
ledh:   lpm                     ;取十位字形
    out portd,r0
    sbi $18,6                   ;关 PB6,硬件设定片选 LED 数码管高电平关,低电平开(灯亮)
    cbi $18,7                   ;开 PB7
    rcall delay
    mov temp,r28                ;显示个位
    andi temp,$0f
    clr zl
ledl:   add zl,temp
    lpm                         ;取个位字形
    out portd,r0
    sbi $18,7                   ;关 PB7,硬件设定片选 LED 数码管高电平关,低电平开(灯亮)
    cbi $18,6                   ;开 PB6
    rcall delay                 ;调用延时
        dec r20
    brne display
    rjmp main
delay:  ldi r27,$10             ;延时子程序
delay1: dec r26
    brne delay1
    dec r27
    brne delay1
    sbi $18,7                   ;关 PB7
    ret                         ;子程序返回
.cseg
.org $0100                      ;字形表首址,显示字形如表 7.7
.dw 0xb0fe,0xf9ed,0xdbb3,0xf0df
.dw 0xf3ff,0x9ff7,0xbdce,0xc7cf
```

表 7.7 LED 字形表

h	a	b	c	d	e	f	g	十六进制码	字形
1	1	1	1	1	1	1	0	FEH	0
1	0	1	1	0	0	0	0	B0H	1
1	1	1	0	1	1	0	1	EDH	2
1	1	1	1	1	0	0	1	F9H	3
1	0	1	1	1	0	1	1	B3H	4
1	1	0	1	1	0	1	1	DBH	5
1	1	0	1	1	1	1	1	DFH	6
1	1	1	1	0	0	0	0	F0H	7
1	1	1	1	1	1	1	1	FFH	8
1	1	1	1	0	0	1	1	F3H	9
1	1	1	1	0	1	1	1	F7H	A
1	0	0	1	1	1	1	1	9FH	B
1	1	0	0	1	1	1	0	CEH	C
1	0	1	1	1	1	0	1	BDH	D
1	1	0	0	1	1	1	1	CFH	E
1	1	0	0	0	1	1	1	C7H	F



硬件设定高电平
LED数码管点亮

7.3.9　星星灯

源程序：SLAVR737. ASM。

用 AVR 单片机 8 位数据产生随机数，由 PORTA 口及 PORTC 口输出随机数，在 8×8 LED 上显示，硬件接线电路见“7.3.10 按钮猜数”。随机数的种子由程序设定（也可外接开关设定），启动种子后，由移位寄存器以互斥的异或逻辑组合返回循环产生。

```
.include"8515def.inc"
    rjmp    RESET
.def    temp    =r16                ;暂存器
.def    templ   =r17                ;暂存器 1
.def    udata   =r21                ;存随机数送 A 口
.def    ddata   =r22                ;存随机数送 C 口
.cseg
.org    0x10
RESET:      ldi     temp,high(RAMEND)   ;设堆栈指针
            out     SPH,temp
            ldi     temp,low(RAMEND)
            out     SPL,temp
            ldi     temp,0xff           ;设 A 口、C 口为输出
            out     ddra,temp           ;送方向寄存器 A
            out     ddrc,temp           ;送方向寄存器 C
start:      wdr                         ;空操作

            ldi     udata,0x6a          ;设置随机数初值
            ldi     ddata,0x3c          ;
startp:     out     porta,udata         ;输出到 A 口
            out     portc,ddata         ;输出到 C 口
            ldi     temp,0x80           ;设延时常数
            rcall   delay               ;调用延时子程序
            rcall   randm               ;调用 16 位随机数子程序
            rjmp    startp
delay:                                  ;通用延时子程序从略
```

16 位移位产生随机数原理图，如图 7.4 所示。8～16 位移位寄存器产生随机数循环组合见表 7.8。

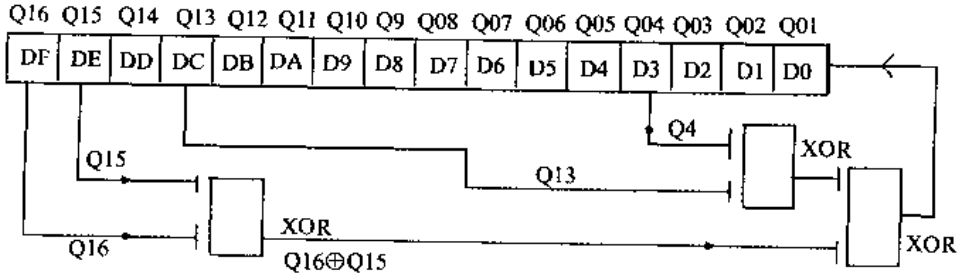


图 7.4　16 位移位产生随机数原理图

```
randm:                          ;产生 16 位随机数子程序
        mov   temp,udata        ;产生 A 口随机数
        mov   temp1,udata       ;
        rol   temp              ;通过进位位左循环移位
        eor   temp1,temp        ;异或
        rol   temp              ;通过进位位左循环移位
        rol   temp              ;通过进位位左循环移位
        eor   temp1,temp        ;异或
        mov   temp,ddata        ;产生 C 口随机数
        swap  temp              ;通过进位位左循环移位
        eor   temp,temp1        ;异或,通过进位位左循环移位
        rol   temp              ;通过进位位左循环移位
        rol   ddata             ;通过进位位左循环移位
        rol   udata             ;通过进位位左循环移位
        ret                     ;子程序返回
```

表 7.8 8～16 位移位寄存器产生随机数循环组合

位 数	循环输入组合 S=2ⁿ−1 Qn XOR Qm
8	Q2 ⊕ Q3 ⊕ Q1 ⊕ Q8（现程序按钮猜数采用 8 位数）
9	Q5 ⊕ Q9
10	Q7 ⊕ Q10
11	Q9 ⊕ Q11
12	Q2 ⊕ Q10 ⊕ Q11 ⊕ Q12
13	Q1 ⊕ Q11 ⊕ Q12 ⊕ Q13
14	Q2 ⊕ Q12 ⊕ Q13 ⊕ Q14
15	Q14 ⊕ Q15
16	Q1 ⊕ Q13 ⊕ Q15 ⊕ Q16

7.3.10 按钮猜数程序

源程序：SLAVR738. ASM。

按钮猜数（电脑摇奖、电脑选出幸运号），开始按钮等待一个不规则且不定序的数据产生，即需要随机数发生器。随机数的种子由程序设定（也可外接开关设定），启动种子后，由移位寄存器以互斥的异或逻辑组合返回循环产生。产生随机数的原理图如图 7.4 所示，产生随机数循环组合如表 7.7 所列。

以 8×8 LED 阵列，开机时为了避免被使用者预测出压按时间对应随机数的变化值，故 LED 字幕以广告动画画面显示，并令随机数随之变化，使其无法预测随机数起始值。广告动画画面共有 4 张，每张有 8 位数据。见"org dpfstb"和图 7.5。

AVR 直接驱动 8×8LED 按钮猜数电路系列图如图 7.6～7.8 所示。

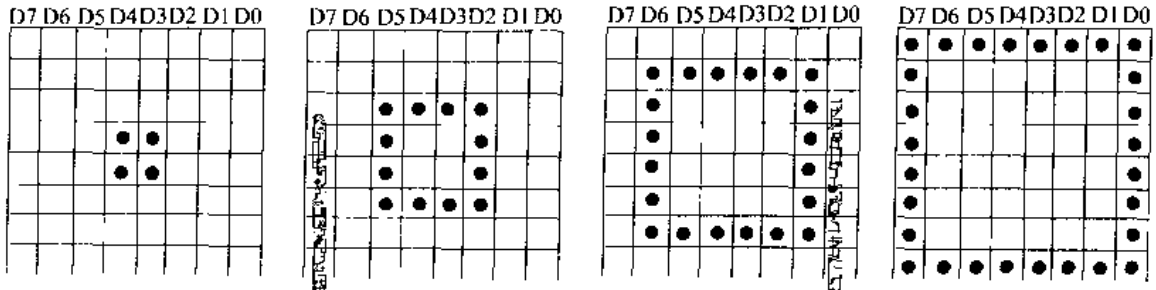


图 7.5.5　LED 字幕广告动画画面图

按下按钮(PD1),AVR 用 8 位数据产生随机数,由 PORTA 口及 PORTC 口输出随机数,在 8×8 LED 上显示真实的按钮猜数。

```
.include "8515def.inc"
.def    peed    =r16
.def    dspn    =r17                ;存显示初始动画次数
.def    temp2   =r18
.def    temp1   =r19
.def    temp    =r20
.def    scndp   =r21
.def    cnt     =r22
.def    rdata   =r23                ;存随机种子数
.def    rdata9  =r24
.equ    dpfstb  =0x01c0             ;大小矩形图表首址
.equ    randtb  =0x0210             ;随机数种子表首址
.equ    numberth=0x0240             ;0~9 数字表首址
.org    $0000
        rjmp    RESET               ;Reset Handle
.cseg
.org $0010
RESET:  ldi     peed,high(RAMEND)   ;设置堆栈 $25F,见器件配置文件"8515def.inc"
        out     SPH,peed
        ldi     peed,low(RAMEND)
        out     SPL,peed
        ldi     peed,0xff           ;对口初始化
        out     ddra,peed           ;设 A 口为输出
        out     ddrc,peed           ;设 C 口为输出
        ldi     peed,0xfd           ;PD1 作输入,且接内部上拉电阻
        out     ddrd,peed           ;PD1 为输入,其余为输出
        ldi     peed,0xff           ;关 D 口
        out     portd,peed
        ldi     peed,0x13           ;显示画面次数
start:  ldi     dspn,0x06           ;显示初始动画
        ldi     zh,high(dpfstb*2)
        ldi     zl,low(dpfstb*2)
dspfm:  rcall   ldtb8               ;调用程序区数送到内存 RAM
        ldi     temp2,0xa0          ;显示动画画面次数
dspfm1: rcall   scan1               ;调用从内存取数显示一次
        sbis    pind,01             ;I/O 口的位被置位跳行,检测到 PD1 按下否
        rjmp    getseed             ;检测到 PD1 按下转
        dec     temp2               ;-1
        brne    dspfm1              ;不为 0 转
        dec     dspn                ;初始画面次数-1
        brne    dspfm               ;不为 0 转
```

```
        rjmp    start                   ;转到显示初始动画
getseed: inc    temp                    ;+1,根据 PD1 按下的时间,选择随机数种子
        sbis    pind,01                 ; I/O 口的位被置位跳行,检测到 PD1 按下否
        rjmp    getseed                 ;检测到 PD1 按下,继续计数
        andi    temp,0x1f               ;按钮松开,取随机数种子与 0X0F 加
        ldi     zh,high(randth * 2)
        ldi     zl,low(randth * 2)
        add     zl,temp
        lpm
        mov     rdata,r0                ;得到随机数种子
next:   ldi     dspn,0x08               ;显示 8 个不同的随机数
repeet: rcall   randm                   ;调用产生随机数子程序
        rcall   dspnumber               ;调用显示 8 个不同的随机数
        dec     dspn                    ;-1
        brne    repeet                  ;dspn 不为 0 转
        rcall   randm                   ;调用产生随机数子程序
guess1: rcall   dspnumber               ;调用显示同一随机数,直到有键按下
        sbic    pind,01                 ;松开后再往下执行(I/O 口清零跳行)
        rjmp    guess1                  ;转显示同一随机数,直到有键按下
wait:   rcall   dspnumber               ;
        sbis    pind,01
        rjmp    wait                    ;等待按钮按下
        ldi     rdata9,0x03             ;显示动画 3 次
start0: ldi     dspn,0x06               ;每次显示 6 幅画面
        ldi     zh,high(dpfstb * 2)
        ldi     zl,low(dpfstb * 2)
dspfm0: rcall   ldtb8                   ;调用从 Z 指向的程序区取数据,送到内存 0080~0087 中
        ldi     temp2,0xa0              ;显示次数
dspfm1a:rcall   scan1                   ;调用从内存 0080~0087 中取数据显示一次
        dec     temp2                   ;-1
        brne    dspfm1a                 ;不为 0 转
        dec     dspn                    ;显示初始动画次数-1
        brne    dspfm0                  ;不为 0 转
        dec     rdata9                  ;显示动画 3 次-1
        brne    start0                  ;不为 0 转
        rjmp    next                    ;转显示 8 个不同的随机数
dspnumber:                              ;显示 1 个 0~9 数字的子程序
        ldi     zh,high(numbertb * 2)
        ldi     zl,low(numbertb * 2)
        add     zl,rdata9
        rcall   ldtb8                   ;取数
        ldi     temp2,0xa0              ;该数字重复显示 A0H 次
dspn1:  rcall   scan1
```

```
        dec     temp2
        brne    dspn1
        ret
scan1:  push    xl                  ;从内存 0080~0087 中取数据显示一次
        ldi     temp,0b01111111
        mov     scndp,temp
        ldi     cnt,0x08
col1:   out     portc,scndp         ;显示屏幕的一列
        ld      r1,x+
        out     porta,r1
        rcall   delay
        sec
        ror     scndp
        dec     cnt
        brne    col1
        pop     xl
        ret
ld1b8:  ldi     xl,0x80             ;从 Z 指向的程序区取数据,送到内存 0080~0087 中
        ldi     xh,0x00
        ldi     temp1,0x08
        push    xl
nexld1: lpm
        st      x+,r0
        ld      r0,z+
        dec     temp1
        brne    nexld1
        pop     xl
        ret
delay:                              ;通用延时子程序从略
randm:  mov     temp,rdata          ;产生 8N(0≤N≤9)随机数子程序
        mov     temp1,rdata
        swap    temp1
        eor     temp,temp1
        rol     temp1
        eor     temp,temp1
        rol     temp1
        eor     temp,temp1
        rol     temp
        rol     rdata
        mov     rdata9,rdata
        andi    rdata9,0x0f
        cpi     rdata9,0x0a
        brsh    randm               ;产生了一个 0≤RDATA9≤9 的随机数
```

```
        lsl     rdata9
        lsl     rdata9
        lsl     rdata9
        ret
.cseg
.org    dpfstb;                         ;大小方框字形表
;small  o
.db         0b00000000,0b00000000,0b00000000,0b00011000
.db         0b00011000,0b00000000,0b00000000,0b00000000
.db         0b00000000,0b00000000,0b00111100,0b00100100
.db         0b00100100,0b00111100,0b00000000,0b00000000
.db         0b00000000,0b01111110,0b01000010,0b01000010
.db         0b01000010,0b01000010,0b01111110,0b00000000
;big    o
.db         0b11111111,0b10000001,0b10000001,0b10000001
.db         0b10000001,0b10000001,0b10000001,0b11111111
.db         0b00000000,0b01111110,0b01000010,0b01000010
.db         0b01000010,0b01000010,0b01111110,0b00000000
.db         0b00000000,0b00000000,0b00111100,0b00100100
.db         0b00100100,0b00111100,0b00000000,0b00000000
.cseg
.org        randtb                  ;随机数种子表
.db         0x5a,0x7b,0x5b,0x4f,0x66,0x6d,0x7d,0x07
.db         0x3b,0x8c,0x67,0x9a,0x99,0x7e,0x2d,0x3e
.db         0x5c,0x6d,0x5b,0x7e,0xf6,0xe7,0x4c,0xc8
.db         0x69,0x9c,0xe2,0x75,0x6c,0xd3,0xe8,0x9a
.cseg
.org        numbertb                ;0～9数字字形表
;0
.db         0b00111000,0b01000100,0b01000100,0b01000100
.db         0b01000100,0b01000100,0b01000100,0b00111000
;1
.db         0b00010000,0b00011000,0b00010000,0b00010000
.db         0b00010000,0b00010000,0b00010000,0b00111000
;2
.db         0b00011100,0b00100010,0b00100000,0b00010000
.db         0b00001000,0b00000100,0b00000010,0b00111110
;3
.db         0b00111100,0b00010000,0b00001000,0b00010000
.db         0b00100000,0b00100000,0b00100010,0b00011100
;4
.db         0b00100000,0b00110000,0b00101000,0b00100100
.db         0b00100010,0b11111110,0b00100000,0b00100000
```

```
;5
.db     0b01111110,0b00000010,0b00111110,0b01000000
.db     0b01000000,0b01000000,0b01000010,0b00111100
;6
.db     0b00110000,0b00001000,0b00000100,0b00111100
.db     0b01000100,0b01000100,0b01000100,0b00111000
;7
.db     0b01111100,0b01000000,0b00100000,0b00010000
.db     0b00001000,0b00001000,0b00001000,0b00001000
;8
.db     0b00111000,0b01000100,0b01000100,0b00111000
.db     0b01000100,0b01000100,0b01000100,0b00111000
;9
.db     0b00111000,0b01000100,0b01000100,0b01111000
.db     0b01000000,0b01000000,0b01000100,0b00111000
```

7.3.11 汉字的输入

源程序:SLAVR739.ASM,硬件电路见"7.3.10 按钮猜数程序"。

```
;* * * * * * * * * * * * * * * * * * * * * * AVR 单片机实验测试程序 * * * * * * * * * * * * *
;*  标题:汉字的输入
;*  版本:1.0
;*  最后更新日期:2000.08.08
;*  支援 E-mail:gzsl@sl.com.cn
;*  描述:
;*  用 AVR Studio 调试软件窗口,观察指令执行变化情况
;*  作者: SL.Z
;*  程序适用于所有单片机;光盘中提供 16×16 汉字显示及汉字库的调用与组成
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
.include "8515def.inc"
.def    dspn    =r23
.def    temp2   =r24
.def    temp1   =r17
.def    temp    =r18
.def    scndp   =r19
.def    cnt     =r20
.equ    dpfstb  =0x01e0
.org    $0000
    rjmp    RESET                   ;Reset Handle
.org $0010
RESET:
    ldi     r16,high(RAMEND)        ;设堆栈为 $025F
    out     SPH,r16
```

```
    ldi     r16,low(RAMEND)
    out     SPL,r16
    ldi     r16,0xff                ;设 A 口、C 口为输出
    out     ddra,r16                ;A 口方向寄存器
    out     ddrc,r16                ;C 口方向寄存器
dspfst: ldi     dspn,0x07           ;显示次数
    ldi     zh,high(dpfstb * 2)     ;高位取数
    ldi     zl,low(dpfstb * 2)      ;低位取数
dspfm:  rcall   ldtb8               ;调用取字形子程序
    ldi     temp2,0xa0              ;循环次数
dspfm1: rcall   scan1
        dec     temp2
        brne    dspfm1
        dec     dspn
        brne    dspfm
        rjmp    dspfst
scan1:  push    xl                  ;XL 进栈
        ldi     temp,0b01111111     ;第 1 次选中 PC7,硬件设定低电平 LED 亮
        mov     scndp,temp
        ldi     cnt,0x08            ;取 1 个字符需 8 次取数
col1:   out     portc,scndp         ;选通数据送 C 口,第 1 次选中 PC7
        ld      r1,x+               ;取数后地址指针加 1
        out     porta,r1            ;数据送 A 口
        ldi     r16,0x10            ;送延时常数
        rcall   delay               ;调用延时
        sec                         ;置位进位位
        ror     scndp               ;通过进位位右循环
        dec     cnt                 ;-1
        brne    col1                ;不为 0 转
        pop     xl                  ;为 0,XL 出栈
        ret                         ;子程序返回
ldtb8:  ldi     xl,0x80             ;取数(字形)子程序
        ldi     xh,0x00             ;
        ldi     temp1,0x08          ;取数(字符)次数
        push    xl                  ;XL 进栈
nexld1: lpm                         ;从程序存储器取数,将 Z 寄存器指向的 1 个字节装入 R0
        st      x+,r0               ;X 寄存器内容(字形)送 R0 后,X 指针加 1
        ld      r0,z+               ;Z 寄存器内容(字形)送 R0 后,Z 指针加 1
        dec     temp1               ;-1
        brne    nexld1              ;未完继续取数
        pop     xl                  ;XL 出栈
        ret                         ;子程序返回
delay:                              ;通用延时子程序从略
```

```
.org    dpfstb                    ;
```

;在 8×8 的方格中填字,硬件设定高电平为 1 点亮 LED。注意:编码的高、低位与习惯编码书写方法正好相反,其目的为汉字、字符书写符合人们习惯(正写)! 8×8 方格填字如图 7.9 所示。

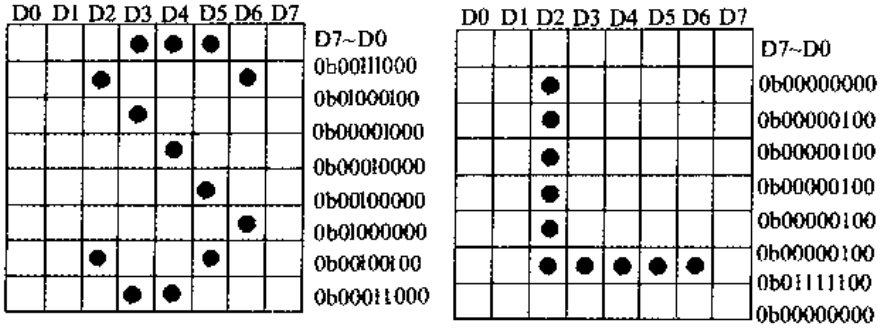


图 7.9 8×8 方格填字

```
;字符 S
.db     0b00111000,0b01000100,0b00001000,0b00010000
.db     0b00100000,0b01000000,0b00100100,0b00011000
;字符 L
.db     0b00000000,0b00000100,0b00000100,0b00000100
.db     0b00000100,0b00000100,0b01111100,0b00000000
;字符"双龙电子"字型表略
```

7.4 复杂实用程序

7.4.1 10 位 A/D 转换

源程序:SLAVR741.ASM。

```
;* * * * * * * * * * * * * * * * * AVR 单片机实验测试程序 * * * * * * * * * * * * * * * * * *
;* 标题:AT90S8535 的 10 位 A/D 转换器
;* 版本:1.0
;* 最后更新日期:2000.08.08
;* 支援 E-mail:gzsl@sl.com.cn
;* 描述
;* 用 AVR Studio 调试软件窗口,观察指令执行变化情况
;* 作者: SL.Z
;* 硬件电路及本程序实测调试通过
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

硬件电路必须按图 7.10 连接!

```
.include"8535def.inc"          ;AT90S8535 器件配置文件
.org $0000
    rjmp reset
.org $000e                     ;INTER 中断入口地址
    rjmp inter
.def    bledbyte=r19           ;存放 ADCH
```

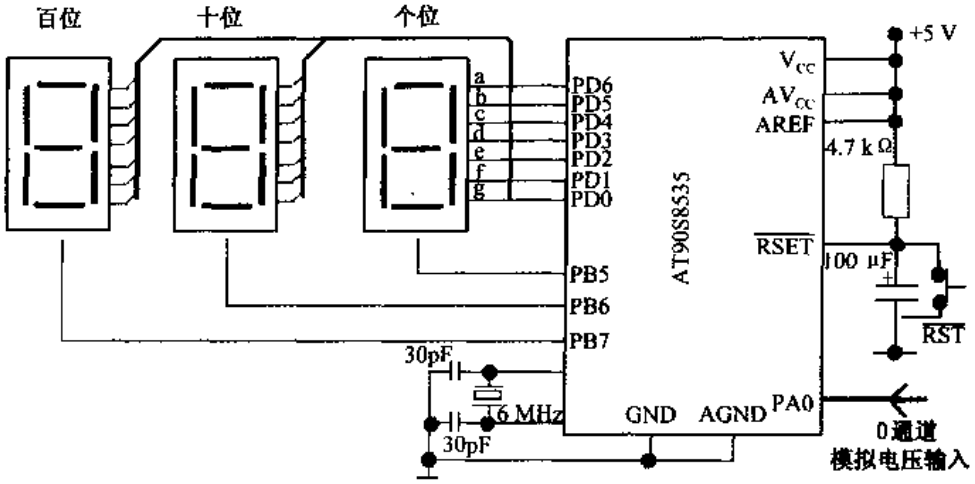


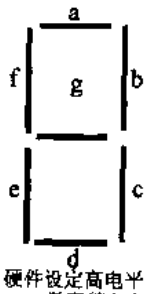
图 7.10　AT90S8535 10 位 A/D 转换电原理图

```
.def     lledbyte=r18          ;存放 ADCL
.equ label=$0100               ;字形表首址
.equ     distime=$38           ;显示次数
.def     temp=r16
reset: ldi temp,$02            ;设堆栈指针 $025F
    out sph,temp
    ldi temp,$5f
    out spl,temp
    ldi temp,$ff               ;B 口、D 口为输出
    out ddrb,temp
    out ddrd,temp
    out portd,temp             ;开通 LED 数码管,硬件设定高电平亮
clr temp
    out ddra,temp              ;A 口为输入
    out porta,temp
main:    clt                   ;清 T 标志
    sei                        ;开中断
    rcall conini               ;调用 A/D 初始化
wait:    brtc wait             ;等待中断
    clt
    ldi r20,distime
start:   ldi zh,high(label*2)
    mov temp,r19
    rcall outpd                ;取出字符
    cbi $18,7                  ;显示百位 LED
    sbi $18,6                  ;显示十位 LED
    sbi $18,5                  ;显示个位 LED
    rcall delay                ;延时
    mov temp,r18
```

```
        swap temp
        rcall outpd
        cbi $18,6                    ;显示十位 LED
        sbi $18,7                    ;显示百位 LED
        sbi $18,5                    ;显示个位 LED
        rcall delay                  ;延时
        mov temp,r18
        rcall outpd
        cbi $18,5                    ;显示个位 LED
        sbi $18,6                    ;显示十位 LED
        sbi $18,7                    ;显示百位 LED
        rcall delay                  ;延时
        dec r20                      ;显示次数减 1
        brne start                   ;显示次数未完转,为 0 顺执
        rjmp main                    ;返回主程序
inter:  in lledbyte,adcl             ;中断后,读取 ADC 数据寄存器低位数据
        in hledbyte,adch             ;取 ADC 数据寄存器高位数据
        cbi adsc,6                   ;停止 ADC 转换
        set                          ;置 T 标志
        reti                         ;中断返回
conini: ldi temp,$8d                 ;设置 ADC 转换,中断触发,ADC 为单一模式且 32MCU 除频
        out adcsr,temp
        clr temp
        out admux,temp               ;选择 0 通道
        sbi adsc,6
        ret
delay:  ldi r27,$10                  ;延时子程序
delay1: dec r26
        brne delay1
        dec r27     brne delay1
        ret
outpd:  andi temp,$0f
        ldi zl,low(label*2)
        add zl,temp
        lpm                          ;取字符
        out portd,r0                 ;输出字符
        ret
.cseg
.org $0100                           ;字形表首地址,字形表如表 7.9 所列
.dw 0xb0fe,0xf9ed,0xdbb3,0xf0df
.dw 0xf3ff,0x9ff7,0xbdce,0xc7cf
```

表 7.9　LED 字形表

h	a	b	c	d	e	f	g	十六进制码	字形
1	1	1	1	1	1	1	0	FEH	0
1	0	1	1	0	0	0	0	B0H	1
1	1	1	0	1	1	0	1	EDH	2
1	1	1	1	1	0	0	1	F9H	3
1	0	1	1	1	0	1	1	B3H	4
1	1	0	1	1	0	1	1	DBH	5
1	1	0	1	1	1	1	1	DFH	6
1	1	1	1	0	0	0	0	F0H	7
1	1	1	1	1	1	1	1	FFH	8
1	1	1	1	0	0	1	1	F3H	9
1	1	1	1	0	1	1	1	F7H	A
1	0	0	1	1	1	1	1	9FH	B
1	1	0	0	1	1	1	0	CEH	C
1	0	1	1	1	1	0	1	BDH	D
1	1	0	0	1	1	1	1	CFH	E
1	1	0	0	0	1	1	1	C7H	F



硬件设定高电平
LED数码管点亮

7.4.2　步进电机控制程序

源程序：SLAVR742. ASM。另光盘中提供两相 H 型桥式驱动电路及驱动程序。

自 60 年代初期步进电机面世以来，它的重要性越来越显著了。它用来驱动时钟和其他采用指针的仪器，如打印机、绘图仪、磁盘光盘驱动器、各种自动控制阀、各种工具以及机器人等。关于步进电机工作原理，请参考有关资料。

下面用单极 1－2 相激磁方法步进电机做实验，即 1 极、2 极、1 极、2 极……依次循环。如何用单极二相激磁方法控制步进电机，由读者或用户自行编制实验程序。

实验选用 4.5V 步进电机，用 5V 即可，实验时节省一组步进电机驱动电源。

型号：MA82135；相数：2 相；电压：4.5V；电流/相：0.12A/相；电阻相：34Ω/相；重量：30g。其激磁波形图如图 7.11 和图 7.12 所示。

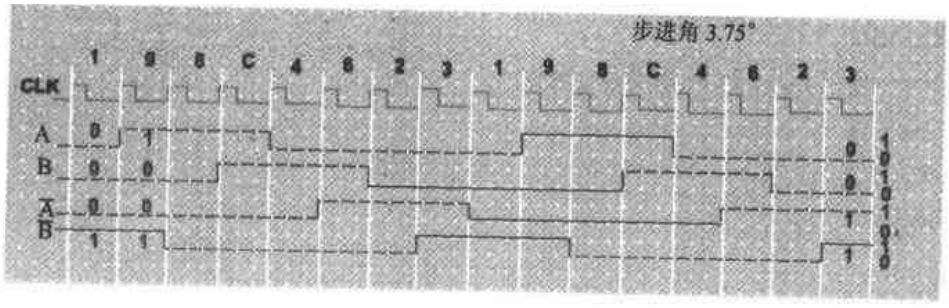


图 7.11　单极 1～2 相激磁波形图

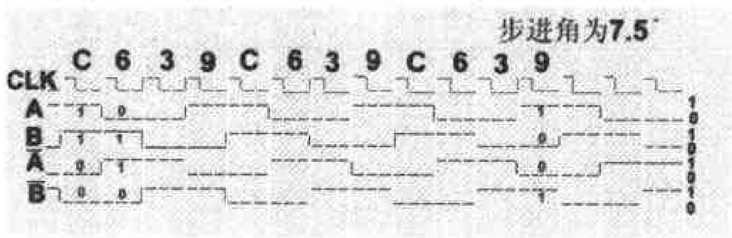


图 7.12 单极二相激磁波形图

步进电机驱动接线图和控制模块见图 7.13 和图 7.14。

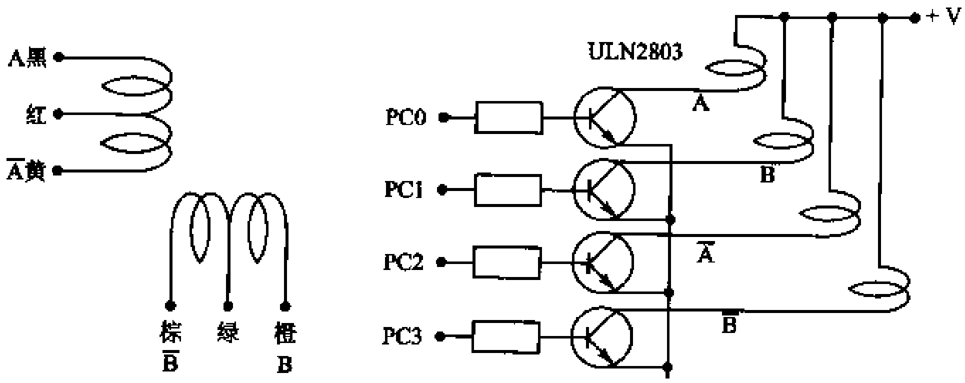


图 7.13 步进电机驱动接线图

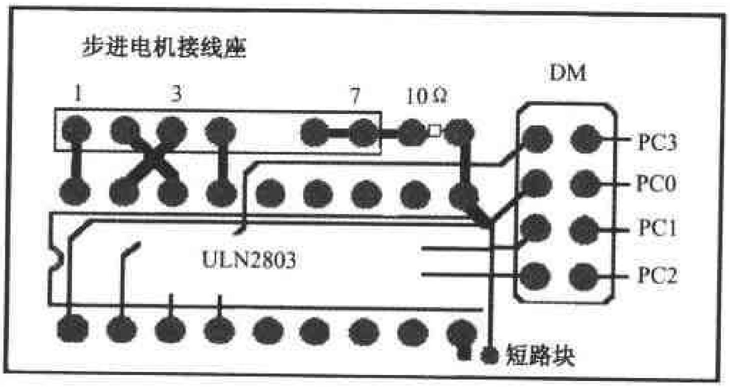


图 7.14 步进电机控制模块

```
;*****************************************************
;*步进电机控制程序(单极 1～2 相)
;* SLAVR742.ASM
;* use ULN2803 ;使用 PC0～PC3 驱动步进电机
;* use 11～17new bord
;*****************************************************
.include"8515def.inc"
.def    temp    =r16
.def    dt      =r19
.def    np      =r17
```

```
.def    step      =r18
.def    TStep     =r20
.def    cnt       =r21
.equ    turntab=0x0200
.org    $0000
        rjmp    RESET
.cseg
.org    0x010
RESET:
        ldi     temp,low(RAMEND)                ;设堆栈
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPL+1,temp
        ser     TEMP                            ;C口设置为输出
        OUT     ddrc,TEMP
        ldi     zl,low(turntab*2)               ;步进电机旋转资料指针
        ldi     zh,high(turntab*2)
        ldi     np,4
        ldi     temp,$44
        out     portc,temp                      ;初始化
        ldi     TStep,$25
        rcall   delay
        ldi     cnt,10
        clt
rep:    ldi     step,192
        ldi     TStep,1                         ;1~255
        rcall   turn
        dec     cnt
        brne    rep
loop:   nop
        rjmp    loop
;*********************************************************
;  t=1          uncircle turn                  ;T=1逆时针转
;  t=0          circle  turn                   ;T=0顺时针转
;  96           step  a  turn
;  TStep is     time  of a step
;*********************************************************
turn:   brts uncircle                           ;判转向
inc     np                                      ;正转
        cpi     np,8
        brne    next
        clr     np
next:   push    zl
```

```
        add     zl,np
        lpm
        out     portc,r0
        pop     zl
        rcall   delay
        dec     step
        brne    turn
        ret
uncircle:                                   ;反转
        dec     np
        cpi     np,$ff
        brne    next
        ldi     np,$07
        rjmp    next
delay:  push    TStep                       ;延时子程序
del1:   ldi     dt,70
del2:   push    dt
del3:   dec     dt
        brne    del3
        pop     dt
        dec     dt
        brne    del2
        dec     TStep
        brne    del1
        pop     TStep
        ret
.org    turntab
;       0    1    2    3    4    5    6    7    ;步进电机旋转资料表
.db     0x11,0x99,0x88,0xcc,0x44,0x66,0x22,0x33
```

7.4.3 测脉冲宽度

源程序:SLAVR743.ASM。

本程序略加修改可应用于测速、测距、测频率等,用 LED 数码管显示。本程序在 SL-AVR 开发下载实验器上验证通过。硬件连接:AT90S8515 的 ICP 作输入,测量输入脉冲宽度时间以 μs 计算,由 6 位 LED 十进制显示。

```
.include"8515def.inc"
        rjmp reset
.def    cnt1d   =r03                ;cnt1d~cnt5d:存放十进制数
.def    cnt2d   =r04                ;存放形式(压缩 BCD 码)
.def    cnt3d   =r05                ;cnt1~cnt5 依次存放个位、十位…
.def    cnt4d   =r06
.def    cnt5d   =r07
```

```
.def    tmp1h  =r08
.def    tmp2h  =r09                ;tmp1h,tmp2h:存放第一次捕捉的 timer1 的值
.def    tim1l  =r10
.def    tim1h  =r11                ;存放 timer1 溢出的次数
.def    tmp5h  =r12
.def    tmp6h  =r13                ;tmp5h~tmp8h:存放二次捕捉 timer1 的差值
.def    tmp7h  =r14
.def    tmp8h  =r15
.def    temp   =r16                ;暂存器
.def    cnt4   =r17
.def    cntn   =r18
.def    cntn1  =r19
.def    tempn  =r20                ;利用 tempn 的 d0 位判断是哪一次捕捉
.def    tempn1 =r21
.cseg
.org 0x03                          ;icp 触发中断向量
icpt1:
    rjmp  captr
.org 0x06                          ;timer1 溢出中断向量
intt1:
    rjmp  calts
.cseg
.org 0x10
captr:                             ;icp 触发中断子程序
    wdr
    sbrc  tempn,  00               ;判断是哪一次捕捉
    rjmp  cap2
cap1:                              ;第一次捕捉处理
    in    tmp1h, icr1l
    in    tmp2h, icr1h             ;把捕捉的 timer1 的值放在 tmp1h 和 tmp2h
    ori   tempn, $01
    ldi   r16,   0b10001000
    out   timsk, r16               ;置 timer1 中断和捕捉中断
    reti
cap2:                              ;第二次捕捉处理
    in    tmp5h, icr1l
    in    tmp6h, icr1h             ;把捕捉的 timer1 的值放在 tmp5h 和 tmp6h
    rcall transd                   ;
    rcall htd4                     ;把 tmp5h~tmp8h 转成十进制
    ldi   tempn, $00               ;令 tempn d0=0
    rcall clrtmm
    ldi   r16,   0b00000000
    out   timsk, r16               ;除 timer1 中断和置捕捉中断
```

```
    ret
calts:                                  ;timer1 溢出中断子程序
    inc   tim1l
    breq  ov
    rjmp  b
ov:     inc   tim1h
    brne  b
    set
b:    reti
transd:                                 ;二次捕捉 timer1 的差值处理
    clc
    sbc   tmp5h, tmp1h                  ;结果放在 tmp5h~tmp8h
    sbc   tmp6h, tmp2h
    brcc  posv
    dec   tim1h
    dec   tim1l
posv:
    mov   tmp7h, tim1l
    mov   tmp8h, tim1h
    ret
reset:
    ldi   temp,  low(ramend)
    out   spl,   temp
    ldi   temp,  high(ramend)           ;设置堆栈
    out   spl+1, temp
    ldi   temp,  $0e                    ;设定 wdtcr 中 wde=1
    out   wdtcr, temp
    cli
    ldi   temp,  $ff                    ;初始化数码管状态
    out   ddrb,  temp                   ;B 口:数码管数据输出
    out   ddrd,  temp                   ;D 口:pd0~pd5 为数码管片选
    out   portd, temp                   ;数码管低电平选中
    ldi   temp,  $00
    out   portb, temp                   ;共阴极,数码管全灭
    ldi   tempn1,00
    rcall clrtm
    ldi   temp,  $00
    out   tccr1a,temp
    out   tccr1b,temp
    out   tcnt1h,temp                   ;装入计数值
    out   tcnt1l,temp                   ;tcnt1h—tcht1l=00
    clt
    sei                                 ;开中断
```

```
    ldi     r16,   0b00001000
    out     timsk, r16
    ldi     r16,   0b11000010
    out     tccr1b,r16                    ;开始计数
reptw:
    sbrs    tempn1,00
    rjmp    reptw
    rjmp    reset
clrtm:
    clr     tmp1h
    clr     tmp2h
    clr     tmp5h
    clr     tmp6h
    clr     tmp7h
    clr     tmp8h
clrtmm:
    clr     tim1h
    clr     tim1l
    clr     cnt4
    clt
    ret
htd4:                                     ;把 tmp5h~tmp8h 转成十进制子程序
    ldi     cntn,  32
    clr     cnt1d
    clr     cnt2d
    clr     cnt3d
    clr     cnt4d
    clr     cnt5d
    clc
loopd:
    rol     tmp5h
    rol     tmp6h
    rol     tmp7h
    rol     tmp8h
    rol     cnt1d
    rol     cnt2d
    rol     cnt3d
    rol     cnt4d
    rol     cnt5d
    dec     cntn
    breq    end
    rcall   adjn
    rjmp    loopd
```

```
end:                                    ;在数码管显出十进制数
    ldi     cntn,   $ff
end1:
    ldi     cntn1,  $ff
cnd2:       mov   temp,   cnt1d
    andi    temp,   $0f                 ;显示个位
    rcall   a
    cbi     portd, 00
    nop
    sbi     portd, 00
    mov     temp,   cnt1d
    andi    temp,   $f0                 ;显示十位
    swap    temp
    rcall   a
    cbi     portd, 01
    nop
    sbi     portd, 01
    mov     temp,   cnt2d
    andi    temp,   $0f                 ;显示百位
    rcall   a
    cbi     portd, 02
    nop
    sbi     portd, 02
    mov     temp,   cnt2d
    andi    temp,   $f0                 ;显示千位
    rcall   a
    swap    temp
    rcall   a
    cbi     portd, 03
    nop
    sbi     portd, 03
    mov     temp,   cnt3d
    andi    temp,   $0f                 ;显示万位
    rcall   a
    cbi     portd, 04
    nop
    sbi     portd, 04
    mov     temp,   cnt3d
    andi    temp,   $f0                 ;显示十万位
    swap    temp
    rcall   a
    cbi     portd, 05
    nop
```

```
    sbi     portd, 05
    dec     cntn1
    brne    end2
    dec     cntn
    brne    end1
    ldi     tempn1, $01
    ret
a:
    ldi     zh,     high(zk * 2)
    ldi     zl,     low(zk * 2)
    add     zl,     temp
    lpm
    out     porth, r0
    ret
adjn:
    push    cntn
    mov     cntn,   cnt1d
    rcall   adjd1
    mov     cnt1d, cntn
    mov     cntn,   cnt2d
    rcall   adjd1
    mov     cnt2d, cntn
    mov     cntn,   cnt3d
    rcall   adjd1
    mov     cnt3d, cntn
    mov     cntn,   cnt4d
    rcall   adjd1
    mov     cnt4d, cntn
    mov     cntn,   cnt5d
    rcall   adjd1
    mov     cnt5d, cntn
    pop     cntn
    ret
adjd1:
    ldi     tempn, 3
    add     tempn, cntn
    sbrc    tempn, 3
    mov     cntn,   tempn
    ldi     tempn, $30
    add     tempn, cntn
    sbrc    tempn, 7
    mov     cntn,   tempn
    wdr
```

```
    ret
.equ    zk=0x0200
.org    zk   ;字形表
.db     0x03f,0x006,0x05b,0x04f
.db     0x066,0x06d,0x07d,0x007
.db     0x07f,0x06f,0x077,0x07c
.db     0x039,0x05e,0x071
```

7.4.4 LCD 显示 8 字循环

源程序:SLAVR744.ASM。

LCD 显示器 1602AT(S)R 简介:LCD 显示器模块 1602AT(S)R 为 2×16 字符。含有 5×10 或 5×7 点 LCD,共 12×16=192 种 CG 显示字形及双组、8 个自由利用软件设定(CGRAM)的 5×8 点图字形。因此,除内部固定 192 种字形外,再加上此 16 个可自由设定图字形,共计 208 种字图形,如字符代码表所示。因 5×8 个点输入设定,故 5 个点仅占用 D4～D0 的 5 位,而 D7～D5 则可为任意值。第 8 行值为游标地址,因此共 8 行占 8 个地址,组成 1 个字形及表示游标地址。总共 8 个设定字图形,因此占有 $8\times8=2^6$ 个地址,CG 地址设定值为 D5～D0。

LCD 引脚功能说明,如图 7.15 所示。

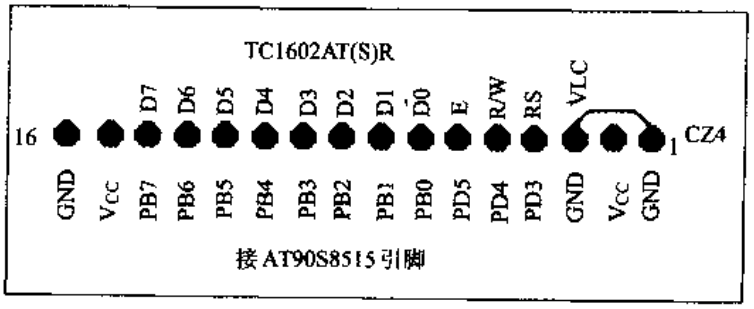


图 7.15 LCD 显示器插座引脚功能

(1) GND:电源地,0V。

(2) V_{CC}:电源+5V。

(3) VLC:LCD 驱动电压 0～5V 对比度调节电压。

(4) RS 寄存器选择信号:

RS=0,指令寄存器 IR 写入(WRITE)。

① 忙(BUSY FLAG)读取(READ);

② 地址计数器(ADDRESS COUNTER)AC 读取(READ)。

RS=1:数据寄存器(DATA REGISTER)读取及写入(READ/WRITE)。

(5) R/W 读/写控制信号(READ/WRITE):R/W=1 读取(READ);R/W=0 写入(WRITE)。

(6) E(ENABLE)片使能信号:写数据控制,下降沿触发。

(7) 7～14 脚为 DB0～DB7:8 位数据总线,三态双向。若作为 4 位传送时应令 DL=0,以

DB4～DB7作传送，将8位数据分2次传送。

(8) 15一般不用(空)，如有背光LED，则接V_{CC}。

(9) 16一般不用(空)，如有背光LED，则接GND。

LCDTC1602 CG RAM字形结构设定输入表及LCD指令表见表7.10和表7.11。

表7.10　LCDTC1602 CG RAM字形结构设定输入表

字形码(DD RAM数据)								CG RAM地址						字形图样(CG RAM)数据							
7	6	5	4	3	2	1	0	5	4	3	2	1	0	7	6	5	4	3	2	1	0
高位							低位	上位					下位	上位			(5×7字形)				下位
											0	0	0	*	*	*	1	1	1	1	0
											0	0	1				1	0	0	0	1
											0	1	0				1	0	0	0	1
											0	1	1				1	1	1	1	0
0	0	0	0	*	0	0	0	0	0	0	1	0	0				1	0	1	0	0
											1	0	1				1	0	0	1	0
											1	1	0				1	0	0	0	1
											1	1	1	*	*	*	0	0	0	0	0
											0	0	0	*	*	*	1	0	0	0	1
											0	0	1				0	1	0	1	0
											0	1	0				1	1	1	1	1
											0	1	1				0	0	1	0	0
0	0	0	0	*	0	0	1	0	0	1	1	0	0				1	1	1	1	1
											1	0	1				0	0	1	0	0
											1	1	0				0	0	1	0	0
											1	1	1	*	*	*	0	0	0	0	0
											0	0	0	*	*	*					
											0	0	1								
0	0	0	0	*	1	1	1	1	1	1											
											1	0	1								
											1	1	0								
											1	1	1								

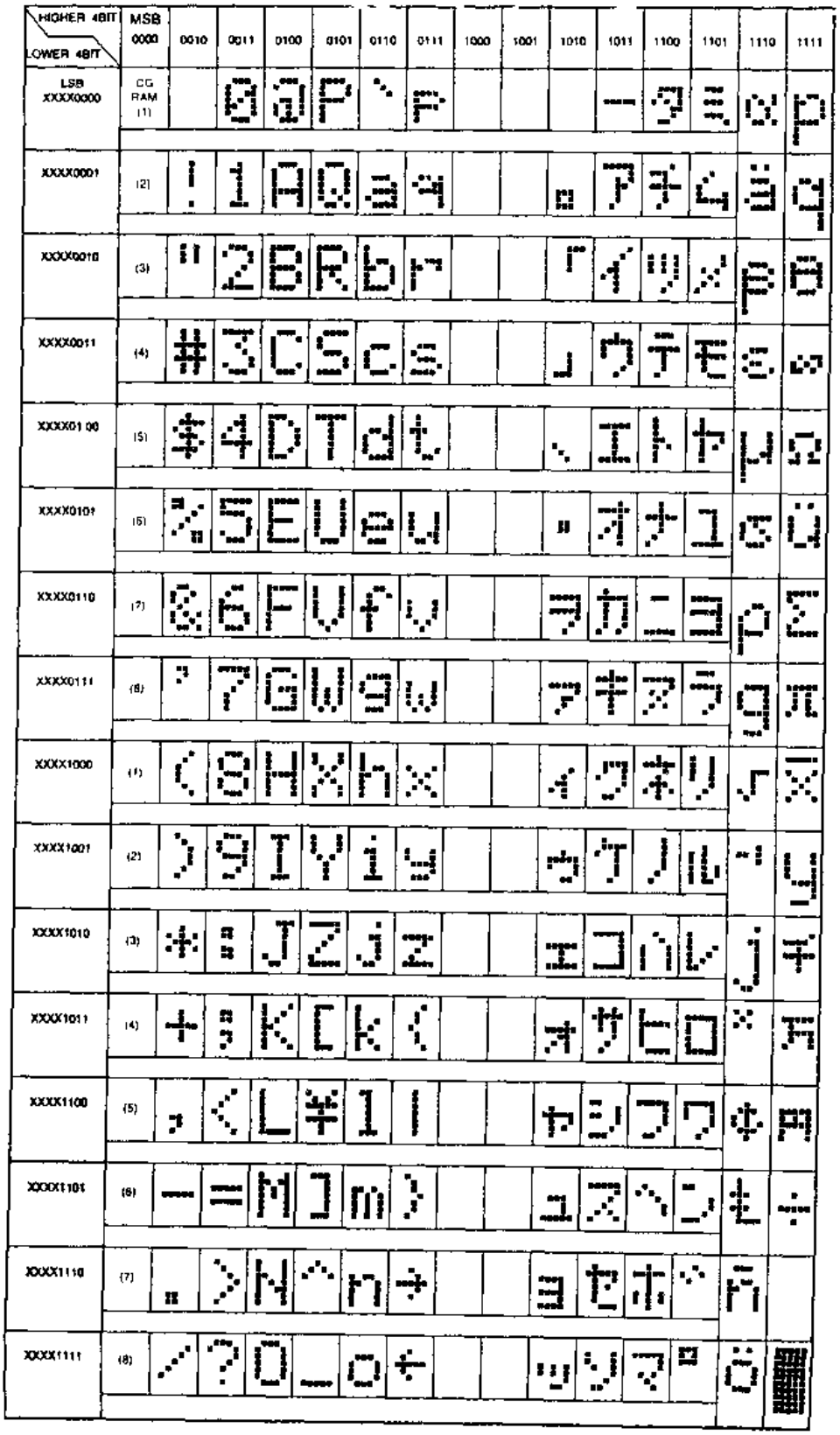
表 7.11 LCD 指令表

指令	指令码										说明	执行周期 f_{osc} = 250kHz
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
清屏	0	0	0	0	0	0	0	0	0	1	请除屏幕，置 AC 为 0，光标回位	1.64ms
光标返回	0	0	0	0	0	0	0	0	1	x	DDRAM 地址为 0，显示回原位，DDRAM 内容不变	1.64ms
设置输入方法	0	0	0	0	0	0	0	1	I/D	S	设置光标移动方向，并指定显示是否移动	40μs
显示开关	0	0	0	0	0	0	1	D	C	B	设置显示开或关(D)，光标开关(C)，光标所在字符闪烁(B)	40μs
移位	0	0	0	0	0	1	S/C	R/L	*	*	移动光标及整体显示，同时不改变 DDRAM 内容	40μs
功能设置	0	0	0	0	1	DL	N	F	*	x	设置接口数据(DL)、显示行数(L)、字符字体(F)	40μs
CGRAM 地址设置	0	0	0	1	ACG						设置 CGRAM 地址，设置后发送接收数据	40μs
DDRAM 地址设置	0	0	1	ADD							设置 DDRAM 地址，设置后发送接收数据	40μs
忙标志/读地址计数器	0	1	BF	AC							读忙标志(BF)正在执行内部操作，并读地址计数器内容	40μs
CGRAM/DD RAM 数据写	1	0	写数据								从 CGRAM 或 DDRAM 写数据	40μs
CGRAM/DD RAM 数据读	1	1	读数据								从 CGRAM 或 DDRAM 读数据	40μs
	I/D=1：增量方式；I/D=0：减量方式 S=1：移位 S/C=1：显示移位；S/C=0：光标移位 R/L=1：右移；R/L=0：左移 DL=1：8 位；DL=0：4 位 N=1：2 行；N=0：1 行 F=1：5×10 字体；F=0：5×7 字体 BF=1：执行内部操作；BF=0：可接收指令										DDRAM：显示数据 RAM CGRAM：字符发生器 RAM ACG：CGRAM 地址 ADD：DDRAM 地址及光标地址 AC：地址计数器用于 DDRAM 和 CGRAM	执行周期随主频改变而改变，例如当 fcp 或 fosc=270kHz 时： (40μs×250)/270=37μs

* ICC AVR C 编译器已提供 LCD 显示组件的软件模块，只需填入一些参数，就能显示你想要做的事情。

字符点阵LCD模块字符代码表见表7.12。

表7.12　字符代码表



```
.include"8515def.inc"
.def temp=r16
.def temp1=r17
.def temp2=r18
.def cnt=r20
.def cnt1=r21
.org $0000
    rjmp reset
.org $0300
reset:   ldi temp,low(ramend)          ;设置堆栈指针
    out spl,temp
    ldi temp,high(ramend)
    out sph,temp
    ldi temp,$ff                       ;设置 D 口输出,B 口作输入
    out ddrd,temp
    out portd,temp
    clr temp
    out ddrb,temp
    out porth,temp
      rcall syset                      ;调用系统设置
lp8:    clr cnt1                       ;循环程序
lp81:    clr cnt
    ldi temp1,$80                      ;设置第 1 行显示寄存器起址(第 2 行为 $a8)
    rcall contd
lp82:    cp cnt1,cnt
    brne lp83
    ldi temp1,$38                      ;字形 8 的代码为 $38
lp84:    rcall writd
    inc cnt
    cpi cnt,$10
    brne lp82
    ldi temp,$55                       ;设置延时常数
    rcall delay
    inc cnt1
    cpi cnt1,$10
    brne lp81
    rjmp lp8
lp83:    ldi temp1,$20
   rjmp lp84
CONTD:      LDI      TEMP,0B00110000   ;写控制字入 LCD 中
    OUT      PORTD,TEMP
    RCALL    DELT3
    CBI      PORTD,$05                 ;使 E=0 ,LCD 片选有效
```

```
      RCALL     DELT3
      SBI       PORTD,$05
BUSYY:   WDR
      SBIC      PINB,$07          ;读取DB7=PINB7是否为0,为0则非忙跳过一行
      RJMP      BUSYY             ;DB7=1为忙,跳回BUSYY再等待DB7=0,以写入
      LDI       TEMP,0b00100000   ;写入数据,写入控制字
      OUT       PORTD,TEMP
      RCALL DELT3                 ;延时,以免AVR速度太快而使LCD无法工作
      LDI       TEMP,$ff          ;设定B口为输出
      OUT       DDRB,TEMP
      OUT       PORTB,TEMP1       ;要写入LCD的数据TEMP1输出到PORTB
      WDR
      CBI       PORTD,$05
      RCALL DELT3
      LDI       TEMP,0B00111000
      OUT       PORTD,TEMP
      CLR       TEMP
      OUT       DDRB,TEMP
      OUT       PORTB,TEMP
      RET
WRITD:   LDI TEMP,0B00110000      ;写数据入LCD中
      OUT PORTD,TEMP
      RCALL DELT3                 ;延时,以免AVR速度太快而使LCD无法工作
      CBI PORTD,$05               ;使E=0,LCD片选有效
      RCALL DELT3
      SBI PORTD,$05
BUZY1:   WDR
      SBIC PINB,$07               ;读取DB7=PINB7是否为0,为0则非忙跳过一行
      RJMP BUZY1                  ;DB7=1为忙,跳回BUSY1再等待DB7=0以写入
      LDI TEMP,0B00101000         ;写控制字入LCD中
      OUT PORTD,TEMP
      OUT PORTB,TEMP1             ;要写入LCD的数据TEMP1输出到PORTB
      LDI TEMP,$ff                ;设定B口为输入
      OUT DDRB,TEMP
      CBI PORTD,$05               ;使E=0,LCD片选有效
      RCALL DELT3                 ;延时,以免AVR速度太快而使LCD无法工作
      LDI TEMP,0B00111000         ;写控制字入LCD中
        OUT PORTD,TEMP
      RCALL DELT3
      CLR TEMP
      OUT DDRB,TEMP               ;PORTB为输入
      OUT PORTB,TEMP              ;PORTB为三态输入
      RET
```

```
syset:    ldi temp1, $01                ;清屏设定
          rcall contd
          ldi temp, $50                 ;设置时间常数
          rcall delay
          ldi temp1, $38                ;2 行 5×7 显示设定
          rcall contd
          ldi temp1, $06                ;自动增量,显示不移位
          rcall contd
          ldi temp1, $0c                ;字形开关 ON,光标开关 OFF
          rcall contd
          ret
DELT3:    ldi temp2, $24
DT111:    wdr
dec temp2
          brne dt111
          ret
delay:                                  ;延时子程序略
```

7.4.5 LED 电脑时钟

源程序:SLAVR745.ASM。硬件连接见“3.3 AVR 单片机开发下载实验器”。

本程序若直接按 SHIFT+EXEC 即从 00:00:00 开始计时。下载后(或上电后),LED 显示 00:00:00 等待设置,请从键盘上输入时、分、秒,要求输入位由小数点作光标提示。输入正确时间后,按执行键(SHIFT+EXEC)执行,电脑钟开始计时。

(1) 请问如何修改程序,可当秒表用?

(2) 又如何修改程序到点发出报时声?

```
;本程序采用 T0,1/1024 分频,设定一次中断为 25ms,40 次中断为 1s
.include"8515def.inc"
.def    TEMP    =r16
.def    TEMP1   =r17
.def    temp2   =r18
.def    temp3   =r19
.def    CNT     =r20
.def    SCNN    =r21
.def    KSNI    =r22
.def    SCNDP   =r23
.def    KEYN    =r24
.def    cnt1    =r25
.def    hour    =r24
.def    minute  =r22
.def    second  =r21
.equ label= $0f00                       ;字形表首址
.org $0000
```

```
        rjmp reset
    .org $007
    intt0:    ldi temp,104          ;因 25ms 内差 40μs,故补上 40/(1/8)即 320 个 CK
    bu:    dec temp                 ;因中断需 4CK,这样:4+104×(1+2)+1+1+1+1=320
        brne bu
        nop
        inc cnt1                    ;cnt1 计数 40 次为 1s
        ldi temp,256-195            ;计数(256-195)次才产生 1 次中断
        out tcnt0,temp              ;CK/1024 分频,这样一次中断需 25ms
        rjmp rccog
    .org $030
    reset:
        ldi temp,low(ramend)        ;设置堆栈指针
        out spl,temp
        ldi temp,high(ramend)
        out sph,temp
        clr r2                      ;清工作寄存器
        clr r3
        clr r4
        clr r5
        clr r6
        clr r7
        clr zh
        clr xh
        clr yh
        clr keyn
        clr second
        clr minute
        clr hour
        clr cnt
        clr yh
        ldi temp,$80
        mov r8,temp                 ;R8=$80
        ldi yl,$60                  ;设置显示内存地址指针 Y 为 $0060
        rcall disram                ;调用 DISRAM
        ld temp,y
        ldi temp1,$80
        add temp,temp1
        st y,temp
    scanad: ldi temp,$07
        ldi yl,$60
    scann:    rcall scan1           ;调用键扫显示子程序 SCAN1
        brts scann
```

```
 scank:    rcall scan1
     brtc scank
     rcall scan1
 scans: nop
          cpi keyn, $10          ;KEYN=$10 转 EXEC
      brcc exec
      rcall wraddram            ;调用 WRADDRAM
     dec temp                   ;TEMP 减 1
     cpi temp, $01
     brne scann                 ;TEMP=1 则转 SCANN
     rjmp scanad
exec:    mov temp1,r7           ;把 r7,r6 的 2 个十进制换成 1 个十六进制入 hour 中
     mov temp,r6
     rcall dechex
     mov hour,temp
     mov temp1,r5               ;把 r5,r4 的 2 个十进制换成 1 个十六进制入 minute 中
     mov temp,r4
     rcall dechex
     mov minute,temp
     mov temp1,r3               ;把 r3,r2 的 2 个十进制换成 1 个十六进制入 second 中
     mov temp,r2
     rcall dechex
     mov second,temp
     ldi temp, $05              ;T0 设置为 CK/1024 分频
     out tccr0,temp
     ldi temp,256-195
     out tcnt0,temp             ;装载 T0 时间常数
     ldi temp, $ff              ;设置 b 口,d 口为输出
     out ddrb,temp
     out ddrd,temp
     sei                        ;开中断总开关
     ldi temp, $02
     out timsk,temp             ;允许 t0 中断
display:rcall disram            ;调用 disram
     clr yh                     ;设置显示内存地址指针 Y 为 $0060
     ldi yl, $60
     ldi scndp, $df             ;设置扫描显示码 SCNDP 起址 0B11011111
agdis:    ld r1,y+
     cpi yl, $62
     brne npoint
     add r1,r8
npoint:    cpi yl, $64
     brne next
```

```
        add r1,r8
next:   out portb,r1            ;把 R1 送 B 口显示
        out portd,scndp         ;扫亮某个数码管
        sec                     ;C=1
        ror scndp               ;右移 SCNDP
        rcall delay             ;延时
        cpi yl,$66
        brne agdis              ;未扫亮最后一位继续
        rjmp display
recog:  cpi cnt1,40             ;40 次中断为 40×25ms=1s
        brne inthome            ;40 次中断未到转 inthome
        clr cnt1                ;40 次中断到则清 cnt1
        inc second              ;秒寄存器加 1
        cpi second,60
        brne change             ;秒寄存器未满转 change
        clr second              ;否则清秒寄存器
        inc minute              ;分寄存器加 1
        cpi minute,60
        brne change             ;分寄存器未满转 change
        clr minute              ;否则清分寄存器
        inc hour                ;时寄存器加 1
        cpi hour,24
        brne change             ;时寄存器未满转 change
        clr hour                ;否则清时寄存器
        reti                    ;中断返回
change: mov temp,second         ;把 second 中的十六进制转换成 2 个十进制数存入 r3,r2 中
        rcall hexdec
        mov r3,temp1
        mov r2,temp
        mov temp,minute         ;把 minute 中的十六进制转换成 2 个十进制数存入 r5,r4 中
        rcall hexdec
        mov r5,temp1
        mov r4,temp
        mov temp,hour           ;把 hour 中的十六进制转换成 2 个十进制数存入 r7,r6 中
        rcall hexdec
        mov r7,temp1
        mov r6,temp
inthome:reti                    ;中断返回
hexdec: clr temp1               ;把 temp 中的十六进制转换成 2 个十进制人 temp1,temp 中的子程序
hexdec1:subi temp,10
        brcs negs
        inc temp1
        rjmp hexdec1
```

```
negs:   subi temp, $f6
        ret                          ;子程序返回
dechex: push temp                    ;把 temp1,temp 的 2 个十进制数转换成 1 个十六进制入 temp 中
        ldi temp2, $0a
        clr temp
dechex1:cpi temp1, $00
        breq dh
        dec temp1
        add temp,temp2
        rjmp dechex1
dh:  mov temp1, temp
        pop temp
        add temp, temp1
        ret                          ;子程序返回
disram: push yl                      ;压栈保护
        push zl
        push xl
        ldi zh,high(label * 2)       ;Z 指针指向字形表首址 label * 2
        ldi zl,low(label * 2)
        clr xh
        ldi xl, $60
        ldi yl, $07
ramag:    ld temp2,y                 ;y 为间址的内容送 temp2
        dec yl
        mov zl,temp2
        lpm
        st x+,r0                     ;把 r0 的内容送到 $0060～$0065 中
        cpi xl, $66
        brne ramag
        pop xl
        pop zl
        pop yl                       ;退栈
        ret                          ;子程序返回
wraddram:push temp                   ;读键存入显示内存及寄存器中
        clr zh
        mov zl,temp
        st z,keyn
        ldi zh,high(label * 2)
        mov zl,keyn
        lpm
        st y+,r0
        cpi yl, $66
        brne pointc
```

```
    ldi yl,$60
pointc:    ld temp2,y
    ldi temp3,$80
    add temp2,temp3
    st y,temp2
    pop temp
    ret                          ;子程序返回
delay:  push temp                ;延时子程序
lp1:    ldi temp2,$10
lp2:    dec temp
        brne lp2
        dec temp2
        brne lp2
        pop temp
        ret                      ;子程序返回
SCAN1:  push xh                  ;键盘扫描显示子程序(注释从略,见7.3.5)。
        PUSH XL
        PUSH TEMP1
        PUSH TEMP
        LDI XL.,$60
            SET
            LDI  SCNN,$00
            LDI  SCNDP,0B11011111
            LDI  CNT,$06
            LDI  KSNI,0B11110111
COL1:   LDI  TEMP,$FF
            OUT  DDRb,TEMP
            OUT  DDRC,TEMP
        OUT  PORTC,TEMP
            OUT  DDRd,TEMP
        OUT  PORTd,SCNDP
            LD   R1,X+
            OUT  PORTb,R1
            RCALL  DELAY
            MOV  TEMP,CNT
            SUBI  TEMP,$03
            BRCS  NOSK
            LDI  TEMP1,$04
            LDI  TEMP,0B00001111
            OUT  DDRc,TEMP
            OUT  PORTc,KSNI
            RCALL  DELYT
            IN   TEMP,PINc
```

```
            ANDI   TEMP,0B11110000
            SWAP   TEMP
KROW: SEC
            ROR   TEMP
            BRCS   NOKEY
            CLT
            MOV    KEYN,SCNN
            SBIS PINd, $07
            ADIW KEYN, $10
NOKEY: INC   SCNN
       DEC   TEMP1
       BRNE   KROW
       SEC
       ROR   KSNI
NOSK: SEC
      ROR   SCNDP
      DEC   CNT
      BRNE   COL1
      LDI    TEMP, $FF
      OUT    DDRC,TEMP
      OUT    PORTC,TEMP
      POP TEMP
      POP TEMP1
      POP XL
      pop xh
      RET
delyt:    ldi temp3, $20
dt31:     dec temp3
          brne dt31
          ret
 .cseg
 .org $0f00                          ;字形表
  .dw 0x063f,0x4f5b,0x6d66,0x077d
  .dw 0x6f7f,0x7c77,0x5e39,0x7179
```

7.4.6 测频率

测频率，信号从 AT90S8515 的 ICP 引脚输入，最大值为 999 999Hz/ms。

源程序：SLAVR746.ASM，在 SL－AVR 开发实验器验证通过。

```
.include"8515def.inc"
    rjmp    reset
.def    temp    = r16                  ;暂存器
.def    cntld   = r17
```

```
.def    cnt2d   = r18                ;cnt1,dcnt2d 和 cnt3d 存放结果的十进制
.def    cnt3d   = r19
.def    count   = r20
.def    cnt     = r21
.def    res1    = r22
.def    res2    = r23                ;res1,res2 和 res3 存放结果的十六进制
.def    res3    = r24
.def    dt      = r25
.def    ovfl    = r26
.def    aa      = r27
.org    0x003                        ;icp 触发中断向量
    rjmp   captr
.org    0x007                        ;timer0 触发中断向量
    rjmp   interru
captr:                               ;icp 触发中断子程序
    brts  b
    inc   res1
    ldi   temp,  0b00000101
    out   tccr0, temp                ;开 timer0
    ldi   temp,  0b00001010
    out   timsk, temp                ;置 timer0 中断和捕捉中断
    ldi   temp,  0b11000000
    out   tccr1b,temp
    set
    reti
b:
    set
    inc   res1                       ;开始计数
    brne  c
    inc   res2
    brne  c
    inc   res3
    cpse  res3,  ovfl                ;溢出处理
    rjmp  c
    rjmp  over1
c:
    ldi   temp,  0b00001010
    out   timsk, temp
    ldi   temp,  0b11000000
    out   tccr1b,temp
    reti
interru:                             ;timer0 溢出中断子程序
    dec   cnt
```

```
    breq  over
    ldi   temp,   0b00001010
    out   timsk,  temp
    reti
over:
    rcall htd3
over1:
    rcall sys
    reti
reset:
    ldi   temp,  low(ramend)
    out   spl,   temp
    ldi   temp,  high(ramend)          ;设置堆栈
    out   spl+1, temp
    ldi   temp,  $ff                   ;初始化数码管状态
    out   ddrb,  temp                  ;B口:数码管数据输出
    out   ddrd,  temp                  ;D口:pd0～pd5为数码管片选
    ldi   temp,  $00
    out   portb, temp                  ;共阴极,数码管全灭
    out   portd, temp
    ldi   cnt1d, 00
    ldi   cnt2d, 00
    ldi   cnt3d, 00
    sei
    rcall sys
loop:                                  ;在数码管显出十进制数
    mov   aa,    cnt1d                 ;个位、十位、百位、千位、万位、十万位
    ......
sys:                                   ;初始化,16转10子程序,计算结果子程序,字形表同"7.4.3测
                                       脉冲宽度"的程序,省略
    ......
```

7.4.7 测转速

测转速,信号从AT90S8515的ICP引脚输入,最大值为999 999r/min。

源程序:SLAVR747.ASM,在SL-AVR开发实验器验证通过。

```
.include"8515def.inc"
      rjmp reset
.def    temp   = r16                 ;暂存器

.def    aa     = r17
.def    cnt    = r18
.def    mc16l  = r19                 ;mc16l和mc16h存放脉冲个数
```

```
.def    mc16h  = r20
.def    mp8u   = r21                ;mp8u=30,因为是每 2s 采样
.def    res1   = r21                ;res1,res2 和 res3 存放结果的十六进制
.def    res2   = r22
.def    res3   = r23
.def    count  = r24
.def    cnt1d  = r25                ;cnt1,dcnt2d 和 cnt3d 存放结果的十进制
.def    cnt2d  = r26
.def    cnt3d  = r27
.def    dt     = r28
.def    dtl      r29
.org 0x003                          ;icp 触发中断向量
icptl:
    rjmp  captr
.org 0x008
    rjmp  interru
.cseg
.org 0x010
captr:                              ;icp 触发中断子程序
    brts  down
    ldi   temp,  0b00000101
    out    tccr0, temp              ;开 timer0
    ldi   temp,  0b00001010
    out    timsk, temp              ;置 timer0 中断和捕捉中断
    ldi   temp,  0b10000000
    out    tccr1b,temp
    set
    reti
down:                               ;下降沿开始计数
    set
    inc   mc16l
    brne  b
    inc   mc16h
    cpse  mc16h, dtl                ;溢出处理
    rjmp  b
    ldi   mc16h, 00
    ldi   mc16l, 00
    rjmp  over
b:
    ldi   temp,  0b00001010         ;置 timer0 中断和捕捉中断
    out    timsk, temp
    ldi   temp,  0b10000000
    out    tccr1b,temp
```

```
    reti
interru:                              ;timer0 溢出中断子程序
    dec   cnt
    breq  over
    ldi   temp,  0b00001010
    out   timsk, temp
    reti
over:
    rcall conver                      ;conver:计算结果子程序
    rcall htd3                        ;htd3
    rcall sys
    reti
reset:
    ldi   temp,  low(ramend)
    out   spl,   temp
    ldi   temp,  high(ramend)         ;设置堆栈
    out   spl+1, temp
    ldi   temp,  $ff                  ;初始化数码管状态
    out   ddrb,  temp                 ;B 口:数码管数据输出
    out   ddrd,  temp                 ;D 口:pd0~pd5 为数码管片选
    ldi   temp,  $00
    out   portb, temp                 ;共阴极,数码管全灭
    out   portd, temp
    ldi   cnt1d, 00
    ldi   cnt2d, 00
    ldi   cnt3d, 00
    sei
    rcall sys
loop:                                 ;在数码管显出十进制数
    mov   aa,    cnt1d                ;个位、十位、百位、千位、万位、十万位
    ......
sys:                                  ;初始化,16 转 10 子程序,计算结果子程序,字形表等同"7.4.3 测
                                      脉冲宽度"的程序,省略
    ......
```

7.4.8 AT90S8535 的 A/D 转换

AT90S8535 10 位 A/D 转换电原理图如图 7.16 所示。

源程序:SLAVR748.ASM,在 SL-AVR 开发实验器上验证通过。

用 AT90S8535 作 0～7 通道 A/D 转换,用 LED 显示,左 1 位(D5)显示通道号,右 3 位(D2～D0)显示转换值(十六进制数 0～3FFH),程序下载即执行。自动从 0～7 通道 A/D 转换扫描显示,当按下 0～7 任一位数字键,该通道显示时间延长一段时间,然后又自动循环显示。

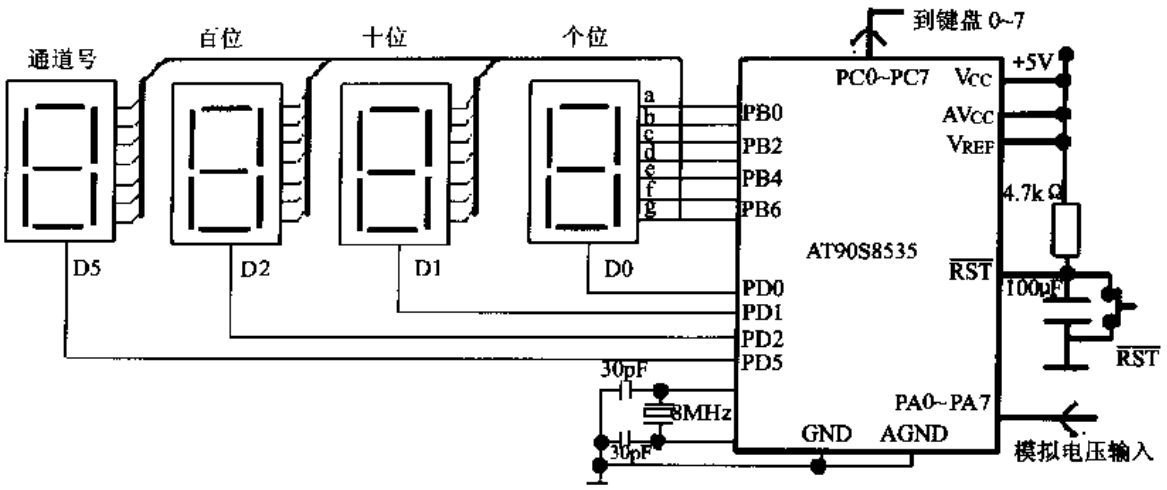


图 7.16　AT90S8535 10 位 A/D 转换电原理图

硬件接口：AT90S8535 的 PB0～PB7 接 LED 段显示(用短路块连接)；PD0～PD5 接 LED 位显示，用接插线连接；PC0～PC7 接键盘线；PA0～PA7 接模拟电压，滑线电位器 A/D VX 端；AGND 接地；最好 AV_{CC} 与 V_{REF} 间接 1 kΩ 电阻，V_{REF} 到地接 100μF 电解电容；AV_{CC} 与 V_{CC} 间接一只 100Ω 电阻，AV_{CC} 接 104 瓷片电容到地；$\overline{RST}$ 接上复位按钮，插上 CZ2 到 AT4 下载线，即连通晶振引脚线。

```
.include   "8535def.inc"
 .org    $0000
      rjmp   reset
 .def    TEMP    =r16
 .def    TEMP1   =r17
 .def    temp2     r18
 .def    temp3   =r19
 .def    CNT     =r20
 .def    scndp   =r21
 .def    KSNI    =r22
 .def    SCNN    =r23
 .def    KEYN    =r24
 .def    temp4   =r25
 .equ    label   =$0f00
 .org 0030
reset:   ldi temp,high(ramend)          ;设置堆栈指针
         out sph,temp
         ldi temp,low(ramend)
         out spl,temp
         clr xh                         ;设置 x 指针为 $0061
         ldi xl,$61
         clr temp                       ;清 $0061、$0062 单元
         st x+,temp
```

```
        st x,temp
init:   clr temp2                    ;由 0 通道开始
next:   ldi temp3, $ 01
next1:  clr temp4
again:  rcall cance                  ;调用 a/d 转换子程序 cance
lp:     rcall scan1                  ;调用键扫显示子程序 scan1
scann:  rcall scan1
        brtc recog                   ;用按键转 recog
        inc temp4                    ;键扫显示次数 temp4+1
        cpi temp4, $ ff
        brne again                   ;temp4 不等于 $ ff 转 again
        dec temp3
        brne again                   ;temp3 不等于 0 转 again
        inc temp2                    ;通道代码 temp2 + 1
        cpi temp2, $ 08
        brne next                    ;8 个通道未结束转下一通道 next
        rjmp init                    ;8 个通道已扫描完再重扫
recog:  cpi keyn, $ 08
        brcc next                    ;无效键转 next
        ldi temp3, $ 04              ;设置有效通道键按下后的循环次数
        mov temp2,keyn               ;通道数送 temp2
        rjmp next1
cance:  mov temp,temp2               ;a/d 转换子程序
        out admux,temp               ;设置通道
        ldi temp, $ 86               ;设置 a/d 转换使能且采用 1/64 分频作转换工作频率
        out adcsr,temp
        sbi adcsr,adsc               ;启动转换
loop:   sbic adcsr,adsc              ;转换结束跳行否则等待
        rjmp loop
        in r2,adcl                   ;把转换结果送 r2、r3
        in r3,adch
        mov temp,temp2
        rcall wrdisram               ;调用把转换的结果转换成显示代码 wrdisram
        ret                          ;转换结束返回
wrdisram:clr xh                      ;使 x 指针为 $ 0060
        ldi xl, $ 60
        rcall fetch                  ;调用 fetch
        st x+,temp                   ;把 temp 存入 $ 0060 单元
        inc xl
        inc xl
        mov temp,r3
        andi temp, $ 0f              ;取 r3 的低 4 位
        rcall fetch                  ;取字形代码
```

```
        st x+,temp
    mov temp,r2
    swap temp
    andi temp,$0f                    ;取 r2 的高 4 位
    rcall fetch                      ;取字形代码
    st x+,temp
    mov temp,r2
    andi temp,$0f                    ;取 r2 的低 4 位
    rcall fetch                      ;取字形代码
    st x+,temp
    ret                              ;返回
fetch:  ldi zh,high(label*2)         ;设置字形表指针 z
    mov zl,temp
    lpm                              ;取字形
    mov temp,r0                      ;字形码送 temp
    ret                              ;返回
SCAN1:push xh                        ;键扫显示子程序
      PUSH XL                        ;将 xl 压入堆栈
      PUSH TEMP3
      PUSH TEMP2
      PUSH TEMP1
      PUSH TEMP
      IDI XL,$60
          SET                        ;T 标志为 1 表示未按键
          LDI  SCNN,$00              ;按键起始扫描码 SCNN 为 00
          LDI  SCNDP,0B11011111      ;令 6 位 7 段 LED 扫描显示码初始为 11011111
          LDI  CNT,$06               ;7 段 LED 共 6 位故 CNT=6 为位数计数
          LDI  KSNI,0BI1110111       ;4×4 键盘扫描码 KSNI 初始为 11110111
   COL1:  LDI  TEMP,$FF              ;PORTB 设定为输出
          OUT  DDRb,TEMP
          OUT  DDRC,TEMP             ;PORTC 设定为输出
          OUT  PORTC,TEMP
          OUT  DDRd,TEMP             ;PORTD 设定为输出
          OUT  PORTd,SCNDP           ;6 位 7 段 LED 扫描显示码输出到 PORTD
          CLR  XH
          LD   R1,X+                 ;要显示于 7 段 LED 的间接寄存器 X 中的内容送入 R1 并
                                     ;令 X 1 1
          OUT  PORTb,R1              ;显示内容输出到 PORTB,以驱动 LED 显示
          RCALL  DELAY               ;调用延时,以显示此位数一段时间
          MOV  TEMP,CNT              ;LED 位数为 6,而按键码行数为 4,故需作 CNT 值检测
          SUBI  TEMP,$03             ;CNT=TEMP 与 3 相减比较
          BRCS  NOSK                 ;位数扫描 CNT 超过 3 则 C 为 1,跳到 NOSK 不作按处理
          LDI  TEMP1,$04             ;一共要检查 4 个按键
```

```
        LDI   TEMP,0B00001111      ;设定 PC0～PC3 为输出,PC4～PC7 为输入
        OUT   DDRc,TEMP
        OUT   PORTc,KSNI           ;KSNI 输出到 PORTC,并令 PC7～PC4 为上拉电阻输入态
        RCALL  DELYT               ;调用延时,以稳定读取键盘 I/O 输入端
        IN    TEMP,PINc            ;读取 C 口检测 PC7～PC4,看是否有按键低电位输入
        ANDI  TEMP,0B11110000      ;取 TEMP 的高 4 位
        SWAP  TEMP                 ;键码顺序为 PC4～PC7,故将 TEMP 的高低 4 位互换成
                                   ;D0～D3
        KROW: SEC                  ;令 C 标志为 1,以便将键盘码 D0～D3 移到 C 标志位检测
        ROR   TEMP                 ;TEMP 的内容右移 1 位,将第 1 个键码 D0=PC4 移到 C
                                   ;标志位检测
        BRCS  NOKEY                ;若有键按下则测到 PC4～D0=0,若 C=1 无按键则转到
                                   ;NOKEY
        CLT                        ;若 PC4～D0=CF=0,表示有按键,令 T=0 表示有按键
        MOV   KEYN,SCNN            ;把按键扫描码 SCNN 送键码 KEYN 中保存
        SBIS PIND,$07
        ADIW KEYN,$10              ;判定 SHIFT 键是否按下,按下则键值+10
        NOKEY: INC  SCNN           ;按键扫描码 SCNN+1
         DEC  TEMP1                ;扫描读取键数 TEMP1-1
         BRNE  KROW                ;每行有 4 个按键,如 TEMP1 不为 0,则跳到 KROW 再检
                                   ;测 PC5～PC7
         SEC                       ;此行 4 个键码检测完后令 C 为 1,以方便键盘扫描码
                                   ;KSNI 内容的移位
         ROR   KSNI                ;键盘扫描码 KSNI=CF=1>11110111,移位以进行下一
                                   ;行按键扫描
         NOSK: SEC                 ;令进位标志 CF=1
         ROR   SCNDP               ;将扫描显示码 SCNDP 左移,作下一位扫描
         DEC   CNT                 ;共需作 6 位数扫描显示故 CNT-1
         BRNE  COL1                ;CNT-1 不为 0,则跳回 COL1,再作扫描显示及读取键盘
                                   ;输入
         LDI   TEMP,$FF            ;若已完成全部扫描显示和读取,按键,则令 TEMP=off
         OUT   DDRC,TEMP           ; TEMP 输出到 DDRC,设定 PORTC 为输出,驱动 LED
         OUT   PORTC,TEMP
         POP TEMP                  ;出栈
         POP TEMP1
         POP TEMP2
         POP TEMP3
         POP XL
         pop xh
         RET                       ;子程序返回
delay:   push temp1                ;延时子程序
         push temp3
         ldi temp1,$10
```

```
        dt11:ldi temp3, $20
        dt21:nop
        dec temp3
        brne dt21
        dec temp1
        brne dt11
        pop temp3
        pop temp1
         ret
delyt:  ldi temp3, $20                    ;延时子程序
        dt31:dec temp3
         brne dt31
         ret
.cseg
.org  $0f00                               ;字形表
.dw 0x063f,0x4f5b,0x6d66,0x077d
.dw 0x6f7f,0x7c77,0x5e39,0x7179
```

第八章　BASCOM－AVR的应用

8.1　基于高级语言BASCOM－AVR的单片机开发平台

传统开发单片机系统主要是用汇编语言编写系统程序。由于汇编语言程序的可读性、可移植性和结构性都较差，因此采用汇编语言编写单片机应用系统程序的周期长，调试和排错也比较困难，产品开发周期长。为了提高编写系统和应用程序的效率，改善程序的可读性和可移植性，缩短产品的开发周期，采用高级语言的开发平台来开发单片机系统已成为发展趋势。概括起来说，基于高级语言开发平台开发单片机系统，具有语言简洁、使用方便灵活、可移植性好、表达能力强、可进行结构化程序设计、可进行软件仿真等优点。实践表明，采用高级语言开发平台进行单片机系统开发的效率比使用汇编语言高几倍甚至几十倍。ATMEL公司的AVR AT90系列单片机是基于新的精简指令RISC结构的，其开发目的就是在于能采用高级语言编程，从而能高效地开发出目标产品。目前国际上已有许多公司推出了C、BASIC等基于高级程序设计语言的AVR开发软件和平台。

本章将采用以高级程序设计语言BASIC为手段的AVR单片机开发平台——BASCOM－AVR。它具有程序设计简洁、方便，专用的面向各种通用接口且功能强大的语句，实物图形化的仿真平台等特点；配合AVR单片机程序存储器可多次编程、在线下载的优点，使学习和使用AVR单片机变得十分容易。使用BASCOM－AVR开发AVR单片机系统，设计人员可以在半个小时之内完成一个功能模块的设计编程和调试；而采用汇编语言，则需要几天甚至几个星期。

本章不是讲述如何去设计和开发一个复杂的实际产品，而是指导和帮助学习者以动手实践为主、掌握使用BASCOM－AVR快速开发AVR单片机系统的方法。尽管本章的实验都是实现一些基本和简单的功能，但它们往往就是一个实际产品的基本功能模块。当把它们有机地结合起来，就可形成一个实际的电子产品。

在学习本章和动手DIY前，先了解和建立一个简单的实践环境。该环境由以下几部分组成：PC机1台，运行Windows95/98；AVR开发仿真软件平台BASCOM－AVRDEMO(http://www.mcselec.com,Free)；AVR程序下载软件AVRProg(http://www.atmel.com,Free)；或双龙网站下载(http://www.sl.com.cn)。BASCOM－AVR开发实验程序由华东师范大学电子科学技术系(ATMEL AVR实验室)实验通过(ma－chao@online.sh.cn)。

AVR单片机开发实验硬件基本系统是SL－AVR开发下载实验器，以及完成实验所需的硬件外围组件和电子元器件。

一、实验硬件系统

AVR单片机开发实验基本系统包括：AVR单片机开发实验最小系统SL－AVR开发下载实验器，RS－232串口通讯电缆，5V直流电源和连接引线等。

利用AVR单片机开发实验基本系统，可以完成一些基本的实验，学习和掌握AVR单片

机的基本特性和使用方法、AVR 开发仿真软件、程序下载软件、AVR 应用程序设计以及设计和开发基于 AVR 单片机的简单电子系统的手段、方法和过程。配合一些硬件外围组件和相应的电子元器件，就可以进一步完成一些复杂实验。AVR 开发实验基本系统不仅可用于高校相关专业开设实验，也可作为高中、职业学校、青少年科技站开设课程和科技制作应用；对于电子工程师和电子产品设计开发人员，本章介绍的开发环境和 AVR 开发实验基本系统也是一套高效、简便、实用的产品设计研制开发工具。

SL－AVR 开发下载实验器详见“3.4 AVR 嵌入式单片机开发下载实验器 SL－AVR”。

二、软件开发平台

BASCOM－AVR 是 MCS 公司推出的面向 AVR 单片机系列、采用高级程序设计语言 Windows BASIC 的软件开发平台。它的运行环境是 Windows95/98/NT。其主要特点有：

采用可带语句标示符的结构型 BASIC 高级程序设计语言编程；结构化的 IF—THEN—ELSE—ENDIF，DO－LOOP，WHILE—WEND，SELECT—CASE 程序设计；变量名和语句标示符长达 32 个字符；有位(bit)、字节(byte)、整型(integer)、字(word)、长型(long)和字符串(string)等多种类型的变量；编译产生的运行代码可在所有带内部存储器的 AVR 单片机中运行；程序语句和 Microsoft VB/QB 高度兼容，为标准的 LCD 显示器；I^2C 芯片和单总线协议芯片等扩充了专用语句；内置模拟终端和程序下载功能；内置软件仿真平台用于测试；优良的程序编辑功能、完善的连机帮助功能和大量的例程；DEMO 版本可生成 2KB 程序代码，完全适用于 AT90S2313。

8.2 BASCOM－AVR 软件平台的安装与使用

目的：掌握 AVR 开发平台 BASCOM－AVR 和程序下载软件 AVRProg 的安装与初步应用。

内容：了解和使用 BASCOM－AVR 开发平台是第 1 步，本实验将介绍这 2 个软件工具的安装、基本参数的设置和初步的应用。

器材与器件：PC 机 1 台，运行 Windows95/98；BASCOM－AVR DEMO 版安装软件包；AVR Prog 安装软件包。

BASCOM－AVR 是 MCS Electronics 公司推出的、基于 AVR 系统的软件开发仿真平台。尽管 DEMO 版本仅可生成 2KB 程序代码，但足以用于实验和学习，而且完全适用于开发 AT90S2313。因为 AT90S2313 的最大程序代码容量即为 2KB。

BASCOM－AVR DEMO 版安装软件包由多张 3 英寸磁盘组成，用户可到 http://www.mcselec.com 免费下载，或从《双龙 AVR 电子书光盘》得到。

AVR Prog 是 ATMEL 公司提供的、用于 AVR 系列单片机程序下载的免费软件。我们用它将 BASCOM－AVR 生成的运行代码(Download)下载到 AVR 芯片中。用户可从 ATMEL公司的网站 http://www.atmel.com 或广州天河双龙电子有限公司的网站 http://www.sl.com.cn 或《双龙 AVR 电子书光盘》得到。

一、BASCOM－AVR 和 AVR Prog131.exe 的安装

1. 安装 BASCOM－AVR

用 Winzip 将 BASCOM－AVR DEMO 版安装软件包 2 张磁盘上的 ZIP 文件分别解压到

硬盘的临时目录 TEMP 下。

双击运行临时目录下的软件安装程序 SETUP. EXE，出现安装画面后单击 Next 继续安装过程。阅读软件版权说明后，单击 Yes 继续安装过程，输入名字和公司名称后单击 Next。以后均单击 Next，采用缺省设置，直到安装结束。

2. 安装 AVR Prog

单击运行 Aprogwing. exe 自解压程序，将下载软件 AVR Prog131. exe 解压到硬盘中，可放在与 BASCOM - AVR 同一个目录下。

二、运行 BASCOM - AVR

编写 BASIC 源程序，运行 BASCOM - AVR。BASCOM - AVR 主窗口如图 8.1 所示。

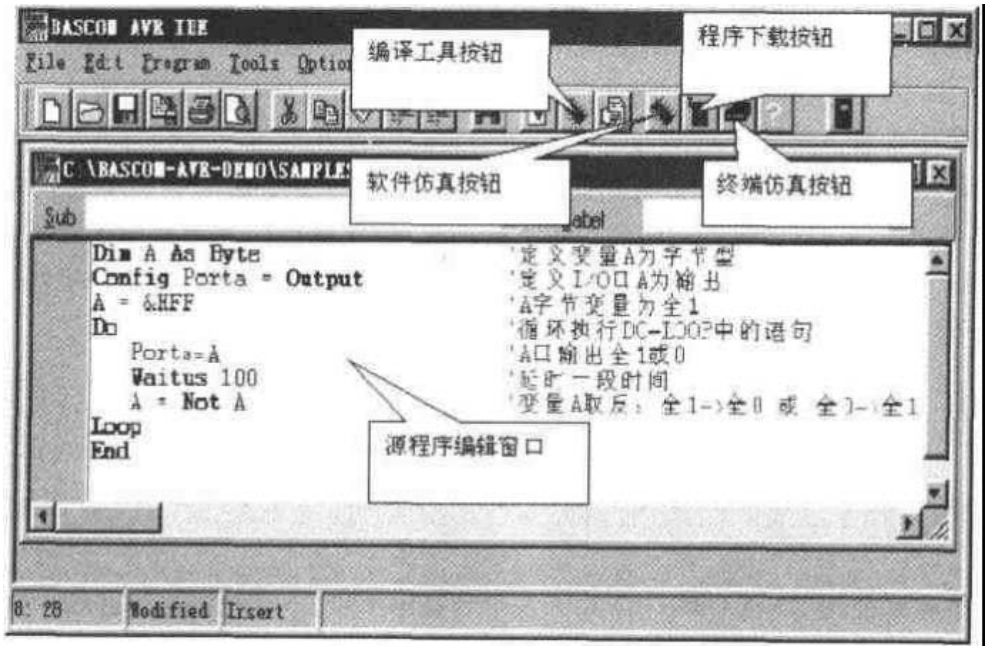


图 8.1 BASCOM - AVR 主窗口

运行 BASCOM - AVR，编写一简单的控制 LED 发光二极管的 BASIC 源程序 exp1. bas。

三、BASCOM - AVR 系统参数设置

选择 Option → Compiler → Chip，进行参数设置(图 8.2)；

选择实验所用芯片 AT90S8515；

继续进入 Output 设置选定输出选项(见图 8.3)；

单击 Programmer，设定使用的执行代码下载程序；

采用外部的程序下载器(external programmer)，设定下载程序所在目录和程序名(本例为 C:\BASCOM - AVR - DEMO\AVR Prog131. exe)；

选择 HEX 格式的下载代码文件类型，单击 Ok，完成参数设置(图 8.4)。

四、编译源程序

编译源程序生成各类代码文件，单击 BASCOM 主窗口工具条中的编译按钮，将 exp1. bas 编译生成可供仿真、下载的 dbg、obj 、hex 等文件。

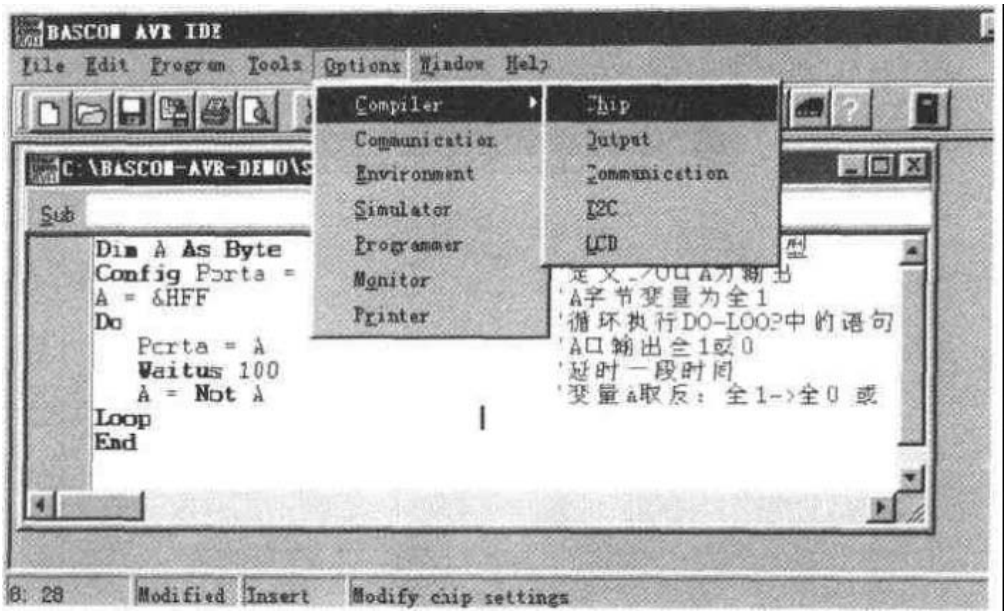


图 8.2　参数设置

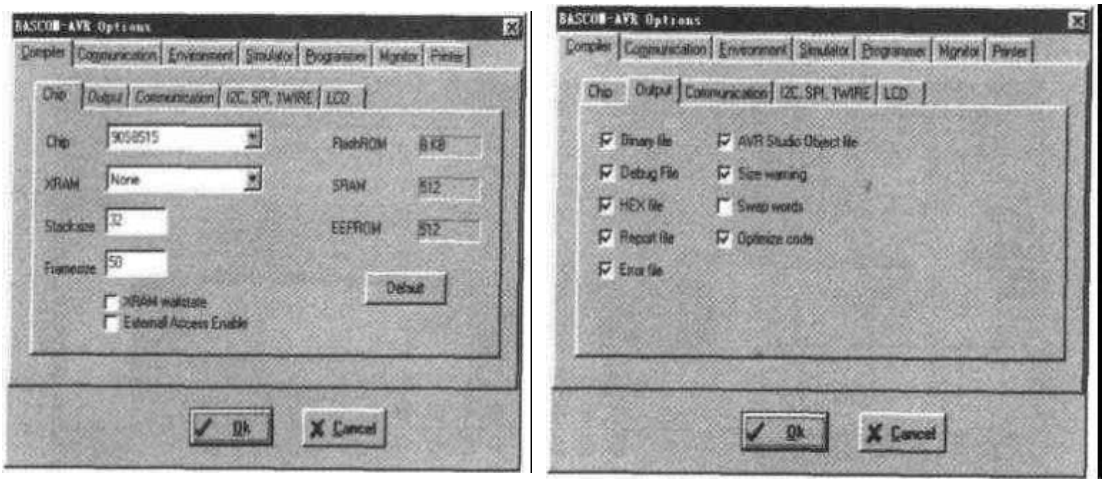


图 8.3　选定输出选项

五、软件仿真

单击 BASCOM 主窗口工具条中的仿真按钮，进入软件仿真窗口。

单击硬件模拟按钮，打开硬件模拟窗口运行程序进行模拟仿真，可看到硬件模拟窗口中 Porta 口的 LED 闪烁(图 8.5)。

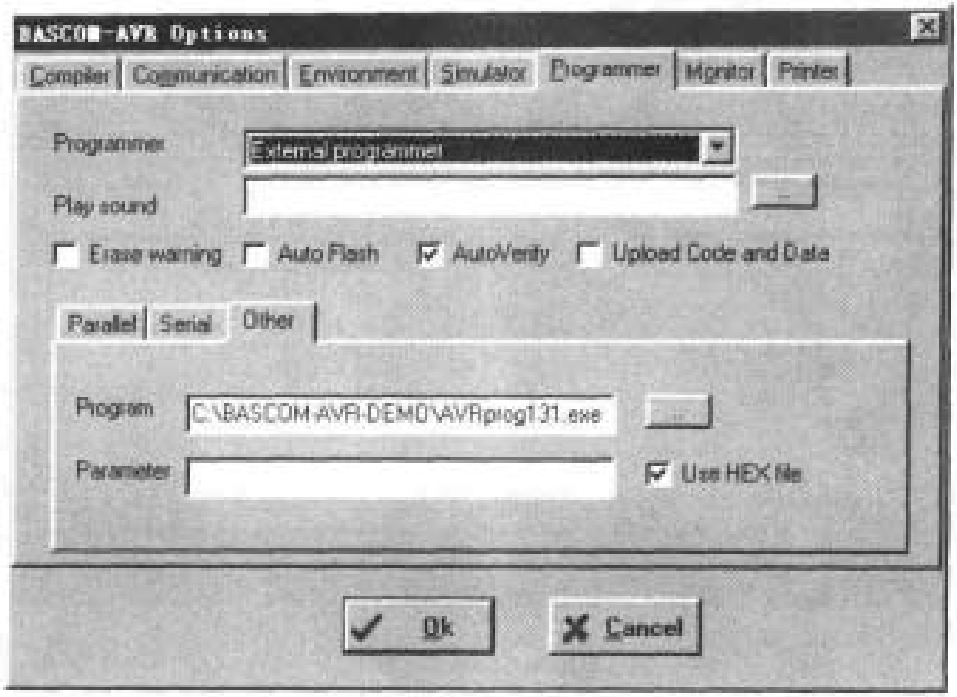


图 8.4 完成参数设置-AVR 主窗口

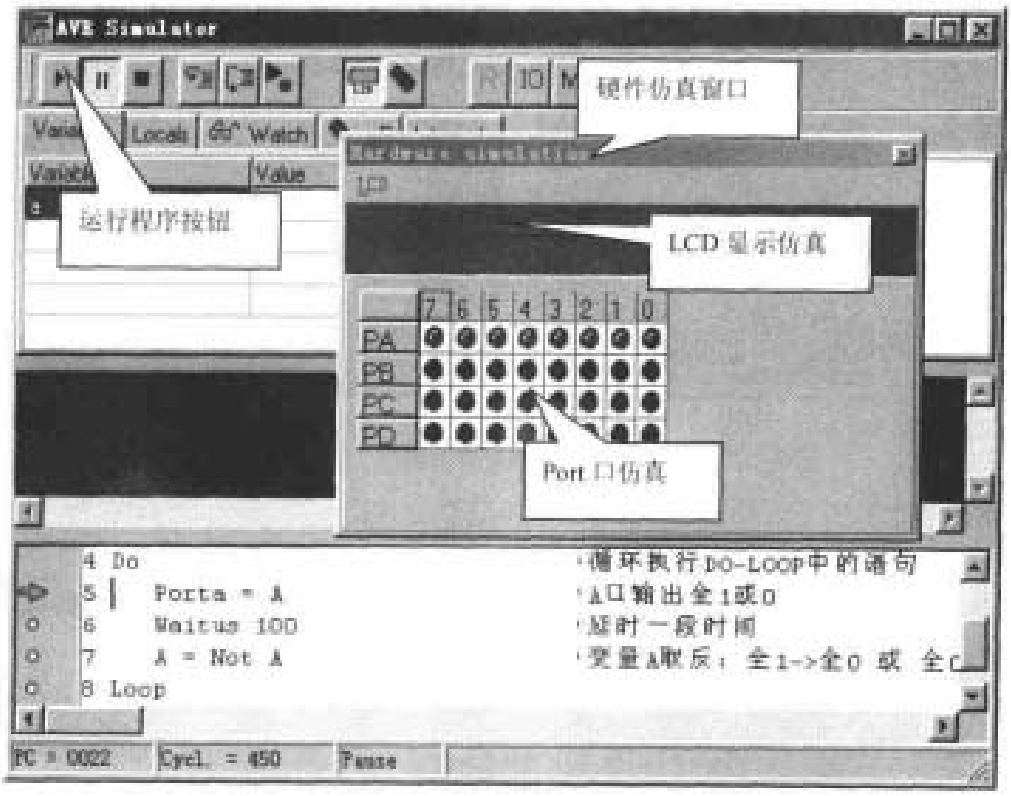


图 8.5 硬件仿真窗口-AVR 主窗口

8.3　AVR I/O 口的应用

AT90S8515 有 Porta、Portb、Portc、Portd 4 个口，共 32 根引脚。每个引脚都可以单独定义为输入或输出使用。

8.3.1　LED 发光二极管的控制

目的：利用 AVR 的 I/O 口控制 LED 发光二极管。

原理：AVR 的 I/O 口输出为低电平“0”，点亮 LED 发光二极管；输出为高电平“1”，LED 发光二极管熄灭。原理如图 8.6 所示。

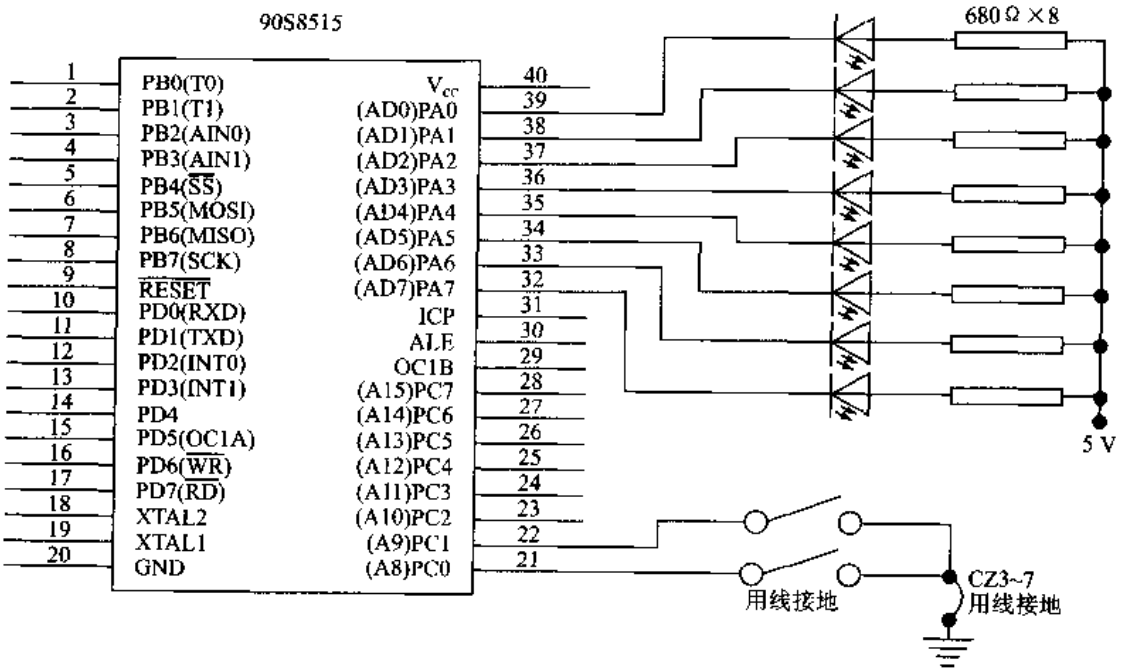


图 8.6　LED 发光二极管的控制原理图

功能：8 个 LED 发光二极管 1s 间歇闪烁。

程序：expl.bas

```
$sim                          '此语句用于软件仿真，程序正式下载时清除
Dim A As Byte                 '定义变量 A 为字节型
Config Porta = Output         '定义 Porta 口为输出
A = &HFF                      '字节变量 A 为 11111111
Do                            '循环执行 DO - LOOP 中的语句
    Porta = A                 'Porta 口输出全“1”或“0”
    Wait 1                    '延时 1s
    A = Not A                 '变量 A 取反：全 1→全 0 或全 0→全 1
Loop
End
```

操作过程：编写输入 BASIC 源程序；编译生成各类文件；利用 BASCOM－AVR 的软件仿

真平台仿真调试;使用 AVRprog 将程序代码下载到 AT90S8515。

8.3.2 简易手控广告灯

目的:利用 AVR 的 I/O 口实现简易手控广告灯。

原理:AVR 的 Porta 口输出控制 8 路发光 LED 作为广告灯,Portc0 和 Portc1 为输入,控制发光 LED 的闪烁方式。

原理图:如图 8.6 所示。

程序:exp2.bas

```
Dim A As Byte , B As Byte                 '定义变量 A,B 为字节型
Config Porta = Output                     '定义 Porta 口为输出,控制 LED
Config Pinc.0 = Input , Pinc.1 = Input    '定义 Portc0,Portc1 为输入控制端
Portc.0 = 1 : Portc.1 = 1                 'Portc0,Portc1 上拉电阻有效,为"11"
Do
   Select Case B
      Case 0:
         A = &H01
         While B = 0                      '输入控制为"00"
            Porta = A
            Wait 1
            Rotate A , Left , 1           '左移循环
            B = Pinc And &H03             '读 Portc 口的控制
                                          '信号
         Wend
      Case 1:
         A = &H80
         While B = 1                      '输入控制为"01"
            Porta = A
            Wait 1
            Rotate A , Right , 1          '右移循环
            B = Pinc And &H03
         Wend
      Case 2:
         A = &H00
         While B = 2                      '输入控制为"10"
            Porta = A
            Wait 1
            A = Not A                     '明暗交替
            B = Pinc And &H03
         Wend
      Case 3:
         While B = 3                      '输入控制为"11"
            Porta = Rnd(255)              '产生随机数,随机闪烁
```

Portc0	Portc1	8 路发光 LED 闪烁方式
0	0	循环左移
0	1	循环右移
1	0	明暗交替
1	1	随机闪烁

```
            Wait 1
            B = Pinc And &H03
        Wend
    End Select
Loop
End
```

8.3.3　简易电脑音乐放音机

目的：利用 AVR 单片机的 I/O 口输出一定长度和频率的脉冲信号，再经过信号放大，由耳机或喇叭放出乐曲声。

原理：如何产生音乐频率 ——利用 BASCOM－AVR 的 SOUND 语句，可以很方便地产生一定长度和频率的脉冲信号。原理如图 8.7 所示。

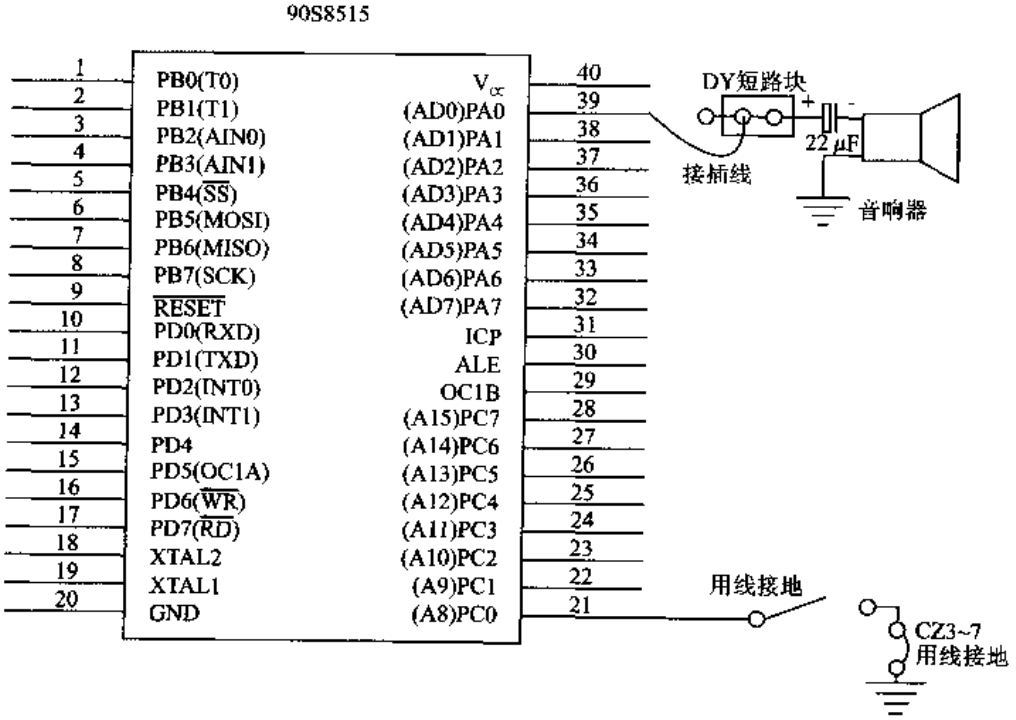


图 8.7　音乐放音机原理图

语句：Sound　pin，duration，pulses

说明：pin 为指定的 I/O 引脚，如 Portb0 等。duration 为产生脉冲的时长。pulses 为产生脉冲的频率数。产生脉冲时长和频率的简单计算如表 8.1 所列。

表 8.1　脉冲时长和频率计算表(晶振频率 4MHz)

节奏：　一拍　100(duration)					
音调	pulses	音调	pulses	音调	pulses
5	1276	2	865	6	568
#5	1205	#2	804	#6	536
6	1136	3	759	7	506

表 8.1(续)

节奏：一拍 100(duration)					
音调	pulses	音调	pulses	音调	pulses
#6	1073	4	716	$\dot{1}$	170
7	1012	#4	676	#$\dot{1}$	451
1	956	5	638	$\dot{2}$	426
#1	903	#5	602		

程序：exp3. bas

```
Dim S As Integer , F As Integer
Dim A As Byte
Config Pinc. 0    Input                    '定义 Portc0 为输入控制端
Portc. 0 = 1                               'Portc0 上拉电阻有效，为"1"
A    Pinc And &H01                         '读选曲控制
Do
   Select Case A                           '第一首歌
      Case 0:
         While A = 0
            Restore Music_1
Do
   Read S : Read F
   If S = 0 And F    0 Then Exit Do
   Sound Porta. 0 , S , F
               Waitms 50
            Loop
            Wait 2
            A = Pinc And &H01
         Wend
      Case 1:                              '第二首歌
         While A    1
            Restore Music_2
Do
 Read S : Read F
 If S = 0 And F    0 Then Exit Do
 Sound Porta. 0 , S , F
 Waitms 50
            Loop
            Wait 2
            A = Pinc And &H01
         Wend
   End Select
Loop
End
```

```
Music_1:                                    "两只老虎"的曲调
Data 100%,956%,100%,865%,100%,759%,100%,956%
Data 100%,956%,100%,865%,100%,759%,100%,956%
Data 100%,759%,100%,717%,200%,638%
Data 100%,759%,100%,717%,200%,638%
Data 50%,638%,50%,568%,50%,638%,50%,717%
Data 100%,759%,100%,956%
Data 50%,638%,50%,568%,50%,638%,50%,717%
Data 100%,759%,100%,956%
Data 100%,865%,100%,1276%,200%,956%
Data 100%,865%,100%,1276%,200%,956%,0%,0%
Music_2:                                    "生日快乐"的曲调
Data 75%,1276%,25%,1276%,100%,1137%,100%,1276%,100%,956%
Data 200%,1012%,75%,1276%,25%,1276%,100%,1137%
Data 100%,1276%,100%,865%,200%,956%
Data 75%,1276%,25%,1276%,100%,638%,100%,759%,100%,956%
Data 100%,1012%,100%,1136%
Data 75%,717%,25%,717%,100%,759%,100%,956%
Data 100%,865%,200%,956%,0%,0%
```

8.4 LCD 显示器

LCD 液晶显示器应用在许多电子产品中。BASCOM-AVR 中有专用的语句,可用于开发标准接口(采用日立公司 HD44780 及其兼容电路控制器)的 40×4,16×1,16×2,16×4,20×2,20×4,16×1a 等多种字符液晶显示器。

8.4.1 标准 LCD 显示器的应用

目的:字符和自定义字符在标准 LCD 显示器上显示。

原理:利用 BASCOM-AVR 的专用语句和特殊点阵设计工具在 LCD 液晶显示器上显示字符和特殊字符。原理如图 8.8 所示。

标准 LCD 液晶显示器简介:

标准点阵 LCD 组件由具有高反差、宽视角液晶显示屏和 CMOS 控制驱动器组成。液晶屏控制器多采用日立公司 HD44780 或兼容的控制芯片,与 MCU 接口容易,显示器上提供了字符发生器和显示数据 RAM。所有显示功能均由指令控制。

特点:结构紧凑,低功耗,数据线 4 位/8 位可选择,96 个 ASCII 字符代码加 92 个特殊字母,内装字符发生器和显示数据 RAM。

引脚功能见表 8.2。

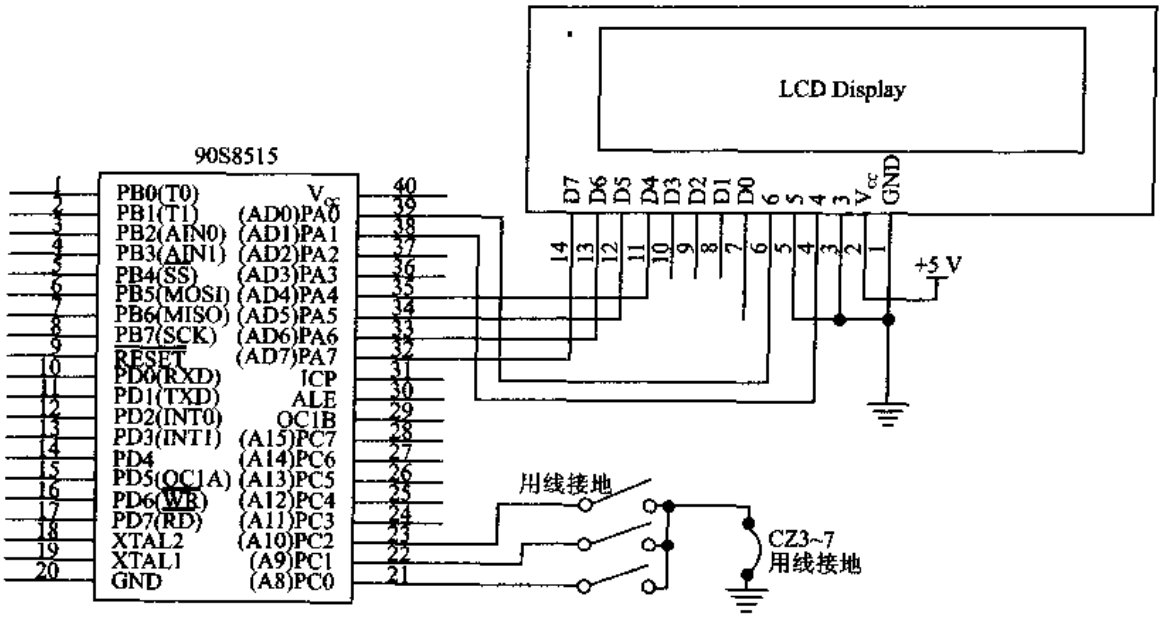


图 8.8 标准 LCD 显示器的应用原理图

表 8.2 标准 LCD 显示器引脚功能

引脚号	符号	功能	引脚号	符号	功能
1	V_{SS}	电源负端,接地(或接-5V)	7	DB0	
2	V_{DD}	电源正端,接+5V	8	DB1	
3	V_O	LCD 反差调整	9	DB2	
4	RS	寄存器选择 RS=0,选指令寄存器 RS=1,选数据寄存器	10	DB3	4 位数据总线 DB4~DB7
			11	DB4	8 位数据总线 DB0~DB7
5	R/W	读/写选择 R/W=0,写数据至 LCD R/W=1,从 LCD 读数据	12	DB5	
			13	DB6	
6	E	输入允许	14	DB7	

程序:exp4.bas

```
$ Sim                                          '此语句软件仿真时使用,正式下载时必须取消
'接 LCD 显示器的 I/O 引脚定义
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
        Db7 = Porta.7 , E = Porta.1 , Rs = Porta.0
Dim A As Byte                                  '定义字节变量 A
Config Lcd = 16 * 2                            '定义使用 16×2 行的 LCD 显示器
Defledchar 0 , 32 , 31 , 14 , 4 , 4 , 14 , 31 , 32   '定义 0 号特殊字符
Defledchar 1 , 31 , 32 , 17 , 27 , 27 , 17 , 32 , 31 '定义 1 号特殊字符
Cls                                            '清屏 LCD
Lcd "Hello world."                             '在第一行显示"Hello world."
```

```
Waitms 250
Lowerline                                          '选择第二行
Lcd Chr(0) ; Chr(1) ; "! @# $ %^&* " ; Chr(1) ; Chr(0)   '在第二行显示自定义字符和特殊字符
Waitms 250
For A = 1 To 16
   Shiftlcd Right                                  '显示内容右移16个位子
   Waitms 250
Next
For A = 1 To 28
   Shiftlcd Left                                   '显示内容左移28个位子
   Waitms 250
Next
For A = 1 To 12
   Shiftlcd Right                                  '显示内容右移12个位子回原位
   Waitms 250
Next
End
```

注:正式下载前可先使用软件仿真(见图8.5),观察显示效果。

8.4.2 简单游戏机——按钮猜数

目的:设计制作简单的按钮猜数游戏机。

原理:按钮猜数在电脑摇奖、电脑选出幸运号中经常使用。游戏开始时,连续不停产生3组0~9之间的随机数,并在LCD显示器显示。按动相应的开关可使对应位停止产生随机数,当3组数停止后,3个数一致或2个数一致便给出得奖提示。原理如图8.8所示。

程序:Exp5.bas

```
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7 = Porta.7 , E = Porta.1
, Rs = Porta.0
Config Lcd = 16 * 2                                '定义LCD接口
Config Pinc.0 = Input , Pinc.1 = Input , Pinc.2 = Input   '定义Portc.0,Portc.1,Portc.2为输入控制端
Portc.0 = 1 : Portc.1 = 1 : Portc.2 = 1            '上拉电阻有效,为"111"
Dim A As Byte , B As Byte , C As Byte , I As Byte
Cursor Off Noblink                                 '消隐LCD光标
Cls                                                '清LCD输出显示
Do
  I = Pinc And &H07                                '读按键值
  Waitms 50
  Select Case I
    Case 7:
      A = Rnd(90)/10 : B = Rnd(90)/10 : C = Rnd(90)/10
      Locate 1 , 6 : Lcd A                         '3个按键均未按下,连续显示3组随机数
      Locate 1 , 8 : Lcd B
      Locate 1 , 10 : Lcd C
```

```
        Case 6:
            A = Rnd(90)/10 : B = Rnd(90) / 10
            Locate 1 , 6 : Lcd A
            Locate 1 , 8 : Lcd B
        Case 5:
            A = Rnd(90)/10 : C = Rnd(90)/10
            Locate 1 , 6 : Lcd A
            Locate 1 , 10 : Lcd C
        Case 4:
            A = Rnd(90)/10
            Locate 1 , 6 : Lcd A
        Case 3:
            C = Rnd(90)/10 : B = Rnd(90)/10
            Locate 1 , 10 : Lcd C
            Locate 1 , 8 : Lcd B
        Case 2:
            B = Rnd(90)/10
            Locate 1 , 8 : Lcd B
        Case 1:
            C = Rnd(90)/10
            Locate 1 , 10 : Lcd C
        Case 0:                                    '3个按键均按下
            While I = 0                            '判断输赢
                Locate 2 , 1
                If A = B And B = C Then
                    Lcd "You Win $10000!"
                Elseif A = B Or B = C Or A = C Then
                    Lcd "You Win $100!"
                Else
                    Lcd "You Lose $10!"
                End If
                    I = Pinc And &H07
            Wend
            Lowerline
            Lcd "                "
    End Select
Loop
End
```

8.5 串口通信 UART

大部分的 AVR 芯片都含有一个硬件串行通信接口 UART，利用该接口可以实现控制系

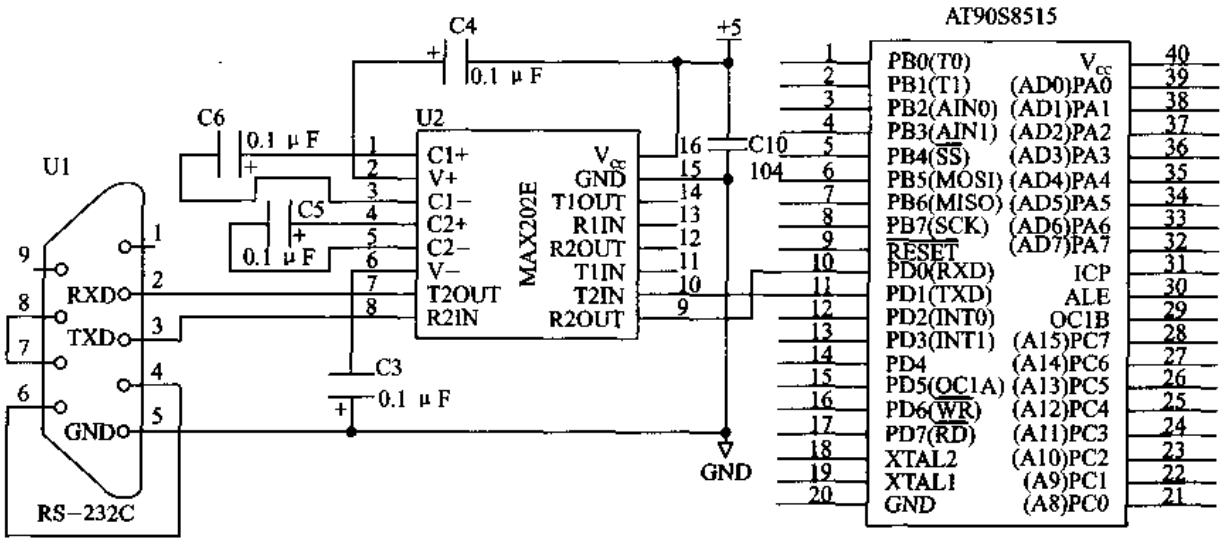
统与 PC 的通信，构成 RS－232、RS－485 或 RS－422 的网络。

8.5.1　AVR 系统与 PC 的简易通信

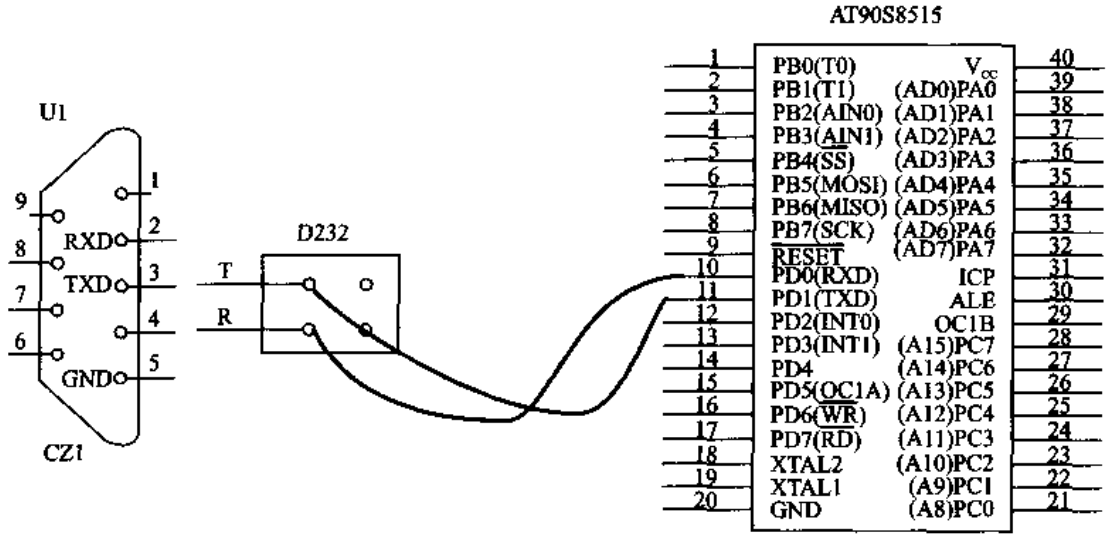
目的：实现 AVR 系统与 PC 之间的串口连网和通信。

原理：利用 AVR 芯片的 URAT 建立与 PC 机 COM 口的简易 RS－232 连网通信，AVR 系统输出字符在 PC 机的屏幕上显示。

原理如图 8.9 所示。



(a) 通信原理图



(b) SL－AVR 通信接口

图 8.9　AVR 系统与 PC 的简易通信

图中 MAX202E 为 TTL 电平与 RS－232 电平转换电路。AVR 系统的接口电平为 5V 的 TTL 电平，必须通过一个电平转换电路才能和 PC 机的串行通信 COM 口连接，否则将损坏 PC 机的 COM 口和 AVR 芯片！

SL－AVR 开发实验器上已有 RS－232 的电平转换电路，用户可利用系统所提供的 RS－232 连接电缆和 SL－AVR 开发实验器上 D232 短路块，通过 D232 的 R、T 接线端用 2 根跳线实现图 8.9(b)的电路。

程序：EXP6.bas

```
$regfile = "8515def.dat"
$crystal = 4000000                    '晶振频率 4MHz
$baud = 9600                          '波特率 9600
                                      'RS－232 缺省设置：8 位数据位，1 位停止位，无校验
Dim A As Byte
A = 0
Do
   Print "This is " ; A ; " Line"      'Print 的内容将在 PC 的屏幕上显示
   A = A + 1
   Waitms 250
Loop
   End
```

实验方法：

实验程序编译下载到 AT90S8515 后，关闭 SL－AVR 开发实验器电源。将 SL－AVR 开发实验器上 D232 短路块断开。使用连线应参照图 8.9(b)正确连接。

关闭 PC 上运行的 AVR 程序下载软件，打开 BASCOM－AVR 的终端仿真功能或 Windows 的超级终端。设置 PC 机的终端仿真口使用的 COM 口号(COM1 或 COM2)及相关通信参数：波特率 9600、8 位数据位、1 位停止位、无校验、无数据流控制。

打开系统电源，在 PC 机的屏幕上将显示 AVR 系统输出的字符。

8.5.2 PC 控制的简易广告灯

目的：实现 PC 机控制的简易广告灯。

原理：由 AVR 系统构成简易广告灯(参见 8.3.2 节)，广告灯的显示方式通过 PC 机控制。原理如图 8.10 所示。

程序：EXP7.bas

```
$regfile = "8515def.dat"
$crystal = 4000000                    '晶振频率 4MHz
$baud = 9600                          '波特率 9600
                                      'RS－232 缺省设置：8 位数据位，1 位停止位，无校验
Dim A As Byte , B As Byte             '定义变量 A，B 为字节型
Config Porta = Output                 '定义 Porta 口为输出，控制 LED
B = "0"                               'B 初始值字符为 "0"
Do
  Select Case B
    Case "0":                         'PC 键盘输入字符为"0"
      A = &H01
      Do
```

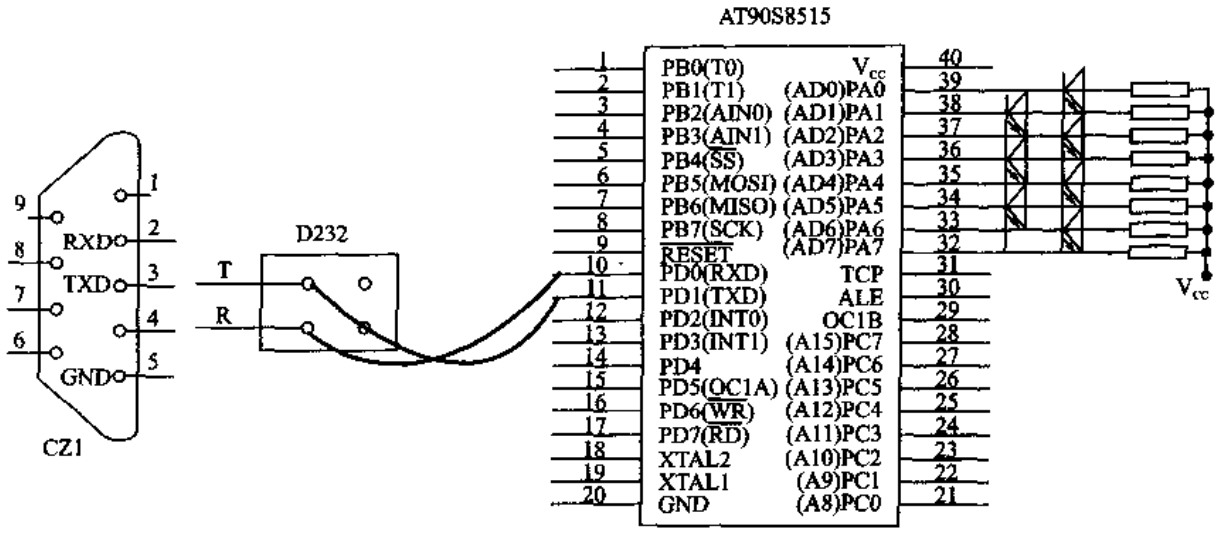


图 8.10 SL-AVR通信接口

```
        Porta = A
        Wait 1
        Rotate A , Left , 1          '左移循环
        B = Inkey()                  '读 PC 的控制命令
    Loop Until B <> 0                'PC 键盘无输入,继续左移循环
Case "1":                            'PC 键盘输入字符为"1"
    A = &H80
    Do
        Porta = A
        Wait 1
        Rotate A , Right , 1         '右移循环
        B = Inkey()
    Loop Until B <> 0
Case "2":                            'PC 键盘输入字符为"2"
    A = &H00
    Do
        Porta = A
        Wait 1
        A = Not A                    '明暗交替
        B = Inkey()
    Loop Until B <> 0
Case Else                            'PC 键盘输入其它字符
    Do
        Porta = Rnd(255)             '产生随机数,随机闪烁
        Wait 1
        B = Inkey()
    Loop Until B <> 0
End Select
```

键盘字符	LED闪烁方式
"0"	循环左移
"1"	循环右移
"2"	明暗交替
其它	随机闪烁

```
Loop
End
```

实验方法：本实验过程、方法同 8.5.1 节。按下 PC 机键盘的字符“0”，“1”，“2”及其它字符，按下的字符通过串口传送到 AVR 系统，AVR 系统将根据接收到的字符控制发光二极管的闪烁方式。

8.6 单总线接口和温度计

“单总线(1-Wire)”接口是美国 DALLAS 公司的特有技术(http://www.dalsemi.com)，它实现了在一条数据线上进行双向数据传输，最大限度地节省了通讯线的数量，使系统布线更方便，成本更低，适用于多点分散系统。同时，DALLAS 公司推出了丰富的单总线产品，如数字温度传感器、电子纽扣(Ibutton)等。这种单总线芯片仅有 1 根信号线和 1 根地线。在信号线上综合了双向数字信号线和电源线的功能，每一块芯片都可提供一个独一无二的登记号，使用户可以灵活地构成不同功能的系统。而且，单总线产品基本为单芯片，成本低、可靠性高，如 DS-1820 集传感器、温度传换、联网通讯于一体。

由于单总线仅使用 1 根信号线进行双向数据传输，因此单总线的通信协议就比其它的串行通信协议要复杂的多，通信时序和时间定时要求也非常严格。在一般的电子产品开发中往往要花费大量的精力和时间，编写和调试单总线接口的底层及通信程序。BASCOM-AVR 提供了简便的 1-Wire 语句，以方便和快速地开发单总线产品。

目的：用 DS-1820 和 AVR 系统构成 0.1℃精度的温度计。

原理：美国 DALLAS 公司生产的单总线数字温度传感器 DS-1820 可把温度信号直接转换成串行数字信号供微机处理。由于每片 DS-1820 含有惟一的硅串行数，所以在 1 条总线上可挂接任意多个 DS-1820。从 DS-1820 读出的信息或写入 DS-1820 的信息，仅需要 1 根数据线(单总线接口)。读写及温度变换功率来源于数据总线，总线本身也可以向所挂接的 DS-1820 供电，而无需额外电源。DS-1820 提供 9 位温度读数，构成多点温度检测系统而无需任何外围硬件。本实验采用 1 片 DS-1820 与 AVR 单片微控制器和 LCD 显示器组成一个温度检测系统。原理如图 8.11 所示。

(1) DS-1820 的特性

单线接口仅需 1 根数据线与 MCU 连接，无需外围元件，可由总线提供电源；测温范围为 55～75℃，精度达 0.1℃；内置 A/D 变换，转换时间为 200ms；9 位温度读数，用户可设定温度报警上下限，其值是非易失性的。

(2) DS-1820 引脚及功能

DS-1820 引脚见图 8.12(PR35 封装)。

GND：地；

DQ：数据输入/输出脚(单总线接口，可作寄生供电)；

V_{DD}：电源电压。

(3) DS-1820 操作指令及时序

参见 Dallas 公司 DS-1820 数据手册。

程序：Exp8.bas

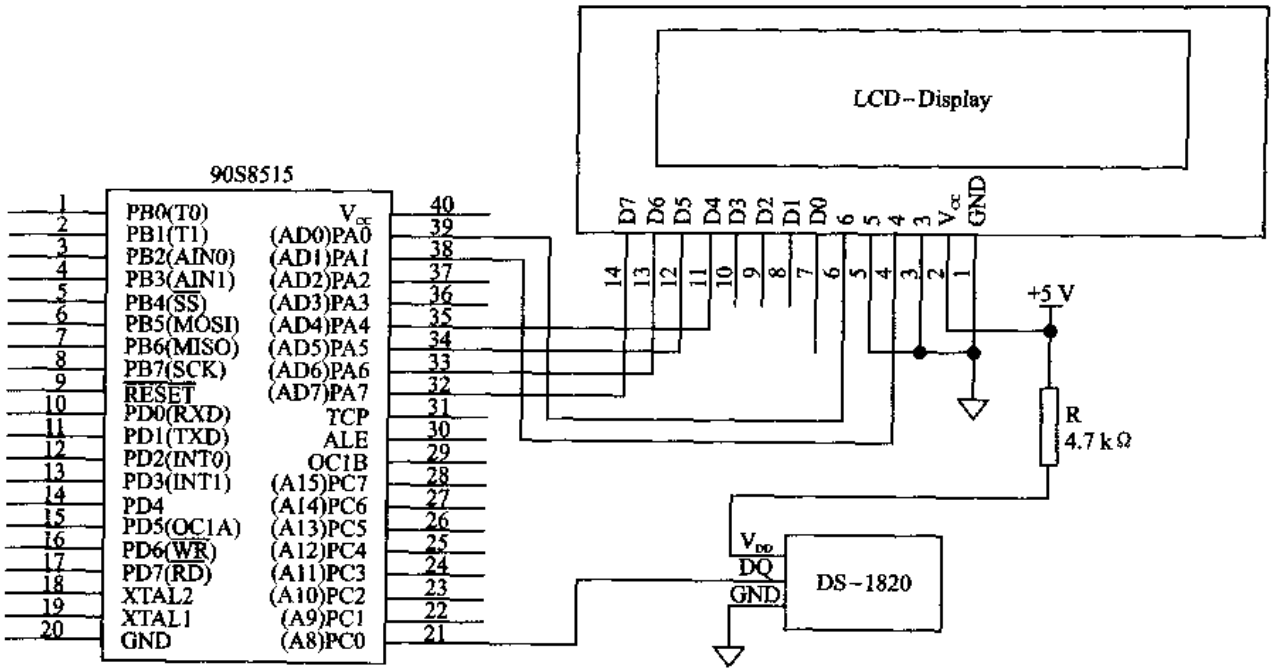


图 8.11　用 DS－1820 和 AVR 系统构成温度计原理图

```
$ regfile = "8515def. dat"
$ crystal = 4000000
Deflcdchar 0 , 14 , 10 , 14 , 32 , 32 , 32 , 32 , 32    '定义"℃"的显示字符
Deflcdchar 1 , 4 , 10 , 17 , 4 , 31 , 2 , 4 , 8         '定义汉字"今"的显示字符
Deflcdchar 2 , 31 , 4 , 31 , 4 , 12 , 10 , 18 , 17      '定义汉字"天"的显示字符
Dim I As Byte , Tmp As Byte , Crc As Byte
Dim T As Integer , T1 As Integer
Dim Data1(9) As Byte
Config 1wire = Portc. 0                                 'Portc. 0 接 DS－
                                                        '1820 的数据线
Config Lcdpin=Pin, Db4=Porta. 4, Db5=Porta. 5, Db6=Porta. 6, Db7
=Porta. 7, E=Porta. 0, Rs=Porta. 1
Config Lcd = 16 * 2                                     '定义 LCD 接口
Cursor Off Noblink                                      '清 LCD 显示
                                                        '无光标
Cls
Locate 1 , 1 : Lcd "Demo for DS1820"                    '第 1 行显示提示字符串
Locate 2 , 1 : Lcd Chr(1) ; Chr(2)                      '第 2 行 1,2 列显示汉字"今天"
Locate 2 , 8 : Lcd Chr(0) ; "C"                         '第 2 行 8 列显示字符"℃"
Do
   1wwrite &HCC : 1wwrite &H44                          '写入 DS－1820 命令,启动温度转换
   Waitms 255
   Waitms 255                                           '等待转换结束
   1wreset                                              'DS－1820 初始化
```

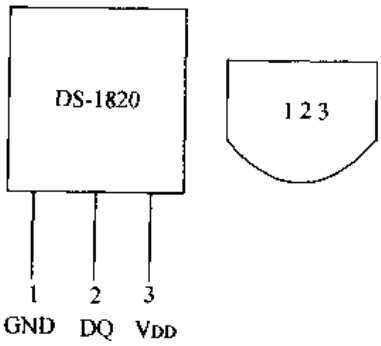


图 8.12　DS－1820 引脚图

```
    1wwrite &HCC                          '写入 DS-1820 命令,跳过 ROM 操作
    1wwrite &HBE                          '写入 DS-1820 命令,读温度值
    Data1(1) = 1wread(9)                  '读 DS-1820 温度,共 9 个字节
    1wreset                               'DS-1820 初始化
    If Err = 1 Then
       Locate 2 , 3 : Lcd "------"        '无 DS-1820 显示"------"
    Else
       Crc = 0
       For I = 1 To 9                     '进行 CRC 校验
          Tmp = Crc Xor Data1(i)
          Crc = Lookup(tmp , Crc8)
       Next I
       If Crc = 0 Then                    'CRC 校验正确,进行温度换算
          Tmp = Data1(1) And 1
          If Tmp = 1 Then Decr Data1(1)
          T = Makeint(data1(1) , Data1(2))
          T = T * 50 : T = T - 25
          T1 = Data1(8) - Data1(7) : T1 = T1 * 100
          T1 = T1 / Data1(8) : T = T + T1 : T = T / 10
       End If
       If Crc = 0 Then                    'CRC 校验正确,温度显示
          If T < 0 Then
             T = Abs(t)
             Locate 2 , 3 : Lcd "-"       '负温度显示"-"号
          Else
             Locate 2 , 3 : Lcd " "       '正温度显示"空格"
          End If
          T1 = T Mod 10 : T = T / 10      '以下显示温度,精度 0.1℃
          If T < 10 Then
             Locate 2 , 4 : Lcd " " ; T ; "." ; T1
          Else
             If T > 99 Then
                Locate 2 , 3 : Lcd T ; "." ; T1
             Else
                Locate 2 , 4 : Lcd T ; "." ; T1
             End If
          End If
       Else
          Locate 2 , 3 : Lcd "****"       'CRC 校验错,显示错误标志"****"
       End If
    End If
    Loop
End
```

```
Crc8:                                                    'CRC 计算用的表格
Data 0 , 94 , 188 , 226 , 97 , 63 , 221 , 131 , 194 , 156
Data 126 , 32 , 163 , 253 , 31 , 65 , 157 , 195 , 33 , 127
Data 252 , 162 , 64 , 30 , 95 , 1 , 227 , 189 , 62 , 96
Data 130 , 220 , 35 , 125 , 159 , 193 , 66 , 28 , 254 , 160
Data 225 , 191 , 93 , 3 , 128 , 222 , 60 , 98 , 190 , 224
Data 2 , 92 , 223 , 129 , 99 , 61 , 124 , 34 , 192 , 158
Data 29 , 67 , 161 , 255 , 70 , 24 , 250 , 164 , 39 , 121
Data 155 , 197 , 132 , 218 , 56 , 102 , 229 , 187 , 89 , 7
Data 219 , 133 , 103 , 57 , 186 , 228 , 6 , 88 , 25 , 71
Data 165 , 251 , 120 , 38 , 196 , 154 , 101 , 59 , 217 , 135
Data 4 , 90 , 184 , 230 , 167 , 249 , 27 , 69 , 198 , 152
Data 122 , 36 , 248 , 166 , 68 , 26 , 153 , 199 , 37 , 123
Data 58 , 100 , 134 , 216 , 91 , 5 , 231 , 185 , 140 , 210
Data 48 , 110 , 237 , 179 , 81 , 15 , 78 , 16 , 242 , 172
Data 47 , 113 , 147 , 205 , 17 , 79 , 173 , 243 , 112 , 46
Data 204 , 146 , 211 , 141 , 111 , 49 , 178 , 236 , 14 , 80
Data 175 , 241 , 19 , 77 , 206 , 144 , 114 , 44 , 109 , 51
Data 209 , 143 , 12 , 82 , 176 , 238 , 50 , 108 , 142 , 208
Data 83 , 13 , 239 , 177 , 240 , 174 , 76 , 18 , 145 , 207
Data 45 , 115 , 202 , 148 , 118 , 40 , 171 , 245 , 23 , 73
Data 8 , 86 , 180 , 234 , 105 , 55 , 213 , 139 , 87 , 9
Data 235 , 181 , 54 , 104 , 138 , 212 , 149 , 203 , 41 , 119
Data 244 , 170 , 72 , 22 , 233 , 183 , 85 , 11 , 136 , 214
Data 52 , 106 , 43 , 117 , 151 , 201 , 74 , 20 , 246 , 168
Data 116 , 42 , 200 , 150 , 21 , 75 , 169 , 247 , 182 , 232
Data 10 , 84 , 215 , 137 , 107 , 53
```

8.7 I^2C 总线接口和简易 IC 卡读写器

I^2C 总线是 PHILIPS 公司推出的串行总线，是各种总线中使用信号线较少，并且具有自动寻址、多主机时钟同步和仲裁等功能的总线。因此，许多接口芯片如 RAM、LCD 驱动、时钟、A/D、D/A 都采用 I^2C 接口。IC 卡在当今社会的各个行业都有着广泛的应用，而且大多数 IC 卡的接口都采用 I^2C 总线。

目的：用 AVR 系统构成简易 IC 卡读写器。

原理：I^2C 串行总线使用 2 根信号线，1 根双向的数据线 SDA，另 1 根是时钟线 SCL。所有连接到 I^2C 总线上的设备的串行数据都接到总线的 SDA 线，各设备的时钟线 SCL 接到总线的 SCL。I^2C 数据总线传输时序如图 8.13 所示。有关 I^2C 总线的详细介绍请参考相关资料。

AVR 系列单片机没有专用的 I^2C 总线硬件接口，因此需要用 2 根 I/O 线来模拟实现 I^2C 总线的功能。这就要编写相应的 I^2C 底层程序，模拟 I^2C 数据总线的传输时序，实现 I^2C 总线起始、停止、读、写及应答的功能。BASCOM－AVR 提供了专用的 I^2C 语句，利用它们就能非

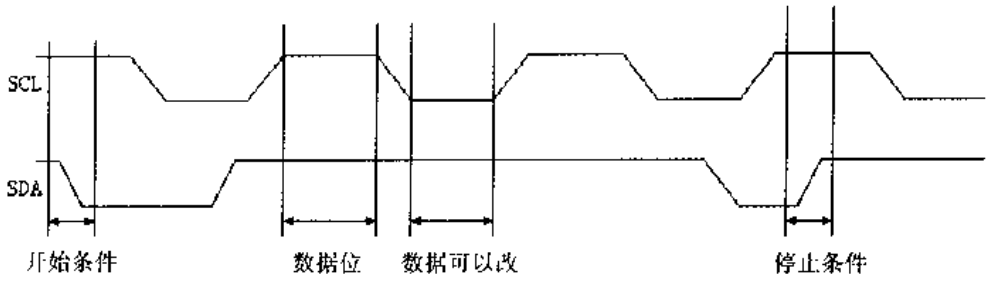


图 8.13 I²C 数据总线传输时序

常方便地设计和实现 I²C 总线的读写。

实验中采用的 IC 卡为 ATMEL 公司的 AT24C01A/2/4/8/16，它是一种通用的读写存储卡。该类 IC 卡上的芯片就是采用 I²C 总线接口的串行 CMOS EEPROM。关于 AT24CXX 系列芯片的详细资料可访问 http://www.atmel.com 网站。

AVR 系列单片机内部还提供了一定容量的 EEPROM，可以用于保存系统的初始数据、设定值或密码口令字等，这一性能为开发产品提供了许多方便，它不仅可使系统设计节省硬件(EEPROM 芯片)和连线，而且还减小了空间，提高了系统的可靠性和保密性。本设计中，使用了 AVR 片内的 EEPROM 来保存密码，并同 IC 卡上的密码核对，判别 IC 卡的合法性。

本设计将 IC 卡的读、写、判别结合在一起，用户使用 PC 的键盘输入 8 位自定密码，通过 RS-232 送到 IC 卡读写器，将密码写入用户的 IC 卡中(也可同时写入 AVR 的 EEPROM 中作为系统密码)。IC 卡读写器还可读取 IC 卡上的密码，并同系统密码核对，用于判别 IC 卡的合法性。IC 卡读写器采用 LCD 液晶显示器，用于显示系统的状态和 IC 卡上的密码等。其原理图和流程图如图 8.14 和 8.15 所示。

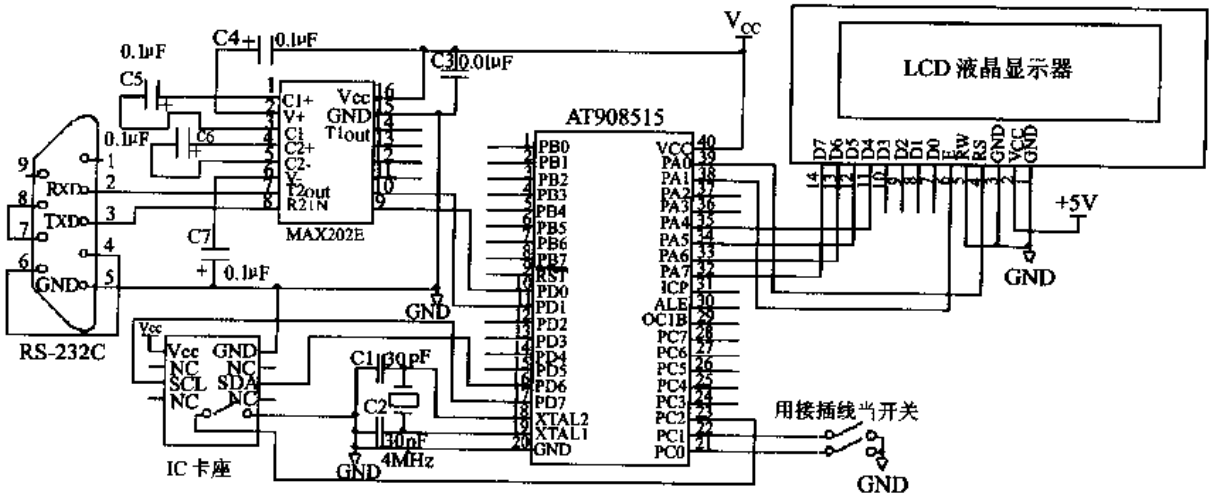


图 8.14 IC 卡读写器原理图

程序：Exp9.bas

```
$ regfile = "8515def.dat"
$ crystal = 4000000
$ baud = 9600
Dim I As Byte , Temp As Byte
```

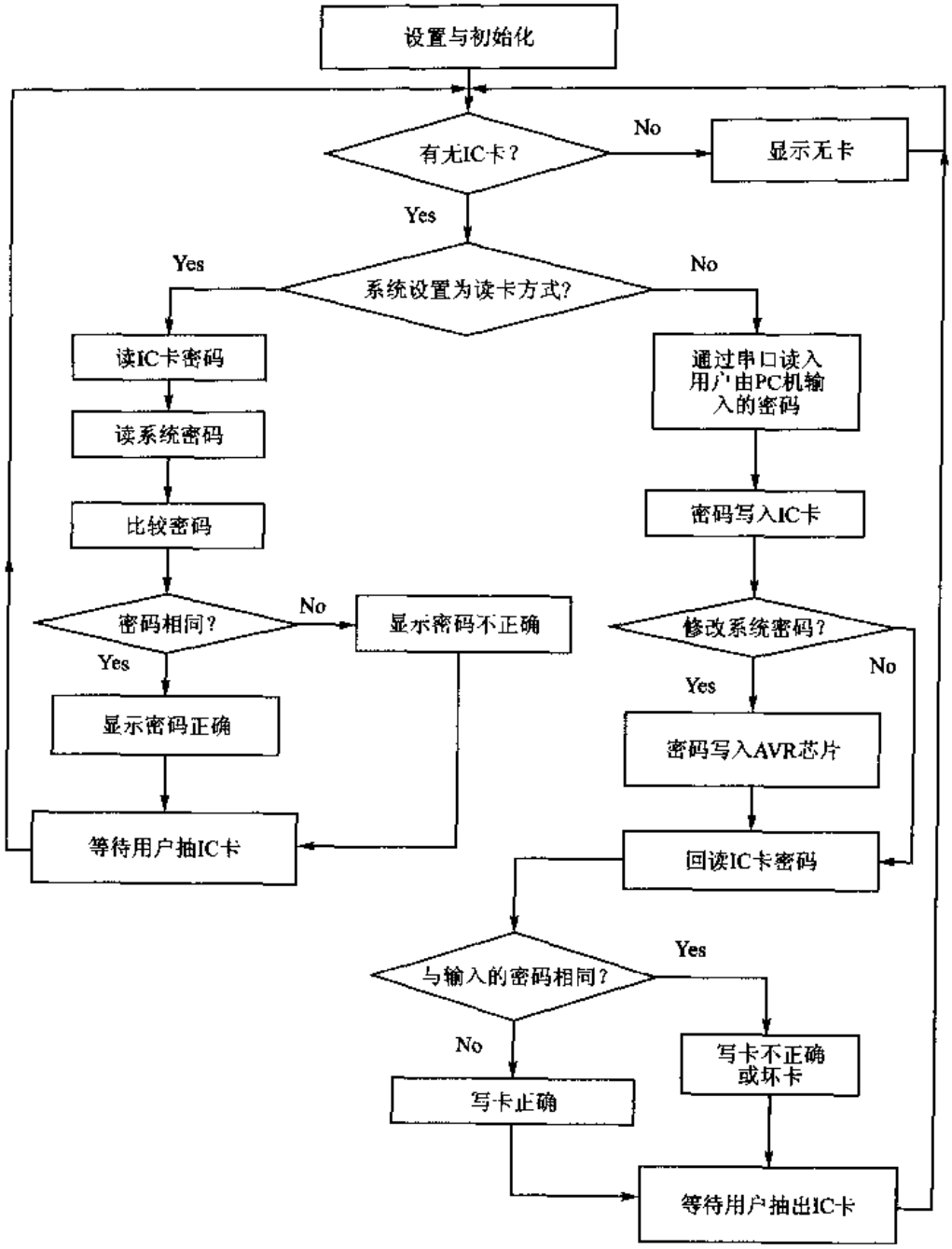


图 8.15　IC 卡读写流程图

```
Dim Data1(8) As Byte , Data2(8) As Byte
Dim Card_ok As Bit
Config Lcdpin=Pin,Db4=Porta.4,Db5=Porta.5,Db6=Porta.6,Db7=Porta.7,E=Porta.0,Rs=Porta.1
Config Lcd = 16 * 2                    '定义 LCD 显示屏接口
Config Scl = Portd.7                   '定义 Portd.7 为 I²C 总线的 Scl
Config Sda = Portd.6                   '定义 Portd.6 为 I²C 总线的 Sda
Const Adresw = &HA0                    '定义 IC 卡的写地址指令字
Const Adresr = &HA1                    '定义 IC 卡的读地址指令字
Config Pinc.0 = Input , Pinc.1 = Input , Pinc.2 = Input
```

```
Portc.0 = 1 : Portc.1 = 1 : Portc.2 = 1                    '定义3个I/O口为输入开关接口
Cursor Off Noblink                                          '清LCD显示
Do
        Cls
        Locate 1 , 1 : Lcd "Demo for IC_Card"               '显示提示字符
        If Pinc.2 = 1 Then                                  '检测有无IC卡插入
          Locate 2 , 1 : Lcd "No IC_Card"                   '无IC卡插入
          Wait 1
        Else                                                '有IC卡插入
          If Pinc.0 = 0 Then                                '检测系统设置为写卡方式?
            Cls
            Locate 1 , 1 : Lcd "Enter Password: "           '写卡方式,要求输入密码字
            I = 1
            Locate 2 , 1
            Do
                Temp = Inkey()                              '由RS-232口接收PC输入的密码字符
                If Temp <> 0 Then
                  Data1(i) = Temp : I = I + 1               '长度为8个
                  Lcd Chr(temp)                             '同时在LCD上显示密码
                  Print Chr(temp);
                End If
            Loop Until I > 8
            Print
            For I = 1 To 8                                  '将8个密码字写入IC卡中
                I2cstart                                    '写入地址为1~8
                I2cwbyte Adresw
                I2cwbyte I
                I2cwbyte Data1(i)
                I2cstop
                Waitms 10
            Next
            If Pinc.1 = 0 Then
                For I = 1 To 8                              '如果系统设置为修改系统密码时
                  Writeeeprom Data1(i) , I                  '将密码写入AVR的EEPROM中
                Next                                        '写入地址为1~8
            End If
            For I = 1 To 8                                  '读IC卡刚写入的密码
                I2cstart
                I2cwbyte Adresw
                I2cwbyte I
                I2cstart
                I2cwbyte Adresr
                I2crbyte Data2(i) , Nack
```

```
            I2cstop
        Next
        Card_ok = 0
        For I = 1 To 8                          '判别写入密码与用户输入的密码
            If Data1(i) <> Data2(i) Then        '是否相同
                Card_ok = 1
                Exit For
            End If
        Next
        Locate 2 , 1
        If Card_ok = 0 Then
            Lcd "Write Card ok!"                '相同,写卡完成
        Else
            Lcd "No or Bad Card!?"              '不同,写卡操作有误或坏的 IC 卡
        End If
        Do
        Loop Until Pinc.2 = 1                   '等待用户抽出 IC 卡
    Else                                        '系统为读卡方式
        Cls : Locate 1 , 1
    For I = 1 To 8
            I2cstart
            I2cwbyte Adresw
            I2cwbyte I
            I2cstart
            I2cwbyte Adresr
            I2crbyte Data1(i) , Nack
            I2cstop                             '读 IC 卡上的密码字 8 个
            Lcd Chr(data1(i))                   '显示在 LCD 上
        Next
        For I = 1 To 8
            Readeeprom Data2(i) , I             '读系统密码字 8 个
        Next
        Card_ok = 0
        For I = 1 To 8                          '将系统密码与 IC 卡上的密码
            If Data1(i) <> Data2(i) Then        '比较
                Card_ok = 1
                Exit For
            End If
        Next
        Locate 2 , 1
        If Card_ok = 1 Then
            Lcd "Password not ok!"              '密码不符
        Else
```

```
                    Lcd "Password Ok!"                    ' 密码相符
                End If
                Do
                Loop Until Pinc.2 = 1                     ' 等待用户将 IC 卡抽出
            End If
        End If
Loop                                                      ' 返回循环
End
```

第九章　ICC AVR C 编译器的使用

C 语言是一门具有结构化的语言。其特点是功能强、效率高和与系统十分接近。用 C 语言，可以采用少量的结构解决复杂的问题；同时，C 表达式的经济性和丰富的操作符集合也是它的一个特点。因而，在很多情况下采用 C 语言对单片机进行编程可以使程序简单，增加可读性。

9.1　ICC AVR 的概述

9.1.1　介绍 ImageCraft 的 ICC AVR

ImageCraft 的 ICC AVR 是一种使用符合 ANSI 标准的 C 语言来开发微控制器(MCU)程序的一个工具。它有以下几个主要特点：

① ICC AVR 是一个综合了编辑器和工程管理器的集成工作环境(IDE)，其可在 Windows 9X/NT 下工作。

② 源文件全部被组织到工程之中，文件的编辑和工程的构筑也在这个环境中完成。

③ 编译错误显示在状态窗口中，并且在单击编译错误时，光标会自动跳转到编辑窗口中引起错误的那一行。

④ 这个工程管理器还能直接产生用户希望得到的、可以直接使用的 INTEL HEX 格式文件。INTEL HEX 格式文件可被大多数的编程器所支持，用于下载程序到芯片中去。

⑤ ICC AVR 是一个 32 位的程序，支持长文件名。

本书仅介绍使用 ICC AVR 所必备的知识，因此，要求读者在阅读本书内容之前，应对 C 语言有一定程度的理解。

9.1.2　ICC AVR 中的文件类型及其扩展名

文件类型是由它们的扩展名决定的，IDE 和编译器可以使用以下几种类型的文件。

(1) 输入文件

.c　　扩展名——表示是 C 语言源文件。

.s　　扩展名——表示是汇编语言源文件。

.h　　扩展名——表示是 C 语言的头文件。

.prj　扩展名——表示是工程文件。这个文件保存由 IDE 所创建和修改的一个工程的有关信息。

.a　　扩展名——库文件。它可以由几个库封装在一起。libcavr.a 是一个包含了标准 C 的库和 AVR 特殊程序调用的基本库。如果库被引用，链接器会将其链接到用户的模块或文件中。用户也可以创建或修改一个符合自己需要的库。

(2) 输出文件

.s 对应每个 C 语言源文件。由编译器在编译时产生的汇编输出文件。
.o 由汇编文件汇编产生的目标文件。多个目标文件可以链接成一个可执行文件。
.hex INTEL HEX 格式文件。其中包含了程序的机器代码。
.eep INTEL HEX 格式文件。包含了 EEPROM 的初始化数据。
.cof COFF 格式输出文件。用于在 ATMEL 的 AvrStudio 环境下进行程序调试。
.lst 列表文件。在这个文件中列举出了目标代码对应的最终地址。
.mp 内存映象文件。它包含了用户程序中有关符号及其所占内存大小的信息。
.cmd NoICE 2.xx 调试命令文件。
.noi NoICE 3.xx 调试命令文件。
.dbg ImageCraft 调试命令文件。

9.1.3 附注和扩充

1. #pragma(编译附注)

这个编译器接受以下附注:

#pragma interrupt_handler <func1>:<vector number> <func2>:<vector> ...

这个附注必须在函数之前定义。它说明函数 func1、func2 是中断操作函数,所以编译器在中断操作函数中生成中断返回指令 reti 来代替普通返回指令 ret,并且保存和恢复函数所使用的全部寄存器;同样编译器根据中断向量号 vector number 生成中断向量地址。

2. #pragma ctask <func1> <func2>...

这个附注指定了函数不生成挥发寄存器来保存和恢复代码。它的典型应用是在 RTOS 实时操作系统中让 RTOS 核直接管理寄存器。

3. #pragma text:<name>

改变代码段名称,使其与命令行选项相适应。

4. #pragma data:<data>

改变数据段名称,使其与命令行选项相适应。这个附注在分配全局变量至 EEPROM 中时必须被使用。读者可参考访问 EEPROM 的例子。

5. #pragma abs_address:<address>

函数与全局数据不使用浮动定位(重定位),而是从<address>开始分配绝对地址。这在访问中断向量和其它硬件项目时特别有用。

6. #pragma end_abs_address

结束绝对定位,使目标程序使用正常浮动定位。

7. C++注释

如果用户选择了编译扩充(Project→Options→Compiler),可以在用户的源代码中使用 C++的//类型的注释。

8. 二进制常数

如果用户选择了编译扩充(Project→Options→Compiler),可以使用 0b<1|0>* 来指定二进制常数。例如,0b10101 等于十进制数 21。

9. 在线汇编

可以使用 asm("string")函数来指定在线汇编代码,读者可参考在线汇编。

9.2 ImageCraft 的 ICC AVR 编译器安装

9.2.1 安装 SETUP.EXE 程序

方法一：

① 打开“我的电脑”；

② 打开光盘驱动器所对应的盘符；

③ 双击光盘中文件“SETUP.EXE”的图标；

④ 按照屏幕提示，选定一个安装路径后进行安装。

方法二：

① 在“开始”菜单中选择运行项目；

② 在“运行”对话框中填入 drive:\setup.exe；

③ 注意，drive 对应用户机器中的光盘驱动器盘符；

④ 按“确定”键，开始安装。

注意：

① 按上述方法进行安装后，得到的是一个只可以使用 30 天的未注册版。用户还要进行第二步的注册，才可以得到一个无时间限制的正式版。

② ICC AVR 正式版分标准版和专业版。在标准版中有一些功能限制，如在标准版中不可以使用代码的压缩、工程和文件的配置检查。

9.2.2 对安装完成的软件进行注册

对首次安装且使用期未超过 30 天的用户，可以按下述步骤注册：

① 启动 ICC AVR 编译器的集成环境(IDE)。

② 将正式版中附带的一张名称为 Unlock Disk 的软盘插入用户机器的软盘驱动器中。

③ 在 IDE 的 Help 菜单中，单击标题为 Importing a License from a Floppy Disk 的一项(见图 9.1)。

④ ICC AVR 软件自动进行注册。当注册完成后，提示用户注册文件已从软盘中移走。当用户“确定”并重新启动 ICC AVR 后，会看到软件已经完成注册。

对不是首次安装或使用时间已超过 30 天的用户，可按下述步骤注册：

① 对这类用户，在程序启动时已不能进入 IDE 环境，而是出现一个提示用户注册的对话框。用户应该选择 Yes 按钮。

② 这时会出现一个注册对话框，对话框上有一个标题为 Importing a License from a Floppy Disk 的按钮。

③ 将正式版中附带的一张名称为 Unlock Disk 的软盘插入用户机器的软盘驱动器中，单击②中提到的按钮。

④ ICC AVR 软件自动进行注册。当注册完成后，提示用户注册文件已从软盘中移走。当用户“确定”并重新启动 ICC AVR 后，会看到软件已经完成注册。

注意：

① Unlock Disk 软盘在注册时应打开写保护，否则无法完成注册。

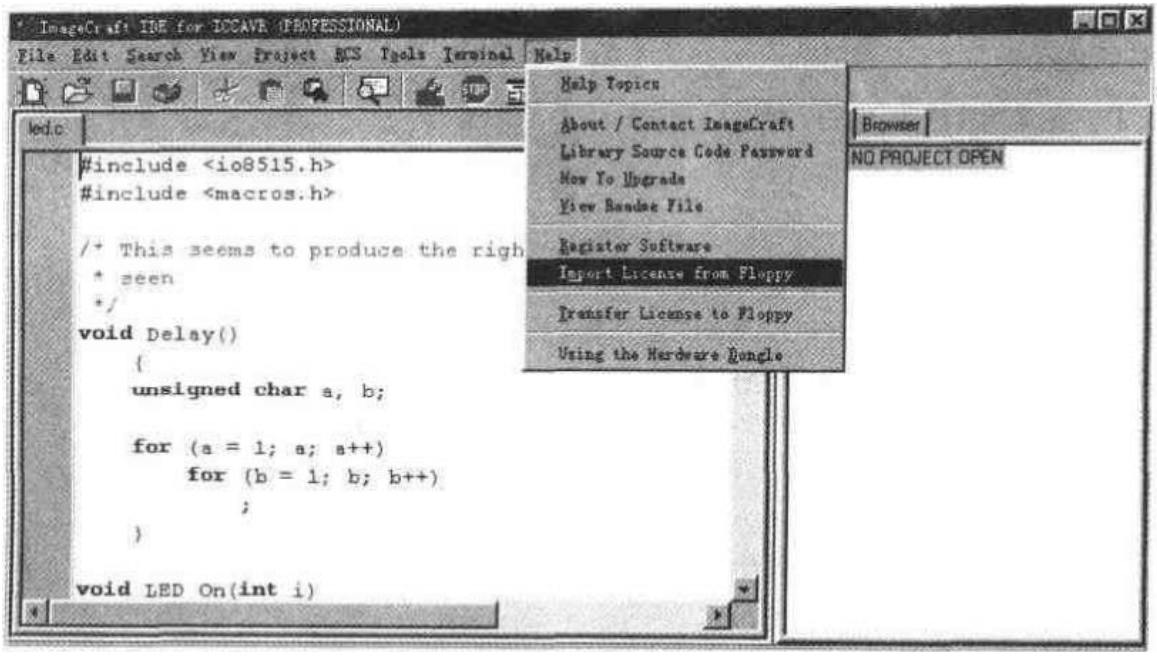


图 9.1 ICC AVR 工作窗口

② 完成注册后，Unlock Disk 软盘成为一张空盘，不可在另一台机器上进行安装和注册。

③ 当用户需要在不同的电脑中使用 ICC AVR 或在同一台电脑中将 ICC AVR 重新安装在与原来不同的目录位置时，应该首先在 Help 菜单中选择 Transferring Your License to a Floppy Disk 一项，将用户的注册文件传送到一张软盘上，然后再按上述方法进行安装注册。

9.3 ICC AVR 导游

9.3.1 起 步

启动 IDE 后，首先从 Project 菜单系统选择 Open 命令，进入\icc\examples. avr 目录，选择并打开 led 工程，工程管理器显示在这个工程中只有一个文件 led. c。然后从 Project 菜单中选择 Options 命令打开工程编译选项，在 Target 标号下选择目标处理器。最后从 Project 菜单中选择 Make Project 命令，IDE 将调用编译器编译这个工程文件，并且在状态窗口中显示所有的信息。

如果没有错误，在与源文件同一个目录(在这个例子中是\icc\examples. avr)中输出一个文件 led. hex。这个文件是 INTEL HEX 格式。大多数能支持 AVR MCU 的编程器和模拟器都支持这种格式，并且能下载这个程序进入用户的目标系统。这样就完成了一个程序的构筑。

如果用户希望用支持 COFF 调试信息的工具来测试自己的程序，比如 AVR Studio，那么需要从 Project 菜单中选择 Options 命令，在编译标签下选择 COFF 输出文件格式。对一些常用的功能，也可使用工具条或右击弹出菜单。例如，可以在工程窗口右击选择编译选项。

在工程窗口中双击文件名，IDE 将使用编辑器打开这个文件。按这个方法打开 led. c，作为实验可设置一些错误，例如从一行中删除分号“;”。现在从 Project 菜单中选择 Make Project命令，IDE 首先自动保存已经改变的文件，并且开始编译这个文件。这时在状态窗口中会显示错误信息，单击状态窗口中错误信息行，或单击其左边的错误符号，光标将移到编辑器

中错误行的下面一行上(基本上所有 C 编译器都是这样)。

开始一个新的工程

从菜单 Project 中选择 New 命令,并且浏览至用户希望输出工程文件的目录。输出文件的名称取决于用户的工程文件名称。例如,若创建一个名称为 foo. prj 的工程,那么输出文件名称为 foo. hex 或 foo. cof 等。

自从创建自己的工程后,用户可以开始写源代码(C 或汇编格式),并且将这个文件加入到工程文件排列中。单击工具栏中 Build 图标,可以很容易地构筑这个工程。IDE 输出与 ATMEL的 AVR Studio 完全兼容 COFF 文件。用户可以使用 ATMEL 的 AVR Studio 来调试自己的代码。

为更容易地使用这个开发工具,可以使用应用程序向导来生成一些有关硬件的初始化代码。

9.3.2 C 程序的剖析

一个 C 程序必须定义一个 main 调用函数,编译器会将用户的程序与启动代码和库函数链接成一个"可执行"文件,因此,也可以在用户的目标系统中执行它。启动代码的用途在启动文件中很详细地被描述了。一个 C 程序需要设定目标环境,启动代码初始化这个目标使其满足所有的要求。

通常,用户的 main 例程完成一些初始化后,然后是无限循环地运行。作为例子,让我们看 \icc\examples 目录中的文件 led. c。

```
#include <io8515.h>
  /* 为了能够看清 LED 的变化图案,延时程序需要有足够的延时时间 */
void Delay()
{
    unsigned char a, b;
      for (a = 1; a; a++)
        for (b = 1; b; b++)
          ;
}

void LED_On(int i)
    {
    PORTB = ~BIT(i);              /* 低电平输出使 LED 点亮 */
    Delay();
    }

void main()4
    {
    int i;
    DDRB = 0xFF;                  /* 定义 B 口输出 */
    PORTB = 0xFF;                 /* B 口全部为高电平,对应 LED 熄灭 */
```

```
while (1)
    {
    /* LED 向前步进 */
    for (i = 0; i < 8; i++)
        LED_On(i);
    /* LED 向后步进 */
    for (i = 8; i > 0; i--)
        LED_On(i);
    /* LED 跳跃 */
    for (i = 0; i < 8; i += 2)
        LED_On(i);
    for (i = 7; i > 0; i -= 2)
        LED_On(i);
     }
}
```

整个 main 例程是很简单的，在初始化一些 I/O 寄存器之后，它运行在一个无限循环中，并且在改变 LED 的步进图案。LED 是在 LED_On 例程中被改变的。在 LED_On 例程中，直接写正确的数值到 I/O 端口。因为 CPU 运行很快，为能够看见图案变化，LED_On 例程调用了延时例程。因为实际延时值不能被确定，这一对嵌套循环只能给出近似延时时间；如果这个实际延时时间是重要的，那么这个例程应该使用硬件定时器来完成延时。

其它的例子，8515intr.c 程序很简单，但同样清楚地显示了如何用 C 语言写一个中断处理过程。这两个例子可以作为用户的程序的起点。

9.4 ICC AVR 的 IDE 环境

9.4.1 编译一个单独的文件

正常建立一个输出文件的次序是，首先应该建立一个工程文件并定义属于这个工程的所有文件。然而，有时也需要将一个文件单独地编译为目标文件或最终的输出文件。这时可以这样操作：从 IDE 菜单 File 中选择 Compile File... 命令，来执行 to Object 和 to Output 中的任意一个。当调用这个命令时，文件应该是打开的，并且在编辑窗口中是可以编辑的。

编译一个文件为目标文件(to Object)，对检查语法错误和编译一个新的启动文件是很有用的。编译一个文件为输出文件(to Output)，对较小的并是一个文件的程序较为有用。注意：这里使用默认的编译选项。

9.4.2 创建一个新的工程

为创建一个新的工程，从菜单 Project 中选择 New 命令，IDE 会弹出一个对话框；在对话框中用户可以指定工程的名称，这也是用户的输出文件的名称。如果用户使用一些已经建立的源文件，可在菜单 Project 中选择 AddFile(s)命令。

另外，可以在菜单 File 中选择 New 命令来建立一个新的源文件来输入代码，可以在菜单 File 中选择 Save 或 Save As 命令来保存文件。然后，可以像上面所述调用 AddFile(s)命令将

文件加入到工程中，也可在当前编辑窗口中右击选择 Add to Project 将文件加入已打开的工程列表中。在工程同一个目录中通常用户输出源文件，但也可以不作这样的要求。

工程的编译选项使用菜单中 Project 的 Options 命令。

9.4.3 工程管理

工程管理允许将多个文件组织在同一个工程中，而且定义它们的编译选项。这个特性允许将工程分解成许多小的模块。当处理工程构筑时，只有一个文件被修改和重新编译。如果一个头文件作了修改，当编译包含这个头文件的源文件时，IDE 会自动重新编译已经改变的头文件。

一个源文件可以写成 C 或汇编格式的任意一种。C 文件必须使用".c"扩展名，汇编文件必须使用".s"扩展名。可以将任意文件放在工程列表中，例如可以将一个工程文档文件放在工程管理窗口中，工程管理器在构筑工程时对源文件以外的文件不予理睬。

对目标器件不同的工程，可以在编译选项中设置有关参数。当新建一个工程时，使用默认的编译选项，可以将现有编译选项设置成默认选项，也可将默认编译选项装入现有工程中。默认编译选项保存在 default.prj 文件中。

为避免工程目录的混乱，可以指定输出文件和中间文件到一个指定的目录，通常这个目录是用户的工程目录的一个子目录。

9.4.4 编辑窗口

编辑窗口是与 IDE 交流信息的主要区域，在这个窗口中用户可以修改相应的文件。

当编译存在错误时，单击有关错误信息时，编辑器会自动将光标定位在错误行的位置。注意：对 C 源文件中缺少分号";"的错误，编辑器定位于其下面一行上。

9.4.5 应用构筑向导

应用构筑向导是用于创建外围设备初始化代码的一个图形界面。可以单击工具条中的 Wizard 按钮或菜单 Tools 中的 ApplicationBuilder 命令来调用它。

应用构筑向导使用编译选项中指定的目标 MCU 来产生相应的选项和代码。

应用构筑向导显示目标 MCU 的每一个外围设备子系统，它的使用是很显而易见的。在这里用户可以设置 MCU 所具有的中断、内存、定时器、I/O 端口、UART、SPI 和模拟量比较器等外围设备，并产生相应的代码。如果需要的话，还可产生 *main*( )函数。

9.4.6 状态窗口

状态窗口显示 IDE 的状态信息。

9.4.7 终端仿真

IDE 有一个内置的终端仿真器，注意它不包含任意一个 ISP(在系统编程)功能，但它可以作为一个简单的终端，或许可以显示目标装置的调试信息，也可下载一个 ASCII 码文件。

从 6.20 版本开始 IDE 加入了对 ISP 的支持。

9.5 C库函数与启动文件

9.5.1 启动文件

这个链接器会自动将启动文件连接到用户的程序之前，并将标准库 libcavr.a 与用户的程序相连接。启动文件根据目标 MCU 的不同在 crtavr.o 和 crtatmega.o 中间任意选择一个。启动文件定义了一个全局符号__start，它也是用户程序的起点。启动文件的功能有：

① 初始化硬件和软件堆栈指针。

② 从 idata 区拷贝初始化数据到直接寻址数据区 data 区。

③ 将 bss 区全部初始化为零。

④ 调用用户主例程 main 函数。

⑤ 定义一个退出点，如果用户的主函数 main( )一旦退出，它将进入这个退出点进行无限循环。

启动文件也定义了复位向量，用户不需要修改启动文件来使用别的中断，具体可参考中断操作部分。

为修改和使用新的启动文件：

```
cd \icc\libsrc.avr                          ; 进入安装的编译器路径
<edit crtavr.s>                             ; 编辑修改 crtavr.s 文件
<open crtavr.s using the IDE>               ; 用 IDE 打开 crtavr.s 文件
<Choose "Compile File To->Object">          ; 选择编译到的目标文件，创建一个新的 crtavr.o
copy crtavr.o ..\lib                        ; 拷贝到库目录
```

如果使用的目标 MCU 是 Mega，应该用 crtatmega 代替 crtavr。注意，Mega 的每一个中断入口地址使用两个字(word)，而非 Mega 芯片每一个中断入口地址使用一个字(word)。

用户可以有多个启动文件，可以在工程选项对话框中很方便地直接指定一个启动文件加入用户的工程中。注意，用户必须指定启动文件的绝对路径或启动文件必须位于工程选项库路径所指定的目录中。

9.5.2 常用库函数

(1) 库源代码

这个库源代码(缺省路径为 c:\icc\libsrc.avr\libsrc.zip)是一个密码保护的 ZIP 压缩文件。用户可以从互联网上任意下载一个 UNZIP 程序进行解压缩。当本软件被开锁后，密码显示在 About 对话框中。例如：

```
unzip-s libsrc.zip
; unzip  提示输入密码
```

(2) AVR 特殊函数

ICC AVR 有许多访问 UART、EEPROM 和 SPI 的特殊函数，堆栈检查函数对检测堆栈是否溢出很有用。另外，我们的互联网上有一个页专门存放用户写的源代码。

(3) io*.h (io2313.h，io8515.h，iom603.h，…)

这些文件是从 ATMEL 公司公开定义的 I/O 寄存器的源文件经过修改得到的，应该用这些文件来代替老的 avr. h 文件。

```
PORTB = 1;
uc = PORTA;
```

(4) macros. h

这个文件包含了许多有用的宏和定义。

(5) 其它头文件

下列标准的 C 头文件是被支持的。如果用户的程序使用了头文件所列出的函数，那么包含头文件是一个好习惯。在使用浮点数和长整型数的程序中，必须用 #include 预编译指令，包含了这些函数原型的头文件。读者可参考返回非整型值的函数。

assert. h — assert()，声明宏。

ctype. h-— 字符类型函数。

float. h —浮点数原型。

limits. h——数据类型的大小和范围。

math. h—— 浮点运算函数。

stdarg. h— 变量参数表。

stddef. h— 标准定义。

stdio. h- —标准输入/输出(I/O)函数。

stdlib. h——包含内存分配函数的标准库。

string. h——字符串处理函数。

9.5.3 字符类型库

下列函数按照输入的 ACSII 字符集字符分类，使用这些函数之前应当用 #include <ctype. h>包含。

1. int isalnum(int c)

如果 c 是数字或字母则返回非零数值，否则返回零。

2. int isalpha(int c)

如果 c 是字母则返回非零数值，否则返回零。

3. int iscntrl(int c)

如果 c 是控制字符(如 FF、BELL、LF 等)则返回非零数值，否则返回零。

4. int isdigit(int c)

如果 c 是数字则返回非零数值，否则返回零。

5. int isgraph(int c)

如果 c 是一个可打印字符而非空格则返回非零数值，否则返回零。

6. int islower(int c)

如果 c 是小写字母则返回非零数值，否则返回零。

7. int isprint(int c)

如果 c 是一个可打印字符则返回非零数值，否则返回零。

8. int ispunct(int c)

如果 c 是一个可打印字符而不是空格、数字或字母则返回非零数值，否则返回零。

9. int isspace(int c)

如果 c 是一个空格字符则返回非零数值，包括空格 CR、FF、HT、NL 和 VT，否则返回零。

10. int isupper(int c)

如果 c 是大写字母则返回非零数值，否则返回零。

11. int isxdigit(int c)

如果 c 是十六进制数字则返回非零数值，否则返回零。

12. int tolower(int c)

如果 c 是大写字母则返回 c 对应的小写字母，其它类型仍然返回 c。

13. int toupper(int c)

如果 c 是小写字母则返回 c 对应的大写字母，其它类型仍然返回 c。

9.5.4 浮点运算库

下列函数支持浮点数运算，使用这些函数之前必须用 # include <math. h>包含。

1. float asin(float x)

以弧度形式返回 x 的反正弦值。

2. float acos(float x)

以弧度形式返回 x 的反余弦值。

3. float atan(float x)

以弧度形式返回 x 的反正切值。

4. float atan2(float x, float y)

返回 y/x 的反正切，其范围在 $-\pi \sim +\pi$ 之间。

5. float ceil(float x)

返回对应 x 的一个整型数，小数部分四舍五入。

6. float cos(float x)

返回以弧度形式表示的 x 的余弦值。

7. float cosh(float x)。

返回 x 的双曲余弦函数值。

8. float exp(float x)

返回以 e 为底的 x 的幂，即 e^x。

9. float exp10(float x)

返回以 10 为底的幂，即 10^x。

10. float fabs(float x)

返回 x 的绝对值。

11. float floor(float x)

返回不大于 x 的最大整数。

12. float fmod(float x, float y)

返回 x/y 的余数。

13. float frexp(float x, int * pexp)

把浮点数 x 分解成数字部分 y(尾数)和以 2 为底的指数 n 两个部分,即 $x=y\times 2^n$,y 的范围为 $0.5\leqslant y<1$,y 值被函数返回,而 n 值存放到 pexp 指向的变量中。

14. float fround(float x)

返回最接近 x 的整型数。

15. float ldexp(float x, int exp)

返回 $x\times 2^{exp}$。

16. float log(float x)

返回 x 的自然对数。

17. float log10(float x)

返回以 10 为底的 x 的对数。

18. float modf(float x, float * pint)

把浮点数分解成整数部分和小数部分,整数部分存放到 pint 指向的变量,小数部分应当大于或等于 0 而小于 1,并且作为函数返回值返回。

19. float pow(float x, float y)

返回 x^y 值。

20. float sqrt(float x)

返回 x 的平方根。

21. float sin(float x)

返回以弧度形式表示的 x 的正弦值。

22. float sinh(float x)

返回 x 的双曲正弦函数值。

23. float tan(float x)

返回以弧度形式表示的 x 的正切值。

24. float tanh(float x)

返回 x 的双曲正切函数值。

9.5.5 标准输入/输出库

标准的文件输入/输出是不能真正植入微控制器(MCU)的,标准 stdio.h 的许多内容不可以使用,不过有一些 I/O 函数是被支持的,同样使用之前应用 #include <stdio.h> 预处理,并且需要初始化输出端口。最低层的 I/O 程序是单字符的输入(getchar)和输出(putchar)程序,如果针对不同的装置使用高层的 I/O 函数,例如用 printf 输出 LCD,需要全部重新定义最底层的函数。

为在 ATMEL 的 AVR Studio 模拟器(终端 I/O 窗口)使用标准 I/O 函数,应当在编译选项中选中相应的单选钮。

注意:作为缺省,单字符输出函数 putchar 是输出到 UART 装置没有修改,无论如何为使输出能如期望的那样出现在程序终端窗口中,'\n'字符必须被映射为成对的回车和换行(CR/LF)。

int getchar( )

使用查寻方式从 UART 返回一个字符。

int printf(char * fmt, ..)

按照格式说明符输出格式化文本 frm 字符串，格式说明符是标准格式的一个子集。

%d 输出有符号十进制整数。

%o ——输出无符号八进制整数。

%x——输出无符号十六进制整数。

%X 除了大写字母使用 A～F 外，同 %x。

%u——输出无符号十进制整数。

%s——输出一个以 C 中空字符 NULL 结束的字符串。

%c——以 ASCII 字符形式输出，只输出一个字符。

%f——以小数形式输出浮点数。

%S——输出在 Flash 存储器中的字符串常量。

printf 支持三个版本，取决于用户的特别需要和代码的大小(越高的要求，代码越大)。

基本形：只有 %c，%d，%x，%u，和 %s 格式说明符是承认的。

长整形：针对长整形数的修改 %ld，%lu，%lx 被支持，以适用于精度要求较高的领域。

浮点形：全部格式包括%f 被支持。

用户使用编译选项对话框来选择版本，代码大小的增加是值得关注的。

① int putchar(int c)

输出单个字符。这个库程序使用了 UART 以查寻方式输出单个字符，注意输出'\n'字符至程序终端窗口。

② int puts(char * s)

输出以 NL 结尾的字符串。

③ int sprintf(char * buf, char * fmt)

按照格式说明符输出格式化文本 frm 字符串到一个缓冲区，格式说明符同 printf( )。

④ const char * 支持功能

cprintf 和 csprintf 是将 Flash 中的格式字符串分别以 prinf 和 sprinf 形式输出。

9.5.6 标准库和内存分配函数

标准库头文件<stdlib.h>定义了宏 NULL、RAND_MAX 和新定义的类型 size_t，并且描述了下列函数。注意在调用任意内存分配程序(比如 calloc、malloc 和 realloc)之前，必须调用_NewHeap 来初始化堆 heap。

1. int abs(int i)

返回 i 的绝对值。

2. int atoi(char * s)

转换字符串 s 为整型数并返回它，字符串 s 起始必须是整型数形式字符，否则返回 0。

3. double atof(const char * s)

转换转换字符串 s 为双精度浮点数并返回它，字符串 s 起始必须是浮点数形式字符串。

4. long atol(char * s)

转换字符串 s 为长整型数并返回它，字符串 s 起始必须是长整型数形式字符，否则返回 0。

5. void * calloc(size_t nelem, size_t size)

分配 nelem 个数据项的内存连续空间，每个数据项的大小为 size 字节并初始化为 0。如果分配成功则返回分配内存单元的首地址，否则返回 0。

6. void exit(status)

终止程序运行，典型的是无限循环，它是担任用户 main 函数的返回点。

7. void free(void * ptr)

释放 ptr 所指向的内存区。

8. void * malloc(size_t size)

分配 size 字节的存储区，如果分配成功则返回内存区地址，如内存不够分配则返回 0。

9. void _NewHeap(void * start, void * end)

初始化内存分配程序的堆。一个典型的调用是将符号_bss_end+1 的地址用做 start 值，符号_bss_end 定义为编译器用来存放全局变量和字符串的数据内存的结束，加 1 的目的是堆栈检查函数使用_bss_end 字节存储为标志字节，这个结束值不能被放入堆栈中。

```
extern char _bss_end;
_NewHeap(&_bss_end+1, &_bss_end + 201);        // 初始化 200 字节大小的堆
```

10. int rand(void)

返回--个在 0 和 RAND_MAX 之间的随机数。

11. void * realloc(void * ptr, size_t size)

重新分配 ptr 所指向内存区的大小为 size 字节，size 可比原来大或小，返回指向该内存区的地址指针。

12. void srand(unsigned seed)

初始化随后调用的随机数发生器的种子数。

13. long strtol(char * s, char * * endptr, int base)

按照 base. 的格式转换 s 中起始字符为长整型数。如果 endptr 不为空，则 * endptr 将设定 s 中转换结束的位置。

14. unsigned long strtoul(char * s, char * * endptr, int base)

除了返回类型为无符号长整型数外，其余同 strtol。

9.5.7 字符串函数

用 # include <string. h>预处理后，编译器支持下列函数。<string. h>定义了 NULL、类型 size_t 和下列字符串及字符阵列函数。

1. void * memchr(void * s, int c, size_t n)

在字符串 s 中搜索 n 个字节长度寻找与 c 相同的字符，如果成功则返回匹配字符的地址指针，否则返回 NULL。

2. int memcmp(void * s1, void * s2, size_t n)

对字符串 s1 和 s2 的前 n 个字符进行比较。如果相同，则返回 0；如果 s1 中字符大于 s2 中字符，则返回 1；如果 s1 中字符小子 s2 中字符，则返回－1。

3. void * memcpy(void * s1, void * s2, size_t n)

拷贝 s2 中 n 个字符至 s1，但拷贝区不可以重叠。

4. void * memmove(void * s1, void * s2, size_t n)

拷贝 s2 中 n 个字符至 s1,返回 s1,其与 memcpy 基本相同,但拷贝区可以重叠。

5. void * memset(void * s, int c, size_t n)

在 s 中填充 n 个字节的 c,它返回 s。

6. char * strcat(char * s1, char * s2)

拷贝 s2 到 s1 的结尾,返回 s1。

7. char * strchr(char * s, int c)

在 s1 中搜索第一个出现的 c,包括结束 NULL 字符。如果成功,返回指向匹配字符的指针;如果没有匹配字符找到,则返回空指针。

8. int strcmp(char * s1, char * s2)

比较两个字符串。如果相同,则返回 0;如果 s1>s2,则返回 1;如果 s1<s2,则返回 -1。

9. char * strcpy(char * s1, char * s2)

拷贝字符串 s2 至字符串 s1,返回 s1。

10. size_t strcspn(char * s1, char * s2)

在字符串 s1 搜索与字符串 s2 匹配的第一个字符,包括结束 NULL 字符,其返回 s1 中找到的匹配字符的索引。

11. size_t strlen(char * s)

返回字符串 s 的长度,不包括结束 NULL 字符。

12. char * strncat(char * s1, char * s2, size_t n)

拷贝字符串 s2(不含结束 NULL 字符)中 n 个字符到 s1。如果 s2 长度比 n 小,则只拷贝 s2,返回 s1。

13. int strncmp(char * s1, char * s2, size_t n)

基本和 strcmp 函数相同,但其只比较前 n 个字符。

14. char * strncpy(char * s1, char * s2, size_t n)

基本和 strcpy 函数相同,但其只拷贝前 n 个字符。

15. char * strpbrk(char * s1, char * s2)

基本和 strcspn 函数相同,但它返回的是 s1 匹配字符的地址指针,否则返回 NULL 指针。

16. char * strrchr(char * s, int c)

在字符串 s 中搜索最后出现的 c,并返回它的指针,否则返回 NULL。

17. size_t strspn(char * s1, char * s2)

在字符串 s1 搜索与字符串 s2 不匹配的第一个字符,包括结束 NULL 字符,其返回 s1 中找到的第一个不匹配字符的索引。

18. char * strstr(char * s1, char * s2)

在字符串 s1 中找到与 s2 匹配的子字符串,如果成功它则返回 s1 中匹配子字符串的地址指针,否则返回 NULL。

19. const char * 支持函数

这些函数除了它的操作对象是在 Flash 中常数字符串外,其余同 c 中的函数。

```
size_t cstrlen(const char * s)
char * cstrcpy(char * dst, const char * src);
```

```
int cstrcmp(const char *s1, char *s2);
```

9.5.8 变量参数函数

<stdarg.h>提供再入式函数的变量参数处理，它定义了不确定的类型 va_list 和三个宏。

1. va_start(va_list foo, <last-arg>)

初始化变量 foo。

2. va_arg(va_list foo, <promoted type>)

访问下一个参数，分派指定的类型。注意，那个类型必须是高级类型，如 int、long 或 double，小的整型类型如 char 不能被支持。

3. va_end(va_list foo)

结束变量参数处理。

例如，printf()可以使用 vfprintf()来实现。

```
#include <stdarg.h>
int printf(char *fmt, ...)
    {
    va_list ap;
va_start(ap, fmt);
    vfprintf(fmt, ap);
    va_end(ap);
    }
```

9.5.9 堆栈检查函数

有几个库函数是用于检查堆栈是否溢出，内存图如图 9.2 所示。如果硬件堆栈增长到软件堆栈中，那么软件堆栈的内容将会被改变，也就是说局部变量和别的堆栈项目被改变。硬件堆栈是用做函数的返回地址，如果用户的函数调用层次太深偶然会发生这种情况。

同样地，软件堆栈溢出、进数据区域，将会改变全局变量或其它静态分配的项目(如果用户使用动态分配内存，还会改变堆项目)。这种情况在用户定义了太多的局部变量或一个局部集合变量太大也会偶然发生。

启动代码写了一个正确的关于数据区的地址字节和一个类似的正确的关于软件堆栈的地址字节作为警戒线。(注意：如果用户使用了自己的启动文件，而其又是以 6.20 版本之前的启动文件为基础的，将需要额外改造为新的启动文件。)

注意：如果用户使用动态分配内存，必须跳过警戒线字节_bss_end 来分配用户的堆。参考内存分配函数。

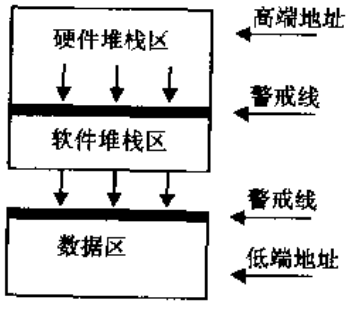


图 9.2 警戒线

(1) 堆栈检查

用户调用_StackCheck(void)函数来检查堆栈溢出：如果警戒线字节仍然保持正确的值，那么函数检查通过；如果堆栈溢出，那么警戒线字节将可能

被破坏。

注意，当程序堆栈溢出的时候，程序将可能运行不正常或偶然崩溃。当_StackCheck 检查错误条件时，它调用了带一个参数的函数_StackOverflowed(char c)。如果参数是1，那么硬件堆栈有过溢出；如果参数是0，那么软件堆栈曾经溢出。在那个例子中制造了两个功能调用，它是两个堆栈都可能溢出的。无论如何，在_StackOverflowed 执行起作用时，第二个调用不可以出现。作为例子，如果函数复位了 CPU，那么将不能返回_StackCheck 函数。

(2) 缺省的 _StackOverflowed 函数

当它被调用时，库会用一个缺省的_StackOverflowed 函数来跳转到 0 的位置，因此复位 CPU 和程序。用户可能希望用一个函数来代替它以指示更多的错误条件，作为一个例子，它可能切断所有的中断并且点亮 LED。注意自堆栈溢出指示故障程序以来，_StackOverflowed 函数或许不能执行任何太复杂的事或实现程序的正常工作。

这两个函数的原型在头文件 macros. h. 中。

9.6 AVR 硬件访问的编程

9.6.1 访问 AVR 的底层硬件

AVR 系列使用高级语言编程时有很高的 C 语言密度，它允许用户对访问目标 MCU 的底层硬件进行访问。由于 AVR 性能，除了要最大程序地优化代码外很少使用汇编。偶然情况下目标 MCU 的硬件特点在 C 语言中不能很好地使用，很显然使用在线汇编和预处理宏能访问这些特点。

头文件 io * . h(如 io8515. h、iom603. h 等)定义了指定 AVR MCU 的 I/O 寄存器细节。这些文件是从 ATMEL 公司发布的文件经过修改，以匹配这个编译器的语法要求。文件 macros. h定义了许多有用的宏，例如宏 UART_TRANSMIT_ON( )能使 UART 开始工作。

这个编译器的效率很高，当访问由 I/O 寄存器映射的内存时，能产生单周期指令，像 in、out、sbis、sbi 等。参考 I/O 寄存器。

注意：老的头文件 avr. h 定义 I/O 寄存器的 bit 有一些模糊，尽管 io * . h 定义了它们的位(bit)的位置。因此使用 io * . h 和 I/O 寄存器的 bit，很多时候将需要使用定义在 macros. h 文件中的 BIT( )宏。例如：

```
avr. h:
    # define SRE  0x80              // 外部 RAM 使能
  ...(你的 C 程序)
   MCUCR |= SRE;
  io8515. h
  # define SRE  7
   ...(你的 C 程序)
   # include <macros. h>
    MCUCR |= BIT(SRE);
```

9.6.2 位操作

一个共同的任务是编程微控制器 MCU 打开或关闭 I/O 寄存器的一些位(bit)。很幸运，

标准 C 有较好的和适用的位操作功能，而没有借助于汇编指令或其它非标准 C 结构。C 定义了一些按位进行的运算是很有用的。

① a|b- —按位或。这个运算中 a 被表达式中的 b 按位进行或运算。这惯用于打开某些位，尤其常用|=的形式。例如：

```
PORTA |= 0x80;      // 打开位 7 (最高位)
```

② a&b- -按位与。这个运算在检查某些位是否置 1 时有用。例如：

```
If ((PORTA & 0x81) == 0)      // 检查位 7 和位 0
```

注意，圆括号需要括在 & 运算符的周围，因为它和 == 相比运算优先级较低。这是 C 程序中很多错误的原因之一。

③ a^b— 按位异或。这个运算对一个位取反有用。例如，位 7 是被翻转的：

```
PORTA^=0x80;      // 翻转位 7
```

④ ~a —按位取反。这个运算在表达式中执行一个取反。当用按位与运算关闭某些位时，与这个运算组合使用尤其有用。例如：

```
PORTA &= ~0x80;      // 关闭位 7
```

这个编译器对这些运算能产生最理想的机器指令。例如：sbic 指令可以用在根据位的状态进行条件分支的按位与运算中。

9.6.3 程序存储器和常量数据

AVR 是哈佛结构的 MCU，它的程序存储器和数据存储器是分开的。这样的设计是有一些优点的。例如，分开的地址空间允许 AVR 装置比传统结构访问更多的存储器；例如，Atmega系列允许有超过 64K 字(word)的程序存储器和 64K 字节的数据存储器。将来的 MCU 装置可能用到更多的程序存储器，而程序计数器仍保留在 16 位上。

不幸的是，C 不是在这种机器上发明的。特别地，C 指针是任意一个数据指针或函数指针，C 规则已经指定用户不可以假设数据和函数指针能被向前和向后修改。可是同是哈佛结构的 AVR，要求数据指针能指向任一个数据内存和程序内存。

非标准 C 解决了这个问题，ImageCraft AVR 编译器使用 const 限定词表示项目是在程序存储器中。注意对指针描述，这个 const 限定词可以应用于不同的场合，不管是限定指针变量自己还是指向项目的指针。例如：

```
const int table[] = { 1, 2, 3 };
const char * ptr1;
char * const ptr2;
const char * const ptr3;
```

table 是表格式样分配进程序存储器；ptr1 是一个项目在数据存储器，而指向数据的指针在程序存储器；ptr2 是一个项目在程序存储器，而指向数据的指针在数据存储器；最后，ptr3 是项目在程序存储器，而指向数据的指针也在程序存储器。在大多数的例子中 table 和 ptr1 是很典型的。C 编译器生成 LPM 指令来访问程序存储器。

注意,C标准不要求const数据是放入只读存储器中,而且在传统结构中,除了正确访问就没有要紧的了。因而,在承认参数的C标准中使用const限定是非传统的。无论如何,这样做与标准C函数定义是有一定冲突的。

例如,标准strcpy的原型是strcpy(char *dst, const char *src),带有const限定的第二个参数表示函数不能修改参数。然而在ICC AVR下,const限定词表示第二个参数指向程序存储器是不合适的。因此,这些函数定义没有const限制。

最后,注意只有常数变量以文件存储类型放入Flash中。例如定义在函数体外的变量或有静态存储类型限制的变量。如果使用有const限制的局部变量,将不被放入Flash中而可能导致有否结果不明确。

9.6.4 字符串

在哈佛结构的AVR中,程序内存和数据内存的分开给程序内存和数据内存的说明带来了一定的复杂性。

这个编译器将带有const说明的表和项目放入程序存储器中,最困难的是字符串的分配和处理,问题在于C中将字符串转换为char指针。如果字符串是分配进程序存储器中,那么所有字符串库函数中的任意一个必须被复制成不同于指针的操作,或者字符串也必须被分配在数据存储器中。

ImageCraft编译器提出了解决这个问题的两个方法:

(1) 缺省的字符串分配

这个缺省的方法是同时分配字符串在数据和程序存储器中,所有涉及的字符串是拷贝进数据存储器的。为了确保它们的值是正确的,在程序启动时字符串是由程序存储器拷贝进数据存储器中的,因此只有单一的字符串拷贝函数是必须的(编译器执行全局变量初始化也是这样处理的)。

如果用户希望节省空间,要能使用常量字符型数组来将字符串只分配进程序存储器中。例如:

```
const char hello[] = "Hello World";
```

在这个例子中,hello可以在上下文中作为字符串使用,但不能用做标准C库中字符串函数的参数。

Printf已被扩展成带%S格式字符来输出只存储于Flash中字符串;另外,新的字符串函数已加入了对只存储于Flash中字符串的支持。

(2) 只分配全部字符串到Flash存储器中

当对应Project→Options→Target→Strings In FLASH Only检查框被选中时,可以指挥编译器将字符串只放在Flash中,这时必须很小心地调用库函数。当这个选项是选中的,字符串类型const char*是有效的,并且必须保证函数获得了合适的参数类型。除了新的const char*与字符串有关系外,创建了cprintf和csprintf函数承认字符串格式的类型。参考标准输入/输出函数。

注意:当选项2(只分配全部字符串到Flash存储器中)时,应使用cprintf()。对const char*及const char ptr[]类型字符串,并且加%S参数。

当选项 1 时，应使用 printf()。对 const char * 及 const char ptr[ ]类型字符串，并且加%S参数。

9.6.5 堆 栈

生成代码使用两个堆栈：一个是用于子程序调用和中断操作的硬件堆栈，一个是用于以堆栈结构传递的参数、临时变量和局部变量的软件堆栈。

硬件堆栈起初是用于存储函数返回的地址，它代表了许多小的软件堆栈。通常，如果程序没有子程序调用，也不调用像带有%f 格式的 printf()等库函数，那么默认的 16 字节应该在大多数的例子中能良好工作。在绝大多数程序中，除了很繁重的递归调用程序(再入式函数)，最多 40 个字节的硬件堆栈应该是足够的。

硬件堆栈是从数据内存的顶部开始分配的，而软件堆栈是在它下面一定数量字节处分配。硬件堆栈和数据内存的大小是受在编译器选项中的目标装置项设定限制的。数据区从 0x60 开始分配，在 I/O 空间后面是正确的。允许数据区和软件堆栈彼此相向生长。

如果选择的目标装置带有 32K 或 64K 的外部 SRAM，那么堆栈是放在内部 SRAM 的顶部，而且向低内存地址方向生长。参考程序和数据内存的使用。

堆栈检查

任意一个程序失败的重要原因是堆栈溢出到其它数据内存的范围。两个堆栈中的任意一个都可能溢出，并且当一个堆栈溢出时会偶然产生坏的事情。可以使用堆栈检查函数检测溢出情况。

9.6.6 在线汇编

除了在汇编文件中写汇编函数外，在线汇编允许写汇编代码进用户的 C 文件中(当然，在用户的工程使用汇编源文件作为一个部件是良好的)。在线汇编的语法是：

```
asm("<string>");
```

多个汇编声明可以被符号\n 分隔成新的一行，String 可以被用来指定多个声明，除了额外增加的 ASM 关键词。为了在汇编声明中访问一个 C 的变量，可使用%<变量名>格式。例如：

```
register unsigned char uc;
    asm("mov %uc,R0\n"
        "sleep\n");
```

任意一个 C 变量都可以被引用。如果在汇编指令中需使用一个 CPU 寄存器，必须使用寄存器存储类(register)来强制分配一个局部变量到 CPU 寄存器中。

通常，使用在线汇编引用局部寄存器的能力是有限的。如果用户在函数中描述了太多的寄存器变量，就很可能没有寄存器可用。在这种情况，用户将从汇编程序得到一个错误。那时也不能控制寄存器变量的分配，所以用户的在线汇编指令很可能失败。作为例子，使用 LDI 指令需要使用 R16～R31 中的一个寄存器，但这里没有请求使用在线汇编，同样也没有引用上半部分的整数寄存器。

在线汇编可以被用在C函数的内部或外部，编译器将在线汇编的每行都分解成可读的。不像AVR汇编器，ImageCraft汇编器允许标签放置在任意地方，所以可以在用户的在线汇编代码中创建标签。当汇编声明在函数外部时，用户可能得到一个警告，可以不理睬这个警告。

9.6.7 I/O寄存器

I/O寄存器，包括状态寄存器SREG，可以被两条路线访问。I/O地址在0x00和0x3f之间，可以使用IN和OUT指令读写I/O寄存器；或者使用在0x20和0x5F之间的数据内存地址，可以使用普通数据访问指令和地址模式。两种方法在C中都可使用：

数据内存地址，一个直接地址可以通过加指针类型符号直接访问。例如，SREG的数据内存在地址是0x5F：

```
unsigned char c = *(volatile unsigned char *)0x5F;          // 读SREG
*(volatile unsigned char *)0x5F |= 0x80;                    // 打开全局断位
```

注意：数据内存地址0到31涉及到CPU寄存器，不要改变CPU寄存器。

当访问在I/O寄存器范围中的数据内存时，编译器自动生成低级指令，像*in*、out、sbrs、sbrc等是首选的方法。

I/O地址可以使用在线汇编和预处理宏来访问：

```
register unsigned char uc;
asm("in %uc,$3F");              // 读SREG
asm("out $3F,%uc");             // 打开全局中断位
```

注意：老的头文件avr.h定义I/O寄存器的bit有一些模糊，尽管io*.h定义了它们的bit的位置。因此，使用io*.h和I/O寄存器的bit，很多时候将需要使用定义在macros.h文件中的BIT()宏。例如：

```
avr.h:
#define SRE   0x80                // 外部RAM使能
...(填充C程序)
    MCUCR |= SRE;
io8515.h
#define SRE          7
...(填充C程序)
#include <macros.h>
MCUCR |= BIT(SRE);
```

9.6.8 绝对内存地址

程序可能需要使用绝对内存地址，例如外部I/O设备通常被映射成特殊的内存。这些可能包括LCD界面和双口SRAM，通常可以使用在线汇编或单独的汇编文件来描述那些定位在特殊内存地址的数据。在稍后版本的编译器中，已在C语言中提供这些能力。

在下面的例子中，假设有一个两字节的LCD控制寄存器定位在0x1000地址，一个两字节的LCD数据寄存器定位在0x1002地址，并且有一个100字节的双口SRAM定位在0x2000

的地址。使用汇编模式，在一个汇编文件中输入以上内容。

```
.area memory(abs)
.org 0x1000
_LCD_control_register:: .blkw 1
_LCD_data_register:: .blkw 1
.org 0x2000
_dual_port_SRAM:: .blkb 100
```

在 C 文件中必须这样描述：

```
extern unsigned int LCD_control_register, LCD_data_register;
extern char dual_port_SRAM[100];
```

注意：界面规定在汇编文件中外部变量名称是带'_'前坠的，并且使用两个冒号定义为全局变量。

使用在线汇编：在线汇编遵守同样的汇编语法规则，除了它被附加一个 asm( )伪函数。在 C 文件中，关于上面的汇编代码被变为如下代码：

```
asm(".area memory(abs)"
    ".org 0x1000"
    "_LCD_control_register:: .blkw 1"
    "_LCD_data_register:: .blkw 1");
asm(".org 0x2000"
    "_dual_port_SRAM:: .blkb 100");
```

在 C 中用户仍然要使用 extern 描述变量，正像上面使用单独的汇编文件那样，否则 C 编译器不会真正知道在 asm 中的声明。

9.6.9 C 任务

作为汇编界面的描述和调用规则，编译器通常用生成代码来保存和恢复保护的寄存器。在一些情况下，这些行为可能是不合适的。例如，若使用 RTOS(实时操作系统)，RTOS 管理着寄存器的保存和恢复并作为任务切换处理的一部分，编译器若再插入这些代码就变得多余了。

为了禁止这种行为，可以使用 #pragma ctask。例如：

```
#pragma ctask drive_motor emit_siren
....
void drive_motor() { ... }
void emit_siren() {...}
```

这个附注(pragma)必须被用在函数定义之前。注意作为默认的情况，从不返回的程序 main 是有这个属性的，它也没有必要为返回保存和恢复任意一个寄存器。

9.6.10 中断操作

(1) C 中断操作

C 中可以使用中断操作。无论函数定义在文件的什么地方,必须用一个附注(pragma)在函数定义之前通知编译器这个函数是一个中断操作:

```
#pragma interrupt_handler <name>:<vector number> *
```

vector number 是中断的向量号。向量号是从 1 开始的,它是复位向量。这个附注有两个作用:对中断操作函数,编译器生成 RETI 指令代替 RET 指令,而且保存和恢复在函数中用过的全部寄存器;编译器生成以向量号和目标 MCU 为基础的中断向量。例如:

```
#pragma interrupt_handler      timer_handler:4
    ...
void timer_handler()
 {
  ...
 }
```

编译器生成的指令为:

```
rjmp _timer_handler            ; 对普通 AVR   MCU
```

或者

```
jmp _timer_handler             ;对 Mega   MCU
```

上述指令定位在 0x06(字节地址,针对普通装置)和 0x0c(字节地址,针对 Mega 装置)。Mega 使用 2 个字作为中断向量,非 Mega 使用 1 个字作为中断向量。

如果希望对多个中断入口使用同一个中断操作,可以在一个 interrupt_handler 附注中放置多个用空格分开的名称,分别带有多个不同的向量号。例如:

```
#pragma interrupt_handler timer_ovf:7 timer_ovf:8
```

(2) 汇编中断操作

可以用汇编语言写中断操作。如果在汇编操作内部调用 C 函数,一定要小心。汇编程序要保存和恢复挥发寄存器(参考汇编界面),C 函数不做这些工作。

如果使用汇编中断操作,那么必须自己定义向量。使用 abs 属性描述绝对区域,用.org 来声明 rjmp 或 jmp 指令的正确地址。注意,这个.org 声明使用的是字节地址。

```
; 对全部除 Atmega 以外的 MCU
 .area vectors(abs)                ;中断向量
.org 0x6
rjmp _timer
; 对 ATMega MCU
.area vectors(abs)                 ; 中断向量
.org 0xC
```

```
jmp _timer
```

9.6.11 访问 UART

默认的库函数 getchar 和 putchar 使用查寻模式从 UART 中进行读写。在\icc\examples.avr 目录,有一个以中断方式工作的 I/O 程序可以代替默认的程序。

9.6.12 访问 EEPROM

EEPROM 在运行时可以使用库函数访问,在调用这些函数之前加入 #include <eeprom.h>。

EEPROM_READ(int location, object)

这个宏调用了 EEPROMReadBytes 函数从 EEPROM 指定位置读取数据送给数据对象,object 可以是任意程序变量包括结构和数组。例如:

```
int i;
EEPROM_Read(0x1, i);                    // 读 2 个字节给 i
EEPROM_WRITE(int location, object)
```

这个宏调用了 EEPROMWriteBytes 函数将数据对象写入到 EEPROM 的指定位置,object可以是任意程序变量包括结构和数组。例如:

```
int i;
EEPROM_WRITE(0x1, i);                   //写 2 个字节至 0x1
```

这些宏和函数可以用于任意 AVR 装置。可是对 EEPROM 单元少于 256 字节的 MCU,即使不需要高地址字节它们也是欠佳的,因为它仍然是要写的。如果它关系重大,用户可以为 EEPROM 较少的目标装置重新编译库源代码。

(1) 初始化 EEPROM

EEPROM 可以在用户的程序源文件中初始化,在 C 源文件中它作为一个全局变量被分配到特殊调用区域 eeprom. 中的。这是可以用附注实现的,结果是产生扩展名为. eep 的输出文件。例如:

```
#pragma data:eeprom
int foo = 0x1234;
char table[] = { 0, 1, 2, 3, 4, 5 };
#pragma data:data
...
int i;
EEPROM_READ((int)&foo, i);              // i 等于 0x1234
```

第二个附注是必须的,为返回默认的 data. 区域需要重设数据区名称。

注意,因为 AVR 的硬件原因,初始化 EEPROM 数据至 0 地址是不可以使用的。

注意,当使用外部描述(比如访问在另一个文件中的 foo),用户不需要加入这个附注。例如:

```
extern int foo;
```

```
int i;
EEPROM_READ((int)&foo, i);
```

(2) 内部函数

如果需要下列函数可以直接使用,但是上面关于宏的描述对大多数装置应该是有能力的。

① unsigned char EEPROMread(int location)

从 EEPROM 指定位置读取一个字节。

② int EEPROMWrite(int location, unsigned char byte)

写一个字节到 EEPROM 指定位置,如果成功返回 0。

③ void EEPROMReadBytes(int location, void * ptr, int size)

从 EEPROM 指定位置处开始读取 size 个字节至由 ptr. 指向的缓冲区。

④ void EEPROMWriteBytes(int location, void * ptr, int size)

从 EEPROM 指定位置处开始写 size 个字节,写的内容由 ptr. 指向的缓冲区提供。

9.6.13 访问 SPI

一个以查寻模式访问 SPI 的函数是提供的,更多的信息参考 spi. h。

9.6.14 相对转移/调用的地址范围

一个带 8K 程序存储器的装置,全部范围内的跳转可以使用相对转移和调用指令(rjmp 和 rcall)。为实现这个目的,相对转移和调用的范围是以 8K 为分界的。例如,一个较远的跳转,跳转到 0x2100 字节处(0x2000 为 8K)实际上会跳转到地址 0x100 处。

这个选项是由工程管理器自动检测的,只要目标装置的程序存储器是 8K 的。

9.6.15 C 的运行结构

数据类型见表 9.1。

表 9.1 数据类型

类 型	长度/B	范 围
unsigned char	1	0～256
signed char	1	−128～127
char *	1	0～256
unsigned short	2	0～65 535
(signed) short	2	−32 768～32 767
unsigned int	2	0～65 535
(signed) int	2	−32 768～32 767
unsigned long	4	0～4 294 967 295
(signed) long	4	−2 147 483 648～2 147 483 647
float	4	+/−1.175e−38～3.40e+38
double	4	+/−1.175e−38～3.40e+38

* char 等同于 unsigned char。

floats 和 doubles 是 IEEE 标准 32 位格式，7 位表示指数，23 位表示尾数，1 位表示符号。

位域类型必须被赋予 unsigned 或 signed 关键字，而且将被包含在一个较小的空间中。如可定义成结构：

```
struct {
        unsigned a : 1, b : 1;
        };
```

这个结构体的长度只有 1 个字节。位域是从右往左放置的。

9.6.16 汇编界面和调用规则

(1) 名称

在汇编文件中 C 语言中的名称是以下划线为前坠的。如函数 main( )在汇编模块中是以 _main( )引用的。名称的有效长度为 32 个字符，在名称后面加两个冒号(::)，可以定义成一个全局变量。例如：

```
  _foo::
      .word 1
(在 C 文件中)
  extern int foo;
```

(2) 传递参数和返回值所使用的寄存器

若第一个参数是整型，则通过 R16/R17 传递；第二个参数是整型，则通过 R18/R19 传递。如果参数是长整型或浮点数，则通过 R16/R17/R18/R19 传递；其余参数通过软件堆栈传递。比整型参数小的(如 char)参数扩展成整型(int)长度传递，即使函数原型也是可用的。如果 R16/R17 已传递了第一个参数，而第二个参数是长整型或浮点数，则第二个参数的低半部分通过 R18/R19 传递，而高半部分通过软件堆栈传递。

整型返回值是通过 R16/R17 返回，而长整型或浮点数返回值则通过 R16/R17/R18/R19 返回。

(3) 保护的寄存器

在汇编函数中必须保护和恢复下列寄存器：

R28/R29 或 Y，这是结构指针。

R10/R11/R12/R13/R14/R15/R20/R21/R22/R23，这些寄存器是调用保护寄存器。这些寄存器的内容在被汇编语言函数调用后必须保持不变。

(4) 挥发寄存器

R0/R1/R2/R3/R4/R5/R6/R7/R8/R9/R24/R25/R26/R27/R30/R31，这些寄存器是调用挥发寄存器。这些寄存器可以在汇编语言函数中使用，而不被保护和恢复。它的内容在被汇编语言函数调用后可以改变。

(5) 中断处理

这不同于普通的函数调用，在中断操作中必须保护和恢复它所使用的全部寄存器。如果使用 C 函数来描述中断处理，那么编译器有能力自动完成中断处理。如果使用汇编写中断处理，而它又调用了普通的 C 函数，那么汇编操作必须保护和恢复挥发性寄存器，普通 C 函数调

用不保护它们。中断处理操作同普通程序操作是异步的，中断处理或它的函数调用不能改变任意一个 MCU 寄存器。

9.6.17 函数返回非整型值

在调用函数前，必须描述其返回的一个长整型、浮点数或结构值。作为例子，在调用任意浮点函数之前，应当用 #include 语句包含头文件<math.h>；否则，在这些程序返回它们的值后程序将不工作。这和那些返回整型值的函数是有不同之处的。

(1) 返回长整型数或浮点数

长整型数或浮点数返回值是设定在一些寄存器 R16～R19 中。

(2) 传递结构值

如果传递结构值，结构允许通过堆栈传递，而不是通过寄存器。传递结构索引(也就是传递结构的地址)和传递任意数据项目的地址是相同的，都是通过一个 2 字节的指针。

(3) 返回结构值

当一个返回结构的函数被调用时，这个调用函数分配一个临时储藏库，而且传递一个隐藏指针给调用函数。当这个函数返回时，它拷贝返回值进这个临时储藏库。

9.6.18 程序和数据区的使用

(1) 程序存储器

程序存储器是被用于保存用户的程序代码、常数表和确定数据的初始值，比如字符串、全局变量。编译器可以生成一个对应程序存储器映像的输出文件(HEX 文件)，这个文件可以被编程器用来对芯片编程。

通常，编译器不能使用任意 64K 字节以上的程序存储器。为了访问 64K 字节边界以上的存储器(如在 Mega103 装置中)，需要在设定 RAMPZ 寄存器后直接调用 ELPM 指令。

(2) 数据存储器(仅指内部 SRAM)

这个数据存储器是被用于保存变量、堆栈结构和动态内存分配的堆。通常，它们不产生输出文件，但在程序运行时使用。一个程序使用数据内存如图 9.3 所示。

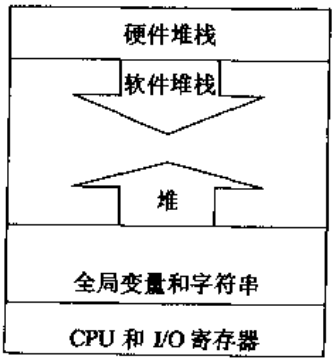


图 9.3 数据内存图

内存图的底部是地址 0，开始的 96(0x60)字节是 CPU 寄存器和 I/O 寄存器。编译器从 0x60 往上放置全局变量和字符串，在变量区域的顶部用户可以分配动态内存。在高端地址，硬件堆栈开始于 SRAM 的最后位置，在它的下面是向下生长的软件堆栈。它要求用户作为程序师，要确保硬件堆栈不生长进软件堆栈，而软件堆栈不生长进堆；否则，将导致意外的结果。

(3) 数据存储器(外部 SRAM)

如果选择带有 32K 或 64K 外部 SRAM 的目标装置，那么堆栈是放置在内部 SRAM 的顶部，并且是朝低端内存地址向下生长。数据内存是开始于硬件堆栈的顶部并且向上生长。这样分配的原因是在多数场合访问内部 SRAM 比访问外部 SRAM 的速度要快，分配堆栈到较快的内存是有很多好处的。

9.6.19 编程区域

编译器生成代码和数据到不同的区域 areas.，区域按照内存地址增高的顺序被编译器使用。

(1) 只读存储器

interrupt vectors—— 这个区域包括中断向量。

func_lit ——函数表区。

这个区的每个字包括了函数入口的地址，为了与代码压缩完全兼容，所有间接的函数索引必须通过间接的额外对准。

如果在 C 中使用函数指针调用函数，这是自动完成的。

在汇编中，举例如下：

```
; 假设 _foo 是函数的名称
. area func_lit
PL_foo:: . word _foo                ; 创建函数表入口
. area text
ldi R30,<PL_foo
ldi R31,>PL_foo
rcall xicall
```

可以间接地在函数表入口地址送入 R30/R31 寄存器对后，使用库函数 xicall 调用这个函数。

lit——这个区域包括了整型数和浮点数常量。

idata——全局变量和字符串的初始值保存在这个区域。

text——这个区域包括程序代码。

(2) 数据内存

data——这个区域包括全局变量、静态变量和字符串。

全局变量和字符串的初始值是保存在 idata 区域，并且是在启动时被拷贝进数据区的。

bss——这个区域包括未初始化的全局变量。

按 ANSI C 定义这些变量在启动时将初始化为 0。

(3) EEPROM 存储器

eeprom——这个区域包括 EEPROM 数据。

EEPROM 数据是写进扩展名为. eep 的输出文件，其为 INTEL HEX 文件格式。

9.6.20 调 试

ICC AVR 可以输出 COFF 格式调试文件，使用户可在 ATMEL 的 AVRStudio 中进行源程序级的调试。

如果用户想使用 AVRStudio 中的模拟 I/O 及终端仿真器，那么在 ICC AVR 的编译选项中必须将“AVR Studio Simulator IO”一项打钩。

ICC AVR 调试详见本书第三章内容。

9.7 应用举例*

9.7.1 读/写口

```
#include <io8515.h>
void main(void)
{                                    //PD口控制PB口的输出
unsigned char  achar;
DDRB = 0xFF;                         //PB口作为输出
DDRD = 0x00;                         //PD口作为输入,内部上拉
PORTD= 0xFF;
for(;;)
    {
    achar = PIND;                    //PD口的引脚状态
    PORTB = achar;                   //PB口输出所读到的PD口的引脚状态
    }
}
```

9.7.2 延时函数

```
#include <io8515.h>                  /*定义8515 */
void delay(int delayValue)
{
int i;                               //定义整型变量i[-32 768,32 767]
for(i=0;i<delayValue;i++) ;          //i<delayValue循环
}
void main(void)
{
unsigned char runner = 0x01;         //定义字符型变量runner
DDRB = 0xff;                         /* Port B输出 */
for (;;)                             /* 死循环 */
    {
    if (runner) runner <<= 1;        // runner! =0   右移一位
    else runner = 0x01;              // runner==0   则重置runner值
     PORTB = runner;                 // PB口输出runner值
    delay(32767);                    // 调用延时函数
    }
}
```

9.7.3 读/写 EEPROM

```
//int EEPROMwrite( int location, unsigned char);
```

---

* 更多应用举例见双龙公司光盘的第十章。

```
//unsigned char EEPROMread( int);
#include <io8515.h>
#include <eeprom.h>
void main(void)
{
  unsigned char temp =0xaa,i;
    EEPROMwrite(0x20,temp);          /* 写 EEPROM 地址 0x20 */
    i=EEPROMread(0x20);              /* 读 EEPROM 地址 0x20 */
    i++;                             //读到的值加 1
    EEPROMwrite(0x30,i);             //写 EEPROM 地址 0x30
}
```

9.7.4 AVR 的 PB 口变速移位

```
#include <io8515.h>
#define BIT(x) (1<<(x))              //定义移位函数 BIT()
void delay(void)
{                                    //延时函数
 unsigned char i,j;
      for (i=1;i;i++)
          for(j=1;j;j++);
}
void led_pb(void)
{                                    //PB 口输出函数
 unsigned char i;
 DDRB=0xff;                          //定义 PB 口作为输出口
 for (i=0;i<8;i++)
   {
    PORTB=~BIT(i);                   //取反后输出
    delay();                         //调用延时函数
   }
}
void main (void)
{
  while (1)                          //死循环
    led_pb();                        //调用 PB 口输出函数
}
```

9.7.5 音符声程序

```
#include <io8515.h>                  /* 预处理命令 */
#define uchar unsigned char
#define uint unsigned int
```

```
void delay(uchar t)
{                                          // 延时函数
 uchar i,j;
  for (i=0;i<t;i++)
      for(j=1;j<150;j++);
}
void sound_pc0(uchar t)
{                                          // PC0 脚输出音符声的函数
 uint i;
 DDRC=0xff;                                // 定义 PC 口作为输出口
 PORTC=0xff;                               // PC 口输出 0xff
 for (i=0;i<350-t*t;i++)                   // i<350-t*t,循环
     {
      PORTC^=0x01;                         // 改变 PC0 的输出状态
      delay(t);                            // 调用延时函数
     }
}
void main (void)
{
 uchar dt;
 for(;;)                                   // 死循环
     {
      for(dt=1;dt<14;dt++)                 // dt<14,循环
          sound_pc0(dt);                   // 调用 PC0 脚输出音符声的函数
      }
}
```

9.7.6 8 字循环移位显示程序

```
#include <io8515.h>
#define uchar unsigned char
#define uint unsigned int
void delay(uint t)
{                                          // 延时函数
 uint i;
 for (i=0;i<t;i++)
     ;
}
void init_disp(void)
{                                          // 初始化函数
DDRB=0xff;                                 //定义 PB 口,PD 口作为输出口
DDRD=0xff;
PORTB=0x7f;                                //PB 口输出 8 的七段码值 0x7f
```

```
}
void scan(void)
{                                          // 扫描函数,动态扫描显示
 uchar i,j;
 for (i=0;i<6;i++)
   {                                       // 扫描 6 位数码管
    j=150;
    do
      {
       PORTD=~(0x01<<i);                   // 扫描移位
       delay(150);                         // 点亮延时
       PORTD=0xff;
       delay(2100);                        // 熄灭延时
       }
    while(--j);
   }
}
void main(void)
{
 init_disp();                              // 调用初始化函数
 for(;;)                                   //死循环
 scan();                                   //调用动态扫描程序
}
```

9.7.7 锯齿波程序

```
#include <io8515.h>
#define uchar unsigned char
void delay(void)
{                                          // 延时函数
}
void main(void)
{
 uchar c;
 DDRA=0xff;                                // 定义 PA 口输出
 for (;;)                                  // 死循环
   {
    PORTA=c++;                             // PA 口输出增量
    delay();                               // 调用延时函数
   }
}
```

9.7.8 正三角波程序

```
#include <io8515.h>
#include <math.h>
#define uchar unsigned char
#define uint unsigned int
void delay(void)
{                                          // 延时函数
}
void main(void)
{
 uchar c;
 DDRA=0xff;                                // 定义 PA 口输出
 for (;;)                                  // 死循环
   {
    for (c=0x00;c<0xff;c++)                // c<0xff,循环
      {                                    //上升
       PORTA=c;                            // PA 口输出
       delay();                            //调用延时函数
      }
    for (c=0xff;c>0x00;c--)                // c>0x00,循环
     {                                     // 下降
      PORTA=c;                             // PA 口输出
      delay();                             // 调用延时函数
      }
  }
}
```

9.7.9 梯形波程序

```
#include <io8515.h>
#define uchar unsigned char
#define uint unsigned int
void delay(uchar t)
{                                          // 延时函数
 uchar i;
 for (i=0;i<t;i++)
     ;
}
void main(void)
{
 uchar c;
 DDRA=0xff;                                // 定义 PA 口输出
```

```
    for (;;)                                    // 死循环
      {
       for (c=0x00;c<0xff;c++)                  // c<0xff,循环
           PORTA=c;                             //上升,PA 口输出
       delay(255);                              //调用延时函数,保持高电平
       for (c=0xff;c>0x00;c--)                  // c>0x00,循环
           PORTA=c;                             //下降,PA 口输出
       delay(255);                              //调用延时函数,保持低电平
       }
}
```

附录 1　AT89 系列单片机简介

AT89 系列单片机是 ATMEL 公司的 8 位 Flash 单片机系列。这个系列单片机的最大特点是在片内含有 Flash 存储器。因此，在应用中有着十分广泛的前途，特别是在便携式、省电及特殊信息保存的仪器和系统中显得更为有用。

一、89 系列单片机特点

AT89 系列单片机是以 8051 核构成的，所以，它和 8051 系列单片机是兼容的系列。这个系列对于以 8051 为基础的系统，是十分容易进行取代和组成的。对于熟悉 8051 的用户，用 ATMEL 公司的 AT89 系列单片机进行取代 8051 的系统设计是轻而易举的事。

1. AT89 系列单片机的优点

(1) 内部含 Flash 存储器　在系统的开发过程中可以十分容易进行程序的修改，这就大大缩短了系统的开发周期。同时，在系统工作过程中能有效地保存一些数据信息，即使外界电源损坏也不会影响到信息的保存。

(2) 和 80C51 插座兼容　AT89 系列单片机的引脚是和 80C51 的引脚一样的，所以，当用 AT89 系列单片机取代 80C51 时，可以直接进行代换。这时，不管采用 40 引脚或是 44 引脚的产品，只要用相同引脚的 AT89 系列单片机取代 80C51 的单片机即可。

(3) 静态时钟方式　AT89 系列单片机采用静态时钟方式，所以可以节省电能，这对于降低便携式产品的功耗十分有用。

(4) 错误编程亦无废品产生　一般的 OTP 产品，一旦错误编程就成了废品。而 AT89 系列单片机内部采用了 Flash 存储器，所以，错误编程之后仍可以重新编程，直到正确为止，故不存在废品。

(5) 可进行反复系统试验　用 AT89 系列单片机设计的系统，可以反复进行系统试验；每次试验可以编入不同的程序，这样可以保证用户的系统设计达到最优。而且，随用户的需要和发展，还可以进行修改，使系统不断能追随用户的最新要求。

2. AT89 系列单片机的内部结构

AT89 系列单片机的内部结构和 80C51 相近，主要含有如下一些部件：

① 8051 CPU　　⑥ 片内 RAM
② 振荡电路　　⑦ 并行 I/O 接口
③ 总线控制部件　　⑧ 定时器
④ 中断控制部件　　⑨ 串行 I/O 接口
⑤ 片内 Flash 存储器　　⑩ 片内 EEPROM

在 AT89 系列单片机中，AT89C1051 的 Flash 存储器容量最小，只有 1K；而 AT89S55 的 Flash 存储器容量最大，有 20K。

在这个系列中，结构最简单的是 AT89C1051，它内部不含串行接口；最复杂的是 AT89S8252，它内部不但含标准的串行接口，还含有一个串行外围接口 SPI、Watchdog 定时器、双数据指针、EEPROM、电源下降的中断恢复等功能和部件。

AT89 系列单片机目前有多种型号，分别为 AT89C1051、AT89C2051、AT89C4051、AT89C51、AT89LV51、AT89C52、AT89LV52、AT89S8252、AT89LS8252、AT89C55、AT89LV55、AT89S53、AT89LS53、AT89S4D12。其中，AT89LV51、AT89LV52 和 AT89LV55 分别是 AT89C51、AT89C52 和 AT89C55 的低电压产品，最低电压可以低至2.7 V；AT89C1051 和 AT89C2051 则是低档型低电压产品，仅有 20 个引脚，最低电压仅为 2.7 V。

3. AT89 系列单片机的型号编码

AT89 系列单片机的型号编码由三个部分组成，它们是前缀、型号和后缀。格式如下：

AT89CXXXXXXXX　其中，AT 是前缀，89CXXXX 是型号，XXXX 是后缀。

下面分别对这 3 个部分进行说明，并且对其中有关参数的表示和意义作相应的解释。

(1)前缀　由字母“AT”组成，表示该器件是 ATMEL 公司的产品。

(2)型号　由“89CXXXX”或“89LVXXXX”或“89SXXXX”等表示。

“89CXXXX”中，9 是表示内部含 Flash 存储器，C 表示为 CMOS 产品。

“89LVXXXX”中，LV 表示低压产品。

“89SXXXX”中，S 表示含有串行下载 Flash 存储器。

在这个部分的“XXXX”表示器件型号数，如 51、1051、8252 等。

(3)后缀　由“XXXX”4 个参数组成，每个参数的表示和意义不同。在型号与后缀部分有“-”号隔开。

● 后缀中的第 1 个参数 X 用于表示速度，它的意义如下：

X=12，表示速度为 12 MHz。　　X=20，表示速度为 20 MHz。

X=16，表示速度为 16 MHz。　　X=24，表示速度为 24 MHz。

● 后缀中的第 2 个参数 X 用于表示封装，它的意义如下：

X=D，表示陶瓷封装。　　X=Q，表示 PQFP 封装。

X=J，表示 PLCC 封装。　　X=A，表示 TQFP 封装。

X=P，表示塑料双列直插 DIP 封装。　　X=W，表示裸芯片。

X=S，表示 SOIC 封装。

● 后缀中第 3 个参数 X 用于表示温度范围，它的意义如下：

X=C，表示商业用产品，温度范围为 0～+70℃。

X=I，表示工业用产品，温度范围为－40～+85℃。

X=A，表示汽车用产品，温度范围为－40～+125℃。

X=M，表示军用产品，温度范围为－55～+150℃。

● 后缀中第 4 个参数 X 用于说明产品的处理情况，它的意义如下：

X 为空，表示处理工艺是标准工艺。

X=/883，表示处理工艺采用 MIL－STD－883 标准。

例如：有一个单片机型号为“AT89C51－12PI”，则表示意义为该单片机是 ATMEL 公司的 Flash 单片机，内部是 CMOS 结构，速度为 12 MHz，封装为塑封 DIP，是工业用产品，按标准处理工艺生产。

二、AT89 系列单片机分类

AT89 系列单片机可分为标准型号、低档型号和高档型号 3 类。

标准型有 AT89C51 等 6 种型号，它们的基本结构和 89C51 是类似的，是 80C51 的兼容产

品。低档型有 AT89C1051 等 2 种型号，它们的 CPU 核和 89C51 是相同的，但并行 I/O 口较少。高档型有 AT89S8252 等型号，是一种可串行下载的 Flash 单片机，可以用在线方式对单片机进行程序下载。

1. 标准型单片机

标准型单片机有 89C51、89LV51、89C52、89LV52、89C55、89LV55 等 6 种型号。

标准型 AT89 系列单片机是和 MCS－51 系列单片机兼容的。在内部含有 4K、8K 或 20K 可重复编程的 Flash 存储器，可进行 1 000 次擦写操作。全静态工作为 0～33 MHz，有 3 级程序存储器加密锁定，有内部含 128～256 字节的 RAM，有 32 条可编程的 I/O 端口，有 2～3 个 16 位定时器/计数器，有 6～8 级中断，有通用串行接口，有低电压空闲及电源下降方式。

在这 6 种型号中，AT89C51 是一种基本型号。AT89LV51 是一种能在低电压范围工作的改进型，可在 2.7～6 V 电压范围工作，其它功能和 89C51 相同。AT89C52 是在 AT89C51 的基础上，在存储器容量、定时器和中断能力上得到改进的型号。89C52 的 Flash 存储器容量为 8K，16 位定时器/计数器有 3 个，中断有 8 级。而 89C51 的 Flash 存储器容量为 4K，16 位定时器/计数器有 2 个，中断只有 6 级。AT89LV52 是 89C52 的低电压型号，可在 2.7～6 V 电压范围内工作。89C55 的 Flash 存储器容量为 20K，16 位定时器/计数器有 3 个，中断有 8 级。AT89 LV55 是 89C55 的低电压型号，可在 2.7～6 V 电压范围内工作。

2. 低档型单片机

低档型单片机有 AT89C1051 和 AT89C2051 2 种型号。除并行 I/O 端口数较少之外，其它部件结构基本和 AT89C51 差不多。之所以被称为低档型，主要是因为它的引脚只有 20 条，比标准型的 40 引脚少得多。

AT89C1051 的 Flash 存储器只有 1K，RAM 只有 64 个字节，内部不含串行接口，内部的中断响应只有 3 种，保密锁定位只有 2 位。这些也是和标准型的 AT89C51 有区别的地方。AT89C2051 的 Flash 存储器只有 2K，RAM 只有 128 个字节，保密锁定位有 2 位。

也由于在上述有关部件上 AT89C1051、AT89C2051 的功能比标准型 AT89C51 要弱，所以它们就处于低档位置。

3. 高档型单片机

高档型单片机有 AT89S53、AT89S8252、AT89S4D12 等型号，是在标准型的基础上增加了一些功能形成的。增加的功能主要有如下几点：

(1) AT89S4D12 有 4K 可下载 Flash 存储器，AT89S8252 有 8K 可下载 Flash 存储器，AT89S53 有 12K 可下载 Flash 存储器。下载功能是由 IBM 微机通过 AT89 系列单片机的串行外围接口 SPI 执行的。

(2)除 8K Flash 存储器外，AT89S8252 还含有一个 2K 的 EEPROM，提高了存储容量。

(3)含有 9 个中断响应的能力。

(4)含标准型和低档型所不具有的 SPI 接口。

(5)含有 Watchdog 定时器(看门狗定时器)。

(6)含有双数据指针。

(7)含有从电源下降的中断恢复。

(8) AT89S4D12 除了 4K 可下载 Flash 存储器之外，还有一个 128K 片内 Flash 数据存储器、12MHz 内部振荡器、5 个可编程 I/O 线。

附录 2　AT94K 系列现场可编程系统标准集成电路

AT94K 系列(Fpslic family) 整合了 ATMEL AT40K 系列 SRAM FPGA 和高性能的带标准外设的 ATMEL AVR 8 位 RISC 微控制器。此器件中包含了扩展数据和指令 SRAM 及器件控制和管理逻辑，以 ATMEL0. 35μm 的 4 层金属 CMOS 工艺制作。10K～40K 门的 AT40K FPGA 带 8 位微控制器和 36K 字节的 SRAM AT40K FPGA 核心，是一个完全符合 3. 3V PCI 标准、带 10 ns 分布式同步/异步可编程的全双工口/单工口的 SRAM，8 个全局时钟、Cache Logic 性能(部分或全部可重新设置而不丢失数据) 及 10K～40K 的可用门数的基于 SRAM 的 FPGA 。

一、AT94K 系列单片机的特点

● 大规模现场可编程系统标准集成电路：

AT94K 基于 SRAM 的 FPGA，具有嵌入式高性能的 RISC AVR®核心及扩展的数据和指令的 SRAM。

● 10K～40K 门基于专利 SRAM 的 AT40K FPGA 带 FreeRAM™：

- -- 4. 6K～18. 4K 位的分布式单/双口 FPGA 的用户 SRAM；

——高性能 DSP 优化的 FPGA 核心单元；

——内置动态可重新编程；

——可存取设置 FPGA。

● AVR 微控制器核心片内支持 Cache Logic®设计：

——极低静态和动态功耗；

----最适于轻便及手持式的应用。

● 专利 AVR 扩展 RISC 结构：

——118 条功能强大的指令；

——绝大多数执行周期为单时钟周期；

——基于 DSP 系统的高性能硬件累乘器；

——可用超过 30 MIPS Performance；

——带 32 个内部寄存器的“C” 代码优化结构；

——低电压休眠，省电及掉电模式。

● 32K 字节动态分配指令和数据 SRAM：

——最多 16K × 16 内部 15ns 指令 SRAM；

——最多 14K × 8 内部 15ns 数据 SRAM。

● AVR Fixed 外设：

- -工业标准的 2 线接口；

· —2 个可编程串行 UART；

——2 个带分立预定比例器 PWM 的 8 位定时器/计数器 和 1 个带分立预定比例器、比较捕获模式及 8 位、9 位或 10 位 PWM 的 16 位定时器/计数器。

● 支持 FPGA 标准的外设：

—— AVR 外设控制；

- ·— 16 解码 AVR 地址线可直接存取 FPGA；

—— 标准外设的 FPGA 宏功能库。

● 16 FPGA 给 AVR 提供内部中断。

● 最多给 AVR 4 个外部中断。

● 8 个全局 FPGA 时钟：

—— 2 个从 AVR 逻辑驱动的 FPGA 时钟；

—— 可从 FPGA 核心存取 FPGA 全局时钟。

● 复合振荡器电路：

— 带片内振荡器的可编程看门狗定时器；

····· AVR 内部时钟电路振荡器；

—— 可软件选择时钟频率；

——定时器/计数器实时时钟振荡器。

● V_{CC}：3.0～3.6V。

● 3.3V、33 MHz PCI 标准的 FPGA I/O。

—— 24 mA 下沉/源高性能 I/O 结构；

-···· 所有 FPGA I/O 单独可编程。

● 引脚与 ATMEL AT40K 系列 FPGA 兼容。

● 高性能、低电压 0.35μm CMOS 4 层金属处理。

● State-of-the-art 基于 PC 的包含协检验的集成软件。

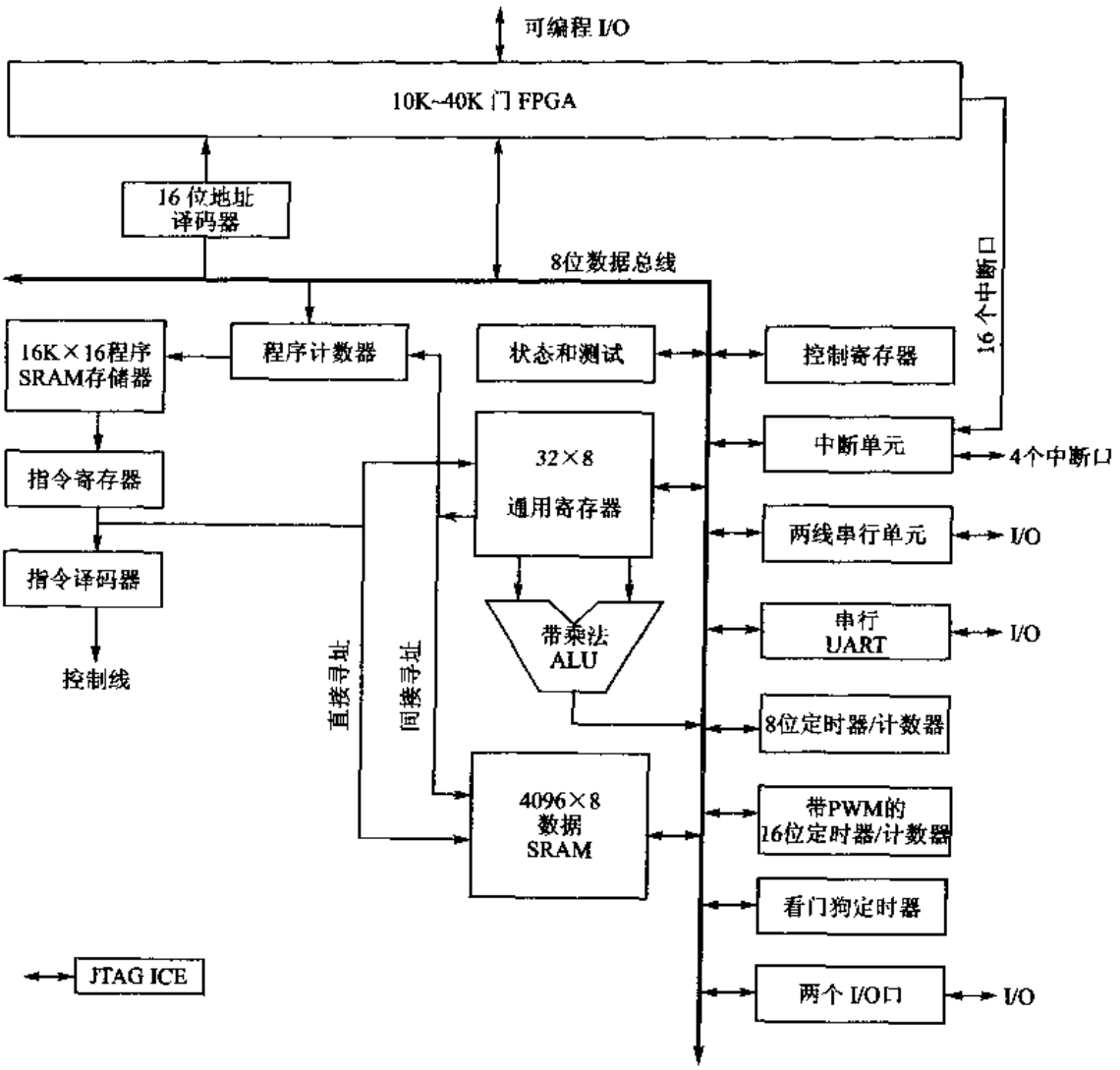
AT94K 系列单片机的性能比较，见附录表 2.1。

附录表 2.1 AT94K 系列单片机的性能比较

器 件	AT94K10	AT94K20	AT94K40
FPGA 门数	10K	20K	40K
FPGA 核心单元	576	1 024	2 304
FPGA SRAM 位数	4 096	8 192	8 432
FPGA 寄存器数(全部)	864	1 408	2 880
最多 FPGA 用户 I/O	144	192	288
可编程 SRAM/B	20K～32K	20K～32K	20K～32K
数据 SRAM/B	4K～16K	4K～16K	4K～16K
硬件类乘数器(8 位)	有	有	有
两线串行接口	有	有	有
UART	2	2	2
看门狗定时器	有	有	有
定时器/计数器	3	3	3
实时时钟	有	有	有
典型的 AVR 吞吐量@40 MHz	30 MIPS	30 MIPS	30 MIPS
工作电压/V	3.0～3.6	3.0～3.6	3.0～3.6

二、AT94K 系列单片机的结构

AT94K 系列单片机的结构见附录图 2.1。



附录图 2.1　AT94K 系列单片机的结构

AT94K 内嵌 AVR 核心，通过在单个时钟周期内执行指令，实现 1 MIPS 每 MHz 的吞吐量，以允许系统设计者优化功耗与处理速度。AVR 核心基于一个包含了丰富指令集和 32 个通用工作寄存器的扩展 RISC 结构。所有 32 个寄存器直接与算数逻辑单元(ALU)连接，在一个时钟周期内执行单条指令时允许存取 2 个独立的寄存器。当吞吐量达到 CLK 频率下的普通 CISC 微控制器 10 倍时，合成的结构可更有效地编码。AVR 可操作片外 SRAM。FPGA设置 SRAM 和 AVR 指令编码 SRAM，都能自动地在系统上电时使用 ATMEL 的内置可编程 AT17 系列 EEPROM 设置存储器来装载。State-of-the-art Fpslic 设计工具“System Designer”，是为了与 Fpslic 结构协调而开发，以减少用来集成微控制器开发及调试的整体时间；FPGA 开发、放置与布线及完成系统检验的功能，集中在易于使用的软件工具中。

附录3　指令集综合

AVR单片机的指令系统，对不同器件有不同的指令。ATmega161等指令，包括所有AVR单片机的指令。它们的关系如下：

(1) 89条指令器件：AT90S1200，是最基本的指令；各种AVR器件指令比较表中都无标记。

(2) 90条指令器件(□)：ATtiny11/12/15/22；90条指令＝□＋89条基本指令。

(3) 118条指令器件(◇)：AT90S2313/2323/2343/2333/4414/4433/4434/8515/8534/8535；　　118条指令＝◇＋90条指令。

(4) 121条指令器件(△)ATmega603/103；　　121条指令＝△＋118条指令。

(5) 130条指令器件(☆)ATmega161/8/16/32；130条指令＝☆＋121条指令。

(6) 133条指令器件ATmega64/128。

(7) 139条指令器件ATmega169，详见附录表4。

附录表 3.1　AVR AT90S1200 器件 89 条指令速查表

算术和逻辑指令		条件转移指令		位指令和位测试指令	
ADD Rd,Rr	加法	SBRC Rr,b	位清零跳行	SBI P,b	置位 I/O 位
ADC Rd,Rr	带进位加	SBRS Rr,b	位置位跳行	CBI P,b	清零 I/O 位
SUB Rd,Rr	减法	SBIC P,b	I/O 位清零跳行	LSL Rd	左移
SUBI Rd,K	减立即数	SBIS P,b	I/O 位置位跳行	LSR Rd	右移
SBC Rd,Rr	带进位减	BRBS s,k	SREG 位置位转	ROL Rd	带进位左循环
SBCI Rd,K	带 C 减立即数	BRBC s,k	SREG 位清零转	ROR Rd	带进位右循环
AND Rd,Rr	与	BREQ k	相等转移	ASR Rd	算术右移
ANDI Rd,K	与立即数	BRNE k	不相等转移	SWAP Rd	半字节交换
OR Rd,Rr	或	BRCS k	C 置位转	BSET s	置位 SREG
ORI Rd,K	或立即数	BRCC k	C 清零转	BCLR s	清零 SREG
EOR Rd,Rr	异或	BRSH k	≥转	BST Rr,b	Rr 的 b 位送 T
COM Rd	取反	BRLO k	小于转(无符号)	BLD Rd,b	T 送 Rr 的 b 位
NEG Rd	取补	BRMI k	负数转移	SEC	置位 C
SBR Rd,K	寄存器位置位	BRPL k	正数转移	CLC	清零 C
CBR Rd,k	寄存器位清零	BRGE k	≥转(带符号)	SEN	置位 N
INC Rd	加 1	BRLT k	小于转(带符号)	CLN	清零 N
DEC Rd	减 1	BRHS k	H 置位转移	SEZ	置位 Z
TST Rd	测试零或负	BRHC k	H 清零转移	CLZ	清零 Z
CLR Rd	寄存器清零	BRTS k	T 置位转移	SEI	置位 I
SER Rd	寄存器置 FF	BRTC k	T 清零转移	CLI	清零 I
条件转移指令		BRVS k	V 置位转移	SES	置位 S
RJMP k	相对转移	BRVC k	V 清零转移	CLS	清零 S
RCALL k	相对调用	BRIE k	中断位置位转移	SEV	置位 V
RET	子程序返回	BRID k	中断位清零转移	CLV	清零 V
RETI	中断返回	数据传送指令		SET	置位 T
CPSE Rd,Rr	比较相等跳行	MOV Rd,Rr	寄存器传送	CLT	清零 T
CP Rd,Rr	比较	LDI Rd,K	装入立即数	SEH	置位 H
CPC Rd,Rr	带进位比较	LD Rd,Z	Z 变址间接取数	CLH	清零 H
CPI Rd,K	与立即数比较	ST Z,Rr	Z 变址间接存数	NOP	空操作
		IN Rd,P	从 I/O 口取数	SLEEP	休眠指令
		OUT P,Rr	存数于 I/O 口	WDR	看门狗复位

注：ATtiny11/12/15/22 为 90 条指令器件，比 AT90S1200 多一条指令 LPM，从程序区取数。

附录表 3.2 AVR 器件 118 条指令速查表

AT90S2313/2333/4414/4433/4434/8515/8534/8535

算术和逻辑指令		BRCC k	C 清零转	位指令和位测试指令	
ADD Rd,Rr	加法	BRSH k	≥转	SBI P,b	置位 I/O 位
ADC Rd,Rr	带进位加	BRLO k	小于转(无符号)	CBI P,b	清零 I/O 位
◇ADIW Rdl,K	加立即数	BRMI k	负数转移	LSL Rd	左移
SUB Rd,Rr	减法	BRPL k	正数转移	LSR Rd	右移
SUBI Rd,Rr	减立即数	BRGE k	≥转(带符号)	ROL Rd	带进位左循环
SBC Rd,Rr	带进位减	BRLT k	小于转(带符号)	ROR Rd	带进位右循环
SBCI Rd,K	带 C 减立即数	BRHS k	H 置位转移	ASR Rd	算术右移
◇SBIW Rdl,K	减立即数	BRHC k	H 清零转移	SWAP Rd	半字节交换
AND Rd,Rr	与	BRTS k	T 置位转移	BSET s	置位 SREG
ANDI Rd,K	与立即数	BRTC k	T 清零转移	BCLR s	清零 SREG
OR Rd,Rr	或	BRVS k	V 置位转移	BST Rr,b	Rr 的 b 位送 T
ORI Rd,K	或立即数	BRVC k	V 清零转移	BLD Rd	T 送 Rr 的 b 位
EOR Rd,Rr	异或	BRIE k	中断位置位转移	SEC	置位 C
COM Rd	取反	BRID k	中断位清零转移	CLC	清零 C
NEG Rd	取补	数据传送指令		SEN	置位 N
SBR Rd,K	寄存器位置位	MOV Rd,Rr	寄存器传送	CLN	清零 N
CBR Rd,K	寄存器位清零	◇ LDI Rd,Rr	装入立即数	SEZ	置位 Z
INC Rd	加 1	◇ LD Rd,X	X 间接取数	CLZ	清零 Z
DEC Rd	减 1	◇ LD Rd,X+	X 间接取数后+	SEI	置位 I
TST Rd	测试零或负	◇ LD Rd,-X	X 间接取数先-	CLI	清零 I
CLR Rd	寄存器清零	◇ LD Rd,Y	Y 间接取数	SES	置位 S
SER Rd	寄存器置 FF	◇ LD Rd,Y+	Y 间接取数后+	CLS	清零 S
条件转移指令		◇ LD Rd,- Y	Y 间接取数先-	SEV	置位 V
RJMP k	相对转移	◇LDD Rd,Y+q	Y 间接取数先+q	CLV	清零 V
◇IJMP	间接转移(Z)	LD Rd,Z	Z 间接取数	SET	置位 T
RCALL k	相对调用	◇LD Rd,Z+	Z 间接取数后+	CLT	清零 T
◇ICALL	间接调用(Z)	◇LD Rd,-Z	Z 间接取数先-	SEH	置位 H
RET	子程序返回	◇LDD Rd,Z+q	Z 间接取数+q	CLH	清零 H
RETI	中断返回	◇LDS Rd,K	从 SRAM 装入	NOP	空操作
CPSE Rd,Rr	比较相等跳行	◇ST X,Rr	X 间接存数	SLEEP	休眠指令
CP Rd,Rr	比较	◇ST X+,Rr	X 间接存数后+	WDR	看门狗复位
CPC Rd,Rr	带进位比较	◇ST -X,Rr	X 间接存数先-		
CPI Rd,K	与立即数比较	◇ST Y,Rr	Y 间接存数		
SBRC Rr,b	位置位跳行	◇ST Y+,Rr	Y 间接存数后+	90 条指令器件为 ATtiny11/12/15/22=□+89 条基本指令器件是 AT90S1200	
SBRS Rr,b	I/O 位清零跳行	◇ST -Y,Rr	Y 间接存数先-		
SBIC P,b	I/O 位置位跳行	◇STD Y+q,Rr	Y 间接存数+q		
SBIS P,b	SREG 位置位转	ST Z,Rr	Z 间接存数		
BRBC s,k	SREG 位清零转	◇ST Z+,Rr	Z 间接存数后+		
BREQ k	相等转移	◇ST -Z,Rr	Z 间接存数先-	118 条指令器件=◇+90 条指令器件	
BRNE k	不相等转移	◇STD Z+q,Rr	Z 间接存数+q		
BRNE k	不相等转移	◇STS k,Rr	数据送 SRAM		
BRCS k	C 置位转	□LPM	从程序区取数		
		IN Rd,P	从 I/O 口取数		
		OUT P,Rdr	存数 I/O 口		
		PUSH Rr	压栈		
		POP Rd,	出栈		

附录表 3.3　各种 AVR 器件指令比较表(指令速查表)

算术和逻辑指令		☆ESPM	扩展存储程序存储器	位指令和位测试指令	
ADD	加法	☆EICALL	延长间接调用子程序	SBI	置位 I/O 位
ADC	带进位加	BRCC	C 清零转	CBI	清零 I/O 位
◇ADIW	加立即数	BRSH	≥转	LSL	左移
SUB	减法	BRLO	小于转(无符号)	LSR	右移
SUBI	减立即数	BRMI	负数转移	ROL	带进位左循环
SBC	带进位减	BRPL	正数转移	ROR	带进位右循环
SBCI	带 C 减立即数	BRGE	≥转(带符号)	ASR	算术右移
◇SBIW	减立即数	BRLT	小于转(带符号)	SWAP	半字节交换
AND	与	BRHS	H 置位转移	BSET	置位 SREG
ANDI	与立即数	BRHC	H 清零转移	BCLR	清零 SREG
OR	或	BRTS	T 置位转移	BST	Rr 的 b 位送 T
ORI	或立即数	BRTC	T 清零转移	BLD	T 送 Rr 的 b 位
EOR	异或	BRVS	V 置位转移	SEC	置位 C
COM	取反	BRVC	V 清零转移	CLC	清零 C
NEG	取补	BRIE	中断位置位转移	SEN	置位 N
SBR	寄存器位置位	BRID	中断位清零转移	CLN	清零 N
CBR	寄存器位清零	数据传送指令		SEZ	置位 Z
INC	加 1	☆MOV	寄存器传送	CLZ	清零 Z
DEC	减 1	◇MOVW	拷贝寄存器字	SEI	置位 I
TST	测试零或负	◇LDI	装入立即数	CLI	清零 I
CLR	寄存器清零	◇LD X	X 间接取数	SES	置位 S
SER	寄存器置 FF	◇LD X+	X 间接取数后+	CLS	清零 S
☆MUL	乘法	◇LD −X	X 间接取数先−	SEV	置位 V
☆MUL S	有符号数乘法	◇LD Y	Y 间接取数	CLV	清零 V
☆MUL SU	有(无)符号数乘法	◇LD Y+	Y 间接取数后 i	SET	置位 T
☆FMUL	小数乘法	◇LD −Y	Y 间接取数先−	CLT	清零 T
☆FMUL S	有符号数乘法	◇LDD Yq	Y 间接取数+q	SEH	置位 H
☆FMUL SU	有(无)符号小数乘法	LD Z	Z 间接取数	CLH	清零 H
条件转移指令		◇LD Z+	Z 间接取数后+	NOP	空操作
RJMP	相对转移	◇LD −Z	Z 间接取数先−	SLEEP	休眠
◇IJMP	间接转移	◇LDD Zq	Z 间接取数+q	WDR	看门狗复位
△JMP	长转移	◇LDS	从 SRAM 装入		
RCALL	相对调用	◇ST X	X 间接存数	90 条指令器件(□)	
◇ICALL	间接调用	◇ST X+	X 间接存数后+	ATtiny11/12/15/22	
△CALL	长调用	◇ST −X	X 间接存数先−		
RET	子程序返回	◇ST Y	Y 间接存数	89 条指令器件	
RET I	中断返回	◇ST Y+	Y 间接存数后+	AT90S1200	
CPSE	比较相等跳行	◇ST −Y	Y 间接存数先−	118 条指令器件(◇)	
CP	比较	◇STD Yq	Y 间接存数+q	AT90S2313/2323/2343/2333	
CPC	带进位比较	ST Z	Z 间接存数	AT90S4414/4433/4434/8515	
CPI	带立即数比较	◇ST Z+	Z 间接存数后+	AT90S8534/8535	
SBRC	位清零跳行	◇ST −Z	Z 间接存数先−	121 条指令器件(△)	
SBRS	位置位跳行	◇STD Zq	Z 间接存数+q	ATmega603/103	
SBIC	I/O 位清零跳行	◇STS	数据送 SRAM	130 条指令器件(☆)	
SBIS	I/O 位置位跳行	□LPM	装程序存储器	ATmega161	
BRBS	SREG 位置位转	☆LPM Z	Z	90 条指令=□+89 条指令	
BRBC	SREG 位清零转	☆LPM Z+	Z+	118 条指令=◇ +90 条指令	
BREQ	相等转移	☆SPM	存储程序存储器	121 条指令=△+118 条指令	
BRNE	不相等转移	◇IN	I/O 口输入	130 条指令=☆+121 条指令	
BRCS	C 置位转	◇OUT	送 I/O 口	更详细资料阅读 英文指令表	
☆ELPM	扩展装载程序存储器	PUSH	压栈		
☆EIJMP	扩展间接跳转	POP	出栈		

附录 4　AVR 单片机选型表

附录表 4.1　AVR 单片机选型表(摘自 2002 年 7 月 ATMEL 网站)

	FLASH/KB	EEPROM/B	RAM/B 32 个工作寄存器	指令	I/O	中断	外部中断	SPI	UART	TWI(I²C)	硬件乘法器	8 位定时器	16 位定时器	PWM	看门狗定时器	定时时钟	模拟比较器	10 位 AD 通道	片内振荡器	BOD	ISP	SPM	V_{CC}/V	时钟/MHz	封　装	
ATtiny11L	1	—	32	90	6	4	1(+5)	—	—	—	—	1	—	—	Y	—	Y	—	Y	—	Y[3]	—	2.7～5.5	0～2	8-Pin DIP, SOIC	now
ATtiny11	1	—	32	90	6	4	1(+5)	—	—	—	—	1	—	—	Y	—	Y	—	Y	—	Y[3]	—	4.0～5.5	0～6	8-Pin DIP, SOIC	now
ATtiny12V	1	64	32	90	6	5	1(+5)	—	—	—	—	1	—	—	Y	—	Y	—	Y[2]	Y	Y	—	1.8～5.5	0～1	8-Pin DIP, SOIC	now
ATtiny12L	1	64	32	90	6	5	1(+5)	—	—	—	—	1	—	—	Y	—	Y	—	Y[2]	Y	Y	—	2.7～5.5	0～4	8-Pin DIP, SOIC	now
ATtiny12	1	64	32	90	6	5	1(+5)	—	—	—	—	1	—	—	Y	—	Y	—	Y[2]	—	Y	—	4.0～5.5	0～8	8-Pin DIP, SOIC	now
ATtiny15L	1	64	32	90	6	8	1(+5)	—	—	—	—	2	—	1	Y	—	Y	4	Y[2]	Y	Y	—	2.7～5.5	1～6	8-Pin DIP, SOIC	now
ATtiny26L	2	128	128+32	118	16	11	1(+8)	1[6]	1[5]	1[4]	—	2	—	4	Y	—	Y	11	Y[2]	Y	Y	—	2.7～5.5	0～8	20-Pin DIP, SOIC 32-Pin MLF	Q302
ATtiny26	2	128	128+32	118	16	11	1(+8)	1[6]	1[5]	1[4]	—	2	—	4	Y	—	Y	11	Y[2]	Y	Y	—	4.5～5.5	0～16	20-Pin DIP, SOIC 32-Pin MLF	Q302
ATtiny28V	2	—	32	90	20	5	1(+8)	—	—	—	—	1	—	--	Y	--	Y	—	Y[2]	—	Y[2]	—	1.8～5.5	0～1	28-Pin DIP 32-Pin TQFP,MLF	now
ATtiny28L	2	—	32	90	20	5	1(+8)	—	—	—	—	1	—	--	Y	--	Y	—	Y[2]	—	Y[2]	—	2.7～5.5	0～4	28-Pin DIP 32-Pin TQFP,MLF	now
AT90S1200	1	64	32	89	15	3	1	—	—	—	—	1	—	—	Y	—	Y	—	Y	—	Y	—	2.7～6.0	0～4	20-Pin DIP, SOIC, SSOP	now
AT90S1200	1	64	32	89	15	3	1	--	—	--	—	1	—	—	Y	—	Y	—	Y	—	Y	—	4.0～6.0	0～12	20-Pin DIP, SOIC, SSOP	now
AT90S2313	2	128	128+32	120	15	20	2	-	1	-	—	1	1	1	Y	—	Y	—	—	—	Y	—	2.7～6.0	0～4	20-Pin DIP, SOIC	now

	FLASH/KB	EEPROM/B	RAM/B 32个工作寄存器	指令	I/O	中断	外部中断	SPI	UART	TWI(I²C)	硬件乘法器	8位定时器	16位定时器	PWM	看门狗定时器	定时时钟	模拟比较器	10位AD通道	片内振荡器	BOD	ISP	SPM	V_{CC}/V	时钟/MHz	封装	
AT90S2313	2	128	128+32	120	15	10	2	—	1	—	—	1	1	1	Y	—	Y	—	—	—	Y	-	4.0~6.0	0~10	20-Pin DIP, SOIC	now
AT90LS2323	2	128	128+32	120	3	2	1	-	—	—	—	1	—	—	Y	—	—	—	—	—	Y	-	2.7~6.0	0~4	8-Pin DIP, SOIC	now
AT90S2323	2	128	128+32	120	3	2	1	—	—	—	—	1	—	—	Y	—	—	—	—	—	Y	-	4.0~6.0	0~10	8-Pin DIP, SOIC	now
AT90LS2343	2	128	128+32	120	5	2	1	-	—	-	—	1	—	-	Y	-	—	-	Y	-	Y	—	2.7~6.0	0~1	8-Pin DIP, SOIC	now
AT90LS2343	2	128	128+32	120	5	2	1	—	—	-	—	1	—	-	Y	-	—	—	Y	—	Y	—	2.7~6.0	0~4	8-Pin DIP, SOIC	now
AT90S2343	2	128	128+32	120	5	2	1	—	—	-	—	1	—	-	Y	-	—	-	Y	-	Y	—	4.0~6.0	0~10	8-Pin DIP, SOIC	now
AT90LS4433	4	256	128+32	120	20	14	2	1	1	—	-	1	1	1	Y	—	Y	6	-	Y	Y	—	2.7~6.0	0~4	28-Pin DIP 32-Pin TQFP	now
AT90S4433	4	256	128+32	120	20	14	2	1	1	—	-	1	1	1	Y	—	Y	6	-	Y	Y	—	4.0~6.0	0~8	28-Pin DIP 32-Pin TQFP	now
AT90S8515	8	512	512+32	120	32	12	2	1	1	—	—	1	1	2	Y	—	Y	—	—	—	Y	—	2.7~6.0	0~4	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90S8515	8	512	512+32	120	32	12	2	1	1	—	—	1	1	2	Y	—	Y	—	—	—	Y	—	4.0~6.0	0~8	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90LS8535	8	512	512+32	120	32	16	2	1	1		—	2	1	3	Y	Y	Y	8	—	-	Y	—	2.7~6.0	0~4	40-Pin DIP 44-Pin PLCC, TQFP	now
AT90S8535	8	512	512+32	120	32	16	2	1	1	—	—	2	1	3	Y	Y	Y	8	—	—	Y	—	4.0~6.0	0~8	40-Pin DIP 44-Pin PLCC, TQFP	now
ATmega8L	8	512	1K+32	130	23	18	2	1	1[5]	1	Y	2	1	3	Y	Y	Y	4+2 6+2	Y[2]	Y	Y	Y	2.7~5.5	0~8	28-Pin DIP 32-Pin MLF, TQFP	now
ATmega8	8	512	1K+32	130	23	18	2	1	1[5]	1	Y	2	1	3	Y	Y	Y	4+2 6+2	Y[2]	Y	Y	Y	4.5~5.5	0~16	28-Pin DIP 32-Pin MLF, TQFP	now
ATmega8535L	8	512	512+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~5.5	0~8	40-Pin DIP 44-Pin TQFP, MLF	Q302

附录表 4.1(续)

	FLASH/KB	EEPROM/B	RAM/B 32个工作寄存器	指令	I/O	中断	外部中断	SPI	UART	TWI(I²C)	硬件乘法器	8位定时器	16位定时器	PWM	看门狗定时器	定时时钟	模拟比较器	10位AD通道	片内振荡器	ROD	ISP	SPM	V_{CC}/V	时钟/MHz	封装	
ATmega8535	8	512	512+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5～5.5	0～16	40-Pin DIP 44-Pin TQFP, MLF	Q302
ATmega8515L	8	512	512+32	130	35	16	3	1	1[5]	1	Y	1	1	2	Y		Y	—	Y[2]	Y	Y	Y	2.7～5.5	0～8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega8515	8	512	512+32	130	35	16	3	1	1[5]	1	Y	1	1	2	Y	–	Y	—	Y[2]	Y	Y	Y	4.5～5.5	0～16	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega161L	16	512	1K+32	130	35	20	3	1	2	—	Y	2	1	4	Y	Y	Y	—	—	Y	Y	Y	2.7～5.5	0～4	40-Pin DIP 44-Pin TQFP	now
ATmega161	16	512	1K+32	130	35	20	3	1	2	—	Y	2	1	4	Y	Y	Y	—	—	Y	Y	Y	4.0～5.5	0～8	40-Pin DIP 44-Pin TQFP	now
ATmega162V	16	512	1K+32	130	35	20	3(+16)	1	2[5]	1	Y	2	2	6	Y	Y	Y	—	Y[2]	Y	Y	Y	1.8～3.6	0～1	40-Pin DIP 44-Pin TQFP, MLF	0302
ATmega162L	16	512	1K+32	130	35	20	3(+16)	1	2[5]	1	Y	2	2	6	Y	Y	Y	—	Y[2]	Y	Y	Y	2.7～5.5	0～8	40-Pin DIP 44-Pin TQFP, MLF	0302
ATmega162	16	512	1K+32	130	35	20	3(+16)	1	2[5]	1	Y	2	2	6	Y	Y	Y	—	Y[2]	Y	Y	Y	4.5～5.5	0～16	40-Pin DIP 44-Pin TQFP, MLF	0302
ATmega163L	16	512	1K+32	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～4	40-Pin DIP 44-Pin TQFP	now
ATmega163	16	512	1K+32	130	32	17	2	1	1	1	Y	2	1	3	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0～5.5	0～8	40-Pin DIP 44-Pin TQFP	now
ATmega16L	16	512	1K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7～5.5	0～8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega16	16	512	1K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5～5.5	0～16	40-Pin DIP 44-Pin TQFP, MLF	now

附录表 4.1(续)

	FLASH/KB	EEPROM B	RAM. B 32个工作寄存器	指令	I/O	中断	外部中断	SPI	UART	TWI(I²C)	硬件乘法器	8位定时器	16位定时器	PWM	看门狗定时器	定时时钟	模拟比较器	10位AD通道	片内振荡器	BOD	ISP	SPM	V_{CC}/V	时钟/MHz	封装	
ATmega323L	32	1K	2K+32	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~5.5	0~4	40-Pin DIP 44-Pin TQFP	now
ATmega323	32	1K	2K+32	130	32	19	3	1	1	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.0~5.5	0~8	40-Pin DIP 44-Pin TQFP	now
ATmega32L	32	1K	2K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~5.5	0~8	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega32	32	1K	2K+32	130	32	20	3	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5~5.5	0~16	40-Pin DIP 44-Pin TQFP, MLF	now
ATmega64L	64	2K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~5.5	0~8	64-Pin TQFP, MLF	Q302
ATmega64	64	2K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5~5.5	0~16	64-Pin TQFP, MLF	Q302
ATmega169V[4]	16	512	1K+32	139	54	22	1(+16)	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	1.8~3.6	0~1	64-Pin TQFP, MLF	Q302
ATmega169L[4]	16	512	1K+32	139	54	22	1(+16)	1	1[5]	1	Y	2	1	4	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~3.6	0~4	64-Pin TQFP, MLF	Q302
ATmega103L	128	4K	4K+32	121	48	16	8	1	1		—	2	1	4	Y	Y	Y	8	—	—	Y	-	2.7~3.6	0~4	64-Pin TQFP	now
ATmega103	128	4K	4K+32	121	48	16	8	1	1	-	—	2	1	4	Y	Y	Y	8	—	—	Y	—	4.0~5.5	0~6	64-Pin TQFP	now
ATmega128L	128	4K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	2.7~5.5	0~8	64 Pin TQFP, MLF	now
ATmega128	128	4K	4K+32	133	53	34	8	1	2[5]	1	Y	2	2	6+2	Y	Y	Y	8	Y[2]	Y	Y	Y	4.5~5.5	0~16	64 Pin TQFP, MLF	now

1) 1个外部中断和唤醒引脚变化(所有I/O引脚)。

2) 高精度(5%)可编程内部RC振荡器。

3) 编程时需在RESET脚提供12V信号。

4) 兼容I²C。

5) 可编程串行USART。

6) 8个引脚可低电平中断。

参 考 文 献

1 宋建国主编. AVR 单片机原理及应用. 北京:北京航空航天大学出版社,1998

2 林容益编. AVR 高速 16 位元 PD 单晶片微控器应用. 台北:台湾全华科技图书股份有限公司,1999

3 ATMEL PRODUCTS JULY 1999 年(光盘文件)

4 徐爱均,彭秀华编著. 单片机高级语言 C51 应用程序设计. 北京:电子工业出版社,1998

5 www. atmel. com

6 www. sl. com. cn

7 www. mcselec. com