AVR 单片机应用技术

李　勋　耿德根　编著

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

内容简介

AVR 单片机是目前最新的单片机系列之一，它以高速度、大驱动能力和丰富的硬件资源而著称于微控制器系列。本书前两章系统深入地讲解了 AVR 单片机系列的硬件组成、运作原理及其指令集。在此基础上，第三、四两章进一步例示了 AVR 汇编语言程序设计技巧及片内各部件的用法。最后两章告诉读者如何构成单片机应用系统及进行软硬件开发。

本书取材先进，叙述条理清晰，语言通畅，例题丰富，可读性强。以高等院校有关专业师生及从事微机控制方面工作的广大科技人员为读者对象。

图书在版编目(CIP)数据

AVR 单片机应用技术/李勋等编著. —北京：北京航空航天大学出版社，2002.6

ISBN 7-81077-177-9

Ⅰ.A… Ⅱ.李… Ⅲ.单片微型计算机，AVR

Ⅳ.TP368.1

中国版本图书馆 CIP 数据核字(2002)第 030287 号

AVR 单片机应用技术

李　勋　耿德根　编著

责任编辑　刘宝俊

责任校对　戚　爽

*

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　传真：(010)82328026

http://www.buaapress.com.cn

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

北京密云华都印刷厂印装　各地书店经销

*

开本：787×1092　1/16　印张：15.5　字数：397 千字

2002 年 6 月第 1 版　2002 年 6 月第 1 次印刷　印数：5 000 册

ISBN 7-81077-177-9/TP·099　**定价：23.00 元**

前　言

AVR 单片机是目前最新单片机系列之一，其突出的特点在于速度高、片内硬件资源丰富、可作为真正意义上的单片机使用。

AVR 采用了现代微处理器流水管线预取指令技术，淘汰了机器周期的概念，以时钟周期为指令执行的最基本时间单位。通常时钟频率用 4 MHz～8 MHz，故最短指令执行时间为 250～125 ns。

这种单片机借鉴了某些机型的高速输入输出 HSIO 和可编程计数阵列 PCA 的概念，实现了本身的输入捕捉、输出比较和脉冲宽度调制输出功能。这是脉冲信号测量、开关量按时控制及某些直流马达调速的得力工具。AVR 除继承了前代器件的异步串行口外，尚增加了高速同步串行口 SPI，因此可方便灵活地与其他器件进行串行通讯。AVR 监视定时器具有独立的时钟，故在掉电方式下仍可正常运行。这样就增加了使 MCU 从掉电方式下苏醒过来的手段。该系列各器件大多内含模拟比较器，有的尚有内部 A/D 转换器，为处理模拟量输入提供了条件。

AVR 摒弃了 EPROM 和掩膜 ROM 方案，以擦写很方便的 Flash 作为程序存储器，且其容量档次齐全，用户可根据实际需要进行选择。加之片内 SRAM 和 EEPROM 可满足一般要求，故通常无系统扩展的必要。所有 I/O 线均可用作通用输入输出，因此可实现与外设之间的直接接口，联络信号由软件产生。

作者认为有必要把这种优秀的芯片介绍给我国广大的有关科技人员，使之成为他们在微控制领域的得力助手。本书的特点是在吃透原文文献的基础上，结合作者多年实践经验写成的。书内大量的例题中，多处体现着作者的思路。这也正是称本书为“编著”的原因之所在。

全书共六章，其中第六章由耿德根高级工程师供稿，其余五章均由李勋教授撰写。本书的两位作者希望看到广大读者对本书的意见和评价。

作　者
天津工业大学
2002.2

目　　录

第一章　体系结构及运作原理

1.1　AT90 系列 …… (1)
1.2　内部结构 …… (1)
1.2.1　结构框图 …… (1)
1.2.2　通用寄存器组合与 ALU …… (4)
1.2.3　存储器组织 …… (5)
1.2.4　程序和数据的寻址方式 …… (6)
1.2.5　I/O 存储器 …… (9)
1.2.6　定时/计数器 0 …… (12)
1.2.7　定时/计数器 1 …… (15)
1.2.8　监视定时器 …… (23)
1.2.9　EEPROM 读写 …… (25)
1.2.10　串行涉外接口 SPI …… (27)
1.2.11　通用异步接收发送器 …… (32)
1.2.12　模拟比较器 …… (38)
1.2.13　复位控制电路 …… (39)
1.2.14　中断系统 …… (41)
1.2.15　I/O 端口 …… (47)
1.3　封装与引脚功能 …… (57)
1.3.1　引脚功能 …… (57)
1.3.2　CPU 时钟 …… (59)
1.3.3　指令执行的时间概念 …… (59)
1.3.4　与片外数据存储器接口 …… (61)
1.4　MCU 的休眠方式 …… (62)
1.4.1　闲置方式 …… (62)
1.4.2　掉电方式 …… (63)
1.5　片内存储器的烧录 …… (63)
1.5.1　并行烧录 …… (63)
1.5.2　串行下载 …… (70)

第二章　指令系统

2.1　数据传送指令 …… (74)
2.1.1　工作寄存器间的数据传送指令 …… (74)
2.1.2　工作寄存器的数据装载指令 …… (74)
2.1.3　工作寄存器的数据存储指令 …… (77)
2.1.4　程序存储器的读指令 LPM …… (78)
2.1.5　输入/输出指令 …… (78)

2.1.6 堆栈操作指令…… (79)
2.2 算术和逻辑运算指令…… (81)
2.2.1 加法类指令…… (81)
2.2.2 减法类指令…… (82)
2.2.3 逻辑与指令…… (85)
2.2.4 逻辑或指令…… (86)
2.2.5 逻辑异或指令…… (87)
2.3 分支指令…… (88)
2.3.1 无条件跳转指令…… (89)
2.3.2 子程序调用指令…… (89)
2.3.3 返回指令…… (90)
2.3.4 条件跨越指令…… (90)
2.3.5 基本条件分支指令…… (92)
2.3.6 派生条件分支指令…… (92)
2.4 位操作及其他指令…… (98)
2.4.1 I/O 寄存器位置位和位清 0 指令 …… (98)
2.4.2 状态寄存器位置位和位清 0 指令…… (99)
2.4.3 位传送指令 …… (100)
2.4.4 移位指令 …… (101)
2.4.5 其他指令 …… (103)
第三章 汇编语言程序设计示范
3.1 AVR 程序的书写规则…… (104)
3.1.1 汇编指令 …… (104)
3.1.2 表达式 …… (110)
3.2 BCD 运算 …… (114)
3.2.1 数制转换 …… (114)
3.2.2 BCD 加减运算 …… (118)
3.2.3 BCD 运算测试程序 …… (119)
3.3 多倍精度运算 …… (120)
3.3.1 多字节加减运算 …… (120)
3.3.2 16 位运算 …… (121)
3.4 数据块传送 …… (123)
3.4.1 程序存储器至数据存储器的数据传送 …… (123)
3.4.2 数据存储器两区域间的数据块传送 …… (123)
3.4.3 数据块传送测试程序 …… (124)
3.5 乘除法运算 …… (124)
3.5.1 8 位无符号数乘法 …… (124)
3.5.2 8 位带符号数乘法 …… (126)
3.5.3 8 位无符号数除法 …… (127)

3.5.4　8 位带符号数除法 …… (128)
3.6　数据排序 …… (129)
3.6.1　冒泡排序算法 …… (129)
3.6.2　数据排序测试程序 …… (130)
3.7　数字滤波 …… (131)
3.7.1　去极值平均滤波法 …… (131)
3.7.2　8 点移动平均值滤波法 …… (132)
第四章　片内硬件资源的应用
4.1　对片内 EEPROM 的访问 …… (135)
4.1.1　对片内 EEPROM 的随机读写 …… (135)
4.1.2　对片内 EEPROM 的顺序读写 …… (136)
4.2　端口的通用 I/O 操作 …… (139)
4.3　监视定时器的应用 …… (142)
4.3.1　启用和中期管理 …… (142)
4.3.2　顺序控制 …… (144)
4.4　定时/计数器 0 的应用 …… (147)
4.4.1　定时功能的实现 …… (147)
4.4.2　计数功能的实现 …… (149)
4.5　定时/计数器 1 的应用 …… (151)
4.5.1　定时功能应用实例 …… (151)
4.5.2　输出比较功能的应用 …… (153)
4.5.3　输入捕捉功能的应用 …… (156)
4.5.4　频率测量 …… (159)
4.5.5　脉宽调制 …… (160)
4.6　模拟比较器的应用 …… (162)
4.6.1　模拟比较器的工作设定 …… (162)
4.6.2　简易 A/D 转换器 …… (164)
4.6.3　逐步逼近式 A/D 转换 …… (166)
4.7　串行口的应用 …… (168)
4.7.1　同步串行传送 …… (168)
4.7.2　异步串行传送 …… (170)
4.7.3　软件 UART …… (173)
4.8　休眠方式的应用 …… (178)
4.8.1　掉电方式的进入和退出 …… (178)
4.8.2　闲置方式的进入和退出 …… (180)
第五章　接口技术
5.1　与显示器接口 …… (181)
5.1.1　发光二极管显示器 …… (181)
5.1.2　液晶显示器 …… (185)

5.2 与键盘接口 …… (195)
5.2.1 独立按键接口 …… (195)
5.2.2 矩阵式键盘接口 …… (196)
5.2.3 键盘和显示器复合接口 …… (200)
5.3 与 BCD 码拨盘接口 …… (212)
5.3.1 与端口直接接口 …… (212)
5.3.2 通过门电路接口 …… (213)
5.4 与微型打印机接口 …… (215)
5.4.1 TPμP 微型打印机 …… (215)
5.4.2 AVR 与 TPμP—40A 接口 …… (216)
5.5 片内 A/D 转换器 …… (217)
5.5.1 内部结构及运作原理 …… (217)
5.5.2 ADC 应用程序 …… (222)
第六章 开发工具及应用
6.1 AVR 实时在线仿真器 ICE200 …… (226)
6.2 JTAG ICE 仿真器 …… (226)
6.3 开发下载实验器 SL—AVR+ …… (227)
6.4 AVR 集成开发环境(IDE) …… (232)
6.4.1 AVR Assembler 汇编器 …… (232)
6.4.2 AVR Studio …… (235)
6.4.3 AVR Prog …… (236)

第一章　体系结构及运作原理

AVR 单片机包括 AT90、ATtiny 和和 ATmega 三个系列。其体系结构大同小异，指令系统相互包容。本章将以 AT90 系列为主线介绍 AVR 微控制器的机理。

1.1　AT90 系列

AT90 系列包括 AT90S1200、AT90S2313、AT90S/LS2323 和 AT90S/LS2343、AT90S/LS2333 和 AT90S/LS4433、AT90S4414/8515、AT90S/LS4434 和 AT90S/LS8535 以及 AT90C8534 等众多品牌的微控制器。它们都是 8 位 CMOS 低功耗器件，其基本框架为 AVR RISC 体系结构，即被命名为 AVR 的精减指令集计算机体系结构。

AT90 系列各单片机内含容量不等的闪速程序存储器(Flash Memory，简称 Flash)，可反复擦写至少 1 000 次。这极大地方便了产品开发和软件修改。这种存储器中的程序可由 PC 机串行下载，亦可在通用写入器上以并行方式写入。

AVR 单片机最大的特点是低功耗和高速度。掉电方式、闲置方式至工作方式下的耗电约在 1 μA～2.5 mA 之间。此类单片机每个时钟周期可执行一条指令。在 12 MHz 频率下，指令的吞吐量为 12 MIPS，这是一般 MCS—51 单片机速度的 12 倍。

AVR 片内含有模拟比较器，与输入捕捉功能配合，可进行多种模拟控制和转换。

AVR 单片机含有内部上电复位电路，因此其低电平有效的$\overline{\text{RESET}}$引脚可直接连到 V_{CC} 上，省却片外复位电路的安排。AT90S1200 在片内还有 1 MHz 的 RC 振荡器，这样就可作为无外加元件的真正意义上的单片机用了。

表 1.1 中给出了 AT90 系列各微控制器片内资源的简要情况。

1.2　内部结构

AVR 微控制器大家族体系结构相同，各成员间内部结构稍有差异。本节将以 AT90S4414/8515 为代表，详细讲述 AVR 单片机内部各部件的组织结构及其运作原理。

1.2.1　结构框图

大家知道，计算机的核心为运算器和控制器，两者合称为中央处理单元，即 CPU。所谓单片机，是把 CPU、一定容量的存储器以及若干输入/输出接口等部件集成在一块小小硅片上的微型计算机，简称单片机。因主要用于控制功能，故亦称微控制器，有时写作 MCU。

如图 1.1 所示，作为 AVR 单片机内核的 CPU 包括下列各部件：

- 运算器——算术逻辑单元 ALU 以及与其配合工作的通用寄存器组和状态寄存器；
- 控制器——程序计数器 PC、指令寄存器、指令译码器和定时与控制部件。

AVR 单片机的存储器有用来存放程序的闪速存储器 Flash、存放数据和中间结果的静态

表 1.1　AT90 系列和 ATtiny 系列微控制器简况

型　号	闪存/KB	EEPROM/B	SRAM/B	指令条数	I/O 引脚	引脚总数	外部中断	SPI	UART	8 位定时器	16 位定时器	PWM	监视定时器	10 位 AD 通道	模拟比较器	V_{CC}/V	时钟频率/MHz
AT90S1200	1	64	—	89	15	20	1	√	—	1	—	—	√	—	√	2.7～6 4～6	0～4 0～12
AT90S2313	2	128	128	118	15	20	2	√	√	1	1	√	√	—	√	2.7～6 4～6	0～4 0～10
AT90S2323	2	128	128	118	3	8	1	√	—	1	—	—	√	—	—	4～6	0～10
AT90LS2323	2	128	128	118	3	8	1	√	—	1	—	—	√	—	—	2.7～6	0～4
AT90S2343	2	128	128	118	5	8	1	√	—	1	—	—	√	—	—	4～6	0～10
AT90LS2343	2	128	128	118	5	8	1	√	—	1	—	—	√	—	—	2.7～6	0～4
AT90S2333	2	128	128	118	20	28/32	2	√	√	1	1	√	√	—	√	4～6	0～8
AT90LS2333	2	128	128	118	20	28/32	2	√	√	1	1	√	√	—	√	2.7～6	0～4
AT90S4433	4	256	128	118	20	28/32	2	√	√	1	1	√	√	—	√	4～6	0～8
AT90LS4433	4	256	128	118	20	28/32	2	√	√	1	1	√	√	—	√	2.7～6	0～4
AT90S4414	4	256	256	118	32	40/44	2	√	√	1	1	√	√	—	√	2.7～6 4～6	0～4 0～8
AT90S8515	8	512	512	118	32	40/44	2	√	√	1	1	√	√	—	√	2.7～6 4～6	0～4 0～8
AT90S4434	4	256	256	118	32	40/44	2	√	√	2	1	√	√	8	√	4～6	0～8
AT90LS4434	4	256	256	118	32	40/44	2	√	√	2	1	√	√	8	√	2.7～6	0～4
AT90S8535	8	512	512	118	32	40/44	2	√	√	2	1	√	√	8	√	4～6	0～8
AT90LS8535	8	512	512	118	32	40/44	2	√	√	2	1	√	√	8	√	2.7～6	0～4
AT90C8534	8	512	256	118	7	48	2	—	—	1	1	—	—	8	—	3.3～6	0～1.5
ATtiny10	1	—	—	90	—	8	1	—	—	1	—	—	√	—	√	2.7～5.5 4.0～5.5	0～2 0～6
ATtiny11	1	—	—	90	—	8	1	—	—	1	—	—	√	—	√	2.7～5.5 4.0～5.5	0～2 0～6
ATtiny12	1	64	—	90	—	8	1	√	—	1	—	—	√	—	√	1.8～5.5 2.7～5.5 4～5.5	0～1 0～4 0～8

随机存取存储器 SRAM 以及存放某些常数的电可擦除的可编程只读存储器 EEPROM。

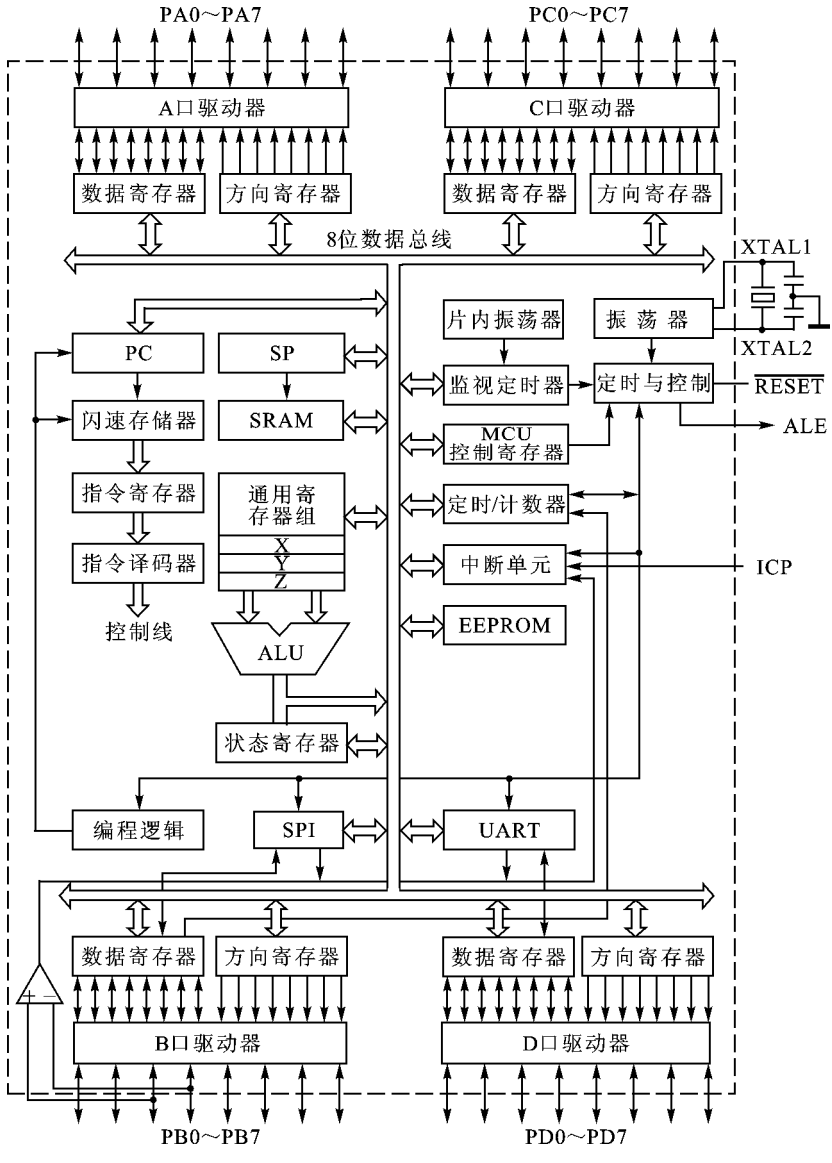


图 1.1　AVR 单片机结构框图

AT90S4414 和 AT90S8515 两机型只是其片内存储器容量不同，其他方面完全一致。前者片内含 4 KB 闪速存储器、256 字节 EEPROM 和 256 字节 SRAM；后者的闪速存储器为 8 KB，其 EEPROM 和 SRAM 均为 512 字节。

CPU 的主要功能是周而复始地读取指令并执行指令。首先由程序计数器 PC 提供行将执行的指令在程序存储器中的地址；该指令代码由那里被读取出来送指令寄存器暂存，进而由指令译码器对此代码进行译码；控制器根据译码结果向运算器各有关部件发出控制信号，最后运算器在这些信号的控制下完成该指令所要求的操作。

单片机与外部世界的沟通靠输入/输出(I/O)接口部件进行。AT90S4414/8515 单片机各

有 A、B、C、D 等四个并行 I/O 端口，辖 32 条 I/O 线。在图 1.1 中这些 I/O 线被标作 PA0～PA7 和 PB0～PB7 等。每个并行端口由驱动器、数据寄存器和数据方向寄存器组成。与外界的串行通讯由异步串行口 UART 和同步串行口 SPI 负责。其他涉外部件尚有定时/计数器和中断单元等。

1.2.2 通用寄存器组合与 ALU

如图 1.2 所示，AVR CPU 含 32 个 8 位通用工作寄存器 R0～R31，它们作为一个整体被称作通用寄存器组合。但其前 16 个寄存器(R0～R15)和后 16 个(R16～R31)之间，有共同点，也有差异。

按 AVR 指令系统的规定，所有涉及到寄存器操作的指令都可直接访问这 32 个工作寄存器。所谓直接访问，是指在 16 位指令代码中，除 11 位操作码用来限定所要求的操作外，尚以 5 位码直接给出该指令所涉及到的寄存器编号 0～31。或者在 16 位代码中有 6 位操作码的情况下，各以 5 位码分别指定目的寄存器和源寄存器编号。以上单寄存器和双寄存器的操作均可在一个时钟周期内完成。

然而，寄存器与立即数间的操作指令，却只能涉及到后 16 个寄存器，即 R16～R31。



图 1.2 AVR CPU 通用工作寄存器

从图 1.2 中可以看到，每个工作寄存器除有编号外，尚分配有一个数据存储器地址。这样，这些寄存器就被直接映射到用户数据空间的最低 32 个地址单元中了。尽管从物理上讲它们并不是像 SRAM 各存储单元那样构成的，但这种存储组织却为访问寄存器提供了很大的灵活性，因为 X、Y 和 Z 三个寄存器可被设置成访问该组合中任一寄存器的地址指针。

组合中最高编号的 8 个寄存器，即 R24～R31，可作 8 位寄存器独立运作，亦可组成寄存器对，作 16 位寄存器使用。这四个寄存器对中，R24:R25(寄存器对的两个字节间以冒号分割)只作数据寄存器。而 R26:R27(合称 X 寄存器)、R28:R29(Y 寄存器)和 R30:R31(Z 寄存器)三个寄存器对在通用功能的基础上又增加了一些专门功能。它们可作对数据空间间接寻址时的地址指针，因此有时也称其为地址寄存器。在不同的寻址方式中，这些地址寄存器可存放基准地址，可自动增量，亦可自动减量。

图 1.3 把图 1.2 中最高编号的六个寄存器画成了 16 位寄存器形式。

应当指出，用户可将任意两个工作寄存器作为寄存器对使用，不过，它们不具备上述三个指针寄存器的功能，也不像 R24:R25 那样有专用指令支持。

如图 1.1 所示，AVR ALU 直接与这 32 个工作寄存器相连，等于有 32 个累加器。这就消除了一些 MCU 中大多数算术逻辑运算都要通过累加器进行的瓶颈现象。此组合中寄存器间的操作，在一个时钟周期内即可完成。ALU 操作有三个主要类型：算术运算、逻辑运算和位操作。

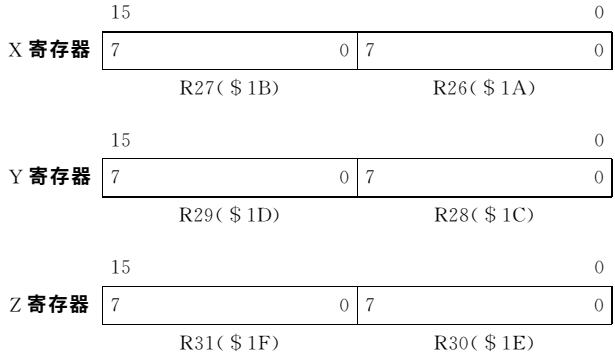


图 1.3　X、Y 和 Z 寄存器

1.2.3　存储器组织

AVR 存储器包括程序存储器 Flash、数据存储器 SRAM 和 EEPROM。

1. 可编程闪速存储器 Flash

AT90S4414 片内含 4 KB 系统内可编程的闪速存储器，简称 Flash，用来存放程序。AT90S8515 有 8 KB Flash。因为所有指令均为 16 位或 32 位字长，所以这里的 Flash 被组织成 2 K×16 和 4 K×16 形式。这种闪速存储器至少可以经受 1 000 次擦写操作。AT90S4414/8515 的程序计数器 PC 为 11/12 位宽，这样就可访问到全部 2 048/4 096 个地址单元。

闪速存储器的程序下载和对程序存储器的寻址方式将另节叙述。

2. 片内和片外 SRAM 数据存储器

图 1.4 所示为 AT90S4414/8515 静态随机存取存储器 SRAM 的组织形式。应当指出，AVR 程序存储器 Flash 和数据存储器 SRAM 分别占据两个独立的地址空间。AT90S4414/8515 Flash 的地址范围为 $000～$7FF/$FFF；而 SRAM 所占空间则是 $0000～$FFFF。由图 1.4 可见，AVR 的片内和片外数据存储器 SRAM 是一个连续的地址空间，总容量可达 64 KB。最低 32 个地址单元 $0000～$001F 系寄存器组合；其后 64 个单元 $0020～$005F 为 I/O 寄存器；片内数据存储器 SRAM 占据此后的 256/512 个地址单元 $0060～$015F/$025F。这样，SRAM 空间的片内部分计有 352/608 个字节。



图 1.4　SRAM 组织

片外 SRAM 根据需要配置，可有可无，可大可小，其最低地址为 $0160/$0260，最高地址可达 $FFFF。指令本身并不区分片内和片外 SRAM，只要访问数据存储空间的地址超出片

内数据 SRAM 的范围，那么这条指令就会自动去访问片外数据 SRAM。当访问片内数据空间时，读写选通信号引脚$\overline{RD}$和$\overline{WR}$在整个访问周期都处于无效状态。片外 SRAM 的操作，可开放也可禁止，这由 MCU 控制寄存器 MCUCR 中的 SRE 位来决定。

与对片内 SRAM 的访问操作相比，每访问片外 SRAM 一个字节，就要多用一个时钟周期。这就意味着，在计算对数据存储空间某字节的读写指令（例如 LD、ST、LDS、STS、PUSH 和 POP）执行时间时，若涉及到片外 SRAM，则勿忘增加一个时钟周期。如果堆栈设置在片外 SRAM 中的话，那么中断响应和返回、子程序调用和返回都要额外多花费两个时钟周期，因为要进行两个字节返回地址的入栈和出栈操作。当片外 SRAM 使用等待状态时，每个字节的操作要增加两个时钟周期。其结果是，数据传送指令要额外多用两个时钟周期，而中断响应和返回以及子程序调用和返回要比指令手册上给出的时间多出 4 个时钟周期。

对数据存储器有五种不同的寻址方式：直接寻址、带偏移量的间接寻址、间接寻址、先减量间接寻址和后增量间接寻址。在寄存器组合中，R26～R31 以可作间接寻址指针寄存器为其特色。

直接寻址方式可以访问整个数据空间。

偏移量间接寻址方式的访问范围可达基准地址后的 63 个地址单元，基准地址由 Y 或 Z 寄存器的内容给出。

当使用先自动减量或后自动增量的间接寻址方式时，地址寄存器 X、Y 或 Z 之内容将自动减量或增量。

AT90S4414/8515 的 32 个通用工作寄存器、64 个 I/O 寄存器、256/512 字节片内 SRAM 以及自由选配的片外 SRAM，均可通过这五种寻址方式访问到。

3. EEPROM 数据存储器

如前所述，AT90S4414/8515 内含 256/512 个字节 EEPROM 数据存储器。它占据独立的数据空间，对此空间可进行单一字节的读写操作。该 EEPROM 的寿命至少可耐 100 000 次擦写操作。有关 CPU 对 EEPROM 的读写、EEPROM 的地址寄存器、数据寄存器和控制寄存器等项内容，稍后将另题讲述。

1.2.4 程序和数据的寻址方式

AVR RISC 微控制器以强力高效的寻址方式支持对程序存储器 Flash 和数据存储器（包括 SRAM、寄存器组合及 I/O 存储器）的访问。寻址方式告诉 CPU 通过什么途径才能找到操作数或下一次行将执行的指令之地址。

1. 单寄存器直接寻址

由图 1.5 可见，在这种寻址方式下，指令的 16 位代码中，其高 11 位为操作码 OP，另 5 位 d 用来直接指定被寻址的寄存器编号。操作数便含在寄存器 d（即 Rd）中。

2. 双寄存器直接寻址

如图 1.6 所示，在双寄存器直接寻址方式下，指令代码中 5 位码 d 用来指定目的寄存器 Rd 的编号，另 5 位码 r 则给出寄存器组合中另一寄存器的编号。两操作数分别位于寄存器 r（即 Rr）和寄存器 d（Rd）中，操作结果将被置于 Rd 内。

应当指出，这些寻址方式图中，d 和 r 在代码中的位置不过是示意性的。在实际代码形成中，d 不一定占据最低 5 位，也有 5 位码不连续的情况，详见指令系统一章。

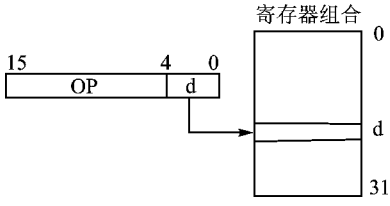


图 1.5　单寄存器直接寻址

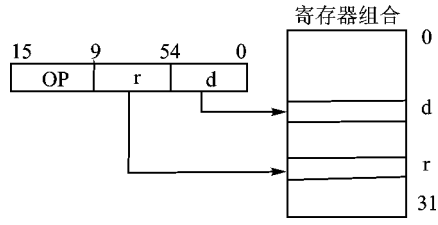


图 1.6　双寄存器直接寻址

3. I/O 直接寻址

这种寻址方式仅用于 IN(输入)和 OUT(输出)指令中。

在图 1.7 所示的这种寻址方式中,指令字的 6 位码 p 代表端口、定时器和配置寄存器等 I/O寄存器地址。5 位码 n 用来指定寄存器组合中某寄存器为目的寄存器 Rd(IN 指令)或源寄存器 Rr(OUT 指令)。

4. 数据直接寻址

这是寄存器组合中某寄存器对数据空间某地址单元进行读写的一种寻址方式。如图 1.8 所示,在双字长的指令代码中,5 位码用来指定某寄存器作目的寄存器 Rd(存放由数据空间读入的数据),或作源寄存器 Rr(把其中数据写入数据空间)。指令代码的低 16 位(16 LSBs)直接给出数据空间某单元地址,数据即从这里读出,或写入此处。

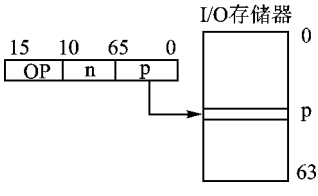


图 1.7　I/O 直接寻址

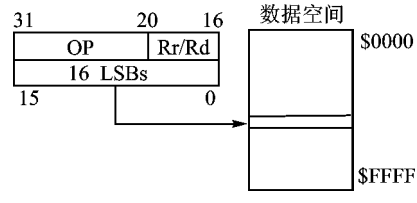


图 1.8　数据直接寻址

5. 带偏移量的数据间接寻址

图 1.9 所示为一种数据间接寻址的实现过程。这里被访问的数据字节之地址不便直接给出,而以基准地址加偏移量的形式间接指定出来。在这种寻址方式下,16 位指令代码中,6 位码 a 作为一个取值范围等于 0～63 的偏移量,而 Y 或 Z 寄存器之 16 位内容为基准地址,两者之和(Y 或 Z+a)便是操作数在数据空间中的地址。指令码中的 n 照例仍负责指定寄存器组合中的某寄存器为目的寄存器 Rd(LDD 指令)或源寄存器 Rr(STD 指令)。

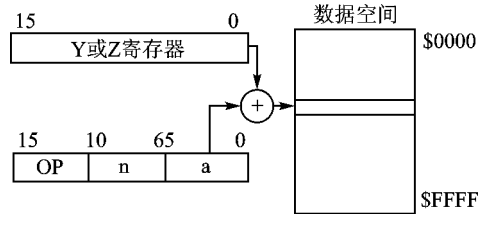


图 1.9　带偏移量的数据间接寻址

6. 数据间接寻址

这是另一种访问数据空间的寻址方式。所谓访问数据空间,就是在寄存器组合和数据空间之间进行数据传送。把数据从数据空间传送到寄存器组合叫作装载(LD,亦称读入);反向传送为存储(ST,写出)。

在数据间接寻址方式下，如图 1.10 所示，X、Y 和 Z 寄存器均可作地址指针寄存器。就是说，它们的 16 位内容便是操作数在存储空间中的地址。

7. 先减量数据间接寻址

有时需要对数据空间中的一个数据块或一张表格进行从其末地址开始的逐项访问，这种寻址方式将为之提供极大的方便。

在这种方式下，如图 1.11 所示，X、Y 或 Z 寄存器的内容在操作前先自动减 1，然后以其减量后的新内容作为操作数在数据空间中的地址。数据或由某源寄存器 Rr 写至此地址单元，或从这里读入到某目的寄存器 Rd 中。

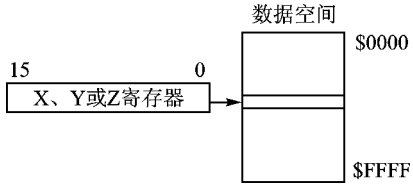


图 1.10 数据间接寻址

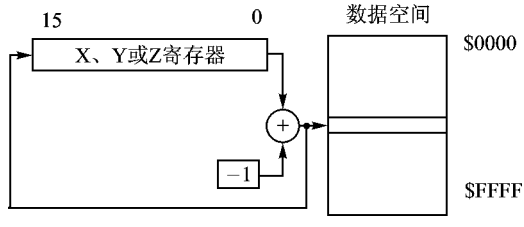


图 1.11 先减量数据间接寻址

当然，在数据间接寻址方式下，不管是偏移量间址、先减量间址、抑或后增量间址，都要在指令代码中以 5 位码指定 Rd 或 Rr。

8. 后增量数据间接寻址

后增量间接寻址方式便于对数据空间中的表格或数据块进行自首地址开始的读写操作。就其寻址过程讲，它和间接寻址并无不同，即以 X、Y 或 Z 等三个地址指针寄存器的内容为操作数的地址。只不过间接寻址后，指针内容保持不变，而后增量间址操作后，指针内容自动增 1，为下次访问做好准备。图 1.12 清楚地演示了这个过程。

9. 程序存储器常数寻址

这是 LPM 指令自程序存储器中读取常数所用的一种寻址方式。前已提及，AVR 单片机的程序存储器每个地址单元存放一个 16 位字。在程序存储器常数寻址方式下，地址指针寄存器 Z 的高 15 位用来指定程序存储器中的某个地址单元（见图 1.13），而 Z 的最低位将对该单元的两个字节做出选择，0 选中低字节，1 选中高字节。

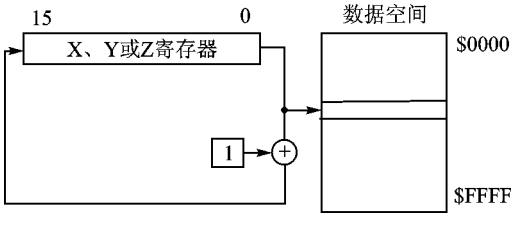


图 1.12 后增量数据间接寻址

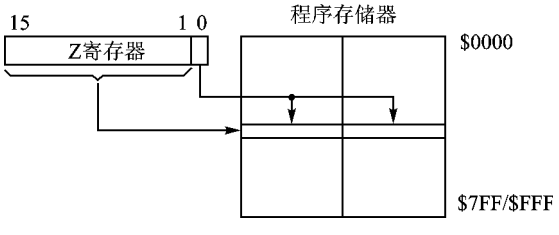


图 1.13 程序存储器常数寻址

10. 程序间接寻址

这是分支指令（跳转 IJMP 和子程序调用 ICALL）所用的一种寻址方式。所谓分支，即从某个非顺序地址单元继续执行程序。由图 1.14 可见，程序间接寻址方式告诉 CPU，地址指针寄存器 Z 的内容代表程序存储空间中的一个地址，接下来应去执行此地址单元所存放的指

令。因为程序计数器 PC 的使命是提供下一次要执行的指令的地址，所以这种寻址方式的运作实际上相当于把 Z 的内容送到 PC 中去。

11. 程序相对寻址

数据寻址是 CPU 寻址操作数的过程；而程序寻址（程序间接寻址、程序相对寻址、有些 AVR 品牌尚用程序直接寻址）则是寻找下一次行将执行的指令代码的过程。在跳转（JMP）和子程序调用（CALL）指令码中，以若干位码的形式直接给出跳转的目的地址或子程序的入口地址，这就是程序直接寻址方式。程序间接寻址则以指针寄存器 Z 的内容作为下一次要执行的指令的地址。

图 1.15 示意了程序相对寻址的运作过程。在执行相对跳转 RJMP 和相对调用 RCALL 指令时，目的地址是以基准地址加偏移量的形式给出的，即 PC＋k＋1。这里 PC 是本 RJMP 或 RCALL 指令代码所在的地址，PC＋1 则为其后一指令的地址，偏移量 k 便由此开始算起。例如，若目的地址为 PC＋2，则 k＝PC＋2－（PC＋1）＝1；若目的地址是 PC－2，则 k＝PC－2－（PC＋1）＝－3，如此等等。不难看出，偏移量是一个带符号的 12 位二进制数，其取值范围为－2 048～2 407。

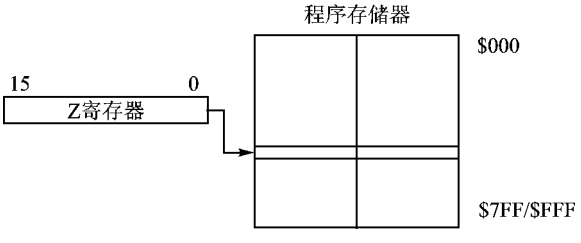


图 1.14　程序间接寻址

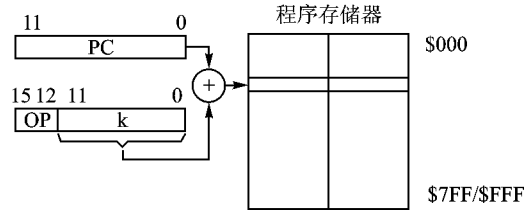


图 1.15　程序相对寻址

1.2.5　I/O 存储器

AT90S4414/8515 的所有 I/O 和涉外寄存器均被安放在了 I/O 空间。用输入（IN）和输出（OUT）指令访问这些 I/O 地址单元，就可在 I/O 空间和 32 个通用工作寄存器间进行数据传送。地址范围为 $00～$1F 的 I/O 寄存器均可按位寻址，即可用 SBI 指令把其中任何一位置 1，用 CBI 指令将之清 0，用 SBIS 和 SBIC 指令对这些寄存器的各位进行测试，看其为 1，抑或为 0。

表 1.2 中列出了 AT90S4414/8515 I/O 空间的所有寄存器。请注意，当使用特定的 I/O 指令 IN 和 OUT 访问这些寄存器时，必须用它们的 I/O 地址；而当把这些 I/O 寄存器作为 SRAM 单元访问时，则应该用它们的 SRAM 地址，即将 I/O 地址加 $20。此表括号中的地址就是各寄存器的 SRAM 地址。

表 1.2　AT90S4414/8515 I/O 寄存器

地　址	符　号	名　称
$3F($5F)	SREG	状态寄存器
$3E($5E)	SPH	堆栈指针高字节

续表 1.2

地 址	符 号	名 称
$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	定时器中断屏蔽寄存器
$38($58)	TIFR	定时器中断标志寄存器
$35($55)	MCUCR	MCU 控制寄存器
$33($53)	TCCR0	定时器 0 控制寄存器
$32($52)	TCNT0	定时/计数器 0(8 位)
$2F($4F)	TCCR1A	定时器 1 控制寄存器 A
$2E($4E)	TCCR1B	定时器 1 控制寄存器 B
$2D($4D)	TCNT1H	定时器 1 高字节
$2C($4C)	TCNT1L	定时器 1 低字节
$2B($4B)	OCR1AH	定时器 1 输出比较寄存器 A 高字节
$2A($4A)	OCR1AL	定时器 1 输出比较寄存器 A 低字节
$29($49)	OCR1BH	定时器 1 输出比较寄存器 B 高字节
$28($48)	OCR1BL	定时器 1 输出比较寄存器 B 低字节
$25($45)	ICR1H	定时器 1 输入捕捉寄存器高字节
$24($44)	ICR1L	定时器 1 输入捕捉寄存器低字节
$21($41)	WDTCR	监视定时器控制寄存器
$1F($3F)	EEARH	EEPROM 地址寄存器高字节(AT90S8515)
$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

续表 1.2

地　址	符　号	名　称
$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 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	模拟比较器控制和状态寄存器

表 1.2 没有列出 I/O 空间尚未派用场的保留地址单元，用户不应对这些保留 I/O 存储地址进行写入操作。如果要向某些含有保留位的 I/O 寄存器进行写入的话，那么，为与未来器件兼容，应把这些保留位写成 0。

表 1.2 中的 I/O 和涉外控制寄存器将分别由后续各节详细讲述，这里我们只介绍其中的 SREG 和 SP 两个。

1. 状态寄存器 SREG

AVR 状态寄存器 SREG 位于 I/O 空间的 $3F($5F)单元，8 位，复位值为 $00，可读亦可写。其各位分布情况如下。

$3F	7	6	5	4	3	2	1	0	
($5F)	I	T	H	S	V	N	Z	C	SREG

位 7——总体中断开放位 I：这是一个中断总开关，它置 1 是开放所有中断的必要条件。在此基础上，各中断源的开放与关闭由各个控制寄存器中的分开关来完成。如果此总体中断允许位被清 0，则各中断源均被关闭，这与各中断源的中断允许位，即上述分开关的状态无关。I 位在某中断获得响应后由硬件自动清 0，中断结束时又被 RETI 指令置 1，以开放此后发生的中断。

位 6——位中转站 T：位复制指令 BLD(位装载)和 BST(位存储)使用 T 位作被操作位的源或目的。用 BST 指令可把寄存器组合中某寄存器的一位复制到 T 位来；BLD 指令则把 T 位之值拷贝到某工作寄存器的一位中去。

位 5——半进位标志 H：在某些算术运算中，H 标志表明是否有低半字节向高半字节的进位或借位。若有，则 H=1；若无，则 H=0。

位 4——符号位 S：符号位总是负标志 N 和补码溢出标志 V 的异或结果。若为 1，则表明运算结果为负值，否则即为正值。

位 3——补码溢出标志 V：V 标志支持补码算术运算。执行过带符号数的加法或减法指令后，若 V 为 1，则表明运算结果发生了溢出，即超出了一个字节或一定位数所能表示的数值

范围。例如,8 位带符号数的数值范围为－128～127,若运算结果为－129 或 128,则 V＝1,说明有溢出发生。

位 2——负标志 N:若负标志 N 为 1,则说明此前的算术和逻辑运算结果为负数。N＝0 表示运算结果为正数。

位 1——0 标志 Z:此标志为 1,表示此前的算术或逻辑运算结果等于 0。当然 Z＝0,说明运算结果非 0。

位 0——进位标志 C:这是算术运算有无进、借位的一个标志。若有进、借位,则 C 标志置 1,否则清 0。

2. 堆栈指针 SP 和 AVR 堆栈

AVR 16 位堆栈指针 SP 是由地址为 $3E($5E)和 $3D($5D)的两个 8 位寄存器组成的。占据高地址 $3E 者为 SP 的高字节,记作 SPH;占据低地址 $3D 者系其低字节 SPL。

堆栈指针指向子程序堆栈和中断堆栈所在的数据 SRAM 区。SP 的复位值为 $0000,由后面所讲内容可知,那里是不能作堆栈的。因此必须在初始化程序段中,用软件对堆栈指针 SP 的内容重新做出安排,最好使其指向数据 SRAM 区的最高地址。

程序所设置的 SP 初值即为堆栈的栈底地址。空栈时,栈底与栈顶是重合的。在 PUSH 指令把某 8 位数据压入堆栈后,SP 内容自动减 1,指向紧邻此数据的一个低地址空白单元。在调用子程序和中断响应时,16 位返回地址被自动压入堆栈保存起来,这时 SP 值减 2。当 POP 指令将栈顶数据弹出堆栈时,SP 内容先自动加 1;而当子程序返回指令 RET 或中断返回指令 RETI 执行时,16 位返回地址由栈顶弹出,故 SP 值加 2。

根据以上讲述不难得出结论,AVR 堆栈是向低地址方向生成的。存入堆栈的数据越多,栈顶地址越低。如果把最后一个存入堆栈的数据所在位置称作栈顶的话,那么 SP 内容等于栈顶地址减 1。或者说,堆栈指针指向紧邻栈顶的一个低地址空白单元。

应当指出,AVR 堆栈可设置在片内 SRAM 区,亦可安排在片外数据 SRAM 区。若在片外,则如 1.2.3 一节之 2 所述,某些操作的执行时间将会有所延长。

1.2.6 定时/计数器 0

AT90S4414/8515 向用户提供了两个通用定时/计数器——一个 8 位的定时器 0 和一个 16 位的定时器 1。此二定时器由同一个 10 位定标计时器获得时钟源,并可经此定标器进行量程选择。所谓定时/计数器,顾名思义,可作定时器用,以内部时钟信号为时基;也可作计数器用,以来自外部引脚的脉冲信号触发计数过程。

1. 定标器

定标器实际上也是一个计时器。如图 1.16 所示,AVR 两定时器所配的定标器是一个 10 位计时器,它对系统时钟 CK 进行分频,分频因子可为 8、64、256 和 1 024。因此,定标器可向两定时器提供 CK/8、CK/64、CK/256 和 CK/1 024 等不同频率的时钟源,这样就可改变定时器的量程。

图 1.16 还显示,除上述不同频率的时钟源外,两定时器尚可由定标器得到 CK、外部时钟以及 0 等时钟源。后者使定时器停止运作。

2. 定时/计数器 0(T/C0)

如图 1.17 所示,定时/计数器 0 主要由 8 位计数器 TCNT0、控制逻辑和 T/C0 控制寄存

器 TCCR0 三个部分组成。下面通过对这三个部件的介绍来讲述 T/C0 的运作机理。

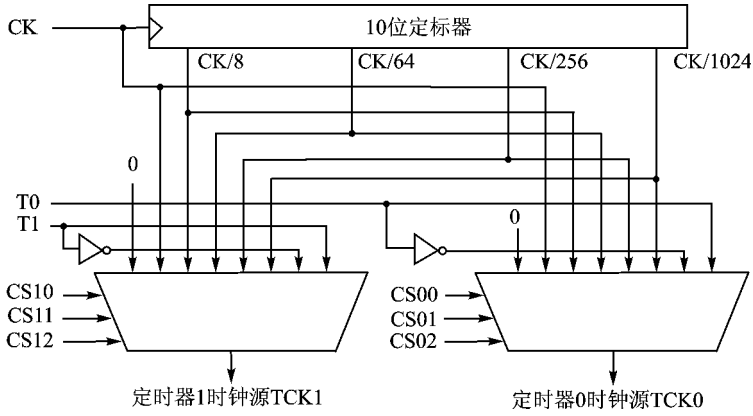


图 1.16　定时/计数器定标器

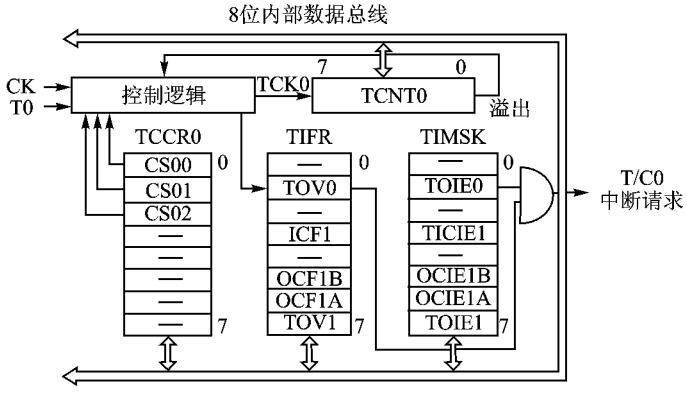


图 1.17　定时/计数器 0 结构框图

(1) TCNT0：这是 T/C0 的 8 位计数器，它对控制逻辑提供的时钟源 TCK0 进行递增计数。当其计数值由 $FF 增至 $00，即发生溢出时，它把定时器中断标志寄存器 TIFR 中的定时器 0 溢出标志 TOV0 置成 1。若 SREG 中的总体中断开放位已被置 1，定时器中断屏蔽寄存器 TIMSK 中的定时器 0 中断允许位 TOIE0 也为 1，则已经置 1 的 TOV0 标志将向 MCU 发出中断请求。

TCNT0 的复位值为 $00，也可用软件向其中写入所需要的任何 8 位值。在写操作后的时钟周期，TCNT0 将继续计数。TCNT0 值亦可用软件读出。

(2) TCCR0：T/C0 控制寄存器，8 位，I/O 地址为 $33($53)，复位值等于 $00。其高 5 位为保留位，恒读作 0，不可写入。其低 3 位自高至低分别是 CS02、CS01 和 CS00，可读亦可写。这是三个时钟选择位，它们将通过控制逻辑为 T/C0 选择适当的时钟源。

(3) 控制逻辑：主要由图 1.16 所示的定标器构成，其任务之一是根据两组时钟选择位的控制，向 T/C0 和 T/C1 提供合适的时钟源。CS02、CS01 和 CS00 三位对 T/C0 时钟源的选择如表 1.3 所列。

[例 1.1]　若 AT90S4414 的时钟频率为 4 MHz，则其 T/C0 的最大定时间隔等于多少？

表 1.3 T/C0 时钟源选择

CS02	CS01	CS00	时钟源 TCK0
0	0	0	时钟源为 0,令 T/C0 停止运作
0	0	1	CK,即单片机时钟
0	1	0	CK/8,对 MCU 时钟 8 分频
0	1	1	CK/64,对 MCU 时钟 64 分频
1	0	0	CK/256,对 MCU 时钟 256 分频
1	0	1	CK/1 024,对 MCU 时钟 1 024 分频
1	1	0	T0 引脚上的外来时钟,下降沿触发
1	1	1	T0 引脚上的外来时钟,上升沿触发

欲得最大定时间隔,第一,必须选频率最低,即周期最长的时钟源,所以应选 CK/1 024 作 TCK0。系统时钟周期为 1/4 MHz=0.25 μs,TCK0 的周期等于 0.25 μs×1 024=256 μs。第二,8 位 TCNT0 的初值应最小。其复位值为 $00,恰可满足此要求。这样,每过 256 个 TCK0 周期,TCNT0 即要溢出。因此最大定时间隔为 256 μs×256=65.536 ms。为达到此目的,应使 TCCR0 值等于 $05(见表 1.3),且以复位值 $00 作 TCNT0 初值。

[例 1.2] 在 CK 为 4 MHz 的情况下,欲使 T/C0 每过 4 ms 请求一次中断,应如安排 TCNT0 和 TCCR0 值?

若将 TCCR0 值写成 $03,则 TCK0 频率将是 4 MHz/64=62.5 kHz,取其倒数,即得 TCK0 周期为 16 μs。再把 TCNT0 初值改写成 $06,则过 250 个周期后,TCNT0 值将为 0,于是便请求溢出中断。这样,周期性地将 TCNT0 初值置成 $06,即可使之每过 16 μs×250=4 ms请求一次中断。

不难理解,分频因子低,TCK0 周期短,可得比较高的分辨率;而分频因子高,则 TCK0 周期长,可得比较大的定时间隔。

表 1.3 最后两行选择的是来自 T0 引脚的外来时钟,这是计数工作方式。外来时钟脉冲要经内部时钟同步,即在 CPU 时钟脉冲的上升沿时对来自 T0(PB0)引脚的时钟信号进行采样。若某上升沿采样到外来信号的高电平,而此后某个上升沿时又发现外来信号为低电平,则两次采样得到了一个外来时钟脉冲的负跳变。如果 TCCR0 值等于 $06 的话,那么这时 TCNT0 值便要加 1。同理,若 TCCR0 值为 $07,则每遇外来时钟脉冲的一个正跳变,TCNT0 值便要加 1。当其值由 $FF 增至 $00 时,照例这将把 TOV0 标志置 1,去向 CPU 请求溢出中断。

为保证对外来时钟脉冲的正确采样,该脉冲信号的高、低电平持续时间必须至少等于一个 CPU 时钟周期,否则就有可能发现不了某些跳变的发生。

[例 1.3] 某脉宽等于 200 ns 的脉冲序列可否在时钟频率为 4 MHz 的 AT90S4414 上通过 T/C0 进行正确的计数?

CPU 时钟周期为 250 ns,而外来时钟脉冲宽度仅为 200 ns,若这样的正脉冲出现在 CPU 时钟的两次上升沿之间,则两次采样值均为低电平,就是说,这个脉冲根本就没有被发现。因此,该问题的答案是否定的。

[例 1.4]　用 AT90S8515 的 T/C0 对某频率相对较低的外来脉冲序列进行计数，要求连续捕捉到 16 个脉冲上升沿时便向 CPU 报告。

要 T/C0 计数，且外来脉冲上升沿使其值加 1，从表 1.3 中可以查到，这时的 TCCR0 值应为 $07。若将 TCNT0 的初值写成 $F0，则捕捉到 16 个脉冲上升沿时，TCNT0 值溢出，变成 $00，定时器中断标志寄存器 TIFR 中的定时器 0 溢出标志 TOV0 被硬件自动置 1，向 CPU 请求中断，报告已有 16 个脉冲到来。

最后再强调一次，外来脉冲应由 T0(PB0)引脚输入到 T/C0 中来。但在计数方式下，PB0 引脚被设定为输出亦无妨，它上面所输出的信号跳变照样可被计数。

1.2.7　定时/计数器 1

定时/计数器 1(T/C1)，16 位，除定时和计数外，又增加了输入捕捉、输出比较和脉冲宽度调制等三项新的功能。下面将以图 1.18 所示的 T/C1 逻辑结构为基础，来讨论这些功能的运作机理。

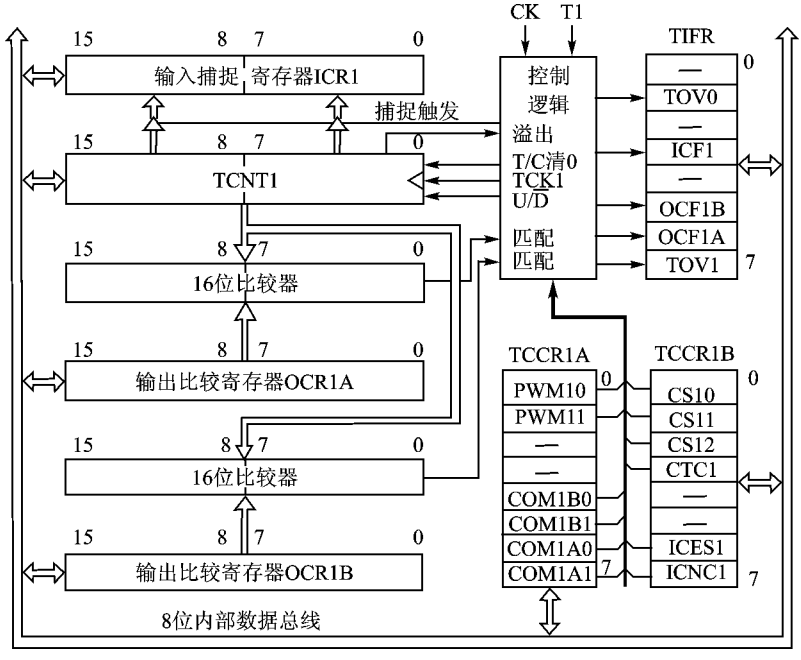


图 1.18　定时/计数器 1 结构框图

1. 定时和计数

如图 1.18 所示，T/C1 的核心是一个 16 位计数器 TCNT1，它可读亦可写，其复位值为 $0000，通常作递增计数器使用。TCNT1 由地址等于 $2D($4D)的高字节 TCNT1H 和地址为 $2C($4C)的低字节 TCNT1L 组成。当 CPU 访问这些寄存器时，为保证高、低字节同时读出或写入，T/C1 内部安排了一个 8 位暂存器 TEMP。对 TCNT1 两个字节的访问就是在它的帮助下进行的。此暂存器在 CPU 访问输出比较寄存器 OCR1A、OCR1B 以及输入捕捉寄存器 ICR1 时也要用到。

应当指出，当主程序和中断服务程序都要访问上述使用 TEMP 的寄存器时，在访问前要

暂时关闭中断。

(1) TCNT1 的写入:当 CPU 写高字节 TCNT1H 时,被写数据暂存 TEMP 中。待 CPU 写低字节 TCNT1L 时,该字节数据将与 TEMP 暂存器中先期存放的高字节数据合成一体,于是 16 位数据便同时写入到 T/C1 的 TCNT1 中。因此,欲进行 16 位寄存器的写操作,必须首先写高字节。

如果在时钟源已选定的情况下向 TCNT1 写入新值,那么,在写入操作完成后的那个 T/C1时钟周期,TCNT1 将以新值为基础继续计数。

(2) TCNT1 的读出:当 CPU 读其低字节 TCNT1L 时,这里的数据被传送至 CPU,而其高字节 TCNT1H 中的数据被暂存入 TEMP 中。待到 CPU 读高字节 TCNT1H 的数据时,CPU 收到的是 TEMP 中的数据。因此,为对 16 位寄存器 TCNT1 进行读操作,必须先读其低字节 TCNT1L。

(3) 时钟源选择:从图 1.16 和图 1.18 中可以看出,在 TCCR1B 寄存器的控制下,由定标器向 T/C1 提供时钟源 TCK1。T/C1 控制寄存器 TCCR1B 地址为 $2E,复位值等于 $00。除两保留位外,其余 6 位可读亦可写。CS12、CS11 和 CS10 为时钟选择位。表 1.4 中列出了这三位不同组合值对时钟源 TCK1 的选择结果。

	7	6	5	4	3	2	1	0	
($2E)	ICNC1	ICES1	—	—	CTC1	CS12	CS11	CS10	TCCR1B

表 1.4 T/C1 时钟源选择

CS12	CS11	CS10	时钟源 TCK1
0	0	0	0 时钟源使 T/C1 停止运作
0	0	1	CK,即 MCU 时钟
0	1	0	CK/8,对 MCU 时钟 8 分频
0	1	1	CK/64,对 MCU 时钟 64 分频
1	0	0	CK/256,对 MCU 时钟 256 分频
1	0	1	CK/1 024,对 MCU 时钟 1 024 分频
1	1	0	引脚 T1 上的外来时钟,下降沿触发
1	1	1	引脚 T1 上的外来时钟,上升沿触发

(4) 定时功能:T/C1 运作的实质是对时钟源 TCK1 的脉冲信号进行计数。若 TCK1 系某已知固定周期的脉冲信号,则这种计数工作便是定时。因此,若选 CPU 时钟 CK 或经分频后的 CK 脉冲信号作为 T/C1 的时钟源 TCK1,那么这就自动选定了 T/C1 的定时功能。这时 TCNT1 值将随每个 TCK1 时钟脉冲的到来而递增。当此值由 $FFFF 递增至 $0000 时,T/C1发生溢出,TIFR 寄存器中的 T/C1 溢出标志 TOV1 被硬件置 1,于是向 CPU 发出溢出中断请求,告知定时间隔已到。

在 CPU 用 4 MHz 时钟的情况下,T/C1 计时的最小单位(分辨率)可为 0.25 μs、2 μs、16 μs、64 μs 和 256 μs。

[例 1.5] CPU 时钟频率为 8 MHz,欲使 T/C1 启动后 10 ms 发出中断请求,应对

TCCR1B和 TCNT1 两寄存器做何安排？

如果 T/C1 计时的分辨率为 1 μs，那么当 TCNT1 的初值等于 65 536－10 000＝55 536＝$D8F0 时，T/C1 便可在启动后 10 ms 产生溢出中断。由表 1.4 中查到，当 TCCR1B 最低 3 位被写成 010 时，TCK1 频率将等于 1 MHz，即最小计时单位为 1 μs。再先后向 TCNT1H 和 TCNT1L 写入 $D8 和 $F0 即可达到目的。

(5) 计数功能：表 1.4 的最后两项是 T/C1 的计数运作方式。此时外来脉冲信号应由 T1(PB1)引脚输送进来。其他诸如采样、增值以及对外来信号的要求等方面与 T/C0 计数完全一致。

2. 输出比较

由图 1.18 中可以看到，T/C1 计数器 TCNT1 的内容可在两个比较器中分别与输出比较器 OCR1A 和 OCR1B 之值进行比较。若 TCNT1 与某输出比较寄存器值相等，则相应比较器便向控制逻辑发出匹配信号。控制逻辑收到匹配信号后要做两件事情：

第一，若系 A 匹配，即 TCNT1 与 OCR1A 两值相等，则按软件的设定，通过 OC1A 引脚(PD5)输出高电平、低电平或将其原输出反相。此外尚可根据软件设定，将 TCNT1 值清 0。若系 B 匹配，则只对 OC1B(独立输出引脚)进行置 1、清 0 或取反三种操作。

第二，把定时器中断标志寄存器 TIFR 中的有关中断标志 OCF1A 或 OCF1B 置 1，使之去请求中断。

(1) 对输出比较引脚操作的选择：匹配时对输出比较引脚所进行的操作，由 T/C1 控制寄存器 TCCR1A 进行选择。

TCCR1A 寄存器，8 位，地址为 $2F($4F)，6 位可读亦可写，2 保留位只可读，且恒为 0。TCCR1A 寄存器的复位值等于 $00。

$2F	7	6	5	4	3	2	1	0	
($4F)	COM1A1	COM1A0	COM1B1	COM1B0	—	—	PWM11	PWM10	TCCR1A

位 7 和位 6——COM1A1 和 COM1A0：这是输出比较方式的两个控制位，由它们确定 A 匹配后对输出比较 A 引脚 OC1A(PD5)应进行的操作(见表 1.5)。这是 I/O 口引脚 PD5 的第二功能，此时要求把相应的方向控制位置 1。

表 1.5　输出比较方式选择

COM1X1	COM1X0	选中方式
0	0	T/C1 与输出引脚 OC1X 脱钩
0	1	OC1X 输出反相
1	0	OC1X 输出 0
1	1	OC1X 输出 1

位 5 和位 4——COM1B1 和 COM1B0：此两控制位用来设定 B 匹配时对输出比较引脚 OC1B 应进行的操作(在表 1.5 中，X 代表 A 或 B)。

(2) A 匹配时 TCNT1 值清 0 与否的选择：在前面的时钟源选择一小节中，我们谈到了控制寄存器 TCCR1B，其位 3 为 CTC1。当控制位 CTC1 置 1 时，TCNT1 值在 A 匹配发生后的

一个时钟周期中被复位成 $0000。若 CTC1 位被清成 0，则 TCNT1 值不受比较匹配的影响，将在原基础上继续计数。因为比较匹配总是在匹配发生后的一个 CPU 时钟周期被检测出来，所以 TCNT1 值被复位成 0 这项功能，在定标器分频因子大于 1 时，将有不同表现。如果分频因子等于 1，且 OCR1A 寄存器值被设置为 c，那么在 CTC1 位已置 1 的情况下，TCNT1 将有如下的计数过程：

……|c－2|c－1|c|0|1|……

而当定标器分频因子为 8 时，TCNT1 的计数过程是：

……|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|……

(3) T/C1 输出比较寄存器 OCR1A 和 OCR1B：两者均为 16 位读/写寄存器，复位值都等于 $0000。OCR1A 由地址等于 $2B($4B)的高字节 OCR1AH 和地址为 $2A($4A)的低字节 OCR1AL 组成。OCR1B 的高 8 位 OCR1BH 占据地址 $29($49)，低 8 位 OCR1BL 位于 I/O 空间的 $28($48)地址单元。

T/C1 输出比较寄存器中存放的数据不断与 TCNT1 值进行比较。比较匹配时应发生的动作，如前所述，由 TCCR1A 和 TCCR1B 两控制寄存器的有关位设定。应当强调指出，比较匹配只在 T/C1 计数过程中当 TCNT1 内容递增至 OCR1X 值时发生。用软件把两者写成相同数值是不会产生比较匹配的。

在比较匹配发生后的 CPU 时钟周期中，比较匹配中断标志 OCF1A 或 OCF1B 被硬件自动置 1。它们可向 MCU 请求中断。

输出比较寄存器 OCR1A 和 OCR1B 均为 16 位寄存器。如前所述，对它们进行写操作时，为保证高、低字节同时更新，必须先写高字节。当 CPU 向 OCR1XH 写数据时，数据字节暂存在 TEMP 寄存器中。待 CPU 写低字节 OCR1XL 时，TEMP 中暂存的数据便同时被写入 OCR1XH 中。

[例 1.6] AT90S8515 微控制器所用的时钟频率为 8 MHz，如何在其 OC1A 引脚上输出一个频率等于 1 kHz 的方波脉冲序列？

- OC1A(PD5)被设定为输出引脚；
- 把 TCCR1B 寄存器的最低 3 位写成 001，即 T/C1 也用 8 MHz 时钟；
- TCCR1A 最高两位写成 01，令 OC1A 引脚电平每遇 A 匹配时便反相一次；
- 向 OCR1AH 寄存器中写入 $0F；
- 向 OCR1AL 寄存器中写入 $A0，$0FA0 即十进制数 4 000；
- 将 TCCR1B 寄存器中的 CTC1 位写成 1，使 TCNT1 值在 A 匹配时复位成 $0000。

应当指出，上述诸点只不过是为叙述方便而分的，并非代表实施的前后顺序。

[例 1.7] 如何在 OC1B 引脚上实现前例所要求的功能？

因为 B 匹配时没有使 TCNT1 复位成 $0000 的功能，所以必须借助中断来修改 OCR1B 寄存器值。其初值设定为 $0FA0，这样，过 4 000 个时钟周期，即 500 μs 后，TCNT1 与 OCR1B 两寄存器值相等，此时 OCR1B 引脚上原输出电平反相，并产生中断。在中断服务程序中，把 OCR1B 寄存器的内容加 $0FA0，使 OC1B 引脚再过 500 μs 后又反相一次。若此过程循环不已，则可在 OC1B 引脚上得一频率为 1 kHz 的方波脉冲序列。

3. 输入捕捉

当输入捕捉引脚 ICP 上发生软件所设定的事件时，T/C1 的控制逻辑发出捕捉触发信号（见图 1.18），打开计数器 TCNT1 与输入捕捉寄存器 ICR1 之间数据传送通路上的三态门。于是事件发生时刻的计时或计数值便被捕捉到 ICR1 寄存器中。同时控制逻辑也把定时器中断标志寄存器 TIFR 中的输入捕捉标志 ICF1 置 1，令其去请求中断。

（1）输入捕捉事件的定义：输入事件的定义是由 T/C1 控制寄存器 TCCR1B 的最高两位确立的。

位 7——输入捕捉噪音消除控制位 ICNC1：若为 0，则不启用输入捕捉触发时的噪音消除功能。欲启用此项功能，必须将 ICNC1 位置 1。

位 6——输入捕捉边沿选择位 ICES1：若此位为 0，则输入捕捉引脚 ICP 上的信号下降沿被定义为一次事件；若 ICES1 位置 1，则上升沿为有效的捕捉事件。当此位所定义的事件发生时，TCNT1 的计时或计数值便被传送到输入捕捉寄存器 ICR1 中去。有时也说事件的发生触发输入捕捉操作。

（2）T/C1 输入捕捉寄存器 ICR1：这是一个 16 位的只读寄存器，其复位值等于 $0000。ICR1 由地址为 $25（$45）的高字节 ICR1H 和占据地址 $24（$44）的低字节 ICR1L 两个 8 位寄存器组成。为同时读出 ICR1 中两个字节的捕获值，必须遵守前面交代过的规则，即先读 ICR1L，再读 ICR1H。当 CPU 读低字节 ICR1L 时，其数据被送往 CPU，而高字节 ICR1H 的内容被暂存于 TEMP 中。待到 CPU 读 ICR1H 时，它得到的将是来自 TEMP 的数据。

（3）输入捕捉引脚 ICP：这是 AVR 单片机的一个独立输入引脚，其结构逻辑如图 1.19 所示。在 ICP 引脚上检测到的事件，即 ICES1 位所定义的输入信号有效边沿，触发 ICR1 对 TCNT1值的捕捉操作，并同时将 ICF1 标志置 1。

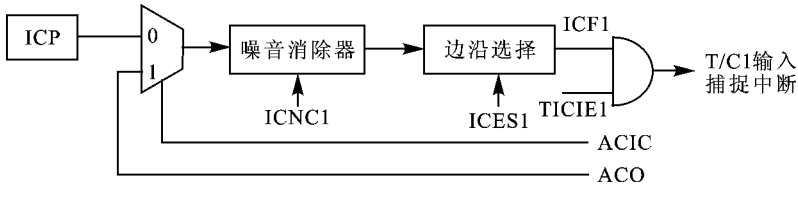


图 1.19　ICP 引脚逻辑

由图 1.19 可见，除 ICP 引脚上的外来输入信号能触发捕捉操作外，AVR 片内模拟比较器也可引起这种捕捉操作。若模拟比较器输入捕捉开放位 ACIC 为 1，则 ICP 与其内部电路脱钩，而将其取而代之者乃是被多路开关选中的模拟比较器的输出信号 ACO（详见 1.2.12 节）。不言而喻，ACIC 位等于 0 时，多路开关将选中 ICP 上的输入信号。

（4）输入捕捉操作中的噪音消除过程：我们以下列假设为前提条件来讨论图 1.20 所示噪音消除电路的工作原理。

- ICNC1 位为 1，启用噪音消除电路，即把 2 号多路开关的输出与 ICF1 接通。
- ACIC 位（见图 1.19）清 0，以 ICP 作触发源。
- ICES1 位置 1，定义 ICP 脉冲上升沿为触发事件，把 PA 与 ICF1 接通。
- ICP 信号初始状态为低电平。

假设某一时刻，ICP 引脚上的输入信号由低电平跳变成了高电平，那么一个时钟周期后，

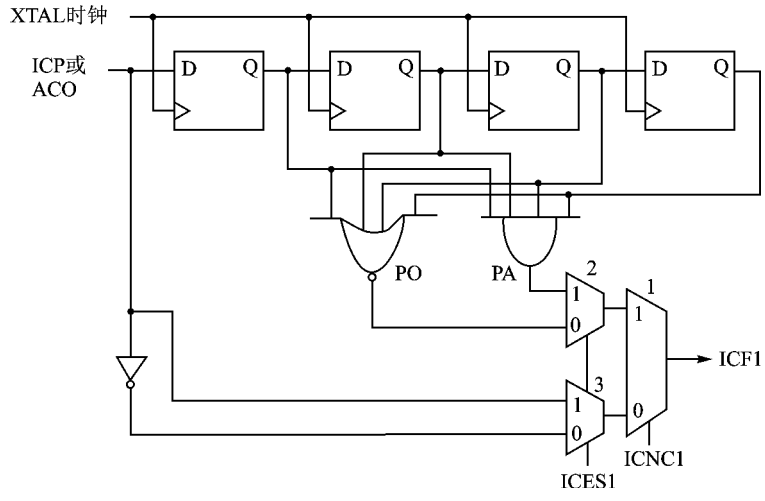


图 1.20　输入捕捉中的噪音消除电路

该高电平便传到最左面一个 D 触发器的输出端 Q。4 个时钟脉冲过后，各 D 触发器均变成为 1 态，这时因与门的 4 个输入均为高电平，故其输出 PA 为 1。后者把 ICF1 标志置 1，并同时触发输入捕捉操作。但如果这个正跳变系噪音干扰所致，那么一般说来，干扰信号非常短暂，不到 4 个时钟周期就会消失。例如，三个时钟周期后，ICP 信号又回到了低电平状态。第四个时钟脉冲到来时，右面三个触发器的输出 Q 仍为高电平，而最左面一个 Q 输出却变成了低电平，于是 PA 不会输出 1 去置位 ICF1。就是说，正跳变后的高电平维持不了四个时钟周期是不算数的，这就消除了噪音的干扰。

现在我们把最后两项前提条件修改一下，ICES1 位清 0，定义 ICP 信号下降沿为触发事件，ICP 信号初态为高电平。

由图 2.20 中可以看到，ICES1 为 0，把或非门的输出 PO 与 ICF1 接通了。如果某时刻 ICP 引脚信号跳变成了低电平，4 个采样周期后，只有 4 个 D 触发器均为 0 态时，或非门输出 PO 才会变成 1 态去置位 ICF1 标志。若 ICP 信号负跳变后的低电平维持不了 4 个采样周期，则 PO 不会把 ICF1 标志置 1。就是说，此负跳变作为一个噪音信号被过滤掉了。

至于 ICNC1 位清 0，1 号多路开关把 3 号多路开关的输出与 ICF1 接通的情况，这里不再讨论，因为这种情况下不启用噪音消除电路。

[例 1.8]　如何测出 ICP 引脚上的输入脉冲宽度？

- 安排 T/C1 以一定的分频因子作定时运行；
- 将 TCCR1B 寄存器的 ICES1 位置 1，定义上升沿为事件；
- 在输入捕捉中断服务程序中，读出 ICR1 值，把 ICES1 位清 0，令下降沿为触发事件；
- 在中断服务程序中读出 ICR1 值，再次将 ICES1 置 1；
- 下降沿触发的捕获值减上升沿捕获值，即两次读到 ICR1 值之差便是被测脉冲宽度。

4. 脉宽调制 PWM 输出

当对控制寄存器 TCCR1A 的最低两位 PWM11 和 PWM10 赋值非 00 时，脉冲宽度调制 PWM 方式即被选中(见表 1.6)。在此方式下，定时器 1、输出比较寄存器 OCR1A 和 OCR1B 以及两个 16 位比较器等部件配套运作，构成两套 8 位、9 位或 10 位 PWM 电路。它们分别以

OC1A(PD5)和 OC1B 两引脚作其输出。此 PWM 电路自由运行,输出波形中无混杂脉冲,相位准确。

表 1.6　PWM 方式选择

PWM11	PWM10	PWM 方式
0	0	禁止 T/C1 的 PWM 运作
0	1	T/C1 为 8 位 PWM
1	0	T/C1 为 9 位 PWM
1	1	T/C1 为 10 位 PWM

在 PWM 方式下,TCNT1 作递增/递减计数运行,由 $0000 递增计数至顶值(见表 1.7),此后开始递减计数至 $0000,如此循环,周而复始。当 TCNT1 之计数值与 OCR1A 或 OCR1B 寄存器的低 10 位值相等时,根据控制寄存器 TCCR1A 最高 4 位的设定,OC1A(PD5)或 OC1B 引脚电平将被置 1 或清 0。

表 1.7　计数顶值及 PWM 频率

PWM 分辨率	计数顶值	PWM 频率
8 位	$00FF(255)	TCK1/510
9 位	$01FF(511)	TCK1/1 022
10 位	$03FF(1023)	TCK1/2 046

如表 1.8 所列,在 PWM 方式下,TCCR1A 寄存器中的 COM1A1、COM1A0、COM1B1 和 COM1B0 等 4 位具有与比较输出方式不同的意义。

表 1.8　PWM 输出方式选择(X=A 或 B)

COM1X1	COM1X0	OC1X 输出
0	0	与 OC1X 无连接
0	1	与 OC1X 无连接
1	0	递增计数匹配时清 0,递减计数匹配时置 1(正向 PWM)
1	1	递减计数匹配时清 0,递增计数匹配时置 1(倒向 PWM)

如何保证在 PWM 输出波形中无混杂脉冲出现,这是一个重要问题。在 PWM 方式下,对 OCR1X 的低 10 位进行写入时,根据片内硬件的安排,这 10 位数据暂先被传送至暂存器内,而当 TCNT1 计数到达顶值时,此数据才被锁进 OCR1X 寄存器中。正是这种 OCR1X 的同步锁存才预防了宽度不定的 PWM 脉冲(杂脉)的出现。

向 OCR1X 写入数据自动经过暂存器缓冲,在计数顶值时,新数据被锁入目的寄存器,如图 1.21 所示,这种同步锁存是不会产生杂脉的。但如果不采取这种同步锁存措施,而是随机写入的话,那么就会出现图 1.21 中那样其宽度无规律可循的混杂脉冲。

在写和锁存操作之间这段时间,读 OCR1X 寄存器之所得将是暂存器的内容。这意味着,尽管最新写入值尚未真正进入 OCR1X,但仍可从此寄存器读得。

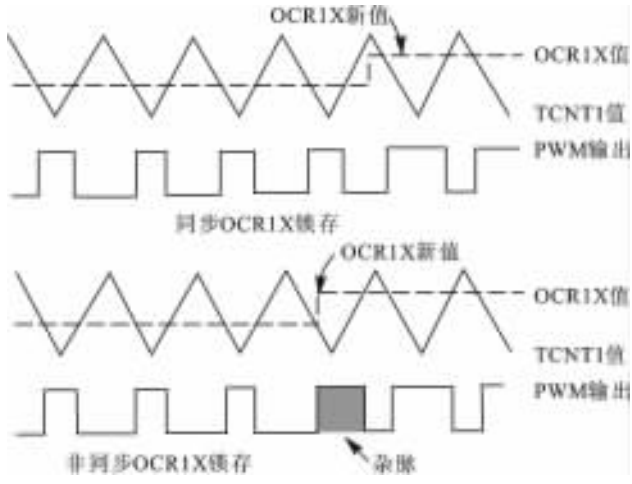


图 1.21　预防杂脉产生的措施

[例 1.9]　图 1.21 所示的 PWM 方式对应于表 1.8 中的哪种情况？

从图 1.21 中可以看到，在 TCNT1 递增计数过程中其值与 OCR1X 内容相等时，OC1X 引脚输出低电平，而在递减计数过程中出现比较匹配时，OC1X 输出高电平。这显然对应的是表 1.8 中 COM1X1 和 COM1X0 等于 10 的情况，即所谓正向 PWM 方式。

不难理解，在图 1.21 的运作情况中，随着对 OCR1X 寄存器赋值的增长，PWM 输出波形中的高电平持续时间将越来越长。当所赋比较值等于计数顶值的一半时，PWM 输出信号中高、低电平段相等，各占半个周期。若将 OCR1X 内容增至顶值，则 PWM 输出恒为高电平。把 OCR1X 值降至 $0000，就会使 PWM 输出恒为低电平。表 1.9 中给出了对应于上述两种极值的正向和倒向 PWM 输出电平。

表 1.9　OCR1X 值＝ $0000 或顶值时的 PWM 输出

COM1X1	COM1X0	OCR1X	OC1X 输出
1	0	$0000	低电平
1	0	顶　值	高电平
1	1	$0000	高电平
1	1	顶　值	低电平

上述这种输出脉冲宽度随着 OCR1X 寄存器赋值不同而变化的过程，就是脉宽调制。它有时可用于 D/A 转换，也可用来控制直流马达的转速、电加热器的温度以及照明设备的亮度等。

在前面讨论的情况中，不管 OCR1X 值如何改变，PWM 输出波形的周期却是固定不变的。欲改变 PWM 输出信号的周期，由表 1.7 得知，可有两个途径。第一，在 PWM 分辨率不变的情况下，改变 T/C1 时钟 TCK1 的频率；第二，在 TCK1 频率固定的情况下，改变 PWM 的分辨率。

[例 1.10]　已知 CPU 时钟频率为 8 MHz，要求根据表 1.10 所给各寄存器的内容，画出 OC1A 和 OC1B 两引脚上的输出波形图，并且标出 PWM 信号的周期值。

表 1.10　例 1.10 的原始数据

寄存器	内　容	寄存器	内　容
TCCR1A	10110011	OCR1A	$0200
TCCR1B	00000001	OCR1B	$0200

根据 TCCR1B 值由表 1.4 中查得，T/C1 的时钟源 TCK1 即 CPU 时钟源 CK，其频率为 8 MHz。

根据 TCCR1A 值由表 1.6 得知，T/C1 工作于 10 位 PWM 方式；进而由表 1.7 查得，TCNT1的计数顶值等于 $03FF，此时的 PWM 频率为 8 MHz/2 046＝3.910 kHz。由此算出 PWM 周期等于 255.75 μs。

表 1.8 告诉我们，TCCR1A 值具体设定 OC1A 作正向 PWM 输出，而 OC1B 则为倒向 PWM 输出。这样，图 1.22 所表达的便是上述诸点的形象总结。

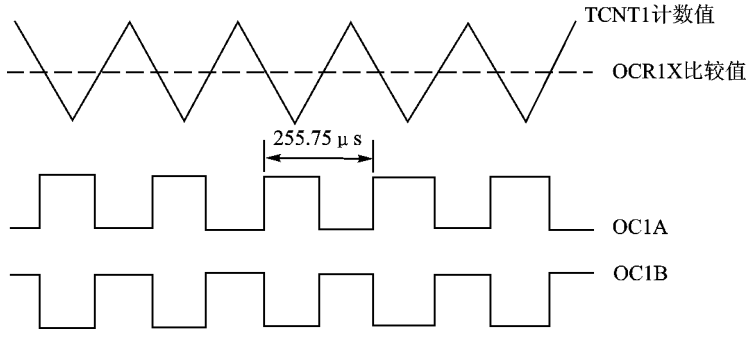


图 1.22　例 1.10 的 PWM 输出波形

最后应当指出，在 PWM 方式下，定时器 1 的溢出标志 TOV1 是在 TCNT1 从 $0000 点起开始改变计数方向时置 1 的。其溢出中断的运作和通常定时/计数方式下的情况完全一致。即在定时器溢出中断和总体中断都开放的情况下，当 TOV1 标志置 1 时，中断将获得响应。就输出比较匹配标志 OCF1A 和 OCF1B 以及它们请求中断的情况来说，PWM 方式也无异于输出比较方式。

1.2.8　监视定时器

监视定时器 WDT，即通常所说的看门狗定时器。其主要任务就是保障 MCU 的安全运行。大家知道，微控制器有时会被噪音、射频干扰等环境因素拖入错误的运行状态。WDT 在特定的时限内使 MCU 复位，将其从错误的状态中解救出来，使之重新开始正常运行。

与诸多同类部件相比，AVR 监视定时器具有两大特色。

第一，有独立的时钟，即使 MCU 处于掉电方式下，CPU 时钟冻结，WDT 仍可照常运作。

第二，有监视定时器复位指令 WDR，故 WDT 的运行控制非常简单。

图 1.23 所示为 AVR WDT 的内部逻辑结构。WDT 的时钟源是一个片内 RC 振荡器 OSC，其振荡频率与所施加电源电压有关。在 5 V 电源的情况下，OSC 的频率为 1.1 MHz 左右。11 位定标器可对此频率进行 16～2 048 分频。就是说，定标器通过多路开关向 10 位

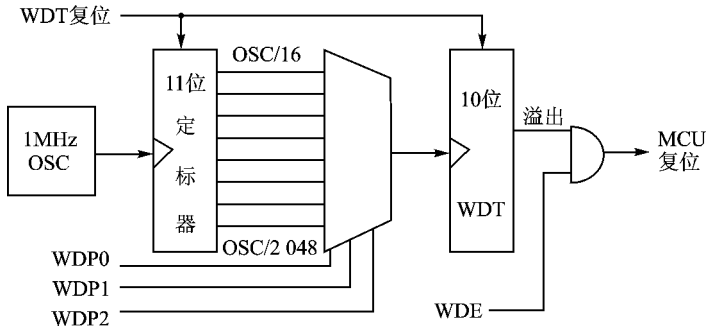


图 1.23 监视定时器框图

WDT 提供 OSC/16～OSC/2 048 等 8 档分频后的时钟。WDT 对此时钟周期进行递增计数。若在此计数过程中无 WDR 指令执行，则 WDT 在计满 1 024 个周期时便要溢出，这样就可引起 MCU 复位了。

通过对定标器的控制，可以调整 WDT 的复位时限。这一任务由看门狗控制寄存器 WDTCR 来承担。WDTCR 的最高 3 位为保留位，只读，其值恒读作 0。其余 5 位意义如下。

$21	7	6	5	4	3	2	1	0	WDTCR
($41)	—	—	—	WDTOE	WDE	WDP2	WDP1	WDP0	复位值 $00

位 4——看门狗关断允许位 WDTOE：WDE 位清 0 时，此位必须置 1。否则，看门狗便不可关断。WDTOE 位一旦置 1，4 个时钟周期后，将被硬件自动清 0。

位 3——看门狗开放位 WDE：WDE 位置 1，就启用了 WDT 功能；WDE 位清 0，即可禁止看门狗的运作。但 WDE 位清 0 只有在 WDTOE 置 1 的情况下才能进行。为关闭已经启用了的看门狗定时器，必须遵循防止 WDT 意外遭禁的两步操作：

- 同时向 WDTOE 和 WDE 两位写入逻辑 1，即使此前 WDE 位已经置 1 也得这样。
- 在此后的 4 个时钟周期内，向 WDE 位写入逻辑 0。这就关闭了看门狗定时器。

位 2～位 0——WDP2、WDP1 和 WDP0：看门狗定标位。这三位在启用看门狗时负责分频因子的选择，即选定 WDT 溢出时限标度（见表 1.11）。

表 1.11 WDT 溢出时限标度选择

WDP2	WDP1	WDP0	WDT OSC 周期数	溢出时限（V_{CC}=3 V）	溢出时限（V_{CC}=5 V）
0	0	0	16 K	47 ms	15 ms
0	0	1	32 K	94 ms	30 ms
0	1	0	64 K	0.19 s	60 ms
0	1	1	128 K	0.38 s	0.12 s
1	0	0	256 K	0.75 s	0.24 s
1	0	1	512 K	1.5 s	0.49 s
1	1	0	1 024 K	3.0 s	0.97 s
1	1	1	2 048 K	6.0 s	1.9 s

应当指出，在开放 WDT 前，必须先用一条 WDR 指令使之复位。否则，看门狗定时器可能不从 0 开始计数，这样溢出时限就不固定了。

在 MCU 程序正常运行时，如果每次赶在 WDT 溢出之前用 WDR 指令使看门狗复位，那么这种有规律的 WDT 复位便是使 WDT 永不溢出的保证。但是，当外界干扰导致程序执行紊乱时，监视定时器因其内容得不到及时复位而溢出。WDT 的溢出将导致 MCU 复位，使之从 $0000 地址处重新开始执行程序，再度正常运行起来。

[例 1.11]　某 AT90S8515 单片机系统在错误的状态下能支持 20 ms，应怎样使用 WDT 来保障系统的安全运行？

首先执行一条 WDR 指令，使看门狗复位。第二，向 WDTCR 寄存器中写入 $08，启用 WDT，选溢出时限 15 ms。第三，对用户软件进行划分，以便在程序中插入 WDR 指令，使每两条 WDR 指令间的程序执行时间不超过 15 ms。这时要充分考虑到条件跳转、子程序调用以及内外中断等因素所带来的影响。对于那些很难估算其执行时间的程序段落，应按最坏情况考虑。

1.2.9　EEPROM 读写

如 1.2.3 一节之 3 所述，AVR 片内的 EEPROM 占据独立的数据空间，对此空间可进行单一字节的读写操作。读写访问所需的地址、数据以及控制信息，由 I/O 空间中的专用寄存器负责提供。

1. EEPROM 地址寄存器 EEAR

AT90S8515 的 EEPROM 地址寄存器 EEAR 由两个 8 位 I/O 寄存器构成。其高字节 EEARH 占据地址 $1F($3F)，低字节 EEARL 位于 I/O 空间的 $1E($3E)单元。EEAR 的取值范围为 0～511，其内容指定 EEPROM 空间 512 个单元中的任一字节地址。

AT90S4414 中没有 EEARH，只有 EEARL，用来保存 EEPROM 空间 256 个字节的某指定地址。EEARH(AT90S8515)和 EEARL 都是可读写的，它们的复位值均为 $00。

2. EEPROM 数据寄存器 EEDR

数据寄存器 EEDR，8 位，可读亦可写，其复位值等于 $00。EEDR 在 I/O 空间的地址为 $1D($3D)。在对 EEPROM 进行写操作时，EEDR 寄存器的内容便是要写入由地址寄存器 EEAR 值所给出的 EEPROM 地址单元中去的数据。对于 EEPROM 读操作来说，EEDR 将含有读自(由 EEAR 值所给定的)EEPROM 某地址单元中的 8 位数据。

3. EEPROM 的读写及控制寄存器 EECR

EEPROM 写操作所需时间与电源电压值有关，一般在 2.5～4 ms 之间。对于 CPU 来说，这是可以执行数千乃至数万条指令的一段很长的时间。EEPROM 的写操作是自定时的，这种自定时功能使用户软件能够检测出本次写操作是否已经完成，即何时可以写下一个字节。如果用户软件中含有向 EEPROM 进行写操作的段落，那么就应当特别注意下述情况。

若供电电源配有大电容滤波器，则在接通或关闭电源时，V_{CC} 上升或下降非常缓慢。这将使单片机在某段时间内运行在欠压条件下，即电源电压低于某时钟频率下最小电压值。在这种条件下的 CPU 运作很可能引起程序计数器计数失误，导致意外跳转，进入向 EEPROM 进行写操作的段落。为防止对 EEPROM 的意外写入，建议用户在这种情况下使用片外欠压复位电路。

AVR 单片机为片内 EEPROM 的写操作规定了严格的步骤，以杜绝意外写操作的发生。这由 EEPROM 控制寄存器 EECR 的 EEMWE 和 EEWE 两位的功能来实现。

	7	6	5	4	3	2	1	0	
($1C)	—	—	—	—	—	EEMWE	EEWE	EERE	EECR

EECR 的高 5 位为保留位，只读，且恒读作 0。其最低 3 位可读亦可写。EECR 寄存器的复位值等于 $00。

位 2——EEPROM 写允许总控位 EEMWE：它宛如一个决定能否对 EEPROM 进行写操作的总开关。在 EEMWE 为 1 的情况下，将 EEWE 位置 1，就会把 EEDR 中的数据写入到 EEAR 所指定的 EEPROM 地址单元中去。若总控位 EEMWE 为 0，则尽管把 EEPROM 写允许位 EEWE 置 1，写操作也不会奏效。当软件已经把 EEMWE 位置成 1 时，4 个时钟周期后，硬件会自动将其清 0。

位 1——EEPROM 写允许位 EEWE：实际上这是一个对 EEPROM 的写选通信号。当地址和数据已经正确地建立起来时，为把此数据值写入 EEPROM 的该地址单元中去，EEWE 位必须置 1。而在向 EEWE 位写入逻辑 1 时，EEMWE 位必须先期置 1，否则不会有 EEPROM 写操作发生。当向 EEPROM 某地址单元中写入数据时，必须遵循下述步骤（第 2 和第 3 两步的前后顺序可调换）：

(1) 等待 EEWE 位变成 0；

(2) 向 EEARL 和 EEARH 中写入 EEPROM 某新的单元地址（可不进行）；

(3) 向 EEDR 寄存器中置入行将写进 EEPROM 某地址单元的新数据（可不进行）；

(4) 向 EECR 寄存器中的 EEMWE 位写入逻辑 1；

(5) 在 EEMWE 置 1 后的 4 个时钟周期内，向 EEWE 位写入逻辑 1。

写访问时间（$V_{CC}=5$ V 时为 2.5 ms，$V_{CC}=2.7$ V 时为 4 ms）一过，EEWE 位即被硬件自动清 0。所以用户软件可对此位进行查询，并待其清 0 后再写入下一个字节。在 EEWE 置 1 后，CPU 暂停两个时钟周期，然后方去执行下条指令。

这里要提请读者注意，第一，若在第 4 和第 5 两步间发生中断，则将导致写周期失效。因为 EEPROM 写允许总控位 EEMWE 将会到时被自动清 0。第二，如果访问 EEPROM 的某中断服务程序打断的是另一个访问 EEPROM 的操作，那么 EEAR 和 EEDR 寄存器的内容就会改变，这将使被打断的 EEPROM 访问操作归于失败。因此，建议在进行后四步的过程中，把总体中断允许位 I 清成 0，以避免上述问题的发生。

位 0——EEPROM 读允许位 EERE：实际上这是对 EEPROM 的读选通信号。当正确的地址码在 EEAR 寄存器中建立起来时，EERE 位必须置 1 才能读出该地址单元的内容。一旦所读数据被传送至 EEDR 寄存器中，EERE 位便被硬件自动清 0。对 EEPROM 的读访问只需一条指令的执行时间，无须通过查询 EERE 位来了解数据是否已经读毕。EERE 位被写成 1 后，CPU 暂停两个周期，此后方去执行下一条指令。

在启动 EEPROM 读操作前，用户应先查询一下 EEWE 位，看是否有写操作在进行。在 EEPROM 写操作正进行之际，如果向它的 I/O 寄存器中写入新地址或新数据，那么正进行着的 EEPROM 写操作便会被打断，且其结果将无定。

4. EEPROM 数据破坏的预防措施

在 V_{CC} 欠压期间，EEPROM 中存放的数据可能被破坏，因为供电电压太低，不足以使 CPU 和 EEPROM 正常运作。这与带有 EEPROM 的板级系统上发生的问题是一致的，故应采用与之同样的设计解决方案。

供电电压太低时，有两种情况可能会引起 EEPROM 数据的破坏。其一，EEPROM 正常的写操作过程要求有一个保障正确运作的最低电压值。其二，供电电压过低，CPU 本身执行指令也可能出错。

不过，只要按下列设计建议之一去做，即可避免 EEPROM 数据破坏现象的发生。

(1) 在欠压供电期间保持 AVR $\overline{\text{RESET}}$信号为低电平有效状态。这一点通过片外 V_{CC} 欠压复位保护电路(BOD)极易实现。

(2) 在 V_{CC} 欠压期间，保持 AVR 内核处于掉电休眠方式之下。这将阻止 CPU 执行指令，有效地保护 EEPROM 各寄存器，使它们免遭意外写入。

(3) 如果不要求软件修改存储内容的话，那么常数可放在程序存储器 Flash 中。CPU 无能更新 Flash 存储器的内容，故其数据不会受到破坏。

1.2.10　串行涉外接口 SPI

串行涉外接口 SPI 可进行 AVR 微控制器与外围设备之间或若干 AVR 器件之间的高速同步数据传送，故称同步串行口。AT90S4414/8515 的 SPI 有如下特点：

- 全双工三线同步数据传送；
- 主控或从控运作；
- 可自最低位亦可自最高位开始传送；
- 四档可编程的位传送率；
- 传送结束中断标志；
- 写冲突标志用于保护；
- 可自闲置方式下唤醒 MCU。

1. 内部结构和运作原理

图 1.24 所示为 SPI 的逻辑结构，其核心部件是 8 位移位寄存器和接收数据缓冲器(两者合称 SPI 数据寄存器)，向其提供时钟信号的包括由分频器、选择器和时钟逻辑等部件构成的时钟发生器，以及由 SPI 控制寄存器、SPI 状态寄存器等环节构成的控制部件。

SPI 的对外引脚有 MISO(PB6)、MOSI(PB5)、SCK(PB7)和$\overline{\text{SS}}$(PB4)。

数据由主机发送，从机接收。向主机的 SPI 数据寄存器 SPDR 装入待发送数据的写操作，自动启动 SPI 时钟发生器，于是该数据便在时钟脉冲的控制下开始逐位左移或右移。若设定最高位 MSB 先传送，则如图 1.25 所示，主机的 MSB 将自 MOSI(PB5)引脚移出，经从机的 MOSI 引脚进入其 8 位移位寄存器中，占据最低位 LSB 的位置。同时，从机的 MSB 由 MISO(PB6)引脚移出，通过主机的 MISO 引脚进入到主机移位寄存器中，作为它的 LSB 位。这样，8 个时钟脉冲之后，主机 SPI 移位寄存器中的一个字节数据，完全传送给了从机，而后者的一个字节数据也同时进入了主机的 SPI 移位寄存器中。这就是说，主从机的数据是互相交换的。

主控方式下的 SCK(PB7)引脚用以输出同步时钟信号，而从控方式下的 SCK 则作为时钟信号输入引脚。由图 1.25 可见，主从机的两个 8 位移位寄存器可被视作一个分布式的 16 位

循环移位寄存器，在统一的时钟控制下运作。

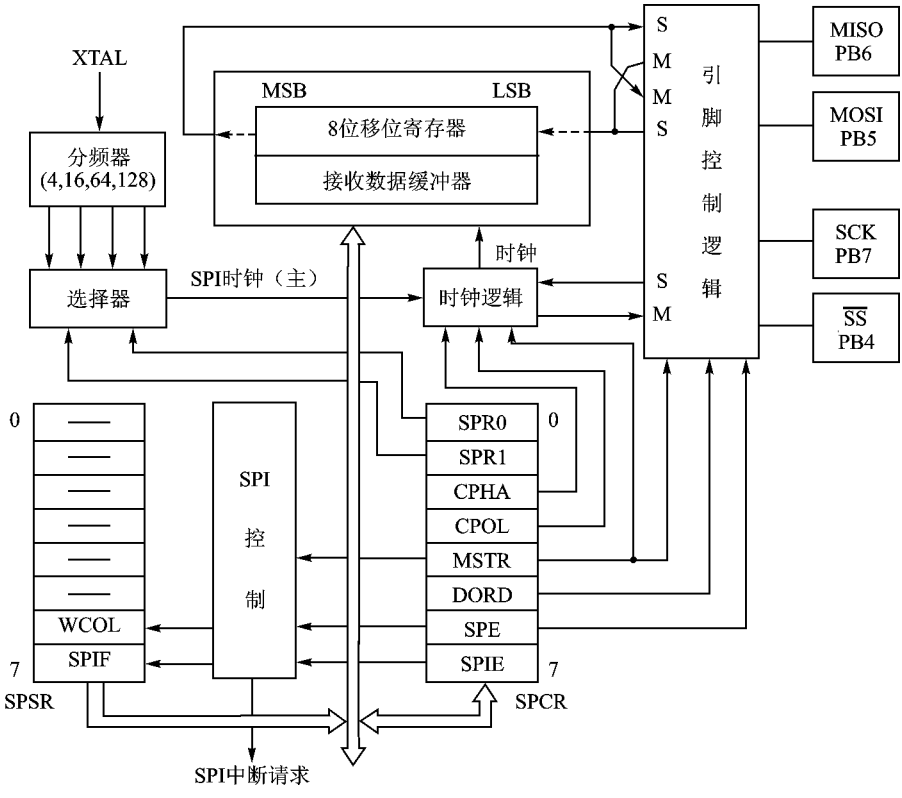


图 1.24 SPI 结构框图

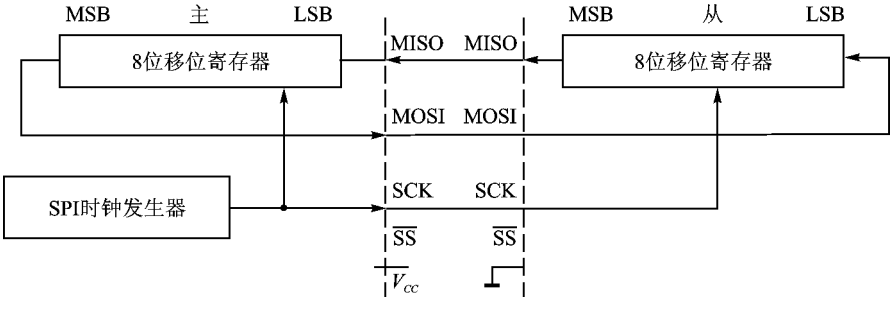


图 1.25 SPI 主从连接

在主 SPI 发送完一个字节数据后，SPI 时钟发生器暂停运行，并将传送结束标志 SPIF 置 1。若 SPI 控制寄存器 SPCR 中的中断开放位 SPIE 已被置 1，那么 SPIF 标志将去请求中断。

应当指出，SPI 系统的发送机构是单缓冲的，即上一个字节尚未发送完毕时，不能向 SPI 数据寄存器 SPDR 中写入待发送的新数据。如前所述，字节发送完毕由 SPIF 标志告知。SPI 的接收机构是双缓冲的，就是说，上一个字节接收完毕但还未被读走时，便可开始接收下一个字节。但在下一个字节完全移入之前，必须把前一字节由 SPDR 中读走，否则第一个字节就会丢失。

2. SPI 寄存器

(1) SPI 控制寄存器 SPCR：8 位，可读亦可写，地址为 $0D($2D)，复位值等于 $00。

$0D	7	6	5	4	3	2	1	0	
($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR

位 7——串行涉外接口中断开放位 SPIE：SPI 中断被响应的条件是总体中断开放位 I、SPI 中断开放位 SPIE 和 SPI 中断标志 SPIF 三者均为 1。但若 SPIE 位被清 0，则尽管 I 和 SPIF 两位均置 1 也无中断请求发出。

位 6——SPI 开放位 SPE：此位置 1 即开放，清 0 则关闭 SPI 的一切操作。

位 5——数据顺序控制位 DORD：DORD 置 1 时，数据字节将由最低位 LSB 开始传送；此位清 0 时，则将首先从最高位 MSB 开始传送。

位 4——主从选择位 MSTR：若置 1，则选中 SPI 主控方式；清 0 则设定为 SPI 从控方式。稍后我们会看到，在某些情况下，MSTR 位也会被硬件自动清 0。

位 3——时钟极性控制位 CPOL：此位置 1 时，时钟信号 SCK 的高电平为闲置态；清 0 时，SCK 之低电平乃闲置态。数据的移位是由时钟脉冲边沿触发的。CPOL 和 CPHA 两位一同决定上升沿抑或下降沿起触发作用(见图 1.26 和图 1.27)。

位 2——时钟相位控制位 CPHA：此位置 1 时，同步时钟脉冲的后沿被用来触发移位寄存器的移位操作。CPHA 位清 0，即设定同步脉冲之前沿为移位操作的触发边沿。图 1.26 和图 1.27 以及 SPI 数据传送格式一小节对此有直观和详尽的说明。

位 1 和位 0——SPI 时钟频率选择位 SPR1 和 SPR0：这两位选择主控 SPI 的时钟频率，它们对从控 SPI 不起作用。SPR1 和 SPR0 两位不同组合值所选 SPI 时钟频率如表 1.12 所列。其中 CK 为振荡器 XTAL 所提供的 CPU 时钟频率。

表 1.12　SPI 时钟频率

SPR1	SPR0	SCK 频率
0	0	CK/4
0	1	CK/16
1	0	CK/64
1	1	CK/128

(2) SPI 状态寄存器 SPSR：8 位，只读，复位值等于 $00，地址为 $0E($2E)。其最低 6 位系保留位，恒读作 0。最高两位意义如下。

$0E	7	6	5	4	3	2	1	0	
($2E)	SPIF	WCOL	—	—	—	—	—	—	SPSR

位 7——SPI 中断标志 SPIF：一次同步串行传送完成时，SPI 被硬件自动置 1。在总体中断开放以及 SPCR 寄存器中的 SPIE 位为 1 的条件下，SPIF 置 1 将引起中断。此外，主控 SPI 的$\overline{SS}$引脚若作输入，且被从外部拉成低电平，这也会将 SPIF 置 1。SPIF 标志在其中断响应过程中被硬件自动清 0。先读 SPI 状态寄存器 SPSR，继而再访问 SPI 数据寄存器 SPDR，这先后两次操作也将把 SPIF 标志清 0。

位 6——写冲突标志 WCOL：在上一个数据尚未发送完毕时，如果向 SPDR 中写入待发送的新数据，那么 WCOL 标志就将置 1，以告知数据发生了碰撞。这再次告诉读者，SPI 发送机构是单缓冲的。在先读 SPSR 寄存器并继而再访问 SPDR 寄存器后，WCOL 标志(以及 SPIF

位)被自动清 0。

(3) SPI 数据寄存器 SPDR:8 位,可读亦可写,地址为 $0F($2F),复位后其值不定。SPDR寄存器用来在寄存器组合与 SPI 移位寄存器间传送数据。自寄存器组合向 SPDR 中写入的数据,实际上进入 SPI 移位寄存器,且此写操作自动启动数据的发送过程。而读此寄存器将得到接收数据缓冲器的内容。

3. $\overline{SS}$ 引脚功能

$\overline{SS}$(PB4)为从机选择输入引脚。若这里输入低电平信号,则本 SPI 将被选作从机。

当 SPI 被设定为主控方式时(SPCR 中的 MSTR 位置 1),用户可决定$\overline{SS}$引脚的方向。若$\overline{SS}$被设定为输出,则它将是一个通用输出引脚,不影响 SPI 系统运作。如果$\overline{SS}$被安排作输入,那么它必须保持为高电平,以保障本 SPI 的主控运作。在 SPI 作主机,$\overline{SS}$为输入且被外围电路拉成低电平时,SPI 系统即把这种情况解释成——另一主机将本 SPI 选作从机,并开始向此从机发送数据。

为避免这种情况下的总线竞争,SPI 系统自动采取下列措施:

(1) SPCR 寄存器中的 MSTR 位被硬件清 0,于是该 SPI 系统便由主控转为从控方式。这种转变的结果使 MOSI 和 SCK 均变成输入引脚。

(2) SPSR 寄存器中的 SPIF 标志置 1,若总体中断和 SPI 中断均已开放,则将执行相应的中断服务程序。

这样,在中断驱动的 SPI 传送用于主控方式,且$\overline{SS}$引脚有可能被从外部拉成低电平的情况下,中断服务程序每次都应检查 MSTR 位是否仍然为 1。一旦发现 MSTR 位因从机选择引脚$\overline{SS}$上有低电平输入而被清 0 的话,用户应再次将其置 1,以重新确立 SPI 的主控方式。

当 SPI 被设定为从控方式时,$\overline{SS}$引脚总是作为输入用的。$\overline{SS}$信号为低电平,SPI 开始运作,MISO 引脚可被用户设定为输出,即作从机输出引脚。其他有关引脚均为输入(见表 1.13)。但当$\overline{SS}$被拉成高电平时,所有 4 引脚均为输入,SPI 退出运作,将不接收输入数据。请注意,一旦$\overline{SS}$输入信号变成高电平,SPI 逻辑将被复位。若在数据传送过程中$\overline{SS}$电平被拉高,则 SPI 将立即停止发送和接收,且所发送和接收的数据必须被视作已经丢失。

表 1.13 SPI 引脚数据方向

SPI 引脚	主机引脚数据方向	从机引脚数据方向
MOSI	由用户定义	输入
MISO	输入	由用户定义
SCK	由用户定义	输入
$\overline{SS}$	由用户定义	输入

在启用 SPI 时,MOSI(主出从入)、MISO(主入从出)、SCK 以及$\overline{SS}$引脚上的数据方向与主从方式的设定有关,如表 1.13 所列。

4. SPI 数据传送格式

如前所述,当 SPCR 寄存器的时钟极性控制位 CPOL 被清成 0 时,同步时钟信号的低电平段为闲置态,即串行数据的移位操作由时钟正脉冲触发。而当 CPOL 位置 1 时,由时钟负脉冲触发。

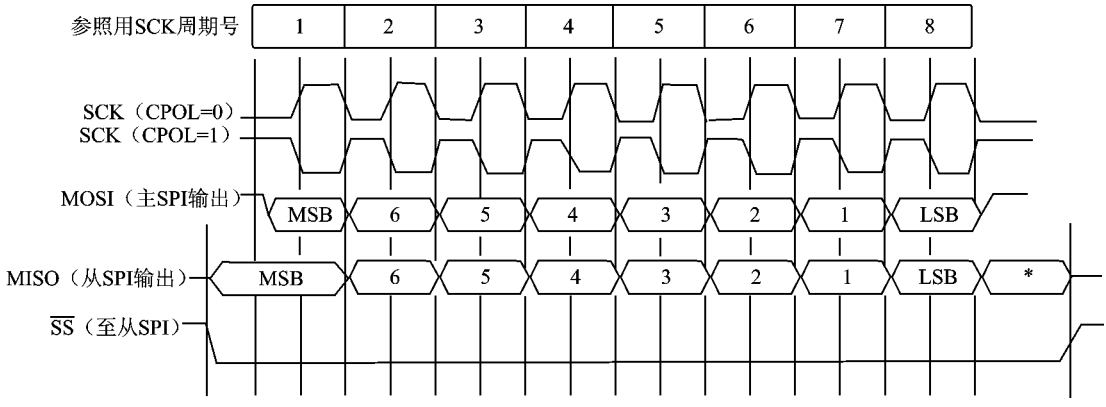
时钟相位控制位 CPHA 置 1，即设定串行数据的移位操作由时钟脉冲后沿触发。CPHA 位清 0，则规定由时钟脉冲前沿触发。

这样，CPOL 和 CPHA 两控制位的设定可有四种不同的组合结果，如表 1.14 所列。

表 1.14　SPI 数据传送同步时钟信号的触发边沿

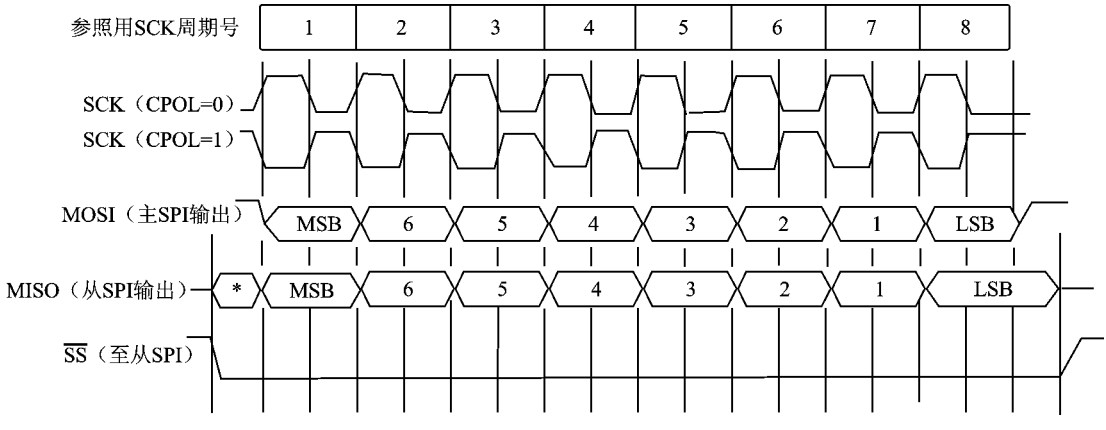
CPOL	CPHA	时钟脉冲触发边沿
0	0	正脉冲前沿触发
0	1	正脉冲后沿触发
1	0	负脉冲前沿触发
1	1	负脉冲后沿触发

图 1.26 和图 1.27 给出了上述四种情况下的的数据传送格式。



* 不确定，但通常是刚收到字符的 MSB。

图 1.26　CPHA＝0，DORD＝0 时的 SPI 传送格式



* 不确定，但通常是前一个所发字符的 LSB。

图 1.27　CPHA＝1，DORD＝0 时的 SPI 传送格式

1.2.11 通用异步接收发送器

AVR 单片机除 8 引脚和 20 引脚的简体者外，大多内含通用异步接收发送器 UART，即异步串行口。AVR UART 一般有下列几个特点：

- 波特率发生器可产生多档波特率；
- XTAL 低频下亦可获得高波特率；
- 数据有 8 位和 9 位两种格式；
- 噪音滤波；
- 超时检测；
- 帧误检测；
- 伪起始位检测；
- 发送完毕、发送数据寄存器空和接收完毕三种中断；
- 接收和发送均有缓冲。

1. 数据发送机理

根据图 1.28 所示 UART 发送器的逻辑结构，我们来分析数据的发送过程。

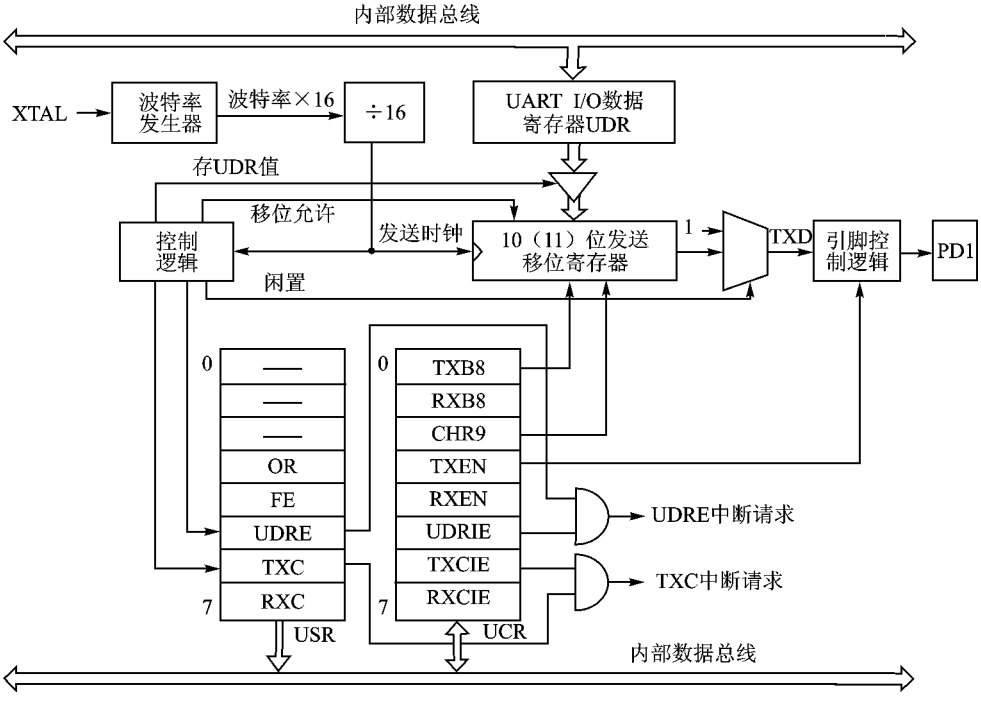


图 1.28 UART 发送器

当向 UART I/O 数据寄存器 UDR 中写入待发送的数据时，此写操作即启动数据的发送过程。此过程的第一步便是把数据由 UDR 送至发送移位寄存器中。后者长度为 10 位或 11 位，包括来自 UDR 的 8 位、UART 控制寄存器 UCR 送来的第 9 数据位 TXB8(可无)以及移位寄存器本身增加的起始位(0)和停止位(1)。在前一个字符的停止位移出之后，如果有新字符写至 UDR 中，那么新字符便被立即装入移位寄存器中。若是在上一个字符的停止位移出

之前有新字符写入 UDR 中，则待原字符的停止位移出之后，新字符方能自 UDR 传至移位寄存器中。

如前所述，若 10(11)位发送移位寄存器空无有效内容，则数据便由 UDR 流入该移位寄存器中。同时，UART 状态寄存器 USR 中的 UART 数据寄存器空标志 UDRE 被硬件置 1。这就是说，UART 已经做好接受下一个字符的准备了。在数据由 UDR 传向移位寄存器的同时，移位寄存器的位 0 被清 0，将作为起始位，位 9 或位 10 被置 1，作为停止位。如果选用 9 位数据字(UART 控制寄存器 UCR 中的 CHR9 位为 1)，那么，UCR 中的 TXB8 位将被传送至发送移位寄存器的位 9 处。因此，在启动发送过程之前，必须把 TXB8 写成用户所需要的数值。

如果说发送过程的第一步是准备阶段的话，那么第二步就是串行发送了。即在发送时钟的控制下，移位寄存器以一定的波特率通过 PD1 引脚逐位向外移出数据。在向移位寄存器装入数据后的一个波特率发生器时钟周期，起始位被移出，到达 TXD 引脚 PD1。随后接踵而来的是数据，最低位居先。若在发送期间已有新数据写入 UDR，则在停止位移出时该数据便被装入移位寄存器中。在此装入过程中，UDRE 标志自动置 1。但在停止位移出时，UDR 寄存器中如无新数据要发送的话，那么上次操作遗留下的 UDRE 标志将一直保持为 1，直至 UDR 有新数据写入为止。在停止位出现于 TXD 引脚上已有一个位周期时间，且不见有新数据写来的情况下，USR 寄存器中的发送完成标志 TXC 自动置 1。

UCR 寄存器中的 TXEN 位被软件置 1 时，引脚控制逻辑即将 UART 发送器与 TXD 引脚(PD1)接通。后者被强制作输出引脚，DDRD 寄存器中 DDD1 位的设定不再起作用。于是发送器得以发挥其串行发送数据的功能。但若 TXEN 位被软件清 0，或保持其复位值 0 不变，则发送器与 PD1 引脚脱钩，故 PD1 可作通用 I/O 引脚使用。

2. 数据接收机理

如图 1.29 所示，UCR 寄存器的 RXEN 位被软件置 1 时，引脚控制逻辑便把 PD0 引脚与 UART 接收器连接起来，并迫使 PD0 作 RXD 输入引脚，不管数据方向寄存器 DDRD 此前对

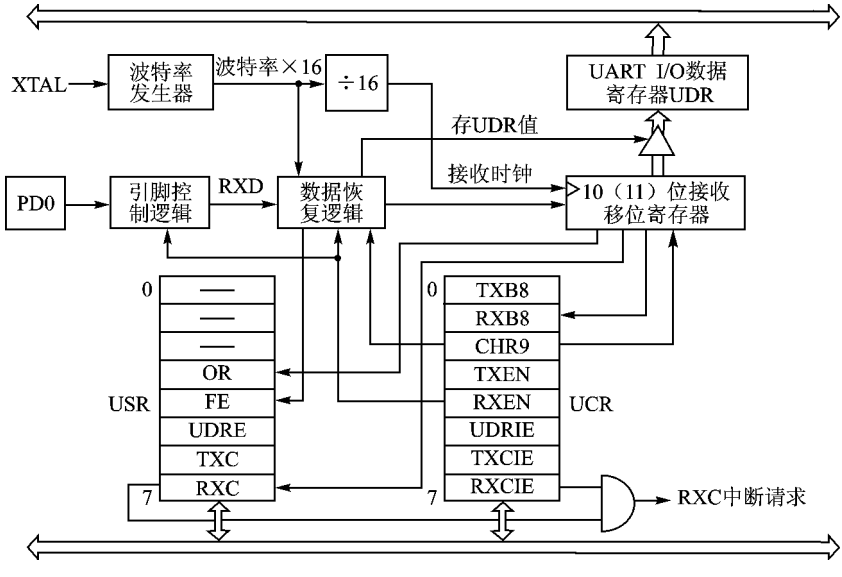


图 1.29　UART 接收器

该引脚做过什么设定。但此时 PORTD0 位仍可用来控制 PD0 引脚内部的上拉电阻。若将 RXEN 位清 0，或保持其 0 复位值不变，则 UART 的接收器将处于禁止状态。此时 PD0 可作通用 I/O 引脚。

UART 的前端逻辑以 16 倍波特率的频率对 RXD 引脚上的输入信号进行采样。在线路空闲期间，输入信号一直保持为高电平，此后采样到的一个逻辑 0 值将被解释为发现了起始位的下降沿。于是这就拉开了起始位检测的序幕。在图 1.30 中，令采样 1 代表第一个 0 采样值。在 1 至 0 的跳变后，接收器在第 8、第 9 和第 10 三个采样脉冲时仍对 RXD 引脚进行采样。这三个采样值中若有两个或全部为逻辑 1，则此起始位被否定，而被视作一个噪音干扰信号。于是接收器将继续寻找下一个 1 至 0 的跳变。

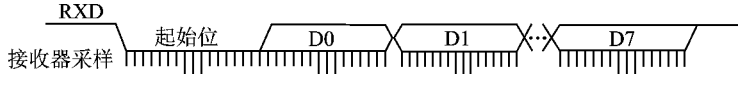


图 1.30　接收数据的采样

但是，如果检测到的确实是一个有效的起始位，那么，对数据位的采样便要跟在起始位后进行。这些数位也是在第 8、第 9 和第 10 三个采样脉冲时被采样的。三次采样的逻辑值中，两次或全部一致者即为该位之值。每采样一位，就向接收移位寄存器中移入一位。

在停止位进入接收器时，三次采样值中必须至少有两次为逻辑 1，才能认定是有效的停止位。若两次或全部采样值为逻辑 0，则 UART 状态寄存器 USR 中的帧误标志 FE 被自动置 1。在读 UDR 寄存器以获得接收数据之前，用户每次都应当检查一下 FE 标志，看是否有帧误。

在每个字符接收周期的末尾，不管是否检测到有效的停止位，所接收到的数据都要被传送至 UDR 寄存器中去。USR 寄存器中的接收完毕标志 RXC 因之而自动置 1。UDR 实际上是两个独立的寄存器，一个只写，一个只读。前者用来存放待发送数据，后者用来存放接收到的数据。若 UCR 寄存器中的 CHR9 位被软件置 1，即选中 9 位数据格式，则在向 UDR 中传送 8 位数据时，接收移位寄存器中的第 9 数据位被送到 UCR 的 RXB8 处(见图 1.29)。

在接收完一个字符后，如果上次接收到的内容仍未从 UDR 中读走的话，那么 USR 寄存器中的超时标志 OR 将被置 1。这就意味着，最后移入接收移位寄存器的数据字节因不能送进 UDR 寄存器中而丢失。应当指出，OR 位是经缓冲的，其值的更新，包括置 1 和清 0，发生在 UDR 中的有效数据字节被读取时。因此，在读过 UDR 寄存器后，用户每次都应检查 OR 位，以便查明在高波特率或 CPU 高负荷的情况下是否有超时错误发生。

3. UART 寄存器

(1) UART I/O 数据寄存器 UDR：8 位，可读亦可写，复位值等于 $00，地址为 $0C ($2C)。

前已谈到，UDR 实际上是共用一个地址的两个独立寄存器，即只可写的 UART 发送数据寄存器和只能读的接收数据寄存器。

(2) UART 状态寄存器 USR：8 位，只读，(TXC 位例外)，复位值等于 $20，地址为 $0B ($2B)。USR 的最低 3 位为保留位，其余各位意义如下。

$0B	7	6	5	4	3	2	1	0	
($2B)	RXC	TXC	UDRE	FE	OR	—	—	—	USR

位 7——UART 接收完成标志 RXC：当接收到的字符自接收移位寄存器送至 UDR 寄存器时，RXC 标志被硬件置 1。尽管偶尔在接收过程中检测到有帧误发生，但所接收到的数据仍要传送至 UDR 中去。故 RXC 标志的置 1 操作不受帧误的影响。如图 1.29 所示，若 UCR 寄存器中的 RXCIE 位为 1，则 RXC 标志置 1 就将向 CPU 发出接收完成中断请求。UDR 寄存器接受读操作后，RXC 标志被自动清 0。这种机制是非常合理的，因为在接收完成中断服务程序中，总要读 UDR 寄存器，以获得 UART 串行输入数据。这就顺便清掉了 RXC 标志，使之在中断返回后不再去为此次接收完成请求中断了。

位 6——UART 发送完成标志 TXC：当发送移位寄存器中的整个字符（包括停止位）已经移出，且无新数据写入 UDR 寄存器时，TXC 标志自动置 1。此标志在半双工通讯接口中尤其有用，这里在发送完成之后发送器必须立即释放通讯总线，进入接收方式。

如果 UCR 寄存器中的 TXCIE 位已由软件置 1，那么 TXC 标志置位就会向 CPU 去请求发送完成中断。TXC 标志在中断响应过程中被硬件自动清 0。应当指出，USR 寄存器从整体上讲是只读的，但 TXC 位例外，它可读亦可写。这样，TXC 的第二个清 0 方法是向该位写入逻辑 1。

位 5——UART 数据寄存器空标志 UDRE：写入 UDR 中的字符被传送到发送移位寄存器时，此位置 1，表明发送器又可接受待发送的新字符了。

在 UCR 中的 UDRIE 位为 1 的情况下，UDRE 标志置 1，如图 1.28 所示，即可请求 UART 数据寄存器空中断。UDRE 位由对 UDR 的写操作清 0。在用中断驱动的数据发送时，UART 数据寄存器空中断服务程序一般都要向 UDR 中写入新的发送数据。这就顺便清掉了 UDRE 标志，使之在中断结束后不去为此番数据寄存器空再次请求中断。

请注意，在系统复位期间，UDRE 标志被置 1，表明发送器随时待命。

位 4——帧误标志 FE：若输入字符的停止位为 0，则称之为帧误，即结构错误。此时 FE 标志被硬件置 1。当收到停止位为 1 的字符时，FE 位被自动清 0。

位 3——超时标志 OR：在一新字符已经移入接收移位寄存器时，若原位于 UDR 寄存器中的前一个字符仍未被读走，则 OR 标志将置 1，表示由于前一字符超时占用 UDR 寄存器，而致使后一字符被拒于 UDR 之外。OR 位是经缓冲的，这意味着，仍位于 UDR 寄存器中的有效数据一旦被读取时，OR 位才更新其值，当然这里指的是置 1。

在有数据收到，且被传送至 UDR 中去的情况下，OR 位被清成 0。

(3) UART 控制寄存器 UCR：8 位，高 6 位可读亦可写，次低位 RXB8 只可读，最低位 TXB8 只可写。UCR 地址为 $0A($2A)，其复位值等于 $02。

$0A	7	6	5	4	3	2	1	0	
($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR

位 7——接收完成中断开放位 RXCIE：此位为 1 时，USR 中的 RXC 位置 1，就会使 CPU 去执行接收完成中断服务程序，当然前提条件是已经总体开放了中断。若 RXCIE＝0，则将不允许发出 RXC 中断请求。

位 6——发送完成中断开放位 TXCIE：在此位为 1 时，USR 寄存器中的 TXC 位置 1 便可产生发送完成中断请求。若此前总体中断开放位 I 已被软件置 1，则该中断请求将获得响应。TXCIE 位为 0，就意味着禁止 TXC 标志去请求中断。

位 5——UART 数据寄存器空中断开放位 UDRIE:若为 1,则允许 UDRE 标志发出中断请求;为 0 则禁止发出此类中断请求。

位 4——接收允许位 RXEN:如数据接收机理一小节针对图 1.29 所述,RXEN 位被软件置 1,即把 PD0 引脚和 UART 接收器连接起来,使它们可以接收数据了。若 RXEN 位为 0,则 PD0 引脚与 UART 接收器分断,使后者处于禁止状态。不难理解,此时 RXC、OR 和 FE 等状态位不可能置 1。但如果这些标志置 1 后把 RXEN 位清 0,那么它们的逻辑 1 态将不会改变。

位 3——发送允许位 TXEN:如前针对图 1.28 所述,TXEN 位被软件置 1,即把 UART 发送器与 PD1 引脚接通,使数据得以发送出去。TXEN 位清 0,则两者断开,UART 处于禁止状态。但在字符发送期间将 TXEN 清 0,发送器不会立即停止运作,而要等到移位寄存器中的字符以及 UDR 中的后续字符完全发出之后方才进入禁止状态。

位 2——9 位字符设定位 CHR9:此位为 1 即设定发送与接收的字符长 9 位,加起始位和停止位共 11 位构成一帧。发送时,第 9 位将自 UCR 中的 TXB8 处提取;接收时,第 9 位将由移位寄存器送入 RXB8 处。CHR9 若清 0,则字符长度将被设定为 8 位,即帧长 10 位。

位 1——RXB8:在 CHR9 为 1 的情况下,RXB8 将是所接收到的字符的第 9 数据位。应当注意,RXB8 复位值等于 1。

位 0——TXB8:当 CHR9 被软件置 1 时,TXB8 将是待发送字符的第 9 数据位。第 9 数据位由用户安排,可作附加停止位、奇偶位,或作具有特定含义的标志。

4. 波特率发生器

UART 的发送与接收都是在时钟控制下进行的,每个时钟周期发送或接收一位。发送或接收时钟的频率称波特率,它等于每秒钟发送或接收的位数,有时写作 bps。波特率发生器的任务就在于向 UART 的发送器和接收器分别提供发送时钟信号和接收时钟信号。

AVR 单片机的波特率发生器实际上是一个两级分频器。第一级对单片机时钟信号 CK 进行 1～256 分频,其分频因子由 UART 波特率寄存器 UBRR 的内容(0～255)进行控制。若后者为 n,则分频因子等于 $n+1$。第一级输出再经第 2 级 16 分频后便作为发送和接收时钟(见图 1.28 和图 1.29),其频率便是波特率 BAUD。频率等于 16 倍波特率(见图 1.29)的第一级输出信号用来作数据接收时的采样信号(见图 1.30)。

UART 波特率寄存器 UBRR 是一个 8 位读/写寄存器,地址为 $09($29),复位值等于 $00。若 UBRR 的内容为 n(8 位无符号数,取值范围 0～255),f_{CK} 代表单片机时钟频率,则波特率

$$BAUD = f_{CK}/[16(n+1)]$$

对于标准晶振频率来说,最常用的波特率可通过表 1.15 所列的 UBRR 值 n 来获得。表中的空白处说明,在某晶振频率下,对应左列的波特率无精确的 n 值。或者说,某 n 值之实得波特率与目标波特率间的误差超过 2%,无实用价值。注意,表中误差超过 1%的波特率建议不要使用。高误差会降低抗干扰能力。

[例 1.12] 某 AVR 单片机所用晶振频率 $f_{CK}=8$ MHz,如果其 UART 以最常用波特率串行发送字符,请问在 CHR9=0 的情况下最高字符传送率等于多少?

CHR9=0,说明字符由 8 个数据位构成,发送时加起始位和停止位,故字符帧长为 10 位。由表 1.15 查得,对应 8 MHz 晶振频率,最高实用波特率为 38 400 bps,故字符传送率等于 3 840字符/秒。

表 1.15　各晶振频率下的 UBRR 值 n

波特率	1 MHz	百分误差	1.843 2 MHz	百分误差	2 MHz	百分误差	2.457 6 MHz	百分误差
2 400	$n=25$	0.2	$n=47$	0.0	$n=51$	0.2	$n=63$	0.0
4 800	$n=12$	0.2	$n=23$	0.0	$n=25$	0.2	$n=31$	0.0
9 600	—	—	$n=11$	0.0	$n=12$	0.2	$n=15$	0.0
14 400	—	—	$n=7$	0.0	—	—	—	—
19 200	—	—	$n=5$	0.0	—	—	$n=7$	0.0
28 800	—	—	$n=3$	0.0	—	—	—	—
38 400	—	—	$n=2$	0.0	—	—	$n=3$	0.0
57 600	—	—	$n=1$	0.0	—	—	—	—
76 800	—	—	—	—	—	—	$n=1$	0.0
115 200	—	—	$n=0$	0.0	—	—	—	—

表 1.15(续 1)

波特率	3.276 8 MHz	百分误差	3.686 4 MHz	百分误差	4 MHz	百分误差	4.608 MHz	百分误差
2 400	$n=84$	0.4	$n=95$	0.0	$n=103$	0.2	$n=119$	0.0
4 800	$n=42$	0.8	$n=47$	0.0	$n=51$	0.2	$n=59$	0.0
9 600	$n=20$	1.6	$n=23$	0.0	$n=25$	0.2	$n=29$	0.0
14 400	$n=13$	1.6	$n=15$	0.0	—	—	$n=19$	0.0
19 200	—	—	$n=11$	0.0	$n=12$	0.2	$n=14$	0.0
28 800	$n=6$	n=1.6	$n=7$	0.0	—	—	$n=9$	0.0
38 400	—	—	$n=5$	0.0	—	—	—	—
57 600	—	—	$n=3$	0.0	—	—	$n=4$	0.0
76 800	—	—	$n=2$	0.0	—	—	—	—
115 200	—	—	$n=1$	0.0	—	—	—	—

表 1.15(续 2)

波特率	7.372 8 MHz	百分误差	8 MHz	百分误差	9.216 MHz	百分误差	11.059 MHz	百分误差
2 400	$n=191$	0.0	$n=207$	0.2	$n=239$	0.0	—	—
4 800	$n=95$	0.0	$n=103$	0.2	$n=119$	0.0	$n=143$	0.0
9 600	$n=47$	0.0	$n=51$	0.2	$n=59$	0.0	$n=71$	0.0
14 400	$n=31$	0.0	$n=34$	0.8	$n=39$	0.0	$n=47$	0.0
19 200	$n=23$	0.0	$n=25$	0.2	$n=29$	0.0	$n=35$	0.0
28 800	$n=15$	0.0	—	—	$n=19$	0.0	$n=23$	0.0
38 400	$n=11$	0.0	$n=12$	0.2	$n=14$	0.0	$n=17$	0.0
57 600	$n=7$	0.0	—	—	$n=9$	0.0	$n=11$	0.0
76 800	$n=5$	0.0	—	—	—	—	$n=8$	0.0
115 200	$n=3$	0.0	—	—	$n=4$	0.0	$n=5$	0.0

[例 1.13] 如果 $f_{CK}=3.2768$ MHz，那么 UART 可选用哪几种常用的数位传送率？

从表 1.15 中可以看到，在晶振频率等于 3.2768 MHz 的情况下，有 5 种常用波特率可选。但其中有三个 n 值所产生的实际波特率与标称波特率间的误差超过 1%，不宜采用。所以只有 2 400 和 4 800 两值最为适宜。

1.2.12 模拟比较器

如图 1.31 所示，模拟比较器接受正输入端 PB2(AIN0)和负输入端 PB3(AIN1)上的输入电压信号，并使两值进行比较。当正输入端 AIN0 上的电压值高于负输入端 AIN1 上的电压值时，模拟比较器的输出 ACO 置 1。此比较器的输出可用来触发定时/计数器 1 的输入捕捉功能。此外，该模拟比较器也可触发其本身专用的中断请求。用户可选择比较器输出波形的上升沿、下降沿或任一边沿作为中断的触发信号。

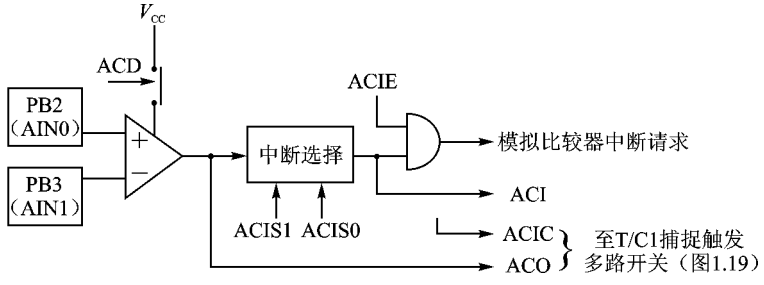


图 1.31 模拟比较器逻辑结构

图 1.31 中的 ACD、ACIE、ACI 以及 ACO 等七个控制和状态信号均来自模拟比较器的控制和状态寄存器 ACSR。

$08	7	6	5	4	3	2	1	0	
($28)	ACD	—	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR

ACSR 寄存器，8 位，地址为 $08($28)，复位值等于 $00。其保留位和 ACO 位只可读，其余各位可读亦可写。

位 7——模拟比较器关断位 ACD：当此位置 1 时，模拟比较器的电源被切断。ACD 可随时用软件置 1，以关断模拟比较器。这可降低 MCU 在工作状态或闲置方式下的功耗。应当注意，欲改变 ACD 位的值，必先将本寄存器的 ACIE 位清 0，以关闭模拟比较器中断。否则 ACD 位发生变化时，可能会产生中断。

位 6——保留位：对 AT90S4414/8515 单片机来说，这是保留位，只可读，且恒读作 0。

位 5——模拟比较器输出 ACO：ACO 位直接与模拟比较器输出端相连，只可读。

位 4——模拟比较器中断标志 ACI：在模拟比较器输出事件触发了 ACIS1 和 ACIS0 两位所定义的中断时，ACI 位置 1。若本寄存器的 ACIE 位和 SREG 寄存器中的 I 位均已被软件置 1，则 ACI 所请求的中断将获得响应。在此响应过程中，ACI 标志被硬件自动清 0。或者用软件向 ACI 位写入逻辑 1，也可将此标志清成 0。然而请注意，在 ACI 已经为 1 的情况下，如果用 SBI 或 CBI 指令修改本寄存器其他位值，那么 ACI 位也将被清 0。

位 3——模拟比较器中断开放位 ACIE：如图 1.31 所示，在 ACIE 位为 1 时，ACI 标志置

位将产生模拟比较器中断请求。若状态寄存器 SREG 中的 I 位也为 1，则此中断请求将获得响应。ACIE 清 0 将使 ACI 标志无法产生中断请求。

位 2——模拟比较器输入捕捉开放位 ACIC：此位置 1 将开放模拟比较器输出 ACO 触发 T/C1 输入捕捉功能的通路。在这种情况下，ACO 直接与输入捕捉前端逻辑相连（见图 1.19），使比较器借用噪音消除器和 T/C1 输入捕捉中断的边沿选择电路。为使比较器能触发 T/C1 输入捕捉中断，定时器中断屏蔽寄存器 TIMSK 中的 TICIE1 位必须置 1。

由图 1.19 可见，ACIC 位清 0，即断开模拟比较器输出 ACO 与输入捕捉电路间的连接，这时 ICP 引脚取代 ACO 将与输入捕捉电路连接起来。

位 1 和位 0——模拟比较器中断方式选择位 ACIS1 和 ACIS0：这两位决定哪些比较器事件能触发模拟比较器中断（见表 1.16）。

表 1.16　ACIS1 和 ACIS0 的中断选择功能

ACIS1	ACIS0	模拟比较器中断	ACIS1	ACIS0	模拟比较器中断
0	0	输出翻转时产生中断	1	0	输出下降沿产生中断
0	1	保　留	1	1	输出上升沿产生中断

应当指出，在修改 ACIS1 和 ACIS0 两位值时，必须先将 ACSR 寄存器中的 ACIE 位清 0，以关闭模拟比较器中断。否则这些位值改变时可能会引发中断。

1.2.13　复位控制电路

AVR 单片机的复位控制是多功能的。AT90S4414/8515 有上电、$\overline{\text{RESET}}$信号及监视定时器等三个复位源。

- 上电复位：当供电电压低于上电复位阈 V_{POT} 时，MCU 复位。
- 外部复位：当$\overline{\text{RESET}}$引脚上的低电平输入信号持续 50 ns 以上时，MCU 复位。
- 监视定时器溢出复位：程序运行故障致使 WDT 溢出时，MCU 复位。

在复位过程中，所有 I/O 寄存器的内容均被复位成它们的初始值。复位后 CPU 将从 $000 地址处开始执行程序。在 $000 地址单元，必须存放一条相对跳转指令 RJMP，以便使程序的执行转移到复位处理程序段去。

图 1.32 所示为 AT90S4414/8515 单片机的复位逻辑，复位电路的电参数及时间参数见表 1.17 所列。

表 1.17　复位参数

符　号	参　数	最小值	一般值	最大值	单　位
V_{POT}	上电复位阈值电压（上升）	0.8	1.2	1.6	V
	上电复位阈值电压（下降）	0.2	0.4	0.6	V
V_{RST}	$\overline{\text{RESET}}$引脚阈值电压	—	—	$0.9V_{CC}$	V
t_{TOUT}	复位延迟时限 FSTRT＝1	11	16	21	ms
t_{TOUT}	复位延迟时限 FSTRT＝0	1.0	1.1	1.2	ms

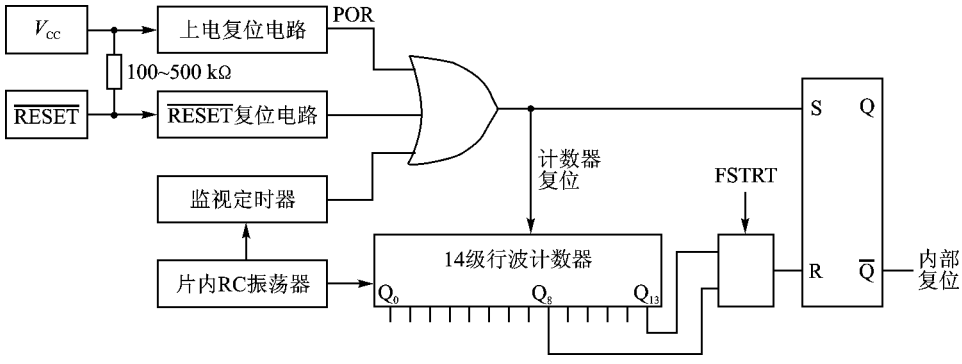


图 1.32 复位逻辑

下面我们将根据复位逻辑图来分析三种复位过程。

1. 上电复位

在 MCU 上电时，供电电压 V_{CC} 有一个上升过程。当上电复位电路 POR 检测到 V_{CC} 达到上电复位阈值 V_{POT} 时，便通过或门(见图 1.32)发一高电平。复位触发器因其 S 端得一高电平信号而进入 1 态，即 $\overline{Q}=0$ 的内部复位状态。当 V_{CC} 达到稳定值时，POR 又将回到 0 态，但这不影响 RS 触发器的状态。

由图 1.32 和图 1.33 可以看到，在内部复位操作开始时，14 级行波计数器也被清 0 并开始对片内 RC 振荡器发出的时钟脉冲进行计数。当闪速存储器中的熔丝位 FSTRT 被编程(即使之为 0)时，由计数器的 Q_8 去触发 RS 触发器的置 0 端 R。若 FSTRT 位未经编程(即为 1)，则 Q_{13} 接至 R 端。在第一种情况下，复位延迟时限(1.0～1.2 ms)一到，Q_8 输出逻辑 1，于是使 RS 触发器翻成 0 态，即 $\overline{Q}=1$，复位过程就此结束(见图 1.33)。第二种情况下，延时 11～21 ms 后，Q_{13} 输出逻辑 1 触发置 0 端 R，从而结束复位过程。

如果使用陶瓷谐振器或其他快速启动振荡器作 MCU 时钟源的话，那么在闪速存储器编程时，也将 FSTRT 位写成 0，这样就可以得到较短的启动时间。

若内在启动延时足够长，则$\overline{\text{RESET}}$引脚可直接或通过外部上拉电阻接至 V_{CC} 引脚。但如果内在启动延时略显不足，那么可在施加供电电压之后，使$\overline{\text{RESET}}$引脚保持一段时间的低电平，这样上电自动复位过程即可有所延长(见图 1.34)。

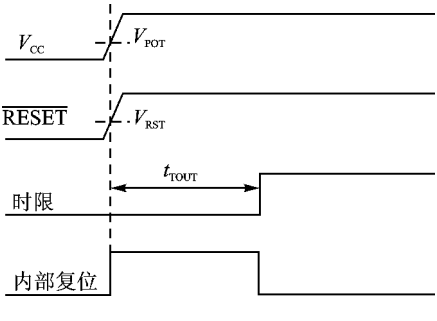


图 1.33 MCU 启动过程($\overline{\text{RESET}}$接 V_{CC})

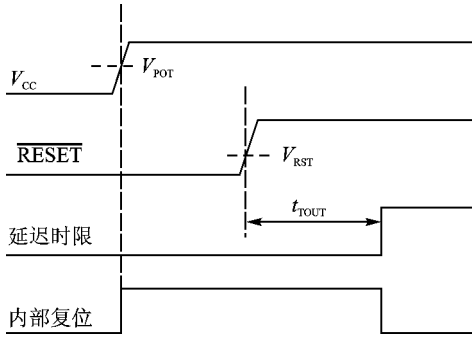


图 1.34 MCU 启动过程($\overline{\text{RESET}}$外控)

不难看出，内部计数器的任务就在于延长复位时间，阻止 MCU 过早启动。这样使系统中的其他外围器件有足够的时间稳定下来，以便与 MCU 一起从初始状态开始运作。

2. 外来信号复位

外部复位是由$\overline{\text{RESET}}$引脚上的低电平输入信号引发的。复位脉冲宽度超过 50 ns 就可产生复位，甚至在 CPU 时钟不运行的情况下也不例外。而较窄的脉冲则不能保证引起复位。当$\overline{\text{RESET}}$信号在上升过程中达到复位阈值电压 V_{RST} 时，如图 1.32 和图 1.35 所示，延时计数器开始对片内 RC 振荡器所提供的时钟脉冲进行计数。延迟时限 t_{TOUT} 一到，复位过程就此结束，MCU 便开始正常运作。

3. 监视定时器复位

监视定时器的时限一到，它就会产生宽度等于一个 XTAL 周期的单一复位脉冲（见图 1.36）。在此脉冲的下降沿处，延时定时器开始对 RC 振荡器输出的脉冲进行计数。在计数值达到延迟时限 t_{TOUT} 所包含的脉冲数目时，Q_8 或 Q_{13} 向复位触发器的置 0 端 R（见图 1.32）输出逻辑 1，从而使复位过程结束，MCU 从 $000 地址处重新执行程序。

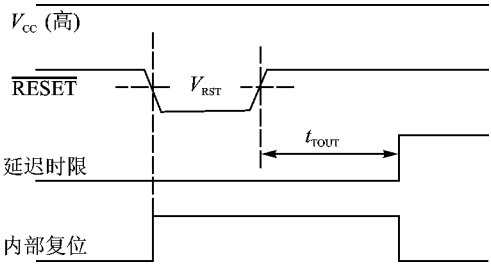


图 1.35　运作过程中的外部复位

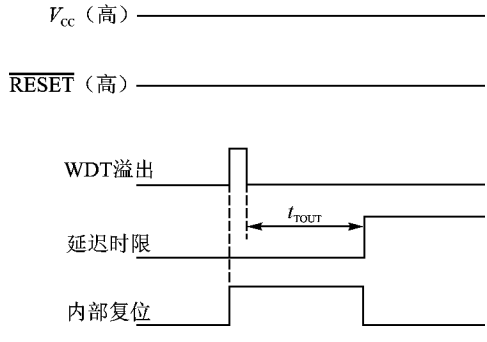


图 1.36　运作过程中的 WDT 复位

1.2.14　中断系统

1. 中断源和中断向量

AT90S4414/8515 有 12 个中断源。每个中断源在程序存储空间某处都有一段服务程序，以其进行中断处理。为进入各自的中断服务程序，在程序存储空间的低地址区，每个中断源均分配给一个称作中断向量的固定单元地址。通常此地址处存放一条两字节的跳转指令，以便转移到相应的中断服务程序去。

表 1.18 所列为上述中断源及其中断向量。表中也列出了各中断源的优先级别，地址越低，优先级别越高。RESET 具有最高的优先级别，其次是外部中断请求 0，等等。就是说，若干个中断源同时请求中断之际，如果看门狗恰好也产生了溢出，那么 MCU 将要复位，而不去响应任何中断。

2. 外部中断

12 个中断源里至今本章尚未交代的便是两个外部中断。它们是由 INT0（PD2）和 INT1（PD3）两个引脚上所发生的事件触发的。请注意，外部中断一经开放，那么 INT0 和 INT1 引脚被设定成输出亦无妨，它们输出的事件同样能触发中断。这一特点可作为产生软件中断的一种方法。触发外部中断的事件可能是信号的上升沿、下降沿或低电平，这由 MCU 控制寄存

器 MCUCR 的最低 4 位来选定。

表 1.18 复位向量和中断向量

向量序号	向量地址	中断源	中断定义
1	$000	RESET	外部复位、上电复位及看门狗溢出复位
2	$001	INT0	外部中断请求 0
3	$002	INT1	外部中断请求 1
4	$003	T/C1 CAPT	定时/计数器 1 捕捉事件
5	$004	T/C1 COMPA	定时/计数器 1 比较 A 匹配
6	$005	T/C1 COMPB	定时/计数器 1 比较 B 匹配
7	$006	T/C1 OVF	定时/计数器 1 溢出
8	$007	T/C0 OVF	定时/计数器 0 溢出
9	$008	SPI STC	SPI 串行传送完成
10	$009	UART RX	UART 接收完成
11	$00A	UART UDRE	UART 数据寄存器空
12	$00B	UART TX	UART 发送完成
13	$00C	ANA COMP	模拟比较器

(1) MCU 控制寄存器 MCUCR:这是一个可读亦可写的 8 位控制寄存器,占据 I/O 存储空间地址 $35($55),复位值为 $00。

$35	7	6	5	4	3	2	1	0	
($55)	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR

位 7——片外 SRAM 开放位 SRE:此位置 1 即开放片外数据 SRAM。这时 A 口的 AD0～AD7 作地址/数据总线;C 口的 A8～A15 作地址总线;而 D 口的$\overline{WR}$和$\overline{RD}$则分别传递读、写控制信号。SRE 位置 1 使上述端口的数据方向寄存器之设定值失效。当 SRE 位清 0 时,片外数据 SRAM 禁止使用,上述引脚的运作要遵从数据方向的设定。

位 6——片外 SRAM 等待状态控制位 SRW:此位清 0 就意味着对片外数据 SRAM 的访问为通常的三个时钟周期机制,无须插入等待状态。而 SRW 位置 1 即设定,在片外数据 SRAM 的访问周期中要插入一个时钟周期的等待状态。即这种访问要用总计 4 个时钟周期,见图 1.57 和图 1.58。

位 5——休眠方式开放位 SE:欲使 MCU 在执行 SLEEP 指令后进入休眠方式,SE 位必须预先置 1。为避免 MCU 在非程序人员意愿情况下进入休眠方式,建议刚好在 SLEEP 指令执行前把 SE 位置 1。

位 4——休眠方式选择位 SM:这位的任务是负责在两种休眠方式间进行选择。若 SM 位为 0,则 SLEEP 指令的执行将使 MCU 进入闲置方式;若 SM 位为 1,则在执行 SLEEP 指令后,MCU 将以掉电方式来休眠。

位 3 和位 2——中断触发控制位 ISC11 和 ISC10:此两位负责定义 INT1 引脚上触发外部

中断的事件(见表 1.19)。

表 1.19　外部中断 1 触发信号

ISC11	ISC10	触发信号
0	0	INT1 引脚上的低电平信号产生中断请求
0	1	保留
1	0	INT1 信号下降沿产生中断请求
1	1	INT1 信号上升沿产生中断请求

位 1 和位 0——中断触发控制位 ISC01 和 ISC00:这是外部中断 0 的触发事件设定位。两位值的四种组合所设定的情况如表 1.20 所列。

表 1.20　外部中断 0 触发信号

ISC01	ISC00	触发信号
0	0	INT0 引脚上的低电平信号产生中断请求
0	1	保留
1	0	INT0 信号下降沿产生中断请求
1	1	INT0 信号上升沿产生中断请求

应当指出,在设定两外部中断的触发信号时,必须先将 GIMSK 寄存器中的有关中断开放位清 0,以暂时关闭 INT0 或 INT1。否则,在对上述 4 位赋新值时,会引发中断。

若选中外部中断的边沿触发方式,则要求触发信号的高低电平持续时间均不得小于一个时钟周期。因为边沿检测是通过对引脚电平的采样来进行的,每个时钟周期采样一次。例如,某时钟周期采样到一高电平,而此后某周期又发现一低电平,则这就检测到了一个 1 到 0 的负跳变,即下降沿。不难理解,如果脉冲宽度小于一个时钟周期,那么,这个脉冲的两个边沿可能都检测不到。

至于电平触发方式,外部中断引脚上的低电平信号应一直保持,直到中断获得响应时方可撤消。

(2) 通用中断屏蔽寄存器 GIMSK:8 位,其中最高两位可读写,其余 6 个保留位只可读,且恒读作 0。GIMSK 占据 I/O 存储空间的地址为 $3B($5B),复位值等于 $00。

$3B	7	6	5	4	3	2	1	0	
($5B)	INT1	INT0	—	—	—	—	—	—	GIMSK

位 7——外部中断请求 1 开放位 INT1:当 INT1 位和状态寄存器 SREG 中的 I 位均被软件置 1 时,INT1(PD3)引脚上的外部中断请求即获准运作。如前所述,MCU 控制寄存器 MCUCR 中的中断触发控制位 ISC11 和 ISC10 决定 INT1 引脚上的信号上升沿、下降沿或低电平触发中断。若已开放了 INT1 中断,则在此引脚被设定为输出的情况下,它上面发生的有效事件仍能引发中断。由表 1.18 得知,INT1 中断向量地址为 $002。

位 6——外部中断请求 0 开放位 INT0:意义同 INT1 位,不过是控制外部中断 0 的。

INT0(PD2)引脚上的有效触发信号是由 MCUCR 寄存器中的 ISC01 和 ISC00 两位决定的。INT0 的中断向量地址为 $001。

(3) 通用中断标志寄存器 GIFR:I/O 地址为 $3A($5A),复位值等于 $00。8 位,其最高两位可读写,其余 6 位为保留位,只可读,且恒读作 0。

$3A	7	6	5	4	3	2	1	0	
($5A)	INTF1	INTF0	—	—	—	—	—	—	GIFR

位 7——外部中断标志 INTF1:INT1 引脚上的事件触发中断时,INTF1 标志被自动置 1。若此时 SREG 寄存器中的 I 位及 GIMSK 寄存器中的 INT1 位已被软件置 1,则 MCU 将跳至 $002 地址处去执行中断服务程序。

INTF1 标志在中断响应过程中被硬件自动清 0。用软件向此位写入逻辑 1 也可将之清 0。

位 6——外部中断标志 INTF0:这是 INT0 的中断标志,其运作机理(包括标志的清 0 方法)同 INTF1,中断向量为 $001。

3. 中断标志和中断屏蔽

每个中断源都有自己的中断标志和中断开关(即中断开放位,亦可称屏蔽位)。只要条件满足,中断标志便被硬件自动置 1,不受屏蔽位的影响。例如,当定时器溢出、UART 发送或接收完成、输入捕获和输出比较匹配时,各自标志均被置 1。但这些置 1 后的标志能否形成中断请求,那要受中断开关的限制。其开放位为 1 者,可形成中断请求;为 0 者便被禁止产生中断请求。

下面我们补述一个中断屏蔽寄存器和一个中断标志寄存器。

(1) 定时/计数器中断屏蔽寄存器 TIMSK:I/O 地址为 $39($59),复位值等于 $00。TIMSK 含 8 位,其中 3 个保留位只可读,且恒读作 0,其余 5 位可读亦可写。

$39	7	6	5	4	3	2	1	0	
($59)	TOIE1	OCIE1A	OCIE1B	—	TICIE1	—	TOIE0	—	TIMSK

位 7——定时/计数器 1 溢出中断开放位 TOIE1:TOIE1 位置 1 和 SREG 寄存器中的 I 位置 1 即开放了定时器 1 溢出中断。这样,在定时器 1 溢出时,定时器中断标志寄存器 TIFR 中的 TOV1 位自动置 1,于是便产生中断请求,并最终将获得响应。即 CPU 暂时脱离背景程序去执行 $006 地址处的指令,从而转至定时器 1 溢出中断服务程序。

若用软件把 TOIE1 位清 0,则定时器 1 溢出时也不能产生中断请求,就是说此中断源被屏蔽了。

位 6——T/C1 输出比较 A 匹配中断开放位 OCIE1A:这是 T/C1 输出比较 A 匹配的中断开关。当 A 匹配发生时,TIFR 中的 OCF1A 标志被自动置 1。若此前 OCIE1A 位已被软件置 1,则将有 A 匹配中断请求发出。OCIE1A 位清 0 将屏蔽掉 OCF1A 标志的中断请求。

位 5——T/C1 输出比较 B 匹配中断开放位 OCIE1B:这是 T/C1 输出比较 B 匹配的中断开关。其功能同 OCIE1A,只不过它所控制的中断标志是 OCF1B。

位 4、位 2 和位 0——保留位,只读,且恒读作 0。

位 3——T/C1 输入捕捉中断开放位 TICIE1:某捕捉触发事件在 ICP 引脚上发生时,

TIFR 中的 ICF1 标志被硬件置 1。TICIE1 位为 1 则允许 ICF1 标志形成中断请求；TICIE1 位清 0 即禁止形成这样的中断请求。

位 1——T/C0 溢出中断开放位 TOIE0：和 TOIE1 意义相同，只不过它是 T/C0 溢出标志 TOV0 的中断开关。

(2) 定时/计数器中断标志寄存器 TIFR：I/O 地址为 $38($58)，复位值等于 $00。TIFR 由 8 位组成，其中 3 个保留位，只读，且恒读作 0。其他可读写的 5 位意义如下。

$38	7	6	5	4	3	2	1	0	
($58)	TOV1	OCF1A	OCF1B	—	ICF1	—	TOV0	—	TIFR

位 7——T/C1 溢出标志 TOV1：当定时/计数器 1 溢出时，TOV1 标志自动置 1。该标志在其中断请求获得响应的过程中被硬件清 0。或者，在不请求中断的情况下，用软件向该位中写入逻辑 1，亦可将之清 0。

在 PWM 方式下，当 T/C1 在 $0000 计数值处改变计数方向时，TOV1 标志也被硬件自动置 1。

位 6——输出比较标志 OCF1A：当 T/C1 的计数值与输出比较寄存器 OCR1A 中的数值相等时，或简称 A 匹配发生时，OCF1A 标志被硬件自动置 1。在 OCIE1A 位为 1 的情况下，此标志可产生中断请求，否则该请求将被屏蔽。

OCF1A 标志的清 0 方法同 TOV1，软、硬件方法均可。

位 5——输出比较标志 OCF1B：B 匹配发生时，OCF1B 标志自动置 1。若 OCIE1B 位为 1，则可产生 B 匹配中断请求。但 OCIE1B 位为 0 将禁止 OCF1B 标志产生中断请求。

OCF1B 标志在其中断请求获得响应时被自动清 0，或者通过用软件向其写逻辑 1 的办法亦可将其清 0。

位 4、位 2 和位 0——保留位，只可读，且恒读作 0。

位 3——输入捕捉标志 ICF1：此位被置 1 标志着某输入捕捉事件发生了，告知 T/C1 值已经被传送到了输入捕捉寄存器 ICR1。

ICF1 标志的清 0 方法同上述几个标志，软、硬件方法均可。

位 1——T/C0 溢出标志 TOV0：这是定时/计数器 0 的溢出标志，T/C0 溢出时置 1，清 0 方法与上述 4 个标志相同。

在结束这一小节的时候，我们把各中断源及其相应的标志、屏蔽位和清 0 方法等项内容列于表 1.21 中，以便读者随时浏览。表中括号里的内容系指某寄存器的某位。

4. 中断响应

任何 AVR 中断的响应时间至少要 4 个时钟周期。就是说，中断标志置 1 后，最少要经历 4 个时钟周期才能去执行中断向量处的指令。在这 4 个时钟周期中，程序计数器的内容(两个字节)被压入堆栈保存起来，堆栈指针值减 2。此堆栈操作耗时 4 个时钟周期，保存的是返回地址，即刚刚结束的那条指令紧后面一条指令的地址。接着，程序计数器被赋予一个新值，即中断向量地址。这意味着马上要执行这里的指令了。如前所述，在中断向量地址处，一般安排一条相对跳转指令，以便使控制转向中断服务程序段去。这条跳转指令又要花费两个时钟周期。

应当指出，程序的执行可以被打断，但指令的执行却不可。例如一条多周期指令执行过程中并非其最后一个时钟周期发现有中断请求，那么 CPU 不能马上响应此中断，而待此指令执

行完毕时才能响应。

表 1.21 中断源、中断标志及中断屏蔽位一览表

中断源	中断标志	屏蔽位	标志清 0 方法
INT0	INTF0(GIFR.6)	INT0(GIMSK.6)	中断响应;软件写 1
INT1	INTF1(GIFR.7)	INT1(GIMSK.7)	中断响应;软件写 1
T/C1 CAPT	ICF1(TIFR.3)	TICIE1(TIMSK.3)	中断响应;软件写 1
T/C1 COMPA	OCF1A(TIFR.6)	OCIE1A(TIMSK.6)	中断响应;软件写 1
T/C1 COMPB	OCF1B(TIFR.5)	OCIE1B(TIMSK.5)	中断响应;软件写 1
T/C1 OVF	TOV1(TIFR.7)	TOIE1(TIMSK.7)	中断响应;软件写 1
T/C0 OVF	TOV0(TIFR.1)	TOIE0(TIMSK.1)	中断响应;软件写 1
SPI STC	SPIF(SPSR.7)	SPIE(SPCR.7)	中断响应;先读 SPSR,再访问 SPDR
UART RX	RXC(USR.7)	RXCIE(UCR.7)	读 UDR 寄存器
UART UDRE	UDRE(USR.5)	UDRIE(UCR.5)	写 UDR 寄存器
UART TX	TXC(USR.6)	TXCIE(UCR.6)	中断响应;软件写 1
ANA COMP	ACI(ACSR.4)	ACIE(ACSR.3)	中断响应;软件写 1;用 SBI 或 CBI 指令修改 ACSR 其他位

在中断响应过程中,总体中断开放位 I 被自动清 0,这为该中断处理不受其他中断源的干扰创造了条件。在中断服务程序中,用户可将 I 位置 1,以允许中断嵌套。当中断服务程序结束(即执行中断返回指令 RETI)时,I 位被自动置 1。

在中断向量地址装入程序计数器时,硬件把产生此次中断请求的中断标志清 0。有些中断标志亦可通过向该标志位写入逻辑 1 的办法来清 0,如表 1.21 所列。

有的中断触发事件,例如脉冲边沿,稍纵即逝。也有的可以维持相当长一段时间,例如比较匹配,只要 T/C1 值尚未增 1,匹配状态就将一直保持。由表 1.4 可见,这种状态最多可维持 1 024 个 CPU 时钟周期。对于这样的事件所触发的中断来说,当然事件一发生,中断标志即被置 1。但是,若中断标志已被清 0,而中断条件却依然存在,则在下一次此类事件发生之前,该标志不会再次置 1。

如果某中断条件已具备,但其相应开放位却为 0,那么中断标志照例要自动置 1,但形不成中断请求。该标志将一直保持,直到此中断开放并获得响应,或至被软件清 0 时为止。

若一个或多个中断条件已具备,但总体中断开放位 I 却为 0,则相应中断标志被自动置 1,且将一直保持至 I 位置 1,并按优先级顺序依次被响应时为止。

应当注意,电平触发外部中断没有标志,只要条件有效,那么中断请求就将一直保持下去。

中断返回指令执行时间为 4 个时钟周期(子程序返回指令亦如此)。在这 4 个时钟周期中,堆栈栈顶的两个字节,即返回地址,被弹出来送入程序计数器中,SP 值加 2,且 I 标志被恢复成置位状态。中断处理完毕时,CPU 的控制总要返回到主程序,在那里再执行一条指令后,才能去响应此前挂起的中断请求。

应当指出,中断响应和子程序调用过程中,硬件只负责保护断点地址(返回地址)。如果也需要保护状态寄存器 SREG 的内容,那么用户得在子程序或中断服务程序的开始部分用软件

来完成。

1.2.15　I/O 端口

作通用数据 I/O 口时，所有 AVR 端口都具有真正的读—修改—写功能。这意味着，某端口引脚的方向可用 SBI 或 CBI 指令来改变，而不会无意中引起任何其他引脚方向的变化。作输出时某引脚输出值的改变以及作输入时其上拉电阻的接通与分断也是如此，即某引脚的变化不会波及“邻居”。

AT90S4414/8515 有 4 个 I/O 端口。

1. 端口 A

端口 A 是一个 8 位双向 I/O 口。

端口 A 分配有三个 I/O 存储地址：$1B($3B)，数据寄存器 PORTA；$1A($3A)，数据方向寄存器 DDRA；$19($39)，A 口引脚 PINA。A 口输入引脚地址是只读的，而数据寄存器及数据方向寄存器可读亦可写。

所有端口引脚都具有各自可通可断的上拉电阻。A 口输出缓冲器可吸收 20 mA 的灌电流，因此能直接驱动 LED 显示器。当 PA0～PA7 引脚作输入用，且从外部被拉成低电平时，在启用了内部上拉电阻的情况下，它们将向外提供电流 I_{IL}。

(1) A 口数据寄存器 PORTA：8 位，可读亦可写，复位值等于 $00，地址为 $1B($3B)。

图 1.37 中的 PORTAn(n=0～7)就是此寄存器的一位锁存器。由图中可以看到，在读锁存器信号 RL 的作用下，三态门开通，于是 PORTAn 这位数据便由 Q 输出端经三态门到达数据总线。而在写 PORTA 脉冲 WP 的作用下，数据总线上来的数据由锁存器的数据输入端 D 被锁入内部。以上分别是对数据寄存器的读和写操作。

(2) A 口数据方向寄存器 DDRA：8 位，可读亦可写，复位值等于 $00，I/O 地址为 $1A($3A)。顾名思义，DDRA 寄存器的任务就在于为 A 口 8 条 I/O 线 PA7～PA0 选择数据方向。即设定哪个作输入，哪个作输出。

A 口的 8 条 I/O 线可作通用数字 I/O，亦可作低位地址/数据总线使用。

① PAn 作通用 I/O 引脚：如果数据方向寄存器 DDRA 中的 DDAn 位被软件置 1，加之 A 口作通用 I/O 时 SRE 位应当为 0，那么与门 2(见图 1.37)和或门 1 将输出逻辑 1，于是三态门开通。这样，数据寄存器 PORTA 的 PORTAn 位便通过与门 4、或门 2 和三态门经引脚 PAn 输出出去。若 DDAn 被清 0，则与门 2 和与门 3 均被封锁，故或门 1 输出逻辑 0，它所控制的三态门呈高阻状态，无输出通路。而此时在读引脚信号 RP 的作用下，与门 7 和或门 3 输出高电平，打开它们控制的三态门，于是来自外界的数据便可通过 PAn 引脚、史密特触发器和三态门到达片内数据总线，最后被读到某个工作寄存器中去。

综上所述，数据方向寄存器某位 DDAn 为 1 时，PAn 被设定为输出引脚；DDAn 清 0，则指定 PAn 作输入引脚。

PAn 作输入引脚时，其内部 MOS 上拉电阻可启用，亦可关断。如前所述，DDAn 为 0 时，PAn 作输入，若此时 PORTAn 位置 1，则与门 1 将输出低电平，于是上拉电阻接通。欲关断作为上拉电阻的 MOS 晶体管，就应把 PORTAn 位清 0，或将引脚设定为输出。

② A 口引脚作低位地址/数据总线：当 AVR 单片机外扩数据存储器 SRAM 时，A 口作访问片外 SRAM 的低位地址/数据总线，用来先发出地址码的低 8 位，再接收或发出所读写的数

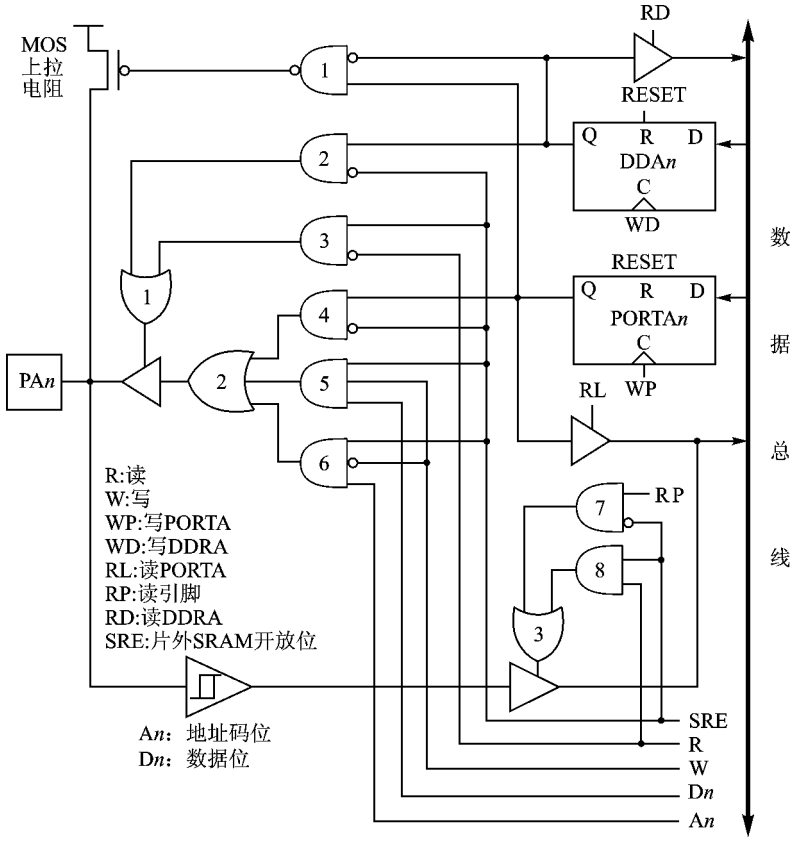


图 1.37　A 口引脚 PA0～PA7 内部结构

据。这是 A 口的第二功能。在这种方式下，其内部上拉电阻是接通的（图 1.37 中无此表示。）欲启用 A 口的第二功能，必须用软件将 MCU 控制寄存器 MCUCR 中的片外 SRAM 开放位 SRE 置 1。此时 A 口数据方向寄存器 DDRA 不再起作用。

SRE 位置 1，与门 2 和与门 4 被封锁。故 DDRA 和 PORTA 两寄存器不再参与端口的 I/O运作。访问片外 SRAM 时，在暂时没有读和写操作的情况下，R＝0，W＝0，因此与门 3 和或门 1 输出高电平，三态门开通。这时地址码 A*n* 通过与门 6、或门 2 和三态门，最后经 PA*n* 引脚输出到片外地址锁存器，并在 ALE 信号的下降沿到来时锁入其中。如果对片外 SRAM 的访问是要向它写数据，那么写信号 W＝1 时，数据位 D*n* 则通过与门 5、或门 2 和三态门，经 PA*n* 引脚写到片外 SRAM 已被选定的地址单元中去。同理，在读片外 SRAM 时，读信号 R＝1，于是与门 8、或门 3 输出高电平，打开输入三态门，数据便可从片外 SRAM 被选定的地址单元中读出，经 PA*n* 引脚到达片内数据总线，最后进入某工作寄存器中。

（3）A 口输入引脚地址 PINA，＄19（＄39）：这不是一个寄存器，但该地址能使用户读到 A 口引脚上的数据。此 I/O 地址单元只可读，复位后 PINA7～PINA0 均呈高阻状态。

应当指出，PA7～PA0 代表引脚，而 PINA7～PINA0 则专指作输入的引脚。

最后，我们把 A 口方向寄存器 DDRA 对各引脚的控制作用以表格的形式列出（见表 1.22），以便读者随时浏览。

表 1.22　DDA*n* 对 A 口引脚的控制作用

DDA*n*	PORTA*n*	PA*n* I/O	上拉电阻	说　明
0	0	输入	无	三态(高阻)
0	1	输入	有	若被从外部拉成低电平,则放出电流
1	0	输出	无	推挽 0 输出
1	1	输出	无	推挽 1 输出

2. 端口 B

B 口也是一个 8 位双向端口。对它的访问可通过三个 I/O 地址进行。

- $18($38):B 口数据寄存器 PORTB,可读亦可写,复位值等于 $00。
- $17($37):B 口数据方向寄存器 DDRB,可读亦可写,复位值为 $00。
- $16($36):B 口输入引脚地址 PINB,只可读,复位后各引脚呈高阻状态。

B 口的驱动能力、上拉电阻的接通与分断、作通用 I/O 时数据方向的选择、复位条件下各引脚呈高阻状态等情况与 A 口无异,故不再赘述。

B 口各引脚除可作通用 I/O 外,尚有其第二功能,如表 1.23 所列。

表 1.23　B 口引脚第二功能

端口引脚	第 二 功 能
PB0	T0(定时/计数器 0 外部计数输入)
PB1	T1(定时/计数器 1 外部计数输入)
PB2	AIN0(模拟比较器正输入)
PB3	AIN1(模拟比较器负输入)
PB4	$\overline{SS}$(SPI 从机选择输入)
PB5	MOSI(SPI 总线主机输出/从机输入)
PB6	MISO(SPI 总线主机输入/从机输出)
PB7	SCK(SPI 总线串行时钟)

下面我们分别讨论各引脚的情况。

(1) PB7——SCK:SPI 通道的主机时钟输出、从机时钟输入引脚。SPI 被选作从机运作将迫使此引脚作为输入,不管 DDB7 设定为何值。而当 SPI 被设定为主控方式时,PB7 的数据方向要受 DDB7 位控制。在此引脚被强制作输入的情况下,上拉电阻仍要受 PORTB7 位控制。

① SPI 被选作从机时,主机选择位 MSTR 应当为 0,SPI 开放位 SPE 为 1。故图 1.38 中的与门 1 输出高电平,打开输入三态门,于是外来 SPI 时钟信号自 PB7(SCK)引脚输入进来,经史密特触发器和三态门最后到达 SPI 部件。此时或门 1 输出逻辑 0,封锁与门 2。所以不管 DDB7 位为何值,输出三态门都是不通的。综上所述,SPI 作从机,PB7 便被强制作 SPI 时钟输入引脚。

② SPI 作主机时,MSTR 和 SPE 两位均为 1,故与门 3 输出逻辑 1。于是 SPI 时钟输出信

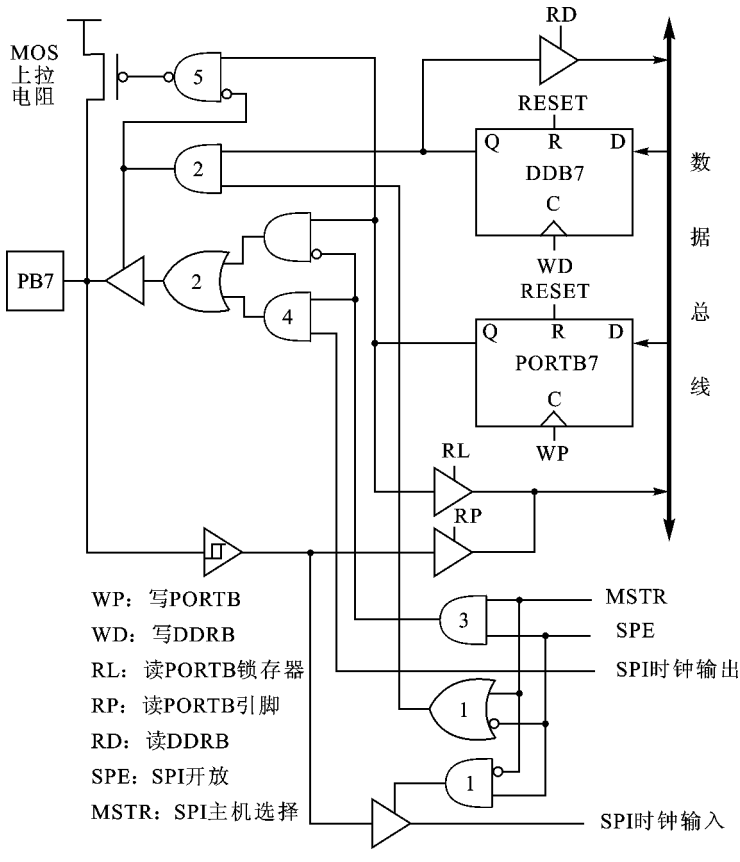


图 1.38　PB7 引脚内部结构

号便可经与门 4 和或门 2 到达输出三态门。此时或门 1 因其一个输入端 MSTR 为 1 而输出高电平，所以与门 2 能否输出高电平，以打开三态门让时钟信号通过 PB7 输送出去，要看与门 2 的另一个输入信号——DDB7。若后者也为 1，即设定 PB7 作输出，则 SPI 时钟输出通路就是畅通的。这就是说，SPI 作主机，PB7 的数据方向要受 DDB7 位控制。

③ 如前所述，PB7 被强制作输入时，与门 2 输出逻辑 0，因此与非门 5 能否输出低电平以开通 MOS 上拉电阻将取决于 PORTB7 位的值。PORTB7 若为 1，则上拉电阻接通；PORTB7 若为 0，则上拉电阻分断。

（2）PB6——MISO：SPI 通道的主机数据输入、从机数据输出引脚。读者可以根据图 1.39 所示的 PB6 引脚之内部结构，参照有关 PB7 的说明，解释下列三点机理：

① 设定 SPI 为主机将迫使此引脚为输入，DDB6 的设定值此时不起作用。

② 如果把 SPI 设定为从机，那么 PB6 引脚的数据方向将受控于 DDB6 位。

③ 当此引脚被强制作输入时，其上拉电阻的通断仍受 PORTB6 位控制。

（3）PB5——MOSI：SPI 通道的主机数据输出、从机数据输入引脚。图 1.40 所示为 PB5 引脚内部结构，不难看出，它与 PB7 的内部结构（见图 1.38）完全一致，只是在主控方式下，PB5 输出数据位，而 PB7 则输出时钟信号。在从控方式下，PB5 输入数据位，PB7 输入时钟信号。有关 PB5 的第二功能，我们强调下列几点，原理不再赘述。

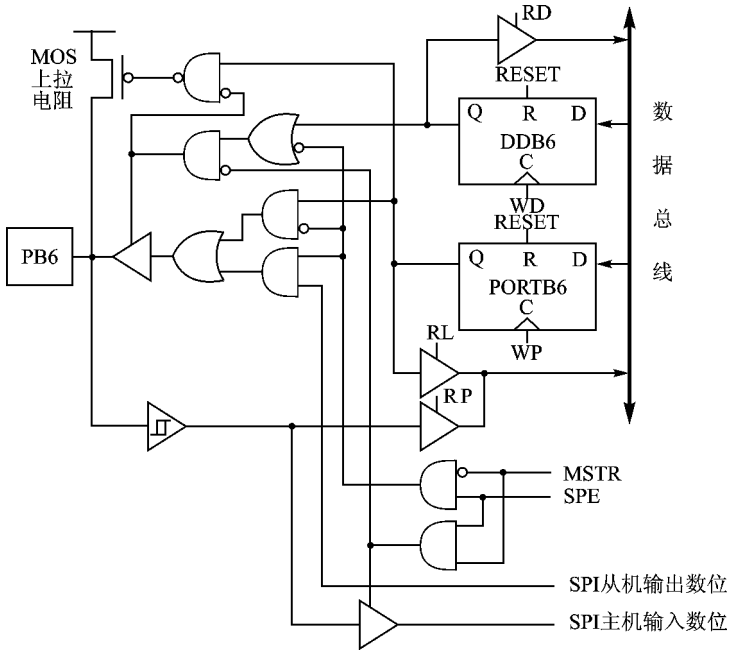


图 1.39　PB6 引脚内部结构

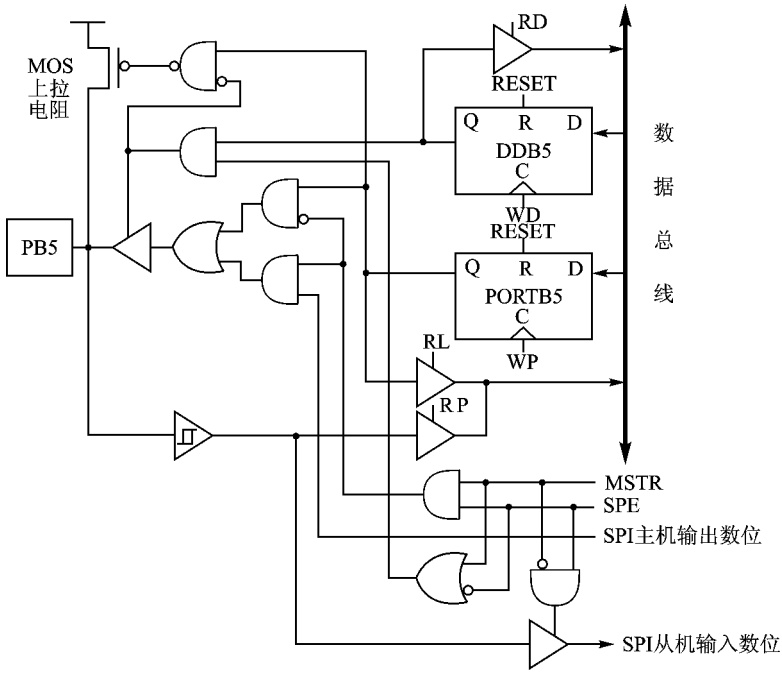


图 1.40　PB5 引脚内部结构

① **若将** SPI **设定为从机，则** PB5 **被强制作输入引脚，以接收主机送来的数据位。而此时数据方向位** DDB5 **不起作用。**

② **当** SPI **被设定为主机时，该引脚之数据方向受控于** DDB5 **的设定值。**

③ 在 SPI 被强制作输入时，内部上拉电阻的通断要受锁存器 PORTB5 值的控制。

(4) PB4——$\overline{SS}$：这是从控端口选择输入引脚。若本 SPI 被设定为从控方式，则图 1.41 中的或门 1 因 MSTR＝0，SPE＝1 而输出逻辑 0。故与门 1 被封锁，不管其另一输入信号 DDB4 位置 1 还是清 0，它都输出逻辑 0，致使输出三态门呈高阻状态。这就是说，SPI 的从控方式迫使 PB4 作输入引脚，而数据方向位 DDB4 此时不起作用。

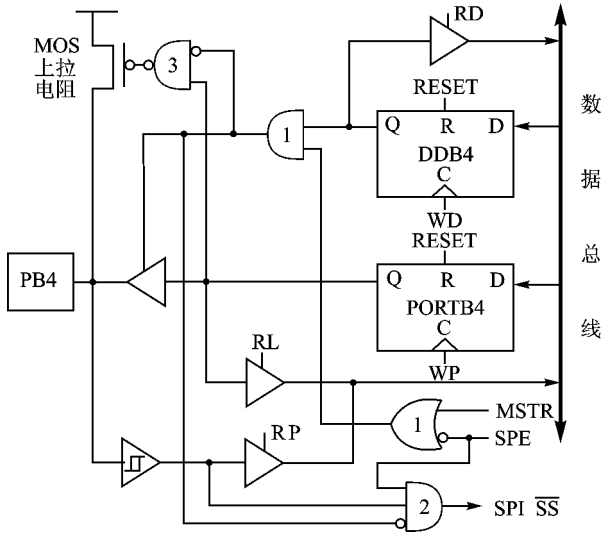


图 1.41 PB4 引脚内部结构

在 SPI 的从控方式下，SPE＝1，与门 1 输出 0，反相后加至与门 2。这样与门 2 的输出，即 SPI 的$\overline{SS}$信号，当然就取决于 PB4 引脚上的输入电平了。若 PB4 被从外部拉成低电平，则与门 2 将向 SPI 部件输入一个低电平的$\overline{SS}$信号，从而启动 SPI 的从控运作。

SPI 的从控方式强制 PB4 作输入引脚时，与门 1 输出 0，于是与非门 3 的输出将取决于 PORTB4 位的值。若后者为 1，则与非门 3 输出 0，使 MOS 上拉电阻投入运行。但如果 PORTB4 位此时为 0，那么与非门 3 将输出高电平，关闭上拉电阻。

当 SPI 被设定为主控方式时，由图 1.41 可知，数据方向是由 DDB4 位确定的。若 DDB4＝1，同时或门 1 因 MSTR＝1 而输出高电平，于是与门 1 输出逻辑 1 打开输出三态门。这样，数据寄存器中的 PORTB4 位便通过三态门经 PB4 引脚输送出去。此时这是一个通用 I/O 引脚，不影响 SPI 系统运作。

(5) PB3——AIN1：模拟比较器负输入引脚。图 1.42 所示为 PB3 和 PB2 引脚的内部结构。从图中可以看到，若将数据方向位 DDB3 写成 0，则输出三态门呈高阻状态，即 PB3 被设定为输入引脚。将数据位 PORTB3 也清成 0，则上拉电阻被关断。这时 PB3 可作通用输入引脚，也

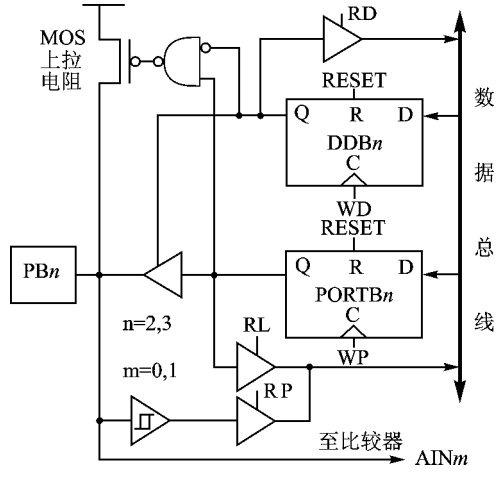


图 1.42 PB2 和 PB3 引脚内部结构

可作片内模拟比较器的负输入引脚 AIN1。

(6) PB2——AIN0：模拟比较器正输入引脚。在图 1.42 中，若将 DDB2 和 PORTB2 均清成 0，则 PB2 也可作片内模拟比较器的正输入引脚 AIN0。

(7) PB1——T1：定时/计数器 1 的计数脉冲源。这对应表 1.4 中的最后两种情况。

(8) PB0——T0：定时/计数器 0 的计数脉冲源，请参阅表 1.3 中最后两项。

图 1.43 所示为 PB0(T0)和 PB1(T1)两引脚的内部结构。最后应当指出，B 口的通用 I/O 用途与 A 口相同，故可参考表 1.22。

3. 端口 C

C 口为 8 位双向 I/O 口，对它的访问可通过三个 I/O 存储地址进行。

● $15($35)：C 口数据寄存器 PORTC，可读也可写，复位值等于 $00。

● $14($34)：C 口数据方向寄存器 DDRC，可读也可写，复位值为 $00。

● $13($33)：C 口引脚地址 PINC，只可读，复位后各引脚呈高阻状态。这不是一个寄存器，但此地址却使用户能够读到 C 口引脚上的逻辑值。而读 PORTC 时，读之所得为数据锁存器的内容。

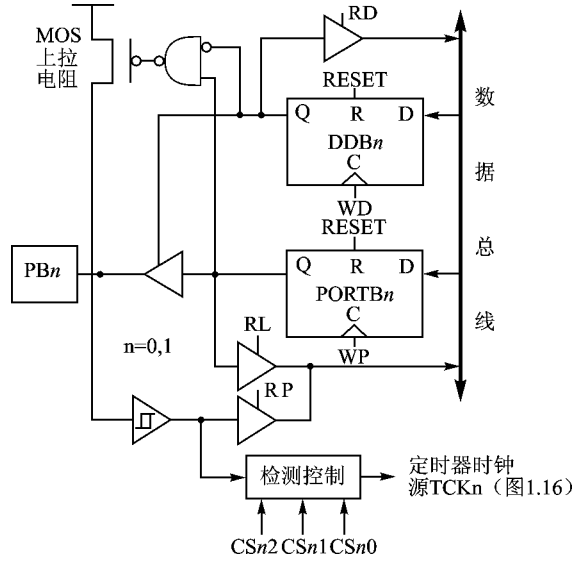


图 1.43　PB0 和 PB1 引脚内部结构

C 口各引脚内部都具有可通可断的上拉电阻。C 口输出缓冲器可吸收 20 mA 灌电流，因此能直接驱动 LED 显示器。当 PC0～PC7 作输入，且从外部被拉成低电平时，如果内部上拉电阻接通的话，便可向外提供电流。

除可作通用 I/O 外，C 口的第二功能是在 MCU 外扩有数据存储器的情况下作地址总线，传送片外 SRAM 地址码的高 8 位。把 MCU 控制寄存器 MCUCR 中的片外 SRAM 开放位 SRE 置 1，就将 C 口设定为地址总线。这种第二功能的设定使数据方向寄存器值失效。

(1) 作通用数字 I/O 口：作通用数字 I/O 引脚时，C 口的 8 条 I/O 线具有同等功能。此时数据方向寄存器 DDRC 中的 DDC*n* 位负责设定相应引脚 PC*n* 的数据方向。如图 1.44 所示，将 DDC*n* 位置 1，就可使或门 1 输出高电平，从而打开输出三态门。于是数据寄存器 PORTC 中的 PORTC*n* 位便通过与门 1、或门 2 和输出三态门并最后经 PC*n* 引脚输出至片外。与门 1 的另一个输入信号为 SRE 位的反态。在作通用 I/O 时，SRE＝0，其反态为 1。故与门 1 能正确地传递 PORTC*n* 值。

若将 DDC*n* 位清 0，则输出三态门呈高阻状态，此时 PC*n* 被设定为输入引脚。如果把 PORTC*n* 位置 1，那么与非门 3 将输出低电平，使 MOS 上拉电阻接通。欲关断上拉电阻，则应将 PORTC*n* 位清 0，或把 PC*n* 置成输出方式。

[例 1.14]　为什么说复位后各端口引脚均呈高阻状态？

由各端口引脚的内部结构图中可以看到，复位信号接到了各口数据方向寄存器和数据寄存器每位锁存器的清 0 端 R，因此两寄存器的复位值均为 $00。这时各引脚都被置成了输入

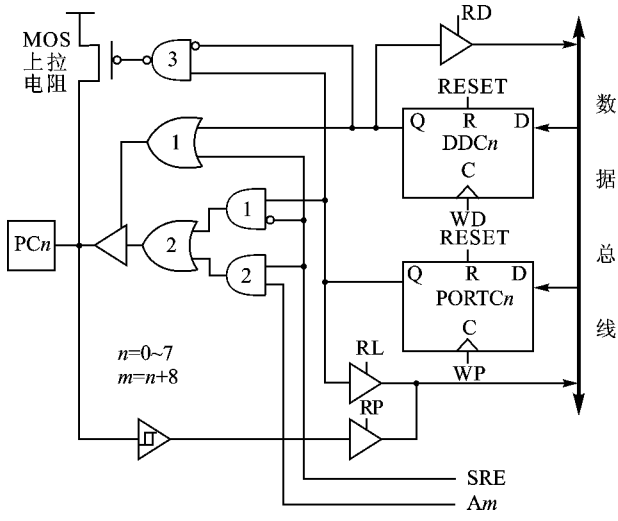


图 1.44　PC0～PC7 引脚内部结构

方式，且上拉电阻关断，输入、输出三态门不通。这就等于说引脚处于浮动状态，和内部无联系，这就是所谓高阻状态（见表 1.22 中的第一项）。应当指出，表 1.22 适用于所有四个端口，只不过把其中的“A”分别读作 A～D 就是了。

（2）作地址总线：由图 1.44 可见，把片外 SRAM 开放位 SRE 置 1，就可使或门 1 输出高电平，打开输出三态门，且此时数据方向控制位 DDC*n* 怎么设定都不起作用。与门 1 因 SRE 位为 1 而被封锁，故向或门 2 输出低电平；而与门 2 则把地址码 A*m* 如实地传递给或门 2，并最后经三态门和引脚 PC*n* 输出至片外 SRAM。C 口输出地址码的高 8 位。

4. 端口 D

这是 AVR 单片机的第四个 8 位双向 I/O 口。它的三个 I/O 存储地址如下。

- $12（$32），D 口数据寄存器，可读亦可写，复位值等于 $00。
- $11（$31），D 口数据方向寄存器 DDRD，其 8 位 DDD7～DDD0 均可读写，复位值为 $00。
- $10（$30），D 口输入引脚地址 PIND，只可读，复位后各引脚呈高阻状态。

D 口各 I/O 线的驱动能力、内部上拉电阻的通断、作通用 I/O 时数据方向的选择等项内容与其他三口相同。下面我们只着重讨论 D 口的第二功能。

除 PD4 外，D 口其他引脚均有第二功能，如表 1.24 所列。

（1）PD7——$\overline{RD}$：这是对片外 SRAM 的读选通信号输出引脚。如图 1.45 所示，若将片外 SRAM 开放位 SRE 置 1，则或门 1 就输出高电平，从而打开输出三态门。此时数据方向控制位 DDD7 无论如何设定均不再有效。与门 1 因 SRE 位为 1 而输出 0，因此与门 2 将如实地把读选通信号$\overline{RD}$传递给或门 2，并经三态门和引脚 PD7 输出到片外 SRAM 的$\overline{RD}$输入端。

（2）PD6——$\overline{WR}$：此引脚传递对片外数据存储器的写选通信号。PD6 与 PD7 两引脚的内部结构完全相同，只不过 PD6 的与门 2 传递的是写选通信号$\overline{WR}$，而非 PD7 的$\overline{RD}$。

（3）PD5——OC1A：T/C1 的 A 匹配输出引脚。由图 1.46 可见，若把方向控制位 DDD5 写成 1，则输出三态门开通，即 PD5 作输出引脚。但这里有两种情况。

表 1.24　D 口引脚的第二功能

端口引脚	第二功能
PD0	RXD(UART 输入线)
PD1	TXD(UART 输出线)
PD2	INT0(外部中断 0 输入)
PD3	INT1(外部中断 1 输入)
PD5	OC1A(定时/计数器 1 的 A 匹配输出引脚)
PD6	$\overline{WR}$(片外 SRAM 的写选通信号输出引脚)
PD7	$\overline{RD}$(片外 SRAM 的读选通信号输出引脚)

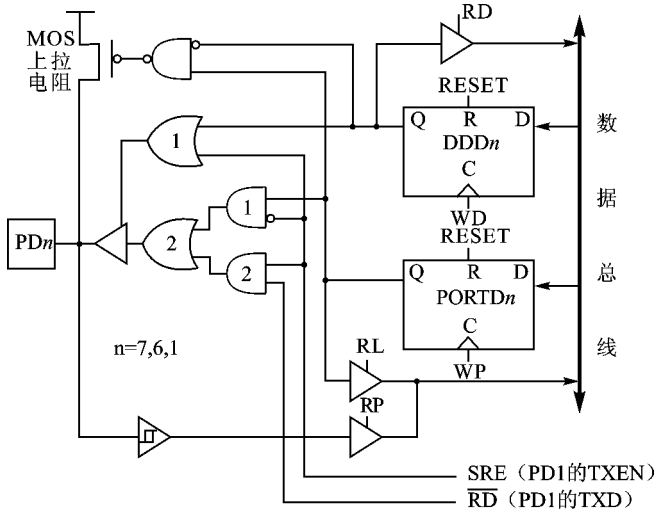


图 1.45　PD7 和 PD1 引脚内部结构

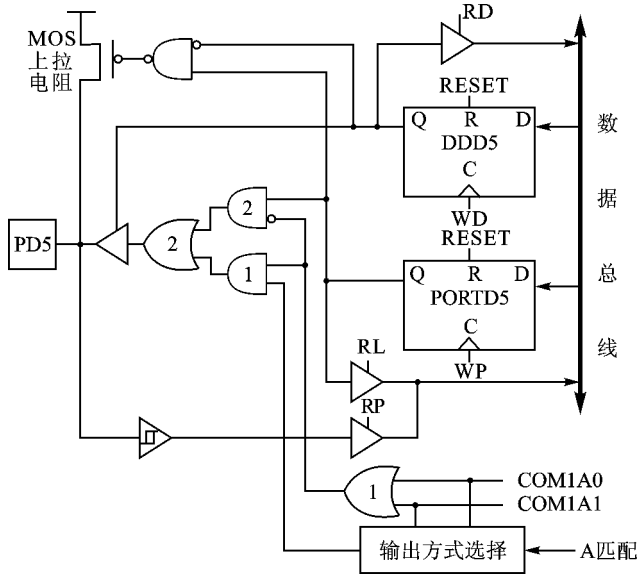


图 1.46　PD5 引脚内部结构

① 定时器控制寄存器 A(TCCR1A)的最高两位 COM1A1 和 COM1A0 中,若有一位为 1,则或门 1 输出高电平,与门 2 输出低电平。但此时与门 1 可根据输出方式选择向或门 2 传递 0、1 或原输出信号的反相电平。这就是表 1.5 中所列的 A 匹配发生时的信号输出。

② COM1A1 和 COM1A0 两位均为 0,则或门 1 和与门 1 均输出低电平。但与门 2 却会把 PORTD5 的值如实地传递给或门 2,并最终经三态门和 PD5 引脚输送出去。这是与 T/C1 输出比较无关的通用输出(表 1.5 第一项)。

如 1.2.7 节之 4 所述,PD5(OC1A)尚可作 PWM 的输出引脚。不过图 1.46 中无此表示。最后应当强调指出,不管作什么输出,前提条件是 DDD5 位置 1。

(4) PD4:此引脚无第二功能,只作通用 I/O,其内部结构如图 1.47 所示。

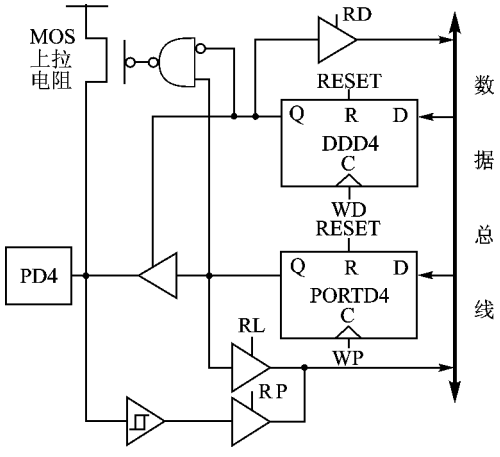


图 1.47 PD4 引脚内部结构

(5) PD3——INT1:外部中断源 1。PD3 引脚的内部结构与 PD4 基本一致,只是在其史密特触发器与引脚输入三态门处又加了一个检测控制电路,如图 1.48 所示。该检测电路由 MCUCR 寄存器的 ISC11 和 ISC10 两位控制(见表 1.19)。

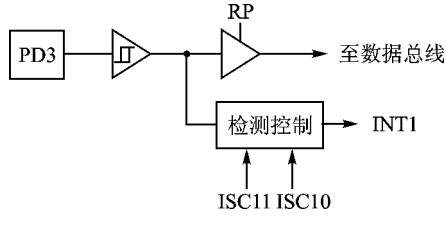


图 1.48 INT1 输入

(6) PD2——INT0:外部中断源 0。PD2 引脚的内部结构同 PD3,只是检测控制位来自 MCUCR 的最低两位,即 ISC01 和 ISC00。

(7) PD1——TXD:UART 的数据输出引脚。其内部逻辑与 PD7(见图 1.45)相同。只是取代 SRE 和$\overline{RD}$信号的分别是 TXEN 和 TXD。如 1.2.11 节之 1 所述,UART 控制寄存器 UCR 中的发送允许位 TXEN 若被软件置 1,则引脚控制逻辑将 UART 发送器与 PD1 引脚接通,迫使 PD1 作输出引脚,向外发送数据 TXD。此时方向控制位 DDD1 失去效力。

上述过程,读者现在完全可以根据图 1.45 作出解释。其实,C 口引脚的主要控制逻辑(图 1.44)与 PD7、PD6 和 PD1 都是相同的。

(8) PD0——RXD:UART 的数据输入引脚。从图 1.49 中可以看到,若把 UCR 寄存器中的接收允许位 RXEN 写成 1,则与门输出低电平,输出三态门呈高阻状态,且 DDD0 无论怎样设定均不起作用。这就是说,UART 开放之后,PD0 自动作输入引脚,用以接收串行数据

RXD。

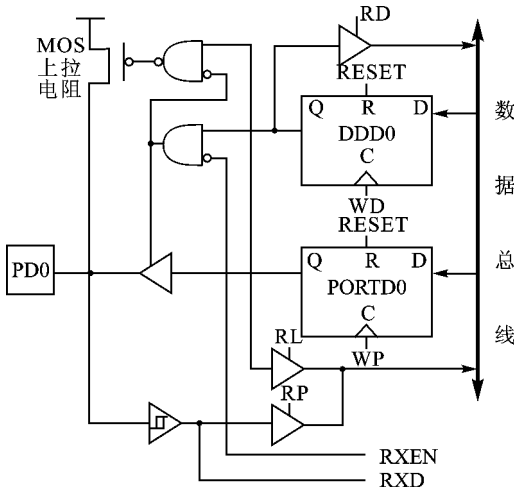


图 1.49　PD0 引脚内部结构

在 PD0 **自动作输入引脚时，其内部上拉电阻的通断和作通用输入引脚时一样，受** PORTD0 **位控制，**1 **则接通，**0 **则关断。**

关于 I/O **端口，最后应当指出，它们的所有引脚都是受系统时钟同步的，不过图中省略了有关的同步锁存部分。**

1.3　封装与引脚功能

就片内资源的多少而言，AVR **单片机从简到繁档次齐全，相应有如下几种封装形式：**

- PDIP——**双列直插式塑料封装；**
- SOIC——**鸥翼式引脚小型塑料封装；**
- SSOP——**收缩式小型塑料封装；**
- PLCC——J **脚芯片载体式塑料封装；**
- TQFP——**超薄方形扁平塑料封装；**
- MLF——**超小型表贴封装。**

1.3.1　引脚功能

AT90S4414/8515 **有** 40 **脚** PDIP、44 **脚** PLCC **和** 44 **脚** TQFP **三种封装形式，如图** 1.50 **所示。图中** NC **为空脚，故三种封装形式均有实用引脚** 40 **个。**

V_{CC}：**供电电源，**2.7～6.0 V(AT90S4414－4 **和** AT90S8515－4)，4.0～6.0V(AT90S4414－8 **和** AT90S8515－8)。

GND：**地。**

端口 A(PA7～PA0)：**这是一个** 8 **位双向** I/O **口，其每条引脚内部均有可通可断的上拉电阻。**A **口各输出缓冲器均可吸收** 20 mA **灌电流，故能直接驱动** LED **显示器。当** PA0～PA7 **用作输入且被从外部拉成低电平时，在接通内部上拉电阻的情况下，这些引脚将向外提供电流。在复位条件有效时，**A **口各引脚呈高阻状态，纵然无时钟信号亦如此。**

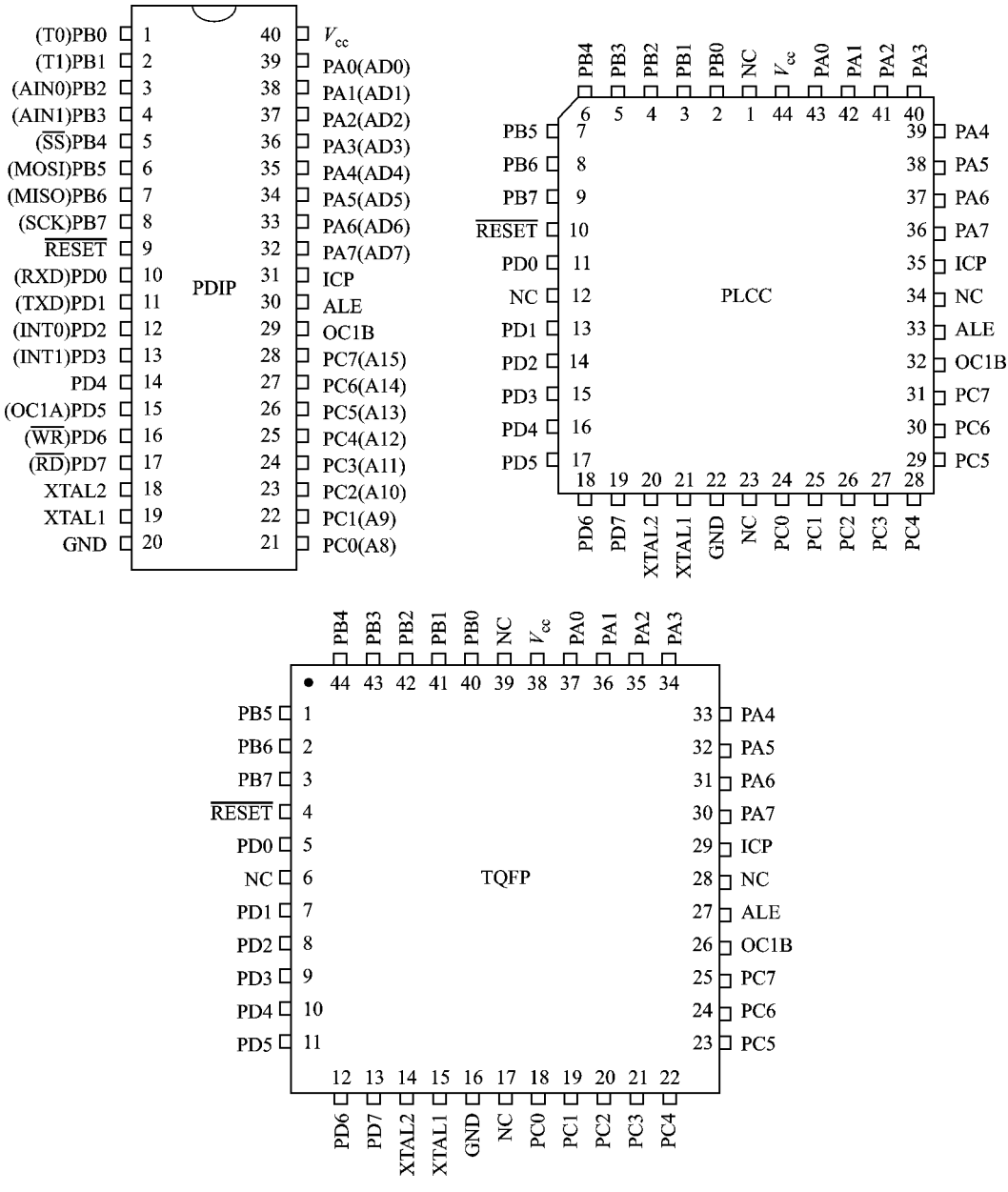


图 1.50　AT90S4414/8515 外观图

在使用片外 SRAM 时，A 口将作为多路切换的地址/数据总线，先输出地址码低 8 位，然后接收或发送数据。

端口 B(PB7～PB0)：8 位双向 I/O 口，其每条 I/O 线的驱动能力、上拉电阻的通断以及复位状态等均无异于 PA7～PA0。

B 口各引脚的第二功能如表 1.23 所列。

端口 C(PC7～PC0)：8 位双向 I/O 口，其各 I/O 线的驱动能力、内部上拉电阻情况以及复位状态等均同上述两口。

当使用片外 SRAM 时，C 口作地址总线，输出地址码高 8 位。

端口 D(PD7～PD0)：8 位双向 I/O 口，主要功能同上述三口。第二功能如表 1.24 所列。

$\overline{\text{RESET}}$：复位信号输入引脚。此引脚上长度超过 50 ns 的低电平输入信号即可引起 MCU 复位，甚至时钟不运行时亦如此。但持续时间不足 50 ns 者不一定能使复位成功。

XTAL1：内部反相振荡放大器的输入，即内部时钟电路的输入。

XTAL2：内部反相振荡放大器的输出。

ICP：定时/计数器 1 输入捕捉功能的输入引脚。

OC1B：定时/计数器 1 的 B 匹配输出引脚。

ALE：地址锁存允许信号输出引脚，在扩接片外数据存储器时使用。在访问片外 SRAM 的第一个时钟周期，ALE 信号的下降沿用来把 AD0～AD7 引脚所传送出的地址码低 8 位锁入片外地址锁存器中。此后在第二个时钟周期，AD0～AD7 引脚用于传送数据。

1.3.2　CPU 时钟

如前所述，XTAL1 和 XTAL2 两引脚内部连接着一个可作 CPU 时钟源的高增益反相放大器。它适于构成反馈振荡电路，为此必须从外部向它提供参考频率。通常的做法是，如图 1.51 所示，在 XTAL1 和 XTAL2 两端跨接一个石英晶体振荡器或陶瓷谐振器以及一端接地的两只电容。这里的石英晶体振荡器为一电感性元件，与外接其上的电容构成并联谐振回路，为片内振荡器提供正反馈和振荡所必需的相移条件，从而构成一个自激振荡器。

也可以用外部时钟源来驱动本微控制器。这时如图 1.52 所示，XTAL2 应作空脚，不外接任何东西，而外来时钟信号由 XTAL1 引脚输入进来。

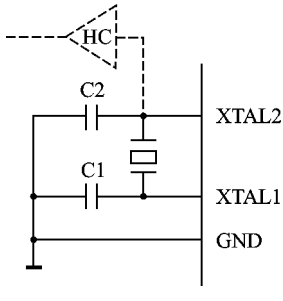


图 1.51　晶体振荡器的连接

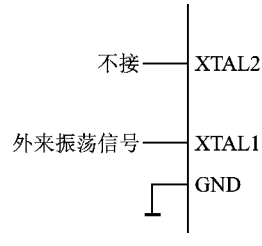


图 1.52　外部时钟驱动

应当指出，若以 MCU 片内振荡器作为某外部器件的时钟，则如图 1.51 虚线所示，应加一个 HC 缓冲器，以增加时钟信号的驱动能力。

1.3.3　指令执行的时间概念

如上节所述，AVR CPU 以系统时钟来驱动，该时钟信号频率直接由片外晶体振荡器产生，片内未加分频环节。本节主要讨论指令执行及片内存储器访问的时间概念。

图 1.53 所示为 AVR CPU 并行读取和执行指令的情况。从这里可以看到，CPU 执行某条指令时，同时读取下一条指令。这样看起来似乎只有执行指令时间，而没有取指令时间。这种流水线结构辅以高速存取的寄存器组合，使 AVR 单片机的速度无与伦比。时钟频率每增加 1 MHz，指令的执行速度便可提高 1 MIPS，即每秒钟可多处理一百万条指令。

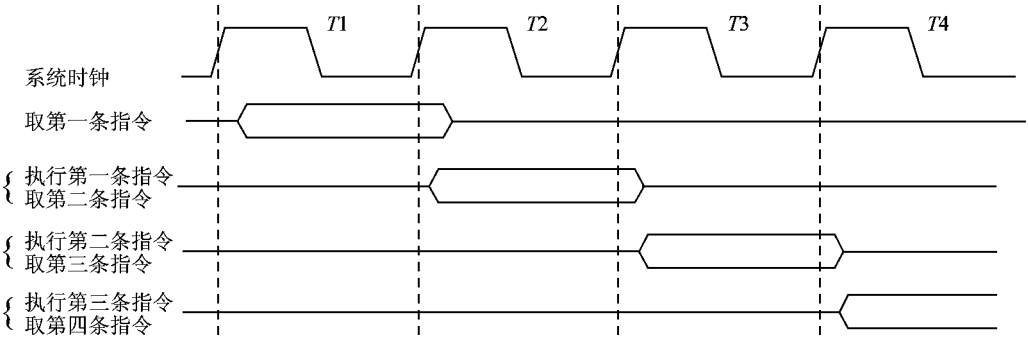


图 1.53　指令的并行读取和执行

图 1.54 所示为访问寄存器组合的时序。从这里可以看到，在一个时钟周期内，可执行完两个寄存器操作数在 ALU 中的运算以及把结果存回到目的寄存器去的操作。正因如此，所以才称其为高速存取的寄存器组合。

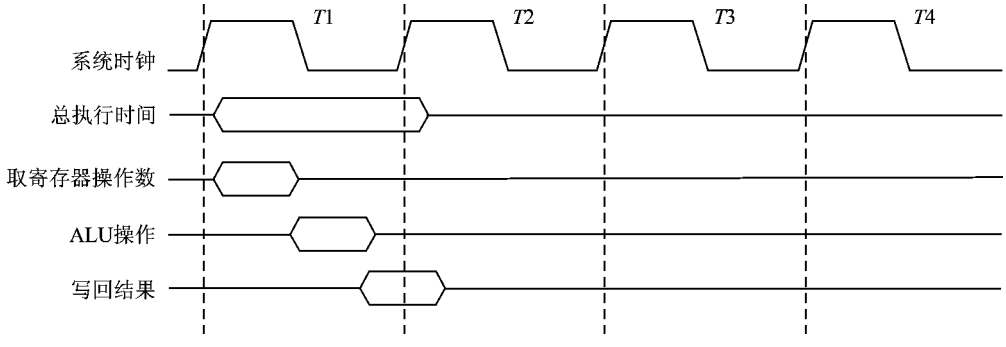


图 1.54　单周期 ALU 操作

对片内 SRAM 的访问则需要两个系统时钟周期，如图 1.55 所示。在第一个时钟周期，内部地址总线上给出源地址，而在第二个时钟周期，它给出目的地址。此时在内部读或写信号控制下，内部数据线上的数据便被传送到目的地址去。

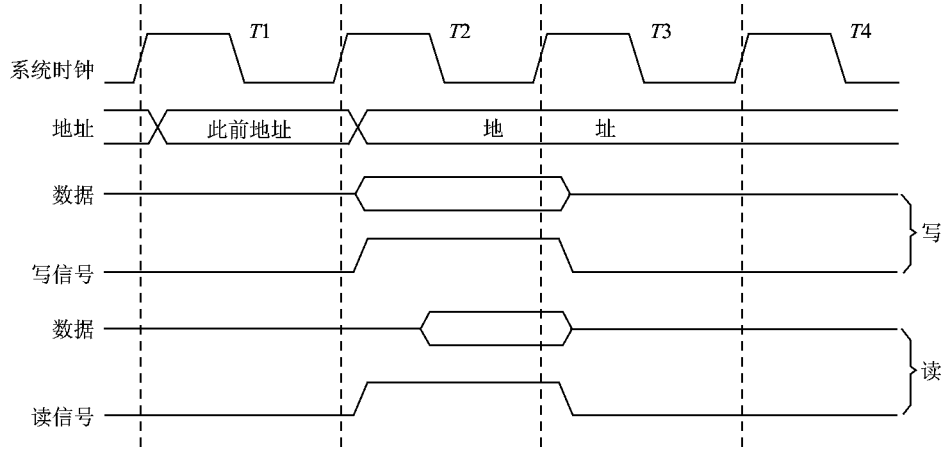


图 1.55　片内 SRAM 访问周期

1.3.4　与片外数据存储器接口

AT90S4414/8515 与片外 SRAM 的接口部件包括：

- 端口 A——多路切换的低位地址总线和数据总线；
- 端口 C——高位地址总线；
- ALE 引脚——发出地址锁存允许信号；
- $\overline{RD}$和$\overline{WR}$——分别作为读和写选通信号的输出引脚。

在存储器组织、中断系统以及 I/O 端口等节中，我们多次谈到，片外数据存储器 SRAM 的开放与禁止是由 MCU 控制寄存器 MCUCR 中的 SRE 位控制的。若 SRE 位置 1，即开放了片外 SRAM 的运作，此时上述接口引脚的数据方向寄存器之设定值不再起作用。当 SRE 位清 0 时，片外 SRAM 被禁止，上述引脚作通用 I/O，服从数据方向寄存器的设定。

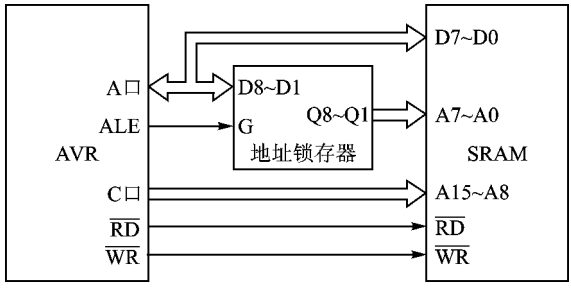


图 1.56　AVR 与片外 SRAM 的连接

在 SRE 位清 0 的情况下，高于片内 SRAM 范围的地址空间并不映射到片内 SRAM 中去，就像不具有片外 SRAM 界面的 AVR 机型那样。

如图 1.56 所示，AVR 单片机在外接 SRAM 时必须外加地址锁存器。当 CPU 执行对 SRAM 的访问指令时，若 SRAM 的地址超出片内数据存储器的范围，则 CPU 自动去访问片外 SRAM。对片外 SRAM 的访问通常要用三个时钟周期，如图 1.57 所示。

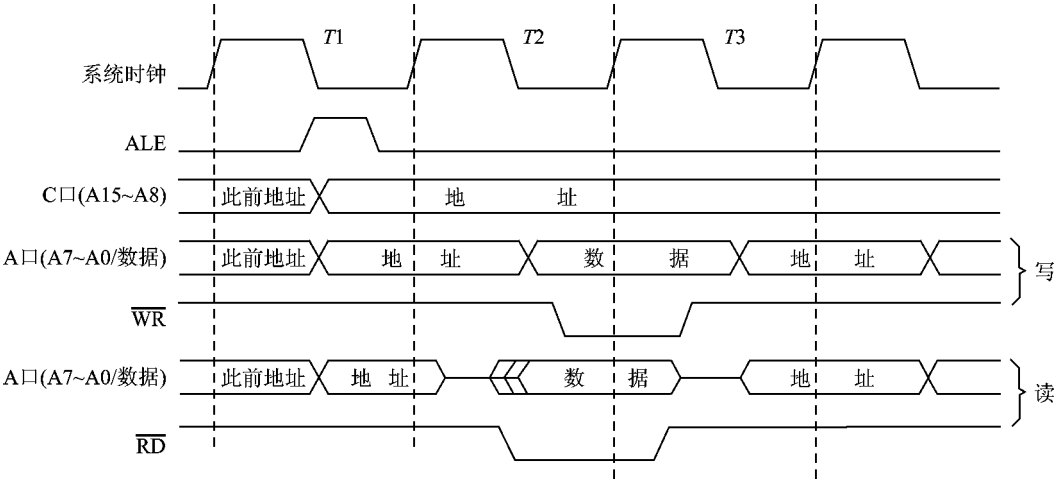


图 1.57　片外 SRAM 存储器周期(无等待状态)

在第一个时钟周期 $T1$ 的负半周期，C 口发出片外 SRAM 地址码的高 8 位 A15～A8，并一直保持着。A 口发出上述地址码的低 8 位 A7～A0。ALE 引脚上产生一个地址锁存允许脉冲，其下降沿把 A 口上的地址码锁入片外地址锁存器中。如果执行的是对片外 SRAM 的写指令，那么 A 口将在 $T2$ 的负半周期至 $T3$ 的正半周期期间送出待写数据，该数据在$\overline{WR}$信

号的作用下被写入片外 SRAM 某指定的地址单元中去。

若系片外 SRAM 的读指令，则 A 口在发出地址码低字节后进入浮动状态一段时间，此后便接收$\overline{RD}$信号从片外 SRAM 某地址单元读出的数据，此数据在 $T3$ 时被传送到寄存器组合内某目的寄存器 Rd 中去。

如果在访问片外 SRAM 的周期中需要增加一个等待状态，即总共用 4 个时钟周期，那么应把 MCUCR 寄存器中的 SRW 位置成 1。这种情况下的时序如图 1.58 所示。

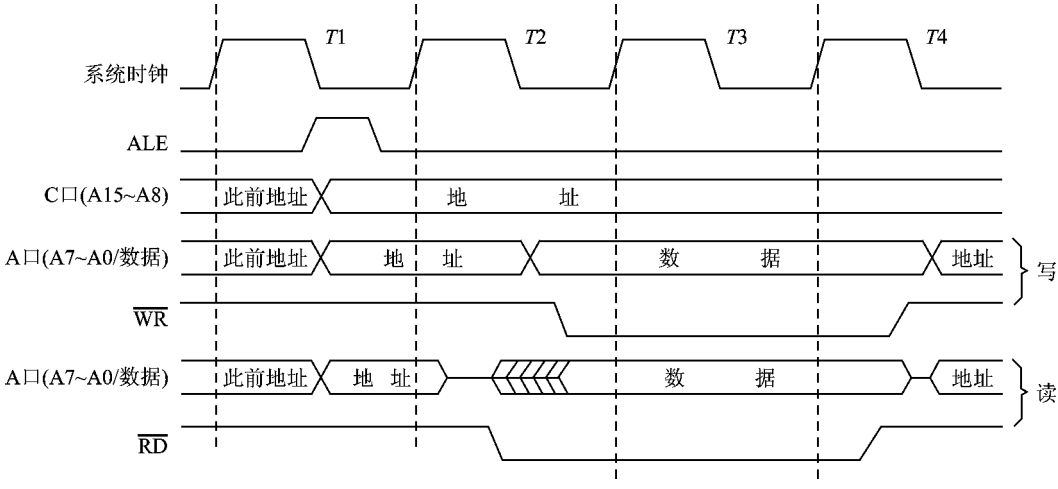


图 1.58 片外 SRAM 存储器周期(加一等待状态)

应当指出，A 口、C 口、ALE、$\overline{RD}$和$\overline{WR}$等接口信号只在访问片外 SRAM 时有效，而在访问片内 SRAM 时不起作用。

1.4 MCU 的休眠方式

现在的 CMOS CPU 或 MCU 都有功耗很低的休眠工作方式。对于 AVR 单片机来说，为使 MCU 进入休眠方式，第一，必须将 MCUCR 寄存器中的休眠方式开放位 SE 置 1；第二，必须执行一条 SLEEP 指令。在 MCU 休眠期间，若有中断发生，则 MCU 将被唤醒，并去执行中断服务程序。中断返回时从 SLEEP 后面一条指令恢复执行程序，寄存器组合、SRAM 以及 I/O存储器的内容均不发生变化。若在休眠期间遇到复位发生，则 MCU 醒来，并从复位向量地址处继续执行程序。

AVR 有两种休眠方式，由 MCUCR 寄存器的休眠方式选择位 SM 来设定。

1.4.1 闲置方式

在 SM 位清 0 的情况下，SLEEP 指令将迫使 MCU 进入闲置方式，CPU 暂停活动，但是允许两定时/计数器、看门狗以及中断系统等部件继续运作。这是因为，在闲置方式下，时钟信号去 CPU 的通路被切断，但至定时器和中断系统等功能部件的途径却畅通无阻，而看门狗定时器则有本身独立的时钟。

中断和复位可用来终止闲置方式。外部触发中断以及内部中断，例如定时器溢出中断等，

均可用来唤醒 MCU，使之继续运作。但是，最好不用模拟比较器中断来做这件事，因为它功耗甚大。故建议在 MCU 进入闲置方式前切断模拟比较器的电源，即把模拟比较器控制状态寄存器 ACSR 中的 ACD 位置 1(见 1.2.12 一节)。这将降低闲置方式下的功耗。闲置方式下通常工作电流为 1 mA。

系统复位和看门狗定时器所引起的复位，也可使 MCU 从闲置方式下醒来，转入正常运作方式。从闲置方式下醒来之后，MCU 将立即恢复程序的执行。不过，若系复位唤醒，则 CPU 将从 $000 地址处继续执行程序，且各 I/O 寄存器的内容均取复位值。

如果是中断使 MCU 脱离了闲置方式，那么 MCU 将去执行中断服务程序。中断返回时从 SLEEP 指令后继续执行。

1.4.2　掉电方式

在 SE 和 SM 位均置 1 的情况下，SLEEP 指令将迫使 MCU 进入掉电方式。在这种方式下，片内振荡器停振，时钟冻结，而外部中断和看门狗却继续运作。掉电方式下耗电约 1 μA。

外部复位、看门狗复位以及 INT0 和 INT1 引脚上的电平触发中断可使 MCU 从掉电方式下苏醒过来。不过应当注意，当用电平触发外部中断来从掉电方式下唤醒 MCU 时，INT0 或 INT1 引脚上的低电平保持时间必须大于复位延迟时限 t_{TOUT}，否则 MCU 可能唤醒不了。

1.5　片内存储器的烧录

本节主要讲述 AVR 程序存储器 Flash 和数据存储器 EEPROM 的写入，即烧录问题。

如前所述，AT90S4414/8515 片内含有 4 K/8 K 字节系统内可重新烧录的闪速存储器和 256/512 个字节的 EEPROM 数据存储器。

市场上购得的这些芯片，均处于擦除状态。就是说，其片内 Flash 和 EEPROM 各字节内容都是 $FF，随时可以写入新的内容。这些器件支持高压(12 V)并行烧录方式以及低压串行烧录方式。该＋12 V 只用于开放烧录操作，此供电引脚所吸收的电流并无明显增加。串行烧录方式为用户提供了一个非常方便的途径，使之能在用户系统内把程序和数据下载到 AT90S4414/8515 中去。

在上述两种烧录方式下，AT90S4414/8515 片内程序和数据存储器阵列均按字节逐一写入。对于 EEPROM 来说，在串行烧录方式下，自定时的写操作期间内含有自擦除周期。

在烧录期间，供电电压必须与表 1.25 之给定值一致。

表 1.25　烧录期间的供电电压值

芯　片	串行烧录	并行烧录
AT90S4414	2.7～6.0 V	4.5～5.5 V
AT90S8515	2.7～6.0 V	4.5～5.5V

1.5.1　并行烧录

下面详细讨论如何对 Flash 程序存储器、EEPROM 数据存储器、封锁位以及熔丝位进行

并行烧录的问题。

1. 信号名称

图 1.59 所示为 AT90S4414/8515 芯片并行烧录时的信号接口情况。这里，某些引脚以其在并行烧录中所传送的信号名称来代替。表 1.26 中列出了这些引脚及相应信号。

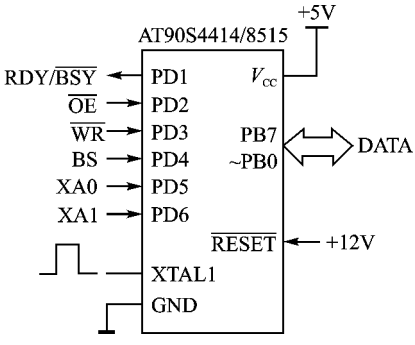


图 1.59 并行烧录信号接口

表 1.26 引脚的烧录功能

信号名称	引脚名称	I/O	功 能
RDY/$\overline{BSY}$	PD1	O	0：器件正在烧录；1：器件准备接受新命令
$\overline{OE}$	PD2	I	输出允许(低电平有效)
$\overline{WR}$	PD3	I	写控制脉冲(低电平有效)
BS	PD4	I	字节选择(0 选择低字节；1 选择高字节)
XA0	PD5	I	XTAL 动作位 0
XA1	PD6	I	XTAL 动作位 1
DATA	PB7～PB0	I/O	双向数据总线($\overline{OE}$为低电平时作输出)

在 XA1 和 XA0 引脚上施加不同的逻辑电平组合，就可决定在 XTAL1 引脚上给一正脉冲时应执行的各种动作，如表 1.27 所列。

表 1.27 XA1 和 XA0 的控制作用

XA1	XA0	XTAL1 引脚被脉冲驱动时的动作
0	0	装入 Flash 或 EEPROM 地址(高或低地址字节由 BS 确定)
0	1	装入数据(Flash 的高或低数据字节由 BS 确定)
1	0	装入命令
1	1	无动作，空闲

表 1.28 汇总了并行烧录方式下的九条命令。这些命令都是单字节的，这里不同位置的“1”代表着不同的功能。在$\overline{WR}$有效脉冲的作用下，命令字节经 DTAT 引脚 PB7～PB0 传入 MCU 中去。

2. 进入烧录方式

为使 MCU 进入并行烧录方式，应当依次实施下列诸点：

(1) 根据表 1.25 所列，在 V_{cc} 和 GND 两引脚间施加供电电压；

(2) 将$\overline{RESET}$和 BS 引脚置成 0 电平，并至少等待 100 ns；

(3) 向$\overline{RESET}$引脚施加 11.5～12.5 V 电

表 1.28 并行烧录方式下的各种命令

命令字节	所执行的操作
1000 0000	芯片擦除
0100 0000	写熔丝位
0010 0000	写封锁位
0001 0000	写 Flash
0001 0001	写 EEPROM
0000 1000	读电子标签各字节
0000 0100	读封锁位和熔丝位
0000 0010	读 Flash
0000 0011	读 EEPROM

压。请注意，在+12 V 电压加至$\overline{\text{RESET}}$引脚后的 100 ns 之内，若 BS 引脚上有任何变故，都将使器件无法进入并行烧录方式。

3. 芯片擦除

芯片擦除命令将把 Flash、EEPROM 以及封锁位全部擦除。在 Flash 和 EEPROM 完全擦除之前，封锁位是不会复原的，但熔丝位不受擦除操作的影响。Flash 和 EEPROM 在重新烧录新内容前，必先擦除其原有内容。装入芯片擦除命令步骤如下：

(1) 如表 1.27 所列，将 XA1 和 XA0 置成 10；

(2) 将 BS 引脚置 0；

(3) 将 DATA 口置成 1000 0000，这是芯片擦除命令(见表 1.28)；

(4) 向 XTAL1 引脚施加一正脉冲，这将把命令装入内部去；

(5) 给$\overline{\text{WR}}$引脚一宽度为 $t_{\text{WLWH\_CE}}$(见表 1.31)的负脉冲，以进行芯片擦除。芯片擦除操作在 RDY/$\overline{\text{BSY}}$引脚上不产生任何信息。

4. Flash 的烧录

(1) 装入写 Flash 命令：分下列四步实施。

① 将 XA1、XA0 置成 10，如表 1.27 所列，这将开放命令装入操作；

② 将 BS 置成 0；

③ 把 DATA 口置成 0001 0000，这是写 Flash 命令(见表 1.28)。

④ 向 XTAL1 引脚施加一正脉冲，这将把 DATA 口上的命令装入内部去。

(2) 装入地址高字节：包括下列四步。

① 将 XA1、XA0 置成 00，开放地址装入操作(见表 1.27)；

② 将 BS 置 1，选择地址高字节；

③ 使 DATA＝地址高字节($00～$07/$0F)；

④ 给 XTAL1 引脚一正脉冲，使地址高字节进入芯片内部。

(3) 装入地址低字节：与地址高字节的装入操作基本相同，只是 BS 不同。

① 将 XA1、XA0 置成 00；

② BS 置 0，选择地址低字节；

③ 使 DATA＝地址低字节($00～$FF)；

④ 向 XTAL1 施加一正脉冲，把地址低字节装入芯片内部。

(4) 装入数据低字节：也分四步进行。

① XA1 和 XA0 置成 01，开放数据装入操作；

② BS 置 0，选中低字节；

③ 使 DATA＝数据低字节($00～$FF)；

④ 向 XTAL1 施加一正脉冲，把数据低字节装入芯片内部。

(5) 写数据低字节：在装入写 Flash 命令和地址的高、低字节以及数据低字节后，就可以进行字节数据的烧录了。

① BS 置 0，选择低字节数据；

② 给$\overline{\text{WR}}$引脚一负脉冲，启动数据字节的烧录，RDY/$\overline{\text{BSY}}$变成低电平，表示正在 BUSY。

③ 待 RDY/$\overline{\text{BSY}}$变成高电平时(READY)，再烧下一个字节。

(6) 装入数据高字节：和数据低字节的装入操作基本相同，只是 BS 应当置 1。

① XA1 和 XA0 置成 01，开放数据装入操作；

② BS 置 1，选数据高字节；

③ 令 DATA＝数据高字节（$00～$FF）；

④ 向 XTAL1 施加一正脉冲，将数据高字节装入内部。

（7）写数据高字节：与数据低字节的烧录基本一致，只是 BS 应为 1。

① BS 置 1，选高字节数据；

② 给$\overline{\text{WR}}$引脚一负脉冲，启动数据字节的烧录，RDY/$\overline{\text{BSY}}$变成低电平；

③ 待 RDY/$\overline{\text{BSY}}$变高后再烧录下一个字节。

图 1.60 和图 1.61 中给出了 Flash 烧录过程中的各信号波形。

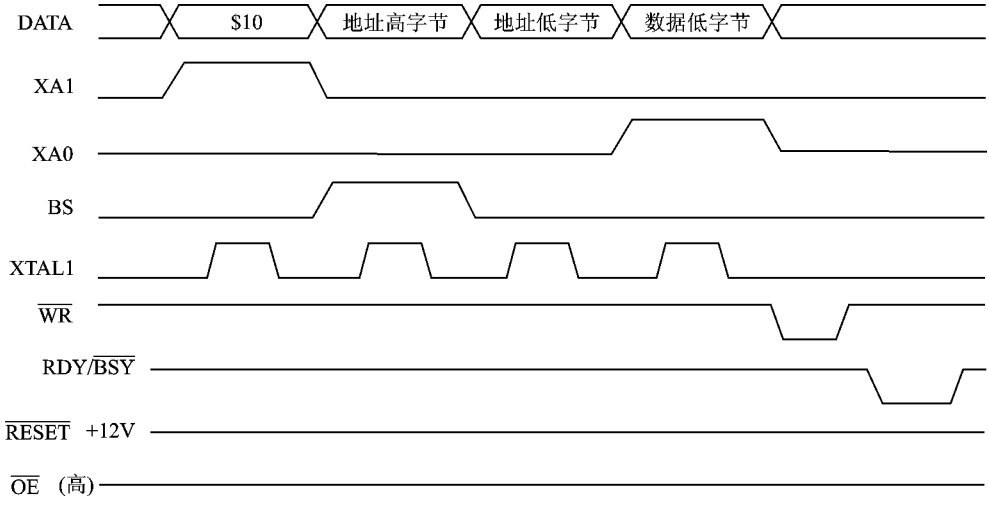


图 1.60 Flash 烧录信号波形

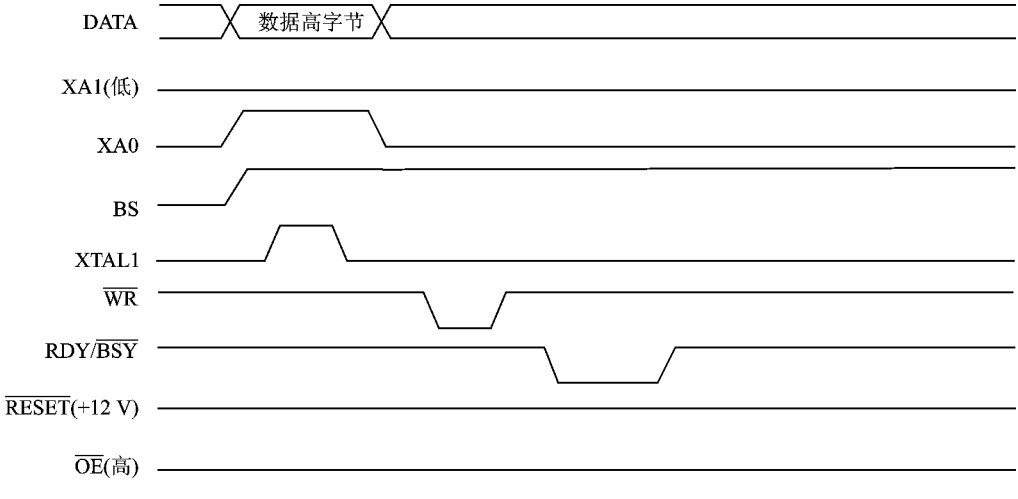


图 1.61 Flash 烧录信号波形（数据高字节）

烧录前所装入的命令和地址在烧录期间将保持不变。因此注意下列几点会使烧录效率更高：

● 在写或读多个存储单元时，命令只须装入一次。

● 地址高字节只须在 Flash 每个新页面(256 字)烧录前装入。

● 数据字节 $FF 可以免写，因为擦除后 Flash 和 EEPROM 各单元均为此值。

上述几点也适应于 EEPROM 烧录以及 Flash、EEPROM 和电子标签的读出。

5. Flash 的读出

读片内闪速存储器的方法如下，其中前三项的细节与 Flash 的烧录相同，故每项下的子条目不再列出。

(1) 装入读 Flash 命令——0000 0010；

(2) 装入地址高字节($00～$07/$0F)；

(3) 装入地址低字节($00～$FF)；

(4) $\overline{OE}$置 0，BS 亦置 0，此时在 DATA 端口处即可读得程序字的低字节；

(5) BS 置 1，在 DATA 端口处可读得程序字的高字节；

(6) $\overline{OE}$恢复成 1。

6. EEPROM 的烧录

AVR EEPROM 数据存储器的烧录步骤如下，其中命令、地址和数据装入的细节与 Flash 的烧录相同，故子条目不再赘述。

(1) 装入写 EEPROM 命令——0001 0001；

(2) 装入地址高字节($00 或 $01，但其 EEPROM 容量不超过 256 字节的 MCU 无须此步操作)；

(3) 装入地址低字节($00～$FF)；

(4) 装入数据低字节($00～$FF)；

(5) 写数据低字节。

7. EEPROM 的读出

AVR EEPROM 存储器的读出步骤如下，其中命令与地址的装入细节因与前同而从略。

(1) 装入读 EEPROM 命令——0000 0011；

(2) 装入地址高字节($00 或 $01，其 EEPROM 容量不超过 256 字节之 MCU 无此步操作)；

(3) 装入地址低字节($00～$FF)；

(4) $\overline{OE}$置 0，BS 置 0，此时在 DATA 口处可读得 EEPROM 数据字节；

(5) $\overline{OE}$恢复成 1。

8. 熔丝位的烧录

AT90S4414/8515 有两个熔丝位——SPIEN 和 FSTRT。

SPIEN 熔丝位被烧录成 0，就意味着本芯片可以进行程序和数据的串行下载。否则，这样的操作是禁止的。该位的本能值是已经烧录的 0 值。

FSTRT 熔丝位被烧录成 0，就选定了短启动时间。该位的本能值为 1。用户可订购此位被烧录成 0 的 AVR MCU。

在串行烧录方式下，两熔丝位是不可访问的。它们的状态不受芯片擦除的影响。

熔丝位的烧录步骤如下，其中命令和数据的装入子条目从略。

(1) 装入写熔丝位命令——0100 0000；

(2) 装入数据低字节,烧录位写 0,擦除位写 1;(位 5 系 SPIEN,位 0 为 FSTRT,其余 6 位是保留位,不应烧录,即均保持为 1);

(3) 向$\overline{WR}$引脚施以宽度为 t_{WLWH_PFB}(见表 1.31)的负脉冲,以启动烧录过程。熔丝位的烧录在 RDY/$\overline{BSY}$引脚上不产生任何信息。

9. 封锁位的烧录

AT90S4414/8515 有两个程序封锁位,它们可以保持原值不变(1 态),也可被烧录成 0 态,以获得表 1.29 所列的附加功能。封锁位只能用芯片擦除命令来擦除。

表 1.29 封锁位的保护方式

存储器封锁位			保护类型
方式	LB1	LB2	
1	1	1	未启用存储器封锁功能
2	0	1	禁止 Flash 和 EEPROM 的进一步烧录 *
3	0	0	同方式 2,但校验也被禁止

注 *:在并行方式下,熔丝位的烧录亦被禁止。熔丝位应在封锁位之前进行烧录。

为对封锁位进行烧录,应依次实施下列诸点,其中命令和数据的装入细节同前。

(1) 装入写封锁位命令——0010 0000;

(2) 装入数据低字节;(该字节之位 2 为 LB2,位 1 为 LB1,其余各保留位均不烧录,应保持为 1。)

(3) 写数据低字节。

封锁位只能通过芯片擦除方能复原,其对程序的封锁作用才被消除。

10. 熔丝位和封锁位的读出

(1) 装入读封锁位和熔丝位命令——0000 0100;

(2) $\overline{OE}$置 0,BS 置 1;(此时熔丝位和封锁位的状态即可在 DATA 口处读得。0 表示已经烧录。在此字节中,位 7=LB1,位 6=LB2,位 5=SPIEN,位 0=FSTRT。请注意,BS 需置 1。)

(3) $\overline{OE}$置 1。

11. 电子标签字节的读出

所有 Atmel 微控制器都有三个字节的标志代码,我们不妨称之为芯片的电子标签。用串行方式和并行方式均可读到此代码。这三个字节驻留在一个单独的地址空间中。表 1.30 中给出了 AT90S4414/8515 两芯片的电子标签。

表 1.30 AT90S4414/8515 电子标签

地 址	标签字节	意 义
$000	$1E	Atmel 公司出品
$001	$93	内含 8 KB 闪速存储器
$002	$01	$001 单元内容为 $93,说明这是 AT90S8515 芯片
$000	$1E	Atmel 公司出品
$001	$92	内含 4 KB 闪速存储器
$002	$01	$001 单元内容为 $92 说明这是 AT90S4414 芯片

应当指出，若两封锁位均已烧录成 0（封锁方式 3），则标签三字节的真实值用串行方式无法读到，读之所得将是 $00、$01 和 $02。

AVR 标签字节的并行读法如下：

（1）装入读电子标签命令——0000 1000；

（2）装入地址低字节（$00～$02）（$\overline{OE}$置 0，BS 也置 0，所选标签字节即可在 DATA 口处读得）；

（3）将$\overline{OE}$恢复成 1。

12. 并行烧录特性

图 1.62 所示为 AVR 并行烧录的定时情况，图中各时间参数请参阅表 1.31。

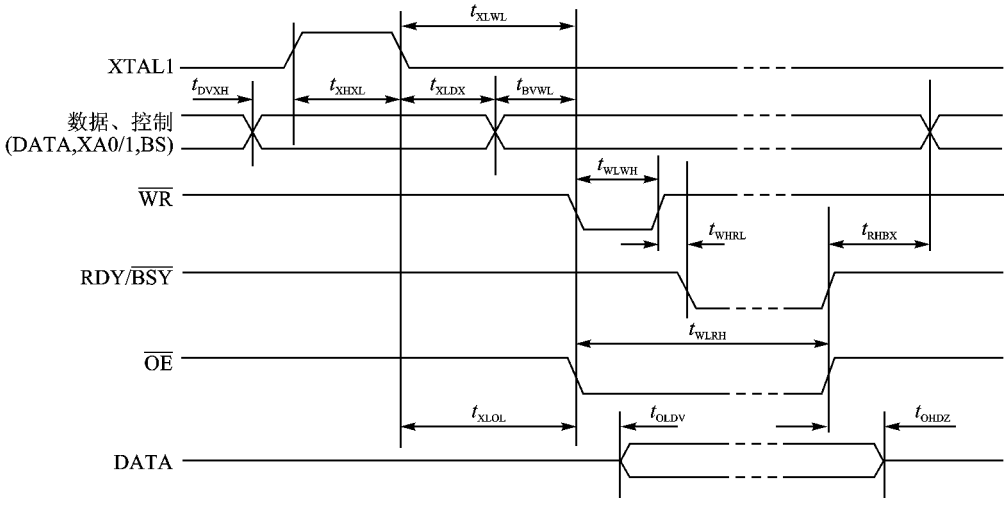


图 1.62　并行烧录时序

表 1.31　并行烧录特性（T_A＝25 ℃±10%，V_{CC}＝5 V±10%）

符　号	参　数	Min	典型值	Max	单　位
V_{PP}	烧录允许电压	11.5		12.5	V
I_{PP}	烧录允许电流			250	μA
t_{DVXH}	XTAL1 高之前数据和控制建立时间	67			ns
t_{XHXL}	XTAL1 正脉冲宽度	67			ns
t_{XLDX}	XTAL1 低后数据和控制保持时间	67			ns
t_{XLWL}	XTAL1 低至$\overline{WR}$低	67			ns
t_{BVWL}	BS 有效至$\overline{WR}$低	67			ns
t_{RHBX}	RDY/$\overline{BSY}$高后 BS 保持时间	67			ns
t_{WLWH}	$\overline{WR}$负脉冲宽度 *	67			ns
t_{WHRL}	$\overline{WR}$高至 RDY/$\overline{BSY}$低 * *		20		ns
t_{WLRH}	$\overline{WR}$低至 RDY/$\overline{BSY}$高 * *	0.5	0.7	0.9	ms
t_{XLOL}	XTAL1 低至$\overline{OE}$低	67			ns

续表 1.31

符　号	参　数	Min	典型值	Max	单　位
t_{OLDV}	$\overline{OE}$低至 DATA 有效		20		ns
t_{OHDZ}	$\overline{OE}$高至数据线三态			20	ns
t_{WLWH_CE}	芯片擦除用$\overline{WR}$负脉冲宽度	5	10	15	ms
t_{WLWH_PFB}	熔丝位烧录用$\overline{WR}$负脉冲宽度	1.0	1.5	1.8	ms

注 * :芯片擦除时用 t_{WLWH_CE};熔丝烧录时用 t_{WLWH_PFB}。

注 * * :若 t_{WLWH}保持时间大于 t_{WLRH},则将无 RDY/$\overline{BSY}$脉冲可见。

1.5.2 串行下载

在$\overline{RESET}$被拉成低电平的情况下,程序存储器 Flash 和数据存储器 EEPROM 均可通过 SPI 总线进行烧录。串行接口信号如图 1.63 所示,包括 SCK、MOSI(输入)和 MISO(输出)。在$\overline{RESET}$引脚被置成低电平后,须先执行烧录允许指令,再执行烧录/擦除指令。

对于 EEPROM 来说,在自定时的写指令执行期间,因有自动擦除周期存在,故无须先执行芯片擦除指令。芯片擦除指令将把程序阵列和 EEPROM 阵列每个存储单元的内容均变成 $FF。

注意,程序存储器和 EEPROM 存储器两阵列具有分立的地址空间:程序存储器空间自 $000 至 $7FF/$FFF(AT90S4414/8515),而 EEPROM 空间则自 $000 至 $0FF/$1FF。

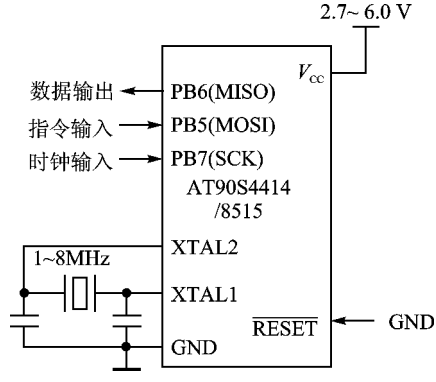


图 1.63 串行烧录和校验

系统时钟信号或由 XTAL1 引脚来的外部时钟源提供,或如图 1.63 所示,由跨接在 XTAL1 和 XTAL2 两引脚间的石英振荡器提供。自 SCK 引脚输入进来的串行时钟信号高低电平持续时间均应大于两个 XTAL1 时钟周期。

1. 串行烧录步骤

在向 AT90S4414/8515 芯片中写入串行数据时,数据位是在 SCK 时钟脉冲上升沿时移入内部的。而由该芯片读取串行数据时,数位是在 SCK 的下降沿时移出的(参见图 1.64 和图 1.65)。

为以串行方式对 AT90S4414/8515 进行烧录和校验,建议遵循下述步骤(参见表 1.32 所列的 4 字节指令格式)。

(1) 上电顺序:在$\overline{RESET}$和 SCK 均被置 0 的情况下,向 V_{CC}和 GND 两引脚间施加供电电压。若在 XTAL1 和 XTAL2 两引脚间未跨接晶振,则应向 XTAL1 引脚加一外来时钟信号。在某些系统中,烧录器不能保证上电期间 SCK 为低电平。在这种情况下,SCK 置 0 后,$\overline{RESET}$引脚上必须给一宽度至少等于两个 XTAL1 周期的正脉冲。

(2) 至少等待 20 ms 后,再向 MOSI(PB5)引脚发送烧录允许串行指令,以开放串行烧录操作。

(3) 如果通讯失去同步,那么串行烧录指令将无所作为。若同步正常,则在发出烧录允许

指令的第三个字节时，其第二个字节（$53）将被返送回来。无论此返回信息正确与否，该指令的 4 个字节均应全部发送出去。如果 $53 未被返送回来，那么在第三和第四两字节发完后，给 SCK 一个正脉冲，再发送一个新的烧录允许指令。若在 32 次尝试中均不见 $53 返回，则说明无功能器件与烧录器相连接。

（4）如果要执行芯片擦除指令（为擦除 Flash，必须进行这一步），那么在该指令后应等待 t_{WD_ERASE} 这样一段时间，然后给 $\overline{RESET}$ 一正脉冲，并且从第（2）步再重新开始。有关 t_{WD_ERASE} 值，请参阅表 1.34。

（5）Flash 或 EEPROM 阵列每次烧录一个字节，这时要把地址、数据和相应的写指令一起提供给待烧录的 MCU。在新数据写入之前，EEPROM 存储单元先自动擦除其原有内容。至于此字节数据何时烧录完毕，即什么时候允许写入下一个字节，用数据查询法可以探知。如果不用查询法，那么可等待 t_{WD_PROG}（参见表 1.35）这样一段时间，然后再传送下一条指令。在已擦除过的器件中，数据文件内的 $FF 字节是无须烧录的。

（6）任何存储单元的内容均可用读指令进行校验，此读指令把所选地址的内容返送至串行输出引脚 MISO（PB6）。

（7）在烧录过程结束时，可把 $\overline{RESET}$ 置成高电平，以便开始正常运作。

（8）撤电顺序：若未用片外晶振，则应将 XTAL1 置成 0；$\overline{RESET}$ 置成 1；关闭 V_{CC}。

2. EEPROM 数据查询

正在向 EEPROM 中烧录某字节数据时，若对此地址单元进行读操作，则在自动擦除操作结束前，将读得数值 $80，而后读值等于 $7F。

当该器件准备接受一 EEPROM 新字节时，刚刚被烧录的数值就可以如实地被读出来了。这一特点正好用来决定何时能写入下一个字节。但所写数据恰为 $80 或 $7F 时，就不好用这种办法了。所以烧录这两个数值时，用户不得不等待所规定的一段时间（t_{WD_PROG}），然后再烧录下一个字节。因为整体被擦除过的器件每个存储单元的内容均为 $FF，所以其内容本应指定为 $FF 的那些存储单元不必再烧录，而应将其越过，继续烧录下一个非 $FF 的单元就是。当然，如果 EEPROM 重新烧录前未经芯片整体擦除，那么这一条是不适用的。

3. Flash 数据查询

欲对 AVR 微控制器的 Flash 存储器进行串行烧录，必须先执行烧录允许指令和芯片整体擦除指令，此后便是逐字节烧录操作了。正在向 Flash 中烧录某字节时，若对此地址单元进行读操作，则读之所得将等于数值 $7F。而当该器件准备接受新字节时，已烧录值即可如实被读得。我们正是用这一特点来探知何时才能写入下一字节的。但当所写字节恰为 $7F 时，不难理解，此法不便使用。这时用户至少应等待 t_{WD_PROG} 这样一段时间，然后才烧录下一个字节。烧录时，其内容本应指定为 $FF 的地址单元可越过，直接烧录其后面内容非 $FF 的地址单元即可。

图 1.64 所示为串行烧录时的波形。表 1.32 则给出了所有串行烧录指令。应当指出，尽管这些指令中有很多无关位，但每条串行烧录指令长度均为 4 个字节。

再次提醒读者，在封锁方式 3 下，即两封锁位均被烧录的情况下，标签三字节不可读。

4. 串行烧录特性

图 1.65 为串行烧录时序图，其中各时间参数值如表 1.33 所列。表 1.34 和表 1.35 中分别给出了芯片擦除指令送出之后的最低等待时间和 Flash 或 EEPROM 每地址单元写操作的

最低等待时间值。

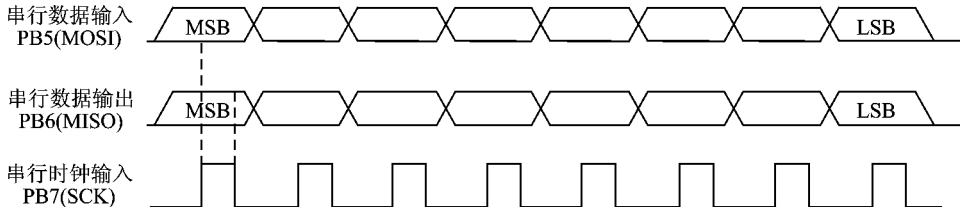


图 1.64 串行烧录波形

表 1.32 串行烧录指令集

指 令	指令格式				操 作
	字节 1	字节 2	字节 3	字节 4	
烧录允许	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	在$\overline{\text{RESET}}$低时允许串行烧录
芯片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Flash 和 EEPROM 存储阵列整体擦除
读程序存储器	0010 H000	xxxx aaaa	bbbb bbbb	oooo oooo	自程序存储器字地址单元 a:b 读 H(高或低)数据 o
写程序存储器	0100 H000	xxxx aaaa	bbbb bbbb	iiii iiii	向程序存储器字地址 a:b 处写入 H(高或低)数据 i
读 EEPROM 存储器	1010 0000	xxxx xxxa	bbbb bbbb	oooo oooo	自 EEPROM 存储地址单元 a:b 读数据 o
写 EEPROM 存储器	1100 0000	xxxx xxxa	bbbb bbbb	iiii iiii	向 EEPROM 存储地址单元 a:b 写数据 i
写封锁位	1010 1100	111x x21x	xxxx xxxx	xxxx xxxx	将 1、2 置 0 烧录封锁位
读标签字节	0011 0000	xxxx xxxx	xxxx xxbb	oooo oooo	在地址 b 处读标签字节 o

注：a=地址高位；b=地址低位；H=0——低字节；1——高字节；o=数据输出；i=数据输入；1=封锁位 1；2=封锁位 2；x=无关位

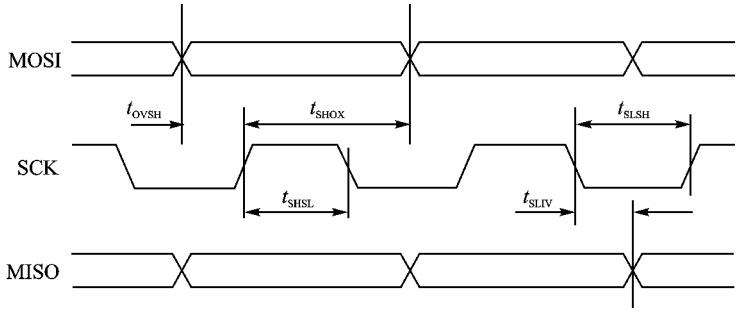


图 1.65 串行烧录时序

表 1.33　串行烧录特性(T_A＝－40～85 ℃ ,V_{CC}＝2.7～6.0V)

符　号	参　数	Min	典型值	Max	单位
$1/t_{CLCL}$	晶振频率(V_{CC}＝2.7～4.0 V)	0		4	MHz
t_{CLCL}	晶振周期(V_{CC}＝2.7～4.0 V)	250			ns
$1/t_{CLCL}$	晶振频率(V_{CC}＝4.0～6.0 V)	0		8	MHz
t_{CLCL}	晶振周期(V_{CC}＝4.0～6.0 V)	125			ns
t_{SHSL}	SCK 正脉冲宽度	$2t_{CLCL}$			ns
t_{SLSH}	SCK 负脉冲宽度	$2t_{CLCL}$			ns
t_{OVSH}	MOSI 建立至 SCK 高	t_{CLCL}			ns
t_{SHOX}	SCK 高后 MOSI 保持时间	$2t_{CLCL}$			ns
t_{SLIV}	SCK 低至 MISO 有效	10	16	32	ns

表 1.34　芯片擦除指令后的最低等待时间值

符　号	3.2 V	3.6 V	4.0 V	5.0 V
t_{WD_ERASE}	18 ms	7 ms	6 ms	4 ms

表 1.35　Flash 或 EEPROM 每单元写操作最低等待时间值

符　号	3.2 V	3.6 V	4.0 V	5.0 V
t_{WD_PROG}	9 ms	7 ms	6 ms	4 ms

第二章　指令系统

本章研究 AVR 单片机的所有指令，这是指挥单片机工作的基础。

命令 CPU 执行一定操作的一组二进制代码称作指令。某 CPU 的指令系统就是它所能执行的指令的集合。指令是计算机硬件能识别的，故属机器语言。但二进制编码的机器语言难记难认，因此各 CPU 的指令系统均以符号语言，即汇编语言形式给出。

AVR 指令系统包括 130 条指令。最基本的 AVR 指令子集是 AT90S1200 的 89 条指令。AT90S4414/8515 指令子集除全部包容这 89 条指令外，又新增了 29 条，计 118 条指令。ATmega 161 的 130 条指令是上述 118 条加 12 条新增指令构成的。本章只讲述含 118 条的指令子集。

这 118 条指令按功能可划分成下列几类：

- 数据传送指令；
- 算术和逻辑运算指令；
- 分支指令；
- 位操作及其他指令。

2.1　数据传送指令

数据传送指令是程序设计中使用最频繁的一类指令。AVR 数据传送指令包括工作寄存器间的数据传送 MOV、工作寄存器与 SRAM 间的数据传送 LD 和 ST、工作寄存器与 I/O 寄存器间的数据传送 IN 和 OUT 以及堆栈操作 PUSH 和 POP 指令。

数据传送指令对状态寄存器 SREG 中的任何标志均无影响，这一点下面将不再一一强调。

2.1.1　工作寄存器间的数据传送指令

通用寄存器组合中，任何两工作寄存器间均可进行数据传送。即把源寄存器 Rr 值复制到目的寄存器 Rd 中去，Rr 值保持不变，Rd 值被 Rr 值取代。此指令的助记符为 MOV。

汇编语言格式：MOV Rd，Rr；　　它所进行的操作：Rd←Rr；

源寄存器和目的寄存器编号：0≤r≤31，0≤d≤31；　　程序计数器：PC←PC+1；

16 位指令代码：0010 11rd dddd rrrr。

[例 2.1]　MOV R16，R0 指令所进行的操作是把 R0 的内容复制到 R16 中去。指令代码是 $2D00。指令执行时间为一个时钟周期（见表 2.9）。

2.1.2　工作寄存器的数据装载指令

把片内外 SRAM 中的数据读到工作寄存器中去以及给工作寄存器赋值的操作称作装载，其助记符为 LD。

1. 立即数装载指令 LDI

工作寄存器 R16～R31 中均可直接装入一 8 位常数值，这就是 LDI 指令。

汇编语言格式：LDI Rd，K；　　指令操作：Rd←K；

操作数：16≤d≤31，0≤K≤255；　　程序计数器：PC←PC＋1；

16 位指令代码：1110 KKKK dddd KKKK。

[例 2.2]　LDI R30，$F0 指令所进行的操作是把 Z 寄存器的低字节 R30 之值置成常数 $F0。此指令代码为 $EFE0，执行时间等于一个时钟周期。

2. 寄存器所有位置 1 指令 SER

SER Rd 指令把 R16～R31 中某个目的寄存器 Rd 的 8 位均置成 1。实际上这是 LDI Rd，$FF 指令的另一种写法。其代码是 1110 1111 dddd 1111。

3. 自数据空间的直接装载指令 LDS

这是一条直接由数据空间某地址单元把一字节数据读到任何一工作寄存器中去的指令。对于那些有 SRAM 的机型来说，数据空间由寄存器组合、I/O 存储器以及内外 SRAM 等部分构成。而无 SRAM 机型的数据空间只包括寄存器组合。EEPROM 占有独立的地址空间。

汇编语言格式：LDS Rd，k；　　指令操作：Rd←(k)；

操作数：0≤d≤31，0≤k≤65 535；　　程序计数器：PC←PC＋2；

32 位指令代码：1001　000d　dddd　0000

kkkk　kkkk kkkk　kkkk。

[例 2.3]　LDS R2，$FF00 指令把数据空间 $FF00 地址单元的 8 位数据装载到 R2 中去，$FF00 单元内容不变，R2 原内容被取代。此指令代码是 $9020FF00，执行时间为 2(3)个时钟周期。

4. 以 X 寄存器为地址指针的间接寻址装载指令

这是间接由数据空间把一字节数据装入某工作寄存器去的操作指令。它不像 LDS 指令那样直接给出数据空间的地址，而是以 X 寄存器的内容作为操作数的地址。故这里 X 称作指针寄存器，其内容在操作中可保持不变，亦可后增量(记作 X＋)或先减量(－X)。这些特点尤其适合对数组、表格的访问，用 X 指针寄存器来仿效堆栈指针也是非常方便的。

表 2.1 汇总了以 X 寄存器为地址指针的三条 LD 指令。

表 2.1　以 X 为指针寄存器的 LD 指令

指令格式		LD Rd，X	LD Rd，X＋	LD Rd，－X
指令操作		Rd←(X)	Rd←(X)，X←X＋1	X←X－1，Rd←(X)
操作数		0≤d≤31	0≤d≤31	0≤d≤31
PC 操作		PC←PC＋1	PC←PC＋1	PC←PC＋1
代码	字节 1	1001 000d	1001 000d	1001 000d
	字节 2	dddd 1100	dddd 1101	dddd 1110

[例 2.4]

```
LDI     R27，$00     ；X 高字节为 $00
LDI     R26，$60     ；X 低字节置成 $60
LD      R0，X＋      ；将 SRAM $0060 单元内容装入 R0 中
```

```
LD      R1,X        ;将 SRAM $0061 单元内容装入 R1 中
LDI     R26,$63     ;X 低字节内容为 $63
LD      R2,X        ;将 SRAM $0063 单元内容装入 R2 中
LD      R3,-X       ;将 SRAM $0062 单元内容装入 R3 中
```

[例 2.5] LD R3,-X 的指令代码是 $903E,执行时间为 2 个时钟周期。执行这条指令时,X 内容先减 1,然后以 X 新内容作为数据地址,从此地址处取数送入 R3 中。

[例 2.6] LD R5,X+的指令代码是 $905D,执行时间为 2 个时钟周期。这条指令所规定的数据传送完毕时,X 内容自动增 1。

5. 以 Y 寄存器为地址指针的间接寻址数据装载指令

Y 寄存器也可作地址指针来间接访问数据空间。Y 的内容在操作中可保持不变,亦可后增量(Y+),或先减量(-Y)。此外尚可用 Y 的内容作基准地址,而以某 6 位二进制无符号数 q 为偏移量,两者之和为有效地址。从有效地址处读一字节数据送往工作寄存器 Rd 中。

以 Y 为指针寄存器的四条 LD 指令如表 2.2 所列。

表 2.2 以 Y 为指针寄存器的 LD(LDD)指令

指令格式		LD Rd,Y	LD Rd, Y+	LD Rd, -Y	LDD Rd, Y+q
指令操作		Rd←(Y)	Rd←(Y), Y←Y+1	Y←Y-1, Rd←(Y)	Rd←(Y+q)
操作数		0≤d≤31	0≤d≤31	0≤d≤31	0≤d≤31, 0≤q≤63
PC 操作		PC←PC+1	PC←PC+1	PC←PC+1	PC←PC+1
代码	字节 1	1000 000d	1001 000d	1001 000d	10q0 qq0d
	字节 2	dddd 1000	dddd 1001	dddd 1010	dddd 1qqq

```
[例 2.7] CLR     R29        ;Y 高字节清 0
         LDI     R28,$60    ;Y 低字节赋值 $60
         LD      R0,Y+      ;把 $0060 地址处的数据装入 R0 中
         LD      R1,Y       ;把 $0061 单元之数据装入 R1 中
         LDI     R28,$63    ;Y 低字节赋值 $63
         LD      R2,Y       ;$0063 单元内容送入 R2 中
         LD      R3,-Y      ;$0062 单元内容送入 R3 中
         LDD     R4,Y+2     ;$0064 单元内容送入 R4 中
```

[例 2.8] LDD R10,Y+50 指令代码为 $A8AA,执行时间等于 2 个时钟周期。

6. 以 Z 寄存器作地址指针的间接寻址数据装载指令

以 Z 寄存器作地址指针进行数据装载,和 Y 寄存器作指针一样,也有 4 条指令(见表 2.3)。

表 2.3 以 Z 为指针寄存器的 LD(LDD)指令

指令格式		LD Rd,Z	LD Rd, Z+	LD Rd, -Z	LDD Rd, Z+q
指令操作		Rd←(Z)	Rd←(Z), Z←Z+1	Z←Z-1, Rd←(Z)	Rd←(Z+q)
操作数		0≤d≤31	0≤d≤31	0≤d≤31	0≤d≤31, 0≤q≤63
PC 操作		PC←PC+1	PC←PC+1	PC←PC+1	PC←PC+1
代码	字节 1	1000 000d	1001 000d	1001 000d	10q0 qq0d
	字节 2	dddd 0000	dddd 0001	dddd 0010	dddd 0qqq

2.1.3　工作寄存器的数据存储指令

任一工作寄存器 Rr 的内容均可以直接或间接寻址方式存储(ST)到数据空间去。

1. 数据空间的直接存储指令 STS

这条指令把某工作寄存器 Rr 中的一个字节数据存放到数据空间某地址单元去，此目的地址由该指令的第二个字直接给出。

指令格式：STS k，Rr；　　**指令操作**：(k)←Rr；

操作数：0≤r≤31，0≤k≤65 535；　　**程序计数器**：PC←PC+2；

32 **位指令代码**：1001　001d　dddd　0000

kkkk　kkkk　kkkk　kkkk。

[**例 2.9**]
```
LDS     R2,$FF00      ; 把 SRAM $FF00 单元内容送至 R2
ADD     R2,R1         ; R2←R2+R1
STS     $FF00,R2      ; 把和存回到 $FF00 单元
```

2. 以 X 寄存器为地址指针的间接寻址数据存储指令 ST

这样的指令有三条，即指针内容不变、指针内容后增量和先减量等三种情况(见表 2.4)。

表 2.4　以 X 为指针寄存器的 ST 指令

指令格式		ST X,Rr	ST X+，Rr	ST −X，Rr
指令操作		(X)←Rr	(X)←Rr，X←X+1	X←X−1，(X)←Rr
操作数		0≤r≤31	0≤r≤31	0≤r≤31
PC **操作**		PC←PC+1	PC←PC+1	PC←PC+1
代码	**字节** 1	1001 001r	1001 001r	1001 001r
	字节 2	rrrr 1100	rrrr 1101	rrrr 1110

[**例 2.10**]
```
CLR     R27           ; X 高字节清 0
LDI     R26,$60       ; X 低字节值等于 $60
ST      X+,R0         ; R0 内容存入数据空间 $0060 单元
ST      X,R1          ; R1 内容存入数据空间 $0061 单元
LDI     R26,$63       ; X 低字节置成 $63
ST      X,R2          ; R2 内容存入 $0063 单元
ST      -X,R3         ; R3 内容存入 $0062 单元
```

3. 以 Y 寄存器为地址指针的间接寻址数据存储指针 ST

这类指令有四条，如表 2.5 所列。

表 2.5　以 Y 为指针寄存器的 ST(STD)指令

指令格式		ST Y，Rr	ST Y+，Rr	ST −Y，Rr	STD Y+q，Rr
指令操作		(Y)←Rr	(Y)←Rr，Y←Y+1	Y←Y−1，(Y)←Rr	(Y+q)←Rr
操作数		0≤r≤31	0≤r≤31	0≤r≤31	0≤r≤31，0≤q≤63
PC **操作**		PC←PC+1	PC←PC+1	PC←PC+1	PC←PC+1
代码	**字节** 1	1000 001r	1001 001r	1001 001r	10q0 qq1r
	字节 2	rrrr 1000	rrrr 1001	rrrr 1010	rrrr 1qqq

4. 以 Z 寄存器为地址指针的间接寻址数据存储指令 ST

和 Y 寄存器作地址指针时的情况一样，Z 作指针的 ST(STD)指令也有四条(见表 2.6)。

表 2.6 以 Z 为指针寄存器的 ST(STD)指令

指令格式		ST Z, Rr	ST Z+1, Rr	ST −Z, Rr	STD Z+q, Rr
指令操作		(Z)←Rr	(Z)←Rr, Z←Z+1	Z←Z−1, (Z)←Rr	(Z+q)←Rr
操作数		0≤r≤31	0≤r≤31	0≤r≤31	0≤r≤31, 0≤q≤63
PC 操作		PC←PC+1	PC←PC+1	PC←PC+1	PC←PC+1
代码	字节 1	1000 001r	1001 001r	1001 001r	10q0 qq1r
	字节 2	rrrr 0000	rrrr 0001	rrrr 0010	rrrr 0qqq

```
[例 2.11]  CLR   R31       ; Z 寄存器高字节清 0
           LDI   R30,$60   ; Z 低字节置成 $60
           ST    Z+,R0     ; R0 内容存入 SRAM $0060 单元
           ST    Z,R1      ; R1 内容存入 $0061 单元
           LDI   R30,$63   ; Z 低字节置成 $63
           ST    Z,R2      ; R2 内容存入 $0063 单元
           ST    -Z,R3     ; R3 内容存入 $0062 单元
           STD   Z+2,R4    ; R4 内容存入 $0064 单元
```

2.1.4 程序存储器的读指令 LPM

这是一条以 Z 寄存器为指针自程序存储器中读一字节送入目的寄存器 R0 的指令。大家知道，AVR 程序存储器 Flash 是以 16 位字为一地址单元的。如图 1.13 所示，Z 寄存器的高 15 位选择 Flash 的 32 K 字地址单元，而其最低位负责选定此字的高或低字节，0 选择低字节，1 选择高字节。

指令格式：LPM；**指令操作**：R0←(Z)；PC **操作**：PC←PC+1；16 **位指令代码**：$95C8。

```
[例 2.12]  CLR   R31       ; Z 寄存器高字节清 0
           LDI   R30,$F0   ; Z 低字节置成 $F0
           LPM             ; 将程序存储器 $00F0 单元之低字节读入 R0 中
```

2.1.5 输入/输出指令

从 I/O 空间(端口、定时器及各配置寄存器，见表 1.2)读出数据并传送至工作寄存器中去的操作为输入(IN)；而把工作寄存器的内容写到 I/O 空间去的操作称为输出(OUT)。在 IN 和 OUT 指令中，均直接给出 I/O 寄存器的地址。表 2.7 中列出了 IN 和 OUT 两条指令的汇编语言书写格式、它们所进行的操作以及指令代码构成等情况。

表 2.7 输入指令 IN 和输出指令 OUT

指令格式	IN Rd,A	OUT A, Rr
指令操作	Rd←I/O(A)	I/O(A)←Rr
操作数	0≤d≤31, 0≤A≤63	0≤r≤31, 0≤A≤63
PC 操作	PC←PC+1	PC←PC+1
指令代码	1011 0AAd dddd AAAA	1011 1AAr rrrr AAAA

［例 2.13］ IN R25，$16 指令把端口 B 输入的数据读到工作寄存器 R25 中去，其指令代码为 $B396，执行时间等于一个时钟周期。

［例 2.14］ 下面一段程序将在 B 口的 8 条输出线 PB7～PB0 上同时各自产生一个由 0 到 1 的正跳变。

```
CLR     R16          ; R16 清 0
SER     R17          ; R17 各位置 1
OUT     $18,R16      ; 向端口 B 写 8 个 0
NOP                  ; 延时一个时钟周期
OUT     $18,R17      ; 向 B 口写 8 个 1
```

2.1.6　堆栈操作指令

AVR 堆栈操作只有工作寄存器才能进行。R0～R31 的内容均可压入堆栈保存(PUSH)，亦可由堆栈栈顶把数据弹出(POP)送到这些工作寄存器中去。在执行入栈指令 PUSH 后，堆栈指针 SP 的内容减 1，指向紧邻栈顶的一个低地址单元。在执行出栈指令 POP 时，SP 内容先自动加 1，然后再把数据由栈顶弹出。堆栈中的源和目的地址均由堆栈指针 SP 的内容间接给出。给 PUSH 的目的地址时，SP 内容后减量；给 POP 的源地址时，SP 先增量。

入栈指令 PUSH 和出栈指令 POP 在表 2.8 中有形象的说明。

表 2.8　入栈指令 PUSH 和出栈指令 POP

指令格式	PUSH Rr	POP Rd
指令操作	(SP)←Rr，SP←SP－1	SP←SP＋1，Rd←(SP)
操作数	0≤r≤31	0≤d≤31
PC 操作	PC←PC＋1	PC←PC＋1
指令代码	1001 001d dddd 1111	1001 000d dddd 1111

［例 2.15］

```
        RCALL   SBRT    ; 调用子程序
        ……
SBRT:   PUSH    R14     ; R14 保护入栈
        PUSH    R13     ; R13 保护入栈
        ……
        POP     R13     ; 恢复 R13
        POP     R14     ; 恢复 R14
        RET             ; 子程序返回
```

表 2.9 汇总了所有上述数据传送指令。除另有注明者外，表中的源和目的寄存器编号 r 和 d 之取值范围均为 0～31。

表 2.9　AVR 数据传送指令一览表

助记符	操作数	指 令 操 作	代码字数	影响标志	时钟周期
MOV	Rd，Rr	Rd←Rr	1	无	1

续表 2.9

助记符	操作数	指 令 操 作	代码字数	影响标志	时钟周期
LDI	Rd，K	Rd←K，d=16～31，K=0～255	1	无	1
LD	Rd，X	Rd←(X)	1	无	2
LD	Rd，X+	Rd←(X)，X←X+1	1	无	2
LD	Rd，－X	X←X－1，Rd←(X)	1	无	2
LD	Rd，Y	Rd←(Y)	1	无	2
LD	Rd，Y+	Rd←(Y)，Y←Y+1	1	无	2
LD	Rd，－Y	Y←Y－1，Rd←(Y)	1	无	2
LDD	Rd，Y+q	Rd←(Y+q)，q=0～63	1	无	2
LD	Rd，Z	Rd←(Z)	1	无	2
LD	Rd,Z+	Rd←(Z)，Z←Z+1	1	无	2
LD	Rd，－Z	Z←Z－1，Rd←(Z)	1	无	2
LDD	Rd,Z+q	Rd←(Z+q)，q=0～63	1	无	2
LDS	Rd，k	Rd←(k)，k=0～65 535	2	无	2
ST	X，Rr	(X)←Rr	1	无	2
ST	X+,Rr	(X)←Rr，X←X+1	1	无	2
ST	－X，Rr	X←X－1，(X)←Rr	1	无	2
ST	Y，Rr	(Y)←Rr	1	无	2
ST	Y+，Rr	(Y)←Rr，Y←Y+1	1	无	2
ST	－Y，Rr	Y←Y－1，(Y)←Rr	1	无	2
STD	Y+q，Rr	(Y+q)←Rr，q=0～63	1	无	2
ST	Z，Rr	(Z)←Rr	1	无	2
ST	Z+，Rr	(Z)←Rr，Z←Z+1	1	无	2
ST	－Z，Rr	Z←Z－1，(Z)←Rr	1	无	2
STD	Z+q，Rr	(Z+q)←Rr，q=0～63	1	无	2
STS	k，Rr	(k)←Rr，k=0～65 535	2	无	2
LPM		R0←(Z)	1	无	2
IN	Rd，A	Rd←I/O(A)，A=0～63	1	无	1
OUT	A，Rr	I/O(A)←Rr，A=0～63	1	无	1
PUSH	Rr	(SP)←Rr，SP←SP－1	1	无	2
POP	Rd	SP←SP+1，Rd←(SP)	1	无	2
SER	Rd	Rd←＄FF，d=16～31	1	无	1

2.2　算术和逻辑运算指令

AVR 算术和逻辑运算指令不限定目的寄存器，AVR 的 32 个工作寄存器均可保存运算结果。这给程序设计提供了很大的灵活性。逻辑运算指令和 8 位算术运算指令都在一个时钟周期内完成。运算中产生的某些状态特征，例如有无进借位、是否有溢出等等，均反映在 SREG 寄存器中的各标志位上。这是数据传送指令所不具备的特点。

2.2.1　加法类指令

AVR 加法指令包括 8 位不带进位的 ADD、带进位的 ADC、寄存器值递增 INC 以及 16 位寄存器对与 6 位常数相加的 ADIW 等指令。

1. 加法指令 ADD

ADD 指令对两工作寄存器的内容进行加法运算，不考虑此前的进位状态，最后把结果存入目的寄存器 Rd 中去。

指令格式：ADD Rd，Rr；　**指令操作**：Rd←Rd＋Rr；　**操作数**：0≤d≤31，0≤r≤31；

PC 操作：PC←PC＋1；　16 位指令代码：0000 11rd dddd rrrr。

ADD 指令对状态寄存器 SREG 中的 6 个运算标志位都有影响。

H：若在加法过程中自位 3 向位 4 有进位，则 H 标志置 1，否则清 0。

S：运算结果的符号位，它等于负标志 N 和溢出标志 V 的异或结果，即 S＝N⊕V。

V：若运算过程中有溢出现象发生，则 V 标志置 1，否则清 0。

N：若运算结果的最高位为 1，则 N 标志置 1，否则清 0。

Z：如果运算结果为 0，那么 Z 标志置 1，否则清 0。

C：如果在加法过程中自最高位有进位，那么 C 标志置 1，否则清 0。

[例 2.16]
```
ADD     R1,R2     ; R1←R1+R2
ADD     R28,R28   ; R28←R28+R28
```

2. 带进位的加法指令 ADC

ADC 指令将两寄存器的内容及此前之进位 C 加在一起，并把结果存放至目的寄存器 Rd 中。对标志的影响与 ADD 同。

指令格式：ADC Rd，Rr；　**指令操作**：Rd←Rd＋Rr＋C；　**操作数**：0≤d≤31，0≤r≤31；

PC 操作：PC←PC＋1；　16 位指令代码：0001 11rd dddd rrrr。

[例 2.17]　16 位加法运算 R3：R2←R3：R2＋R1：R0 用下列两条指令进行。

```
ADD     R2,R0     ; 两低字节相加
ADC     R3,R1     ; 两高字节及低字节进位相加
```

3. 字加立即数指令 ADIW

某寄存器对的 16 位内容与一 6 位立即数(0～63)相加，结果存回到此寄存器对中去。这条指令特别适于对 X、Y 和 Z 三指针内容进行调整。

指令格式：ADIW Rd，K；　**指令操作**：Rd＋1：Rd←Rd＋1：Rd＋K；

操作数：d∈{24，26，28，30}，0≤K≤63；　PC 操作：PC←PC＋1；

16 位指令代码：1001 0110 KKdd KKKK。

ADIW 指令对 H 标志无影响。对其他标志的影响基本上与 ADD 同，只不过 S、V、N、Z 和 C 都是与 16 位操作数有关的。

[例 2.18]
```
ADIW    R24,1     ; R25：R24 寄存器对内容加 1
ADIW    R30,63    ; Z 寄存器内容加 63
```

4. 递增指令 INC

INC 指令把某寄存器 Rd 的内容增 1，结果仍送回到此目的寄存器 Rd 中去。这种操作对 C 标志无影响，因此 INC 指令可作多字节运算时的循环计数器使用。

指令格式：INC Rd；　　指令操作：Rd←Rd+1；　　操作数：0≤d≤31；

PC 操作：PC←PC+1；　　16 位指令代码：1001 010d dddd 0011。

受影响的标志有 S、V、N 和 Z。当且仅当操作前 Rd 的内容为 $7F 时，V 标志才被置 1。

[例 2.19]
```
    CLR     R22         ; R22 清 0
LOOP: INC   R22         ; R22 值加 1
      ……
      CPI   R22,$4F     ; R22 值与常数 $4F 比较
      BRNE  LOOP        ; 不等则循环
```

2.2.2 减法类指令

和加法类指令相比，减法类指令里增加了 8 位常数作减数的指令。对标志的影响也同加法指令，只不过 H 和 C 这里表示半借位和借位，而非半进位和进位。

此外，比较指令通常也被归入减法类指令的范畴。

1. 减法指令 SUB

SUB 指令所进行的操作是两工作寄存器内容相减，结果被存回到 Rd 寄存器中去。

指令格式：SUB Rd，Rr；　　指令操作：Rd←Rd−Rr；　　操作数：0≤d≤31，0≤r≤31；

PC 操作：PC←PC+1；　　受影响标志：H、S、V、N、Z 和 C；

16 位指令代码：0001 10rd dddd rrrr。

[例 2.20]
```
    SUB     R13,R12     ; R13←R13−R12
    BRNE    NOTEQ       ; 若 R13≠R12，则跳转
    ……
NOTEQ: ……
```

2. 减立即数指令 SUBI

该指令把寄存器组合中某寄存器作被减数，一 8 位无符号二进制数作减数，进行减法运算，并将减法结果存回目的寄存器 Rd 中去。应当指出，这条指令只能使用组合中后 16 个工作寄存器，即 R16～R31。因此非常适用于 X、Y 和 Z 各指针寄存器的内容调整。

指令格式：SUBI Rd，K；　　指令操作：Rd←Rd−K；　　操作数：16≤d≤31，0≤K≤255；

PC 操作：PC←PC+1；　　受影响标志：H、S、V、N、Z 和 C；

16 位指令代码：0101 KKKK dddd KKKK。

[例 2.21]
```
    SUBI    R22,$11     ; R22←R22−$11
    BRNE    NOTEQ       ; 若 R22≠$11，则跳转
```

……

NOTEQ：　……

3. 带借位减法指令 SBC

这是带借位的减法指令，它执行的操作是两寄存器内容相减，再减去此前 C 标志的内容，结果存回到目的寄存器 Rd 中。

指令格式：SBC Rd，Rr；　**指令操作**：Rd←Rd－Rr－C；　**操作数**：0≤d≤31，0≤r≤31；

PC **操作**：PC←PC＋1；　**受影响标志**：H、S、V、N、Z **和** C；

16 **位指令代码**：0000 10rd dddd rrrr。

[**例 2.22**]　R3：R2←R3：R2－R1：R0 **的运算可用下列两条指令进行**。

```
SUB     R2,R0      ；低字节相减
SBC     R3,R1      ；高字节相减，考虑低字节借位
```

4. 带借位的减立即数指令 SBCI

SUBI 指令只进行目的寄存器和常数的减法，不理会此前 C 标志的状态。而 SBCI 指令除进行上述减法外，还要减去此前 C 标志之内容。

指令格式：SBCI Rd，K；**指令操作**：Rd←Rd－K－C；　**操作数**：16≤d≤31；0≤K≤255；

PC **操作**：PC←PC＋1；　**受影响标志**：H、S、V、N、Z **和** C；

16 **位指令代码**：0100 KKKK dddd KKKK。

[**例 2.23**]　R17：R16 **寄存器对的内容减去立即数** $4F23。

```
SUBI    R16,$23    ；低字节相减
SBCI    R17,$4F    ；高字节带借位相减
```

5. 字减立即数指令 SBIW

某寄存器对的内容与一 6 位无符号二进制数相减，所得结果被存回到该寄存器对中去。此指令对指针寄存器内容调整非常有用。

指令格式：SBIW Rd，K；　**指令操作**：Rd＋1：Rd←Rd＋1：Rd－K；

操作数：d∈{24，26，28，30}，0≤K≤63；　PC **操作**：PC←PC＋1；

16 **位指令代码**：1001 0111 KKdd KKKK；　**受影响标志**：S、V、N、Z **和** C。

[**例 2.24**]　SBIW　R24,1　；R25：R24 **寄存器对内容减** 1

SBIW　R28,63　；Y **指针内容减** 63

6. 递减指令 DEC

某工作寄存器值减 1，所得结果仍留在此寄存器中。

和 INC 指令类似，DEC 指令不影响 C 标志，因此可作多字节运算时的循环计数用。

指令格式：DEC Rd；　**指令操作**：Rd←Rd－1；　**操作数**：0≤d≤31；

PC **操作**：PC←PC＋1；　16 **位指令代码**：1001 010d dddd 1010；

受影响标志：S、V、N **和** Z。**若此前** Rd **值为** $80，**则** DEC Rd **指令执行后**，V **标志置** 1。**也只有这种情况下** DEC **指令才将** V **标志置** 1。

[**例 2.25**]　**以简短程序实现表达式** R1←R1＋16＊R2 **所要求的运算**。

```
LDI     R17,$16    ；R17 赋初值 16
```

```
LOOP:  ADD   R1,R2        ; R1←R1+R2
       DEC   R17          ; R17←R17-1
       BRNE  LOOP         ; 若 R17 不为 0 则循环
```

7. 取反指令 COM

此指令将 Rd 寄存器的内容按位取反码，原为 1 者变成 0，为 0 者变成 1。

指令格式：COM Rd；指令操作：Rd←$FF－Rd；操作数：0≤d≤31；PC 操作：PC←PC+1；

16 位指令代码：1001 010d dddd 0000；受影响标志：S、N 和 Z，V=0，C=1。

[例 2.26]
```
       COM    R4     ; R4 内容取反
       BREQ   ZERO   ; 为 0 则跳转
       ……
ZERO:  ……
```

8. 取补指令 NEG

此指令把 Rd 寄存器的内容转换成其二进制补码形式。若 Rd 原内容为 $80，则 NEG 指令执行后，Rd 内容将仍是 $80。

指令格式：NEG Rd；指令操作：Rd←$00－Rd；操作数：0≤d≤31；PC 操作：PC←PC+1；

16 位指令代码：1001 010d dddd 0001；

受影响标志：H、S、V、N、Z 和 C。V 标志置 1 只有在操作结果等于 $80 时才会发生。C 标志在 NEG 指令后总被置 1，只是操作结果等于 $00 时例外。

[例 2.27]
```
       SUB    R11,R0     ; R11←R11-R0
       BRPL   POSIT      ; 若结果为正则跳转
       NEG    R11        ; R11 内容取补
POSIT: ……
```

9. 比较指令 CP

CP 指令进行 Rd 和 Rr 寄存器内容的比较，但不改变它们的内容，只是把比较时的特征反映到状态寄存器各标志位中。CP 指令后可接用任一条件跳转指令。

指令格式：CP Rd，Rr；　指令操作：Rd－Rr；　操作数：0≤d≤31，0≤r≤31；

PC 操作：PC←PC+1；　受影响标志：H、S、V、N、Z 和 C；

16 位指令代码：0001 01rd dddd rrrr。

[例 2.28]
```
       CP     R4,R19     ; R4-R19
       BRNE   NOTEQ      ; 若 R4≠R19 则跳转
       ……
NOTEQ: ……
```

10. 带借位的比较指令 CPC

CPC 指令进行 Rd 和 Rr 两寄存器内容的比较，同时也考虑到此前的 C 标志状态。比较后两寄存器内容均不发生变化，SREG 各标志位集中反映比较操作的特征。故此指令后可接用任一条件分支指令。

指令格式：CPC Rd，Rr；　指令操作：Rd－Rr－C；　操作数：0≤d≤31，0≤r≤31；

PC 操作：PC←PC+1；　受影响标志：H、S、V、N、Z 和 C；

16 位指令代码：0000 01rd dddd rrrr。

应当强调指出，CPC 指令对 Z 标志的影响是很特殊的。CPC 指令执行后，Z 标志置 1(即参与比较的两数相等)的条件是必须同时满足下面两点：

- CPC 指令执行前 Z=1；
- Rd−Rr−C=0。

因此，例 29 中，只有 R3：R2=R1：R0 时，Z 标志才能置 1。

[例 2.29]　寄存器对 R3：R2 与寄存器对 R1：R0 比较，应该用下面几条指令。

```
        CP    R2,R0     ；低字节比较
        CPC   R3,R1     ；高字节带 C 比较
        BRNE  NOTEQ     ；不相等则跳转
        ……
NOTEQ:  ……
```

11. 与立即数比较指令 CPI

这是某寄存器 Rd 值与一无符号 8 位二进制常数进行比较的指令。比较后 Rd 内容不发生变化，只是 SREG 寄存器中的各标志位用来反映比较结果的特征。故此指令后可接用任一条件分支指令。

指令格式：CPI Rd，K；　指令操作：Rd−K；　　操作数：16≤d≤31，0≤K≤255；

PC 操作：PC←PC+1；　受影响标志：H、S、V、N、Z 和 C；

16 位指令代码：0011 KKKK dddd KKKK。

[例 2.30]
```
        CPI    R19,3     ；R19 的内容=3？
        BRNE   ERROR     ；不等则跳转
        ……
ERROR:  ……
```

2.2.3　逻辑与指令

AVR 的逻辑与指令包括寄存器间的逻辑相与(AND)、寄存器和常数逻辑相与(ANDI)、寄存器某些位清 0(CBR)以及寄存器内容测试(TST)等指令。

1. 逻辑与指令 AND

AND 指令对 Rd 和 Rr 两寄存器的内容逐位进行逻辑与运算，并将结果存回到目的寄存器 Rd 中去。

指令格式：AND Rd，Rr；　指令操作：Rd←Rd & Rr；　　操作数：0≤d≤31，0≤r≤31；

PC 操作：PC←PC+1；　受影响标志：S、N 和 Z，V 被清 0；

16 位指令代码：0010 00rd dddd rrrr。

[例 2.31]
```
        AND    R2,R3     ；R2 和 R3 内容按位相与，结果在 R2 中
        LDI    R16,1     ；R16 中存入屏蔽字节 0000 0001
        AND    R2,R16    ；保留 R2 中最低位不变，其他位清 0
```

2. 和立即数逻辑相与指令 ANDI

ANDI 指令对某寄存器 Rd 的内容和一 8 位立即数进行逻辑与运算，并将结果存入目的寄存器中。这里需要提醒读者的是，凡是和 8 位立即数打交道的寄存器，必定是 R16～R31 中的一个。R0～R15 无此功能。

指令格式:ANDI Rd, K; **指令操作**:Rd←Rd & K; **操作数**:16≤d≤31,0≤K≤255;
PC **操作**:PC←PC+1; **受影响标志**:S、N **和** Z,V **被逻辑指令清** 0;
16 **位指令代码**:0111 KKKK dddd KKKK。

[**例 2.32**]
```
ANDI    R17,$0F     ; R17 高 4 位清 0
ANDI    R18,$10     ; 保留 R18 的位 4 不变,其他位清 0
ANDI    R19,$AA     ; R19 中偶数位清 0
```

3. 寄存器位清 0 指令 CBR

CBR **指令将** Rd **寄存器中的指定位清** 0。**实际上这条指令执行的是** Rd **寄存器内容和屏蔽常数** K **的反码间的逻辑与运算,以及把结果存入** Rd **中的数据传送操作。**

指令格式:CBR Rd, K; **指令操作**:Rd←Rd & ($FF−K);
操作数:16≤d≤31,0≤K≤255;
PC **操作**:PC←PC+1; **受影响标志**:S、N **和** Z,V **被逻辑指令清** 0;
16 **位指令代码**:**同** ANDI,**只是应将** K **按位取反码。**

[**例 2.33**]
```
CBR     R16,$F0     ; R16 高 4 位清 0
CBR     R18,1       ; R18 最低位清 0
```

[**例 2.34**] ANDI R16, $0F **的指令代码为** $700F;CBR R16, $F0 **的指令代码也是** $700F。**故它们执行的是同样的操作,即把** R16 **的高** 4 **位清成** 0。

4. 寄存器内容测试指令 TST

此指令对某工作寄存器 Rd **的内容进行测试,看其是否为** 0 **或负数。实际上,这条指令对** Rd **的内容和其本身进行逻辑与运算,使其内容保持不变,但却把全部特征都反映到了** SREG **寄存器的各标志位上。**

指令格式:TST Rd; **指令操作**:Rd←Rd & Rd; **操作数**:0≤d≤31;
PC **操作**:PC←PC+1; **受影响标志**:S、N、**和** Z,V **被清** 0;
16 **位指令代码**:0010 00dd dddd dddd。

[**例 2.35**]
```
        TST     R0          ; 测试 R0
        BREQ    ZERO        ; 若 R0=0 则跳转
        ……
ZERO:   ……
```

2.2.4 逻辑或指令

这类指令有寄存器间的逻辑相或(OR)、**寄存器与常数相或**(ORI)**以及寄存器某些位置** 1 (SBR)**等指令。**

1. 逻辑或指令 OR

OR **指令对** Rd **和** Rr **两工作寄存器的内容逐位进行逻辑或运算,并将结果存回到目的寄存器** Rd **中去。**

指令格式:OR Rd, Rr; **指令操作**:Rd←Rd∨Rr; **操作数**:0≤d≤31,0≤r≤31;
PC **操作**:PC←PC+1; **受影响标志**:S、N **和** Z,V=0;
16 **位指令代码**:0010 10rd dddd rrrr。

[例 2.36]
```
        OR      R15,R16     ; 寄存器间按位逻辑或运算
        BST     R15,6       ; 将 R15 的位 6 存入 T 标志中
        BRTS    OK          ; 若 T=1,则跳转
        ……
OK:     ……
```

2. 和立即数逻辑相或指令 ORI

此指令对某工作寄存器 Rd 的内容和一 8 位常数进行逻辑或运算，并将结果存回到目的寄存器 Rd 中去。

指令格式：ORI Rd，K；　**指令操作**：Rd←Rd∨K；　**操作数**：16≤d≤31，0≤K≤255；

PC 操作：PC←PC+1；　**受影响标志**：S、N 和 Z，V 被逻辑指令清 0；

16 位指令代码：0110 KKKK dddd KKKK。

[例 2.37]
```
        ORI     R16,$F0     ; R16 高 4 位置 1
        ORI     R17,1       ; R17 最低位置 1
```

3. 寄存器位置 1 指令 SBR

SBR 指令把工作寄存器 Rd 中的某些指定位置成 1。实际上，这条指令所进行的操作，一是 Rd 寄存器内容和屏蔽常数 K 间的逻辑或运算，二是把结果存回到目的寄存器 Rd 中去。

指令格式：SBR Rd，K；　**指令操作**：Rd←Rd∨K；　**操作数**：16≤d≤31，0≤K≤255；

PC 操作：PC←PC+1；　**受影响标志**：S、N 和 Z，V 被清 0；

16 位指令代码：0110 KKKK dddd KKKK。

读者不难发现，ORI 和 SBR 两条指令实际上只不过是一种操作的不同写法而已。

[例 2.38]
```
        SBR     R16,3       ; R16 最低两位置 1
        SBR     R17,$F0     ; R17 高 4 位置 1
```

2.2.5　逻辑异或指令

这里只有两工作寄存器 Rd 和 Rr 间按位进行异或运算的一条 EOR 指令及其一个特例 CLR 指令。

1. 逻辑异或指令 EOR

EOR 指令对 Rd 和 Rr 两工作寄存器的内容逐位进行逻辑异或运算，最后把结果存入目的寄存器 Rd 中。

指令格式：EOR Rd，Rr；　**指令操作**：Rd←Rd⊕Rr；　**操作数**：0≤d≤31，0≤r≤31；

PC 操作：PC←PC+1；　**受影响标志**：S、N 和 Z，V 被清 0；

16 位指令代码：0010 01rd dddd rrrr。

[例 2.39]
```
        EOR     R4,R4       ; R4 清 0
        EOR     R0,R22      ; R0 和 R22 按位异或
```

2. 寄存器清 0 指令 CLR

CLR Rd 指令是 EOR 指令的一个特例，即 EOR Rd，Rd 的另一种写法，故其操作码为 0010 01dd dddd dddd。大家知道，某 8 位数和其自身进行逻辑异或运算，其结果肯定为 0。所以这是一条把寄存器内容清 0 的指令。这条指令执行后，S=V=N=0，Z=1。

表 2.10 中汇总了上述所有算术和逻辑运算指令。

表 2.10 AVR 算术和逻辑运算指令一览表

助记符	操作数	指 令 操 作	受影响标志	代码字数	时钟周期
ADD	Rd,Rr	Rd←Rd+Rr	H,S,V,N,Z,C	1	1
ADC	Rd,Rr	Rd←Rd+Rr+C	H,S,V,N,Z,C	1	1
ADIW	Rd,K	Rd+1：Rd←Rd+1：Rd+K，K=0～63	S,V,N,Z,C	1	2
SUB	Rd，Rr	Rd←Rd−Rr	H,S,V,N,Z,C	1	1
SUBI	Rd，K	Rd←Rd−K，K=0～255	H,S,V,N,Z,C	1	1
SBC	Rd，Rr	Rd←Rd−Rr−C	H,S,V,N,Z,C	1	1
SBCI	Rd，K	Rd←Rd−K−C，d=16～31，K=0～255	H,S,V,N,Z,C	1	1
SBIW	Rd，K	Rd+1：Rd←Rd+1：Rd−K，K=0～63	S,V,N,Z,C	1	2
CP	Rd，Rr	Rd−Rr	H,S,V,N,Z,C	1	1
CPC	Rd，Rr	Rd−Rr−C	H,S,V,N,Z,C	1	1
CPI	Rd，K	Rd−K，d=16～31，K=0～255	H,S,V,N,Z,C	1	1
AND	Rd，Rr	Rd←Rd & Rr	S,N,Z,V=0	1	1
ANDI	Rd，K	Rd←Rd & K，d=16～31，K=0～255	S,N,Z,V=0	1	1
OR	Rd，Rr	Rd←Rd∨Rr	S,N,Z,V=0	1	1
ORI	Rd，K	Rd←Rd∨K，d=16～31，K=0～255	S,N,Z,V=0	1	1
EOR	Rd，Rr	Rd←Rd⊕Rr	S,N,Z,V=0	1	1
COM	Rd	Rd←$FF−Rd	S,N,Z,V=0	1	1
NEG	Rd	Rd←$00−Rd	H,S,V,N,Z,C	1	1
SBR	Rd，K	Rd←Rd∨K，d=16～31，K=0～255	S,N,Z,V=0	1	1
CBR	Rd，K	Rd←Rd &（$FF−K),d=16～31,K=0～255	S,N,Z,V=0	1	1
INC	Rd	Rd←Rd+1	S,V,N,Z	1	1
DEC	Rd	Rd←Rd−1	S,V,N,Z	1	1
TST	Rd	Rd←Rd & Rd	S,N,Z,V=0	1	1
CLR	Rd	Rd←Rd⊕Rd	S=V=N=0,Z=1	1	1

2.3 分支指令

分支指令可分无条件分支与条件分支两类。无条件分支包括跳转、子程序调用及返回等指令。条件分支又有条件跨越和条件跳转等诸多指令。这类指令的共性是它们都不影响任何标志位，只有中断返回指令 RETI 例外。

2.3.1　无条件跳转指令

对于 PC 寄存器不超过 12 位的小容量 Flash 单片机来说，没有直接跳转指令，只有 PC 相对寻址和间接寻址无条件跳转指令。

1. 相对跳转指令 RJMP

相对跳转到 PC－2 K＋1 至 PC＋2 K(字)范围内的某地址单元去继续执行程序。用汇编语言编程时，跳转的目的地址在指令中以标号形式取代相对地址 k。对于程序存储器不超过 4 K字(8 K 字节)的那些微控制器来说，这条指令可从任一地址单元跳转到整个程序存储空间的任一地点去。

指令格式：RJMP k；　　**指令操作**：PC←PC＋k＋1；　　**操作数**：－2 K≤k≤2 K；

PC 操作：PC←PC＋k＋1；　　16 位指令代码：1100 kkkk kkkk kkkk。

[例 2.40]
```
        CPI     R16,$42     ; R16 与 $42 相比较
        BRNE    ERROR       ; 若不等则跳转
        RJMP    OK          ; 无条件跳转
ERROR:  ADD     R16,R17     ; R16←R16＋R17
        INC     R16         ; R16←R16＋1
   OK:  ……                  ; RJMP 的目的地址
```

2. 间接跳转指令 IJMP

间接跳转到 Z 指针寄存器的内容所指定的地址单元去。

指令格式：IJMP；　　**指令操作**：PC←Z；　　16 位指令代码：$9409。

[例 2.41]　欲无条件跳转到 $0789 地址单元去，可用下列几条指令：

```
LDI     R31,$07     ; Z 寄存器高字节＝$07
LDI     R30,$89     ; Z 寄存器低字节＝$89
IJMP                ; 间接跳转至 $0789 单元
```

2.3.2　子程序调用指令

和跳转指令的情况类似，这里我们不讨论直接寻址的长调用指令 CALL，而只介绍 118 条指令子集中的 RCALL 和 ICALL。

1. 相对调用子程序指令 RCALL

RCALL 指令可用来调用 PC－2 K＋1 至 PC＋2 K 范围内以某地址单元为入口的子程序。这条指令执行时，返回地址，即 RCALL 后面一条指令之地址，被自动压入堆栈保存起来。在汇编语言写法中，要用标号地址来取代相对地址 k。对于程序存储容量不超过 4 K 字(8 K 字节)的 AVR 微控制器来说，此指令可从任一地址单元出发访问整个程序空间任何一处的子程序。在 RCALL 指令执行期间，使用后减量机制。

指令格式：RCALL k；　　**指令操作**：(SP)←PC＋1，SP←SP－2，PC←PC＋k＋1；

操作数：－2 K≤k≤2 K；　　16 位指令代码：1101 kkkk kkkk kkkk。

[例 2.42]
```
        RCALL   RTNE        ; 调用子程序
        ……
RTNE:   PUSH    R14         ; 保护 R14 入栈
```

```
……
POP     R14     ; 恢复 R14 内容
RET             ; 子程序返回
```

2. 间接调用子程序指令 ICALL

此指令间接调用以 Z 指针寄存器的内容为其入口地址的子程序。在 ICALL 指令执行期间，自动保存返回地址入栈时，堆栈指针 SP 使用后减量机制。

指令格式：ICALL；指令操作：(SP)←PC+1，SP←SP−2，PC←Z；16 位指令代码：$9509。

[例 2.43] 已知某子程序入口地址的低 8 位保存在 R0 寄存器中，其高 4 位在 R31 中，那么可用下列两条指令来调用这段子程序。

```
MOV     R30,R0      ; 入口地址低 8 位送入 Z 的低字节中
ICALL               ; 以 Z 寄存器为指针调用子程序
```

2.3.3 返回指令

返回指令包括子程序返回和中断返回两条指令。

1. 子程序返回指令 RET

各段子程序通常均用 RET 指令作其结尾。执行这条指令时，堆栈栈顶的两个字节被弹出来送入程序计数器 PC 中。大家知道，在子程序调用时，RCALL 或 ICALL 后面一条指令的地址作为返回地址被自动压入堆栈保存了起来。现在执行 RET 指令时，又把这两个字节的返回地址从栈顶弹出来送入 PC 中。这就是说，要从此地址处恢复程序的执行了。

指令格式：RET；　　指令操作：SP←SP+1，PCL←(SP)，SP←SP+1，PCH←(SP)；16 位指令代码：$9508。

2. 中断返回指令 RETI

中断服务程序一般都以 RETI 指令作其结尾。RETI 指令有两项功能：一是恢复断点，此功能同 RET 指令，即把返回地址从栈顶弹出送入 PC 中；二是恢复 SREG 寄存器中总体中断开放位 I 的置位状态。后者是在中断响应过程中被自动清 0 的。

指令格式：RETI；　指令操作：同 RET；　16 位指令代码：$9518；　受影响标志：I←1。

[例 2.44] 某中断服务程序的开头与结尾可有如下形式。

```
EXTINT: PUSH    R0      ; 保护 R0 入栈
        ……              ; 中断处理，略
        POP     R0      ; 恢复 R0 原值
        RETI            ; 中断返回
```

2.3.4 条件跨越指令

条件跨越指令，顾名思义，当条件满足时，将跨越其后第一条指令，直接去执行其后第二条指令。但当条件不满足时，则将无跨越地接续执行其后第一条指令。这种跨越指令的条件有：两工作寄存器内容相等，寄存器某位为 0、为 1 以及 I/O 寄存器某位为 0 和为 1 等。

1. 比较相等时的跨越指令 CPSE

CPSE 指令首先比较 Rd 和 Rr 两寄存器内容，若相等，则跨越其后一条指令，直接去执行其后第二条指令；但若两值不等，则将接续执行其后一条指令。

指令格式：CPSE Rd，Rr；**指令操作**：**若** Rd＝Rr，**则** PC←PC＋2(**或** 3)，**否则** PC←PC＋1；

操作数：0≤d≤31，0≤r≤31；　16 **位指令代码**：0001 00rd dddd rrrr。

应当指出，若无跨越发生(PC←PC＋1)，**则此类指令的执行时间为一个时钟周期；如果跨越单字指令**(PC←PC＋2)，**那么执行时间等于两个时钟周期；跨越双字指令**(PC←PC＋3)**则将需三个时钟周期。**

[例 2.45]　R4 **的内容加** 1 **后与** R0 **内容相比较，若两者不等，则要对** R4 **内容求补；如果两值相等，即无须求补。这段程序如下：**

```
INC     R4          ; R4 值增 1
CPSE    R4,R0       ; R4 与 R0 比较
NEG     R4          ; R4≠R0,R4←$00－R4
……                  ; R4＝R0,跨越 NEG 指令,由此接续
```

2. 寄存器位为 0 时的跨越指令 SBRC

这条指令对某工作寄存器的给定位 b **进行测试，若为** 0，**则跨越其后一条指令，否则接续执行其后一条指令。**

指令格式：SBRC Rr，b；**指令操作**：**若** Rr(b)＝0，**则** PC←PC＋2(**或** 3)，**否则** PC←PC＋1；

操作数：0≤r≤31，0≤b≤7；　16 **位指令代码**：1111 110r rrrr 0bbb。

[例 2.46]　**做减法** R0←R0－R1，**若所得结果最高位为** 1，**则** R0 **值还原，否则保留差值。**

```
SUB     R0,R1       ; R0←R0－R1
SBRC    R0,7        ; 若位 7 为 0,则无须执行下一条指令
ADD     R0,R1       ; 位 7 等于 1 时才执行这条指令
```

3. 寄存器位为 1 时的跨越指令 SBRS

指令格式：SBRS Rr，b；**指令操作**：**若** Rr(b)＝1，**则** PC←PC＋2(**或** 3)，**否则** PC←PC＋1；

操作数：0≤r≤31，0≤b≤7；　16 **位指令代码**：1111 111r rrrr 0bbb。

[例 2.47]　R0 **与** R1 **两寄存器内容相减，若差值最高位为** 1，**则结果有效，否则差值应求补。**

```
SUB     R0,R1       ; R0←R0－R1
SBRS    R0,7        ; R0 之位 7 为 1 则跨越
NEG     R0          ; R0 位 7 为 0 时,对差值求补
```

4. I/O 寄存器位为 0 时的跨越指令 SBIC

SBIC **指令对某** I/O **寄存器的指定位进行测试，若该位为** 0，**则跨越其后一条指令，去执行其后第二条指令，否则接续执行。此指令只适用于低地址的** 32 **个** I/O **寄存器，即其** I/O **地址为** 0～31 **者。**

指令格式：SBIC A，b；**指令操作**：**若** I/O(A，b)＝0，**则** PC←PC＋2(**或** 3)，**否则** PC←PC＋1；

操作数：0≤A≤31，0≤b≤7；　16 **位指令代码**：1001 1001 AAAA Abbb。

[例 2.48]　**检查** EECR **寄存器的** EEWE **位是否为** 0，**若是，则进行下一字节的写入，否则等待。**

```
E2WAIT:SBIC    $1C,1     ; 若 EEWE 位为 0 则跨越
       RJMP    E2WAIT    ; EEPROM 写操作尚未完成,等待
       ……                ; 此时可写下一个字节了
```

5. I/O 寄存器位为 1 时的跨越指令 SBIS

指令格式:SBIS A,b;**指令操作**:若 I/O(A,b)=1,则 PC←PC+2(或 3),否则 PC←PC+1;

操作数:0≤A≤31,0≤b≤7;　　16 位指令代码:1001 1011 AAAA Abbb。

[例 2.49]　D 口最低位若输入 1,则进行某种操作,输入 0 时应等待。

```
WAITSET: SBIS    $10,0     ; 若 PIND0=1,则跨越
         RJMP    WAITSET   ; PIND0=0,跳回等待
         ……                ; 此时可进行操作
```

2.3.5　基本条件分支指令

AVR 条件分支指令虽多达 20 条,但最基本的却只有两条,其余均为其派生产物。所有这些指令欲跳往的目的地址皆以 7 位带符号二进制数的偏移量形式给出,其取值范围为 −64～+63。该偏移量由本指令后的第一条指令(即 PC+1)算起。因此,条件分支指令的跳转范围是 PC−63～PC+64。

1. 以状态位为 0 作条件的分支指令 BRBC

这是相对寻址的条件分支指令,它对 SREG 寄存器的某给定位 s 进行测试,若该位为 0,则跳转到目的地址去,否则接续执行下一条指令。

指令格式:BRBC s, k;**指令操作**:若 SREG(s)=0,则 PC←PC+k+1,否则 PC←PC+1;

操作数:0≤s≤7,−64≤k≤63;　　16 位指令代码:1111 01kk kkkk ksss。

应当指出,在用汇编语言编写程序时,指令中的偏移量 k 应以目的地址的标号代替。

[例 2.50]　若 R20 寄存器的内容等于 5,则应进行某种操作,而不相等时便无此操作。

```
        CPI     R20,5     ; R20 与立即数 5 比较
        BRBC    1,NOTEQ   ; 若 Z=0,则转 NOTEQ
        ……                ; Z=1 时的处理
NOTEQ:……                 ; Z=0 时的目的地址
```

2. 以状态为 1 作条件的分支指令 BRBS

指令格式:BRBS s, k;**指令操作**:若 SREG(s)=1,则 PC←PC+k+1,否则 PC←PC+1;

操作数:0≤s≤7,−64≤k≤63;　　16 位指令代码:1111 00kk kkkk ksss。

[例 2.51]　对 R0 的位 3 进行测试,若该位为 1,则跳转到 BITSET 去,否则接续执行下一条指令。可用下列几条指令实现这一要求。

```
        BST     R0,3      ; 将 R0 之位 3 送入 T 中
        BRBS    6,BITSET  ; T 为 1 则跳转至 BITSET 处
        ……                ; T 为 0 则接续执行这里的指令
BITSET:……                ; T 为 1 时的目的地址
```

2.3.6　派生条件分支指令

把上一节两条基本指令代码中的参数 s 固定成 000～111,即可派生出下列 18 条指令来。

把这些指令换成新的助记符写法，读者便可不必记忆各标志位究竟在 SREG 寄存器中的什么位置上。

1. 测试 0 标志 Z(s=001)

(1) 相等则跳转指令 BREQ：BREQ k 是 BRBS 1，k 指令的另一种写法。它对 Z 标志进行测试，若 Z=1，则将做相对 PC 的跳转，否则接续执行其后一条指令。若此指令紧接在 CP、CPI、SUB 或 SUBI 指令后执行，则只有在参与比较或参与减法的两数 Rd、Rr(或立即数)相等时，才会发生跳转现象。

指令格式：BREQ k；指令操作：若 Rd=Rr(Z=1)，则 PC←PC+k+1，否则 PC←PC+1；

操作数：−64≤k≤63；　16 位指令代码：1111 00kk kkkk k001(同 BRBS 1，k)。

[例 2.52]
```
        CP      R1,R0       ; R1 和 R0 比较
        BREQ    EQUAL       ; 相等则跳转
        ……                  ; 不等时执行这里的段落
EQUAL:  ……                  ; 跳转目的地址
```

(2) 不相等则跳转指令 BRNE：这是 BRBC 1，k 指令的另一种写法。它对 Z 标志进行测试，若 Z=0，则跳转到偏移量 k 所指定的地址去，否则接续执行下面一条指令。如果此指令紧跟在 CP、CPI、SUB 或 SUBI 指令后面，那么只有参与比较或减法运算的两数 Rd、Rr(或立即数)不相等时，才能发生跳转现象。

指令格式：BRNE k；指令操作：若 Rd≠Rr(Z=0)，则 PC←PC+k+1，否则 PC←PC+1；

操作数：−64≤k≤ 63；　16 位指令代码：1111 01kk kkkk k001(同 BRBC 1，k)。

[例 2.53]
```
        EOR     R27,R27     ; R27 清 0
LOOP:   INC     R27         ;R27 内容加 1
        ……
        CPI     R27,5       ;R27=5？
        BRNE    LOOP        ; 不等则循环
        ……                  ; 相等则结束循环
```

2. 测试进位标志 C(s=000)

(1) C 置位则跳转指令 BRCS：等于 BRBS 0，k 指令，若 C 标志为 1，则跳转，否则接续执行下一条指令。

指令格式：BRCS k；　指令操作：若 C=1，则 PC←PC+k+1，否则 PC←PC+1；

操作数：−64≤k≤63；　16 位指令代码：1111 00kk kkkk k000(同 BRBS 0，k)。

[例 2.54]
```
        CPI     R26,$56     ; R26－$56
        BRCS    CARRY       ; 有借位，则跳转
        ……
CARRY:  ……                  ; 跳转目的地址
```

(2) C 清 0 则跳转指令 BRCC：这是 BRBC 0，k 指令的另一种写法。若 C 标志为 0，则跳转，否则接续执行下一条指令。

指令格式：BRCC k；　指令操作：若 C=0，则 PC←PC+k+1，否则 PC←PC+1；

操作数：−64≤k≤63；　16 位指令代码：1111 01kk kkkk k000(同 BRBC 0，k)。

[例 2.55]
```
        ADD     R22,R23     ; R22←R22+R23
```

```
        BRCC    NOCARRY  ; 若 C=0,则跳转
        ……               ; C=1 时的处理
NOCARRY: ……              ; 跳转目的地址
```

(3) **相等或高于(无符号数)则跳转指令** BRSH:**这是** BRBC 0, k **指令的第三种写法。** BRSH **主要是针对两无符号数之比较结果的。如果此指令紧跟在** CP、CPI、SUB **或** SUBI **指令之后,那么只有原** Rd **中的无符号二进制数大于或等于** Rr **中的无符号二进制数或立即数时(此时无借位,即** C=0),**才会发生跳转现象。**

指令格式:BRSH k;**指令操作、操作数情况及指令代码与** BRCC k **和** BRBC 0,k **两指令相同。只是在指令操作中,这里强调的是** Rd≥Rr,**即** C=0。

[**例 2.56**] **若** R19 **中的无符号二进制数大于或等于** 4,**则程序应跳转到** HISM **地址单元去。此任务可用下面几条指令来完成。**

```
        SUBI    R19,4    ; R19←R19-4
        BRSH    HISM     ; 若 R19≥4,则跳转
        ……
HISM: ……                ; 跳转目的地址
```

(4) **低于(无符号数)则跳转指令** BRLO:**这是** BRBS 0,k **指令的第三种写法,也是针对无符号数比较结果的。如果** BRLO **指令紧跟随在** CP、CPI、SUB **或** SUBI **指令后面,那么只有原** Rd **中的无符号二进制数小于** Rr **中无符号数或立即数时,才会发生跳转现象。**

指令格式 BRLO k;**指令操作、操作数情况及指令代码与** BRBS 0, k **指令、**BRCS **指令完全相同,只不过指令操作中,这里强调的是** Rd<Rr,**即** C=1。

[**例 2.57**]
```
        EOR     R19,R19  ; R19 清 0
LOOP:   INC     R19      ; R19 内容加 1
        ……
        CPI     R19,$10  ; R19-$10
        BRLO    LOOP     ; R19<$10(无符号数)则循环
        ……               ; 退出循环
```

3. 测试负标志 N(s=010)

(1) **负则跳转指令** BRMI:**这是** BRBS 2,k **指令侧重此前运算结果之正、负号的更加明确的写法。** BRMI **指令对** N **标志进行测试,若** N **为** 1,**则跳转,否则循序前进。**

指令格式:BRMI k; **指令操作:若** N=1,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:-64≤k≤63; 16 **位指令代码:**1111 00kk kkkk k010(**同** BRBS 2,k)。

[**例 2.58**]
```
        SUBI    R18,4    ; R18←R18-4
        BRMI    NEGT     ; 结果为负则跳转
        ……
NEGT:   ……               ; 跳转目的地址
```

(2) **正则跳转指令** BRPL:**这是** BRBC 2, k **指令的另一种写法,它强调此前运算结果的符号。**

指令格式:BRPL k; **指令操作:若** N=0,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:-64≤k≤63; 16 **位指令代码:**1111 01kk kkkk k010(**同** BRBC 2,k)。

[例 2.59]
```
SUBI    R26,$50     ; R26←R26－$50
BRPL    POSIT       ; 结果为正则跳转
……
POSIT:  ……         ; 跳转目的地址
```

4. 测试符号标志 S＝N⊕V(s＝100)

(1) 大于或等于(有符号数)则跳转指令 BRGE:这是 BRBC 4,k 指令的另一种写法,它更明确地强调 G(大于)或 E(等于)。如果这条指令紧跟在 CP、CPI、SUB 或 SUBI 指令后,那么只有 Rd 寄存器中的有符号数曾经大于或等于 Rr 中的有符号数或作减数的立即数时,才会有跳转现象发生。

指令格式:BRGE k;　　操作数:－64≤k≤63;

指令操作:若 Rd≥Rr(s＝N⊕V＝0),则 PC←PC＋k＋1,否则 PC←PC＋1;

16 位指令代码:1111 01kk kkkk k100(同 BRBC 4,k)。

[例 2.60]
```
CP      R11,R12     ; R11 与 R12 比较
BRGE    GRTER       ; 若 R11≥R12(有符号数),则跳转
……
GRTER:  ……         ; 跳转目的地址
```

(2) 小于(有符号数)则跳转指令 BRLT:这是 BRBS 4,k 指令的另一种写法。BRBS 4,k 只是说,若 SREG 中的位 4 为 1 则跳转;而 BRLT 则明确强调带符号的被减数若小于减数则跳转。但两者的实质都是以符号位 S＝1 作条件的。

如果 BRLT 指令紧跟随在 CP、CPI、SUB 或 SUBI 指令后,那么只有 Rd 寄存器中的带符号数小于 Rr 中的带符号数或一立即数时,才会发生跳转。

指令格式:BRLT k;　　操作数:－64≤k≤63;

指令操作:若 Rd＜Rr(s＝N⊕V＝1),则 PC←PC＋k＋1,否则 PC←PC＋1;

16 位指令代码:1111 00kk kkkk k100(同 BRBS 4,k)。

[例 2.61]
```
CP      R16,R1      ; R16 与 R1 中的有符号数相比较
BRLT    LESS        ; 若 R16＜R1 则跳转
……
LESS:   ……         ; 跳转目的地址
```

5. 测试半进位标志 H(s＝101)

(1) 半进位标志清 0 则跳转指令 BRHC:这是 BRBC 5,k 指令的一种以 H 标志为 0 作条件的明确表示方式。

指令格式:BRHC k;　　指令操作:若 H＝0,则 PC←PC＋k＋1,否则 PC←PC＋1;

操作数:－64≤k≤63;　　16 位指令代码:1111 01kk kkkk k101(同 BRBC 5,k)。

(2) 半进位标志置 1 则跳转指令 BRHS:这是 BRBS 5,k 指令的另一种表现形式。

指令格式:BRHS k;　　指令操作:若 H＝1,则 PC←PC＋k＋1,否则 PC←PC＋1;

操作数:－64≤k≤63;　　16 位指令代码:1111 00kk kkkk k101(同 BRBS 5,k)。

6. 测试 T 标志位(s＝110)

(1) T 标志清 0 则跳转指令 BRTC:BRBC 6,k 指令的明确表现形式。

指令格式:BRTC k;　　指令操作:若 T＝0,则 PC←PC＋k＋1,否则 PC←PC＋1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 01kk kkkk k110(**同** BRBC 6,k)。

[**例 2.62**]　**测试工作寄存器中某一位是否为** 0,**可用下列指令。**

```
        BST     R3,5      ; R3 之位 5 存入 T 中
        BRTC    TCLR      ; 若 T=0 则跳转
        ……
TCLR:   ……                ; 跳转目的地址
```

(2) T **标志置** 1 **则跳转指令** BRTS:**和** BRBS 6,k **指令等效。**

指令格式:BRTS k;　　**指令操作**:**若** T=1,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 00kk kkkk k110(**同** BRBS 6,k)。

7. 测试溢出标志 V(s=011)

(1) **无溢出则跳转指令** BRVC:BRBC 3,k **指令的另一种写法,它明确指出此前运算无溢出作为跳转的条件。**

指令格式:BRVC k;　　**指令操作**:**若** V=0,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 01kk kkkk k011(**同** BRBC 3,k)。

[**例 2.63**]
```
        ADD     R3,R4     ; R3←R3+R4
        BRVC    OVOK      ; 无溢出则跳转
        ……                ; 溢出处理
OVOK:   ……                ; 跳转目的地址
```

(2) **有溢出则跳转指令** BRVS:**与** BRBS 3,k **指令等效。**

指令格式:BRVS k;　　**指令操作**:**若** V=1,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 00kk kkkk k011(**同** BRBS 3,k)。

[**例 2.64**]
```
        ADD     R3,R4     ; R3←R3+R4
        BRVS    OVFL      ; 有溢出则跳转
        ……
OVFL:   ……                ; 溢出处理
```

8. 测试总体中断允许标志 I(s=111)

(1) **中断关闭则跳转指令** BRID:**对总体中断开放位** I **进行测试,若为** 0,**即在关闭所有中断的情况下,跳转到目的地址去,否则接续执行下一条指令。它与** BRBC 7,k **指令是等效的。**

指令格式:BRID k;　　**指令操作**:**若** I=0,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 01kk kkkk k111(**同** BRBC 7,k)。

(2) **中断开放则跳转指令** BRIE:**与** BRBS 7,k **指令等效,不过助记符** BRIE **中的** IE(**中断开放**)**更醒目。**

指令格式:BRIE k;　　**指令操作**:**若** I=1,**则** PC←PC+k+1,**否则** PC←PC+1;

操作数:−64≤k≤63;　　16 **位指令代码**:1111 00kk kkkk k111(**同** BRBS 7,k)。

表 2.11 **汇总了本节讲述的所有** AVR **分支指令。**

表 2.11　AVR 分支指令一览表

助记符	操作数	指 令 操 作	影响标志	代码字数	时钟周期
RJMP	k	PC←PC+k+1,k=−2 K～2 K	无	1	2
IJMP		PC←Z	无	1	2
RCALL	k	PC←PC+k+1, (SP)←PC+1, SP←SP−2	无	1	3
ICALL		PC←Z,(SP)←PC+1,SP←SP−2	无	1	3
RET		SP←SP+1,PCL←(SP),SP←SP+1,PCH←(SP)	无	1	4
RETI		SP←SP+1, PCL←(SP), SP←SP+1, PCH←(SP)	I=1	1	4
CPSE	Rd,Rr	若 Rd=Rr,则 PC←PC+2(或 3),否则 PC←PC+1	无	1	1/2/3
SBRC	Rr,b	若 Rr(b)=0, 则 PC←PC+2(或 3),否则 PC←PC+1	无	1	1/2/3
SBRS	Rr,b	若 Rr(b)=1,则 PC←PC+2(或 3),否则 PC←PC+1	无	1	1/2/3
SBIC	A,b	若 I/O(A,b)=0,则 PC←PC+2(或 3),否则 PC←PC+1	无	1	1/2/3
SBIS	A,b	若 I/O(A,b)=1,则 PC←PC+2(或 3),否则 PC←PC+1	无	1	1/2/3
BRBS	s,k	若 SREG(s)=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRBC	s,k	若 SREG(s)=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BREQ	k	若 Rd=Rr(Z=1),则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRNE	k	若 Z=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRCS	k	若 C=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRCC	k	若 C=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRSH	k	若 Rd≥Rr(C=0),则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRLO	k	若 Rd<Rr(C=1),则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRMI	k	若 N=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRPL	k	若 N=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRGE	k	若 Rd≥Rr(N⊕V=0),则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2

续表 2.11

助记符	操作数	指 令 操 作	影响标志	代码字数	时钟周期
BRLT	k	若 Rd<Rr(N⊕V=1),则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRHS	k	若 H=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRHC	k	若 H=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRTS	k	若 T=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRTC	k	若 T=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRVS	k	若 V=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRVC	k	若 V=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRIE	k	若 I=1,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2
BRID	k	若 I=0,则 PC←PC+k+1,否则 PC←PC+1	无	1	1/2

2.4 位操作及其他指令

位操作指令包括位置位、位清 0、位传送和移位等指令。

2.4.1 I/O 寄存器位置位和位清 0 指令

如 1.2.5 一节所述,地址范围为 $00～$1F 的 32 个 I/O 寄存器均可按位寻址,即用 SBI 指令将其中任何一位置 1,用 CBI 指令将之清 0。这样的指令不影响任何标志。

1. I/O 寄存器位置位指令 SBI

SBI 指令将某 I/O 寄存器的指定位置成 1。详细情况请参见表 2.12。

2. I/O 寄存器位清 0 指令 CBI

CBI 指令将某 I/O 寄存器的指定位清成 0。表 2.12 给出了它的写法、操作及代码等。这两条指令的执行时间均为两个时钟周期。

表 2.12 I/O 寄存器位置位指令 SBI 和位清 0 指令 CBI

指令格式	SBI A,b	CBI A,b
指令操作	I/O(A,b)←1	I/O(A,b)←0
操作数	0≤A≤31,0≤b≤7	0≤A≤31,0≤b≤7
PC 操作	PC←PC+1	PC←PC+1
指令代码	1001 1010 AAAA Abbb	1001 1000 AAAA Abbb

[例 2.65] 由 AT90S4414 的 EEPROM 空间读一字节数据送入 R1 中,被读字节的地址位于 R0 中。

```
OUT     $1E,R0      ; 写 EEPROM 地址
SBI     $1C,0       ; 将 EECR 寄存器中的读允许位置 1
IN      R1,$1D      ; 读 EEPROM 数据
```

[例 2.66]　端口 D(PORTD 寄存器地址为 $12)最高位输出 0,可用 CBI $12,7 指令。

2.4.2　状态寄存器位置位和位清 0 指令

状态寄存器位置位和位清 0 操作各有一条基本指令负责进行。然而,对应每个标志的置位和清 0 尚各有一条专用助记符的派生指令。这样的指令只影响到所指定的标志位,对其他位无影响。

1. 状态位置位指令 BSET

BSET 指令将 SREG 寄存器中的某标志位置 1。表 2.13 汇总了 BSET 和 BCLR 指令的情况。

表 2.13　状态位置位指令 BSET 和清 0 指令 BCLR

指令格式	BSET s	BCLR s
指令操作	SREG(s)←1	SREG(s)←0
操作数	0≤s≤7	0≤s≤7
PC 操作	PC←PC+1	PC←PC+1
指令代码	1001 0100 0sss 1000	1001 0100 1sss 1000

2. 状态位清 0 指令 BCLR

BCLR 指令的功能是把状态寄存器中的某指定位清 0,详细情况见表 2.13。

[例 2.67]
```
BSET   6      ; T 位置 1
BSET   7      ; 总体开中断
```

[例 2.68]
```
BCLR   0      ; 清 C 标志
BCLR   7      ; 关中断
```

3. BSET 派生指令

(1) 进位标志 C 置位指令 SEC:这是 BSET 0 指令的另一种写法,它把进位标志 C 置成 1。表 2.14 中列出了所有这些派生指令的详细情况。

[例 2.69]
```
SEC                  ; C←1
ADC    R0,R1         ; R0←R0+R1+C
```

(2) 0 标志 Z 置 1 指令 SEZ:即 BSET 1 指令的另一种写法,它把 0 标志 Z 置成 1。

(3) 负标志 N 置 1 指令 SEN:与 BSET 2 指令等效,执行过这条指令后,N 标志被置成 1。

(4) 溢出标志 V 置 1 指令 SEV:等于 BSET 3 指令,其功能在于将溢出标志 V 置 1。

(5) 符号标志 S 置 1 指令 SES:这是 BSET 4 指令的变体,其功能是把符号标志 S 置 1。

(6) 半进位标志 H 置 1 指令 SEH:与 BSET 5 指令等效,其功能是将半进位标志置 1。

(7) T 位置 1 指令 SET:BSET 6 指令的另一种写法,其功能在于将 T 位置成 1。

(8) 总体中断开关位 I 置 1 指令 SEI:BSET 7 指令的变体,其功能是总体开放中断。

[例 2.70]
```
CLI                  ; 暂关中断
IN     R13,$16       ; 读 B 口
SEI                  ; 重开中断
```

表 2.14 状态位置 1 及清 0 派生指令

派生指令	等效指令	指令操作	PC 操作	指令代码	时钟周期
SEC	BSET 0	C←1	PC←PC+1	$9408	1
SEZ	BSET 1	Z←1	PC←PC+1	$9418	1
SEN	BSET 2	N←1	PC←PC+1	$9428	1
SEV	BSET 3	V←1	PC←PC+1	$9438	1
SES	BSET 4	S←1	PC←PC+1	$9448	1
SEH	BSET 5	H←1	PC←PC+1	$9458	1
SET	BSET 6	T←1	PC←PC+1	$9468	1
SEI	BSET 7	I←1	PC←PC+1	$9478	1
CLC	BCLR 0	C←0	PC←PC+1	$9488	1
CLZ	BCLR 1	Z←0	PC←PC+1	$9498	1
CLN	BCLR 2	N←0	PC←PC+1	$94A8	1
CLV	BCLR 3	V←0	PC←PC+1	$94B8	1
CLS	BCLR 4	S←0	PC←PC+1	$94C8	1
CLH	BCLR 5	H←0	PC←PC+1	$94D8	1
CLT	BCLR 6	T←0	PC←PC+1	$94E8	1
CLI	BCLR 7	I←0	PC←PC+1	$94F8	1

4. BCLR 派生指令

(1) 进位标志 C 清 0 指令 CLC:这是 BCLR 0 指令的一种醒目写法。从助记符的三个字母中就可以看出,此指令的功能在于将 C 标志清 0。

(2) 0 标志 Z 清 0 指令 CLZ:BCLR 1 指令的醒目形式,其功能是把 Z 标志清 0。

(3) 负标志 N 清 0 指令 CLN:和 BCLR 2 指令等效,其功能是将 N 标志清 0。

(4) 溢出标志 V 清 0 指令 CLV:BCLR 3 指令的异体形式,其功能在于将 V 标志清 0。

(5) 符号标志 S 清 0 指令 CLS:BCLR 4 指令的变体,其功能是把符号标志 S 清 0。

(6) 半进位标志 H 清 0 指令 CLH:与 BCLR 5 指令等效,它们的任务都是把 H 标志清 0。

(7) T 位清 0 指令 CLT:与 BCLR 6 指令等效,它们的任务都是把 T 位清成 0。

(8) 总体中断开放位 I 清 0 指令 CLI:即关中断指令,是 BCLR 7 指令的另一种写法。

2.4.3 位传送指令

位传送指令包括位存储和位装载两条基本指令。所谓位存储,就是把任何一个工作寄存器的 8 个位变量之一写到 SREG 寄存器的 T 位上去。而把 T 位上的位变量读到某工作寄存器的指定位去的操作称位装载。

1. 位存储指令 BST

BST 指令把 Rr 寄存器的位 b 写到状态寄存器 SREG 的 T 位上去(详见表 2.15)。

2. 位装载指令 BLD

BLD 指令把 SREG 寄存器中 T 位上的变量装载到 Rd 寄存器的位 b 上去(见表 2.15)。

[例 2.71]　把 R1 的位 2 复制到 R0 的位 4 上去。

因为没有工作寄存器间的位传送指令,所以必须通过 T 位中转。

```
BST     R1,2      ; T←R1(2)
BLD     R0,4      ; R0(4)←T
```

表 2.15　位存储指令 BST 和位装载指令 BLD

指令格式	BST Rr,b	BLD Rd,b
指令操作	T←Rr(b)	Rd(b)←T
操作数	0≤r≤31, 0≤b≤7	0≤d≤31,0≤b≤7
PC 操作	PC←PC+1	PC←PC+1
指令代码	1111 101r rrrr 0bbb	1111 100d dddd 0bbb

2.4.4　移位指令

移位操作分开环移位和闭环移位两种情况,闭环移位称作旋转。移位和旋转的主体都是工作寄存器 Rd。

1. 逻辑左移指令 LSL

实际上这是 ADD Rd,Rd 指令的另外一种表达方式,其任务是把 Rd 寄存器的所有位均依次向左移动一位,原最高位 b7 进入 C 标志位中,最低位 b0 处由外面补充一个 0 进来。这等于原 Rd 值乘以 2。表 2.16 汇总了移位指令的详细情况。

[例 2.72]
```
ADD     R0,R4     ; R0←R0+R4
LSL     R0        ; R0←2 * R0
```

2. 逻辑右移指令 LSR

此指令把 Rd 寄存器的内容依次向右移动一位,原最低位 b0 移出寄存器,进入 C 标志,最高位 b7 处从外面补充进一个 0 来。右移操作相当于把某个无符号 8 位数除以 2。

[例 2.73]
```
ADD     R0,R4     ; R0←R0+R4
LSR     R0        ; R0←R0/2
```

3. 算术右移指令 ASR

ASR 指令把 Rd 寄存器的内容依次右移一位,原最高位 b7 除移至次高位 b6 外,本身尚保持不变,原最低位 b0 移出 Rd 进入 C 标志中。算术右移操作等于把某个带符号的 8 位二进制数除以 2。详细情况请参见表 2.16。

[例 2.74]
```
LDI     R16,$10   ; R16←16
ASR     R16       ; R16←R16/2
LDI     R17,$FC   ; R17←-4
ASR     R17       ; R17←R17/2
```

4. 左旋指令 ROL

实际上这是 ADC Rd,Rd 指令的另一种写法。ROL 指令把 Rd 和 C 作为一个整体向左旋

转一位，C 之原内容进入 Rd 作为其最低位 b0，而 Rd 之原最高位 b7 移出寄存器进入 C 标志。此指令与 LSL 配合，可进行多字节带符号数或无符号数的乘 2 运算。

[例 2.75] 把寄存器对 R19:R18 的内容乘以 2，可用下列几条指令完成。

```
        LSL     R18       ; R18 逻辑左移
        ROL     R19       ; R19 与 C 一起左旋
        BRCS    ONEC      ; 若有进位则跳变
        ……
ONEC:   ……                ; 跳转目的地址
```

表 2.16 AVR 移位指令

指令格式	LSL Rd	LSR Rd	ASR Rd
指令操作	Rd(n+1)←Rd(n) Rd(0)←0,C←Rd(7)	Rd(n)←Rd(n+1) Rd(7)←0,C←Rd(0)	Rd(n)←Rd(n+1) C←Rd(0),Rd(7)←Rd(7)
操作数	0≤d≤31, 0≤n≤6	0≤d≤31,0≤n≤6	0≤d≤31,0≤n≤6
PC 操作	PC←PC+1	PC←PC+1	PC←PC+1
影响标志	H,S,V(N⊕C),N,Z,C	S,V(N⊕C),N=0,Z,C	S,V(N⊕C),N,Z,C
指令代码	0000 11dd dddd dddd	1001 010d dddd 0110	1001 010d dddd 0101

5. 右旋指令 ROR

此指令把 Rd 和 C 作为一个整体向右循环移动一位，Rd 原最低位 b0 移至 C 标志中，而 C 之原内容则移进 Rd 作其最高位 b7。此指令与 ASR 配合，可对带符号多字节数进行除以 2 运算；与 LSR 配合，可对无符号多字节数进行除以 2 运算。

[例 2.76] R19:R18 寄存器对的内容为无符号双字节整数，而 R17:R16 中则是带符号双字节整数，两数均除以 2，所得之商仍放在原寄存器对中。

```
LSR     R19     ; 无符号数多字节除法
ROR     R18     ; R19:R18←R19:R18/2
……
ASR     R17     ; 有符号数多字节除法
ROR     R16     ; R17:R16←R17:R16/2
```

6. 半字节交换指令 SWAP

SWAP 指令将工作寄存器 Rd 的高 4 位和低 4 位进行交换。此指令不影响标志位。

[例 2.77] R16 寄存器中有一个两位 BCD 数，将其两位分离开，十位数放入 R17 中，个位数留在 R16 寄存器中。

```
MOV     R17,R16     ; R17←R16
ANDI    R16,$0F     ; R16 的 BCD 十位数清 0
SWAP    R17         ; 个位数与十位数交换位置
ANDI    R17,$0F     ; 个位数清 0,保留十位数
```

表 2.17 中详细列出了上述三条旋转指令的操作、代码以及对标志的影响等项内容。

表 2.17　AVR 字节循环移位指令

指令格式	ROL Rd	ROR Rd	SWAP Rd
指令操作	Rd(n+1)←Rd(n) Rd(0)←C,C←Rd(7)	Rd(n)←Rd(n+1) Rd(7)←C,C←Rd(0)	Rd(7～4)↔Rd(3～0)
操作数	0≤d≤31,0≤n≤6	0≤d≤31,0≤n≤6	0≤d≤31
PC 操作	PC←PC+1	PC←PC+1	PC←PC+1
影响标志	H,S,V(N⊕C),N,Z,C	S,V(N⊕C),N,Z,C	无
指令代码	0001 11dd dddd dddd	1001 010d dddd 0111	1001 010d dddd 0010

2.4.5　其他指令

1. 空操作指令 NOP

NOP 指令什么事情都不做，代码为 $0000，执行时间等于一个时钟周期，不影响任何标志。通常用来得到一个时钟周期的延时。

[例 2.78]
```
CLR     R16          ; R16 清 0
SER     R17          ; R17← $FF
OUT     $18,R16      ; 向 B 口写 $00
NOP                  ; 延时一个周期
OUT     $18,R17      ; 向 B 口写 $FF
```

2. 休眠指令 SLEEP

SLEEP 指令使 AVR 微控制器进入 MCU 控制寄存器所定义的休眠方式。不影响任何标志，指令执行时间为 3 个时钟周期。指令代码等于 $9588。

[例 2.79]
```
MOV     R0,R11       ; R0←R11
LDI     R16,$20      ; 拟将 SE 位置 1
OUT     MCUCR,R16    ; SE=1,SM=0
SLEEP                ; MCU 进入闲置方式
```

3. 看门狗复位指令 WDR

此指令使监视定时器复位并重新从 0 开始递增计数。在正常情况下，WDR 指令必须在看门狗定时器所给的时限内执行一次，以使 WDT 不致溢出。指令代码为 $95A8。WDR 指令不影响任何标志。

最后应当指出，2.4 一节所讲述的各条指令，除另有说明者外，均为单周期指令。

第三章　汇编语言程序设计示范

学过 AVR 内部硬件资源及指令系统后，读者自然就要尝试进行程序设计了。随后再把他所设计的汇编语言源程序通过汇编软件译成机器语言的目的程序，经调试、修改、定稿并最后写入 Flash 存储器。本书今后几章将沿着这条路线为读者提供帮助。

3.1　AVR 程序的书写规则

用汇编语言编写程序必须按汇编软件所限定的规则进行，否则将不能顺利地通过汇编来得到机器码的目的程序。我们所编写的源程序由指令助记符、标号及汇编指令等组成。每行不得超过 120 个字符。

源程序每行前面都可能有一个标号，它由字符数字串构成，以冒号作结尾。标号通常用来代表程序存储器中跳转或分支指令的目的地址，或 RAM 中的变量名称。

3.1.1　汇编指令

汇编指令，顾名思义，是汇编软件所使用的指令，亦称伪指令。意思是说，它们不属于任何 CPU 指令系统，因此在汇编时不生成机器码。这些伪指令被用来分配指令代码在存储器中的单元地址、定义宏指令、对某些存储单元写入常数值等等。

AVR 编辑汇编软件 WAVRASM 所使用的汇编指令如表 3.1 所列。下面我们将逐条详述。

表 3.1　WAVRASM 汇编指令一览表

汇编指令	功　能	汇编指令	功　能
BYTE	给变量预留字节存储单元	ESEG	EEPROM 段
CSEG	程序代码段	EXIT	文件出口
DB	定义常数字节	INCLUDE	援引其他文件
DEF	为工作寄存器定义符号名称	LIST	启用列表文件生成功能
DEVICE	定义为何器件进行汇编	LISTMAC	列出宏扩展
DSEG	数据段	MACRO	宏指令开始
DW	定义常数字	NOLIST	关闭列表文件生成功能
ENDMACRO	宏指令结束	ORG	设置程序起始地址
EQU	为某符号设定一等值表达式	SET	为某符号设定一等值表达式

1. BYTE——为变量预留字节存储单元

此语句前通常要有既当符号地址又作变量名称的标号，其后必须跟着参数，以指明要预留多少字节。BYTE 伪指令只能用在数据区段，其功能是告诉汇编软件在 SRAM 区段中，从标

号所代表的地址处开始，为以此标号作名称的变量保留本语句参数所指定数量的字节单元。请注意，它只要求预留位置，而在预留地址单元中不写入具体数据。

［例 3.1］　演示 BYTE 语句的用法。

```
.DSEG                          ；数据段
    var1：.byte   1            ；为 var1 预留一个字节
    table：.byte   tab_size    ；为 table 预留 tab_size 个字节
```

2. CSEG——代码段

用来宣告以下是代码段，告诉汇编软件，此后的常数或指令代码将要存入 Flash 程序存储器中。一个汇编文件可能含有若干个代码段，但在汇编过程中它们将被连成一个整体。若无任何段的宣告，则汇编软件将本能地认为这是代码段。代码段拥有本身的字地址计数器，负责按顺序安排本段内指令代码及常数的字地址。CSEG 语句不带任何参数。应当强调指出，BYTE 语句不可用于代码段。

［例 3.2］　演示 CSEG 语句的用法。

```
.DSEG                          ；以下是数据段
    vartab：.BYTE   4          ；SRAM 中预留 4 个字节
.CSEG                          ；以下是代码段
    const：.DW      2          ；在 Flash 中写入常数 $0002
    mov     r1,r0              ；指令代码
```

3. DB——在程序存储器或 EEPROM 中定义常数字节

DB 语句前面一般带有标号，后面必须跟着参数，只能用在代码段或 EEPROM 段内。其功能是告诉汇编软件，在程序存储器或 EEPROM 段中，该语句前之标号所代表的地址处，将写入此语句后之参数所指定的常数。DB 语句后的参数可能是以逗号分隔的多个表达式，但每个表达式值均必须在 8 位带符号或无符号二进制数所能表达的范围内。

若多参数的 DB 语句用在代码段，则在汇编过程中，每两个参数组成一个字，并分配给一个字地址。如果这样组合之后，尚余一个 8 位参数，那么它将以字的形式被放在程序存储器前一字地址之后。即使下一行语句也含一个 DB 伪指令，前后行也不合并。

［例 3.3］　演示 DB 语句的用法。

```
. CSEG
  const：.DB   0，255，0b01011010，－128，0xaa
.ESEG
 eeconst：.DB   $ff
```

［例 3.4］　在程序存储器中，自 $050 开始的 4 个连续字地址内，依次存放 $063F、$4F5B、$6D66 和 $077D 等 4 个字常数。请读者注意 DB 语句后面参数的顺序。

```
.EQU     DATAB= $050
.CSEG
.ORG     DATAB
.DB       $3F，$06，$5B，$4F，$66，$6D，$7D，$07
```

在阅读这个例子时，读者也许会感到困惑。题义要求 $050 单元存放数据 $063F，但在

DB 语句后为什么先 $3F，而后 $06 呢？

现在我们用 LPM 指令来读 $050 单元的常数。由图 1.13 得知，LPM 指令用 Z 寄存器的高 15 位来指定程序存储器的某地址单元，而以其最低位来选择该存储字的两个字节。在本例中，Z 寄存器的高 15 位应当等于 $050。如果将 $050 乘以 2 后（得 $0A0）再送往 Z 寄存器中，那么 Z 之高 15 位值为 $050，而最低位为 0。这将读到 $050 单元的低字节，即 $3F。下次 Z 的内容加 1 后再读，读之所得将是 $06。这样的顺序和 DB 后的参数是完全一致的。

但是，还应再提醒一句，DB 语句用在 EEPROM 段中，不会出现上述情况，因为这里每个地址单元中只存放一个字节常数。

4. DEF——为工作寄存器定义一符号名称

DEF 语句用来为工作寄存器取符号名称。已定义过的符号可以写在此后的程序段中，特指它所代表的工作寄存器。一个寄存器允许有若干个符号名称。符号可在此后的程序中重新定义。

［例 3.5］ 举例说明 DEF 伪指令的用法。

```
.DEF    temp=r16
.DEF    ior=r0
.CSEG
        ldi   temp, 0xf0    ; r16←$f0
        in    ior, 0x3f     ; 将 SREG 内容读入 r0
        eor   temp, ior     ; r16←r16⊕r0
```

5. DEVICE——定义为何器件进行汇编

DEVICE 语句后面要写出 AT90 系列 MCU 的具体型号，以告诉汇编软件应针对这个器件的指令系统及硬件资源进行汇编。使用此语句后，如果遇到指定器件所不支持的指令，那么汇编软件将发出警告。若代码段或 EEPROM 段的规模超出所指定 MCU 的支持范围，则也将有警告发出。在未用 DEVICE 语句的情况下，就认为所有指令都支持，且无存储器规模的限制。

［例 3.6］

```
.DEVICE   AT90S1200     ; 使用 AT90S1200
.CSEG                   ; 代码段
    PUSH   R30          ; 此指令将受到警告，因为 AT90S1200 无此指令
```

6. DSEG——数据段

DSEG 宣告这里是数据段，即告知汇编软件此处是 SRAM 空间。一汇编文件可能会含有若干数据段，不过它们在汇编过程中将被连成一个整体。数据段有自己的字节地址计数器，负责分配本段中 BYTE 语句要求预留字节的单元地址。BYTE 语句只能用在数据段。

关于 DSEG 的具体应用例子请参见例 3.1。

7. DW——在程序存储器或 EEPROM 中定义常数字

DW 语句前通常有标号作为符号地址，其后必须跟着参数，或为表达式，或为具体数据。该参数便是行将写入 Flash 或 EEPROM 中去的常数。DW 后也可跟有以逗号分隔的若干个参数，但每个参数必须在 16 位带符号或无符号二进制数所能表达的范围之内。

［例 3.7］

```
.CSEG
const:.DW   0,0xffff, -32 768, 65 535
```

```
.ESEG
    eeconst: .DW    $ffff
```

[例 3.8]　用 DW 语句改写例 3.4。

```
.CSEG
.ORG    $050
.DW     $063F,$4F5B,$6D66,$077D
```

8. EQU——设定某符号等于一表达式

EQU 意即“等于”,用来设定某符号等于一表达式或一具体数据。该符号经 EQU 设定后便一成不变,不可事后更改。

[例 3.9]

```
.EQU    porta=$1b       ; porta I/O 地址
.EQU    portb=porta+3   ; portb I/O 地址
.CSEG                   ; 代码段
    clr   r2            ; R2 清 0
    out   portb,r2      ; 向 portb 写入数据
```

9. ESEG——EEPROM 段

ESEG 语句用来宣告以下是 EEPROM 地址空间。若某汇编文件含有若干 EEPROM 段,则汇编时它们将被连成一个统一的空间。EEPROM 段也有一个字节地址计数器,负责分配本段各伪指令所涉及到的数据应占有之字节地址。

应当指出,CSEG、DSEG 和 ESEG 等三个存储器段别宣告语句之后均不带参数。

[例 3.10]　演示各段别宣告语句的用法。

```
.DSEG                       ; 数据段
    vartab: .BYTE   4       ; SRAM 中预留 4 个字节
.ESEG                       ; EEPROM 段
    eeconst: .DW    $ff0f   ; 写入两字节初始值
.CSEG                       ; 代码段
    const:  .DW     2       ; 写入常数 $0002
            mov     r1,r2
```

10. EXIT——文件出口

EXIT 语句告诉汇编软件停止对该语句所属文件的汇编。通常汇编软件一直运行到文件的末尾。如果 EXIT 语句出现在被援引的文件中,那么汇编软件将从主文件中用来援引其他文件的 INCLUDE 语句后面一行继续运行。

11. INCLUDE——援引其他文件

INCLUDE 语句告诉汇编软件开始读取此语句所指定的文件。随之汇编软件便对该文件进行汇编,直到文件结束或遇到 EXIT 语句为止。实际上这是把其他文件借用到主文件中来的一个命令。被援引的文件本身也可能作为主文件以 INCLUDE 语句去借用其他文件。这就是所谓被援引文件的嵌套。

本语句后面要有带双引号的行将被借用的文件名称。

[例 3.11]

```
.EQU    sreg=0x3f
```

```
.EQU   sph=0x3e
.EQU   spl=0x3d
```

把上述三个 EQU 语句以 IODEFS.ASM 为文件名建立文件后，便可使用下面的伪指令。

```
.INCLUDE    "iodefs.asm"      ;借用 iodefs.asm 文件
    in      r0,sreg           ; 读状态寄存器
```

12. MACRO——宏指令开始

AVR 汇编软件具有很强的宏汇编能力，这使程序设计人员可用若干条基本 AVR 指令构成一条虚拟的宏指令。每条宏指令最多可有十个参数，在宏指令定义中，这些参数被标作@0至@9。定义过的宏指令在随后的程序中可随时使用（术语称调用），宛如一条 AVR 指令集中的合法指令。调用宏指令时，其参数应顺序列出，参数间以逗号分隔。汇编时，这条宏指令被就地扩展成定义时所使用的那些 AVR 基本指令。汇编后形成的清单中，这些宏扩展可列出，亦可免于列出。

MACRO 语句告诉汇编软件这是某宏定义的开始。MACRO 语句以宏指令名称为其参数。宏定义用 ENDMACRO 语句作其结尾。

[例 3.12] 定义一条宏指令 SUBI16，令其做寄存器对与一 16 位立即数的减法运算。

```
.MACRO     SUBI16                ; 宏定义开始
    subi   @1, low(@0)           ; 低字节相减
    sbci   @2, high(@0)          ; 高字节相减
.ENDMACRO                        ; 宏定义结束
.CSEG                            ; 代码段
    SUBI16 0x1234, r16, r17      ; r17:r16←r17:r16－$1234
```

这里宏调用中的参数@0 等于 0x1234，参数@1 和@2 则分别为 R16 和 R17。

13. ENDMACRO——宏定义结束

本语句用来宣告宏定义到此结束，其后无任何参数。用例请参见例 3.12。

14. LISTMAC——清单中列出宏指令扩展

此语句用来告知汇编软件，在汇编后所生成的列表文件中，应列出调用宏指令时的宏扩展内容。若无此语句，则在列表文件中仅列出宏调用及其参数，而无宏扩展部分的详细内容。

在汇编后的清单中，宏调用指令在操作码域以“＋”号予以标识，详见例 3.14。

15. LIST——启用列表文件生成功能

LIST 语句告诉汇编软件启用列表文件生成功能。汇编软件对某源程序进行汇编后要产生一列表文件，其中包括汇编语言源语句、地址及相应的机器码。即使无 LIST 语句宣告，也要有这样的清单产生。该语句的功能主要是与 NOLIST 语句配合使用，以便有选择地列出汇编源文件中的某些部分。

16. NOLIST——关闭列表生成功能

如前所述，若无任何宣告，则汇编后要产生一包括汇编源语句、地址及相应机器码的列表文件。NOLIST 语句可用来禁止生成这样的列表文件。

[例 3.13] 主程序中援引了两个其他文件，但汇编后无须在清单中列出这些文件的内容。

```
.NOLIST                    ；以下内容无须列表
.INCLUDE  "macro.inc"      ；此二被援引文件
.INCLUDE  "const.def"      ；无须列入清单
.LIST                      ；以下部分恢复列表
```

应当指出，在程序清单中，宏扩展部分列出与否，取决于宏调用之前有无 LISTMAC **语句，这不属于** LIST **语句的管辖范围。**

［例 3.14］　举例说明 NOLIST、LIST **和** LISTMAC **语句的用法。**

```
.NOLIST
.INCLUDE    "8515def.inc"          ；此被援引文件在汇编后的清单中无须列出
.LIST                              ；以下在清单中必须列出
    RJMP    RESET
.MACRO      SUBI16                 ；宏定义
    SUBI    @1, LOW(@0)
    SBCI    @2, HIGH(@0)
.ENDMACRO                          ；宏定义结束
.CSEG                              ；程序代码段
.ORG        0x10                   ；从$010地址开始
RESET:
.LISTMAC
    SUBI16  0x1234, R16,R17        ；宏调用
    OUT     DDRA,R16
    OUT     PORTA,R17
    SUBI16  0x5A4D, R26,R30        ；宏调用
    IN      R26,PINA
    OUT     PORTA,R30
```

上段程序经汇编后生成如下清单，其最左列为地址及机器码。

```
        .NOLIST
000000 c00f  RJMP    RESET
        .MACRO     SUBI16                    ；宏定义
                   SUBI   @1, LOW(@0)
                   SBCI   @2, HIGH(@0)
        .ENDMACRO                            ；宏定义结束
        .CSEG                                ；程序代码段
        .ORG       0x10                      ；从$010地址开始
        RESET:
        .LISTMAC
000010  +          SUBI16  0x1234, R16, R17  ；宏调用
000010  5304       SUBI   R16, LOW(0x1234)
000011  4112       SBCI   R17, HIGH(0x1234)
        .ENDMACRO
000012  bb0a       OUT    DDRA, R16
```

```
000013  bb1b          OUT   PORTA, R17
000014  +             SUBI16  0x5A4D, R26, R30   ; 宏调用
000014  54ad          SUBI   R26, LOW(0x5A4D)
000015  45ea          SBCI   R30, HIGH(0x5A4D)
        .ENDMACRO                                ; 宏定义结束
000016  b3a9          IN     R26, PINA
000017  bbeb          OUT    PORTA,R30
Assembly complete with no errors.
```

17. ORG——设置程序起始地址

ORG 语句用来为地址计数器设置初始值。此设置值以 ORG 参数的形式给出。若 ORG 语句用在数据段中,则被设置初值的是 SRAM 的地址计数器;如果该语句用在代码段,那么被设置初值者为程序存储器地址计数器;若用在 EEPROM 段,则系 EEPROM 地址计数器被置初值。汇编开始时,程序存储器和 EEPROM 地址计数器的默认值为 0,但 SRAM 地址计数器的默认值则是 32,因为工作寄存器占据了地址 0～31。应当指出,EEPROM 和 SRAM 的地址计数器按字节计数,而程序存储器的地址计数器则按字计数。

[例 3.15] 演示 ORG 伪指令的用法。

```
.DSEG                          ; 数据段
.ORG    0x67                   ; SRAM 地址 $67
    variable: .BYTE  1         ; 在此地址处预留一字节
.ESEG                          ; EEPROM 段
.ORG    0x20                   ; EEPROM 地址计数器置初值
    eevar:   .DW  0xfeff       ; 此处写入一个字数据
.CSEG                          ; 程序代码段
.ORG    0x10                   ; 程序地址计数器初值
        mov r0,r1
```

18. SET——为某符号设定一等值表达式

SET 语句用来为标号赋值。这样的标号可以用在此后的表达式中。已被 SET 语句赋过值的标号此后在程序中可以改变,不像 EQU 赋值语句那样一成不变。

[例 3.16]

```
.SET    io_offset=0x23
.SET    porta=io_offset+2
.CSEG
        clr    r2              ; 清 r2
        out    porta, r2       ; A 口输出
```

3.1.2 表达式

汇编软件把对表达式的处理也纳入了自己的业务范围。表达式可由操作数、运算符及函数组成。所有表达式值在内部都是 32 位的。

1. 操作数

在 AVR 的汇编文件中,可以使用下列操作数:

- 用户定义的标号,即符号地址,它等于在该标号出现处地址计数器的计数值。

- 用户用 SET 语句定义的变量。
- 用户用 EQU 语句定义的常数。
- 整常数：常数可用若干种格式给出，其中包括：

(1) 十进制形式(无特殊标识)，如 10，255；

(2) 十六进制形式(两种表示法)，如 0x0A，$0a，0xff，$ff；

(3) 二进制形式，如 0b00001010，0b11111111。

- PC——程序存储地址计数器的现行值。

2. 函　数

AVR 汇编软件定义了下列函数：

- LOW(表达式)表示要某表达式的低字节。
- HIGH(表达式)表示要某表达式的高字节。
- BYTE2(表达式)同 HIGH 函数。
- BYTE3(表达式)表示要某表达式的第三个字节，即位 16～23。
- BYTE4(表达式)表示要某表达式的第四个字节，即位 24～位 31。
- LWRD(表达式)表示要某表达式的低 16 位，即位 0～位 15。
- HWRD(表达式)表示要某表达式的高 16 位，即位 16～位 31。
- PAGE(表达式)表示要某表达式的页码值，即位 16～位 21。
- EXP2(表达式)函数的返回值为 2^n，n 为表达式值。
- LOG2(表达式)函数的返回值为 $\log_2$(表达式)，即表达式值以 2 为底的对数之整数部分。

3. 运算符

AVR 汇编软件支持下列运算符，它们都具有一定的优先级别，表 3.2 中优先级别按递减顺序排列，同级者不分先后。表达式可能位于括号内，这样的表达式必须先得到具体数值后方可与括号外的任何符号相结合。

(1) !(逻辑非)：一元运算符，若其后之表达式值为 0，则返回值等于 1。若表达式值非 0，则返回 0。

[例 3.17]　ldi　r16，!0xf0　；向 r16 中装入数值 0x00

(2) ～(按位取反)：一元运算符，它所返回的是各位均取其反码的输入表达式。

[例 3.18]　ldi　r16，～0xf0　；r16 赋初值 0x0f

(3) －(负号)：一元运算符，它所返回的是某表达式值对 0 的补数。

[例 3.19]　ldi　r16，－2　；r16 赋初值－2，即 $fe

(4) *(乘号)：二元运算符，其返回值为两表达式之积。

[例 3.20]　ldi　r30，label*2　；r30 赋初值 label*2

请注意，若乘积超过 8 位数值，则仅取其低 8 位值。

(5) /(除号)：二元运算符，其返回值为左表达式除以右表达式所得之整数商。

[例 3.21]　ldi　r30，label/2　；r30 赋初值 label/2

(6) ＋(加号)：二元运算符，其返回值为两表达式之和。

[例 3.22]　ldi　r30，c1＋c2　；r30←c1＋c2

(7) －(减号)：二元运算符，其返回值等于左表达式减右表达式所得之差。

[例 3.23]　ldi　r17，c1－c2　；r17←c1－c2

(8) ≪(左移):二元运算符,它使左表达式左移 n 位,n 为右表达式值。

[例 3.24] 将 1 左移 bitmask 位后装入 r17 中。若 bitmask=2,则 1≪bitmask 将 0000 0001 左移 2 位,得 $04。

```
ldi   r17, 1≪bitmask   ; r17←$04
```

(9) ≫(右移):二元运算符,它使左表达式右移 n 位,n 为右表达式值。

[例 3.25] ldi r17, c1≫c2 ; c1 右移 c2 位后装入 r17

(10) <(小于号):二元运算符,若左边带符号的表达式值真的小于右边带符号的表达式值,则返回 1,否则返回 0。

[例 3.26] 若 c1<c2,则 r18 将和 bitmask+1 相或,否则只和 $01 相或。下面一条指令便实现了这个要求。

```
ori   r18, bitmask*(c1<c2)+1
```

(11) <=(小于或等于号):二元运算符,若左边的带符号表达式真的小于或等于右边带符号的表达式,则返回 1,否则返回值为 0。

[例 3.27] ori r18, bitmask*(c1<=c2)+1

若条件为真,则 r18 将和 bitmask+1 相或;如果条件为假,那么 r18 将只和 $01 相或。

(12) >(大于号):二元运算符,若左边的带符号表达式值真的大于右边的带符号表达式值,则返回 1,否则返回 0。

[例 3.28] ori r18, bitmask*(c1>c2)+1

(13) >=(大于或等于号):二元运算符,若左边的带符号表达式值真的大于或等于右边的带符号表达式值,则返回 1,否则返回 0。

[例 3.29] ori r18, bitmask*(c1>=c2)+1

(14) ==(等号):二元运算符,若其两侧之带符号表达式值相等,则返回 1,否则返回值为 0。

[例 3.30] andi r19, bitmask*(c1==c2)+1

这条指令的意思是,若 c1 和 c2 两表达式不相等,则 r19 将和 $01 相与;若两表达式相等,则 r19 将和 bitmask+1 相与。

(15) !=(不等号):二元运算符,若其两侧之带符号表达式值不相等,则将返回 1,否则返回值为 0。

[例 3.31] .SET flag=(c1!=c2) ; 将 flag 置成 1 或 0。

(16) &(按位相与):二元运算符,其返回值为两表达式按位相与之结果。

[例 3.32] 令 c1 和 c2 两表达式按位相与,并将相与结果的第二个字节送入 r18 中。

```
ldi   r18, high(c1 & c2)
```

(17) ^(按位异或):二元运算符,其返回值等于两表达式按位异或所产生的结果。

[例 3.33] 令 c1 和 c2 两表达式按位进行异或运算,将所得结果的最低字节装入 r18 寄存器中。

```
ldi   r18, low(c1^c2)
```

(18) |(按位相或):二元运算符,其返回值等于两表达式按位相或所产生结果。

[例 3.34]　ldi　r18, low(c1|c2)

这条指令把 c1 和 c2 两表达式按位相或所得结果之最低 8 位装入 r18 寄存器中。若 c1=$567a,c2=$67ad,则 c1|c2=$77ff。故此指令为 r18 赋值 $ff。

(19) &&(逻辑与):二元运算符,若两表达式均不为 0,则该运算的返回值为 1,否则其返回值将等于 0。

[例 3.35]　low(c1 && c2)可以理解成 low(c1)&&low(c2)。若 c1=$00aa, c2=$9955,则两值之低字节均不为 0,故 low(c1 && c2)=1。因 c1 之高字节为 $00,故 high(c1 && c2)=0。这样,下面两条指令执行后,r18=$01,r19=$00。

```
ldi   r18, low(c1 && c2)
ldi   r19, high(c1 && c2)
```

(20) ‖(逻辑或):二元运算符,若两表达式中一个不为 0,或两者均不为 0,则该运算之返回值为 1,否则其返回值便等于 0。

[例 3.36]
```
ldi   r18, low(c1 ‖ c2)   ; r18←$01,c1 和 c2 值同上
ldi   r19, high(c1 ‖ c2)  ; r19←$01
```

表 3.2 中汇总了 WAVRASM 汇编软件所支持的全部运算符。

表 3.2　AVR 编辑汇编软件所支持的运算符一览表

运算符	名　称	优先级	功　　能
!	逻辑非	14	一元符,表达式值为 0,返回 1,表达式值非 0,则返回 0
~	按位取反	14	一元符,将表达式按位取反码,0 变 1,1 变 0
−	负　号	14	一元符,返回值等于表达式对 0 的补数
*	乘　号	13	二元符,返回值等于两表达式值之乘积
/	除　号	13	二元符,返回值等于左表达式除以右表达式之整数商
+	加　号	12	二元符,返回值等于两表达式值之和
−	减　号	12	二元符,返回左表达式减右表达式所得之差
≪	左　移	11	二元符,使左表达式左移 n 位,n 为右表达式值
≫	右　移	11	二元符,使左表达式右移 n 位,n 为右表达式值
<	小　于	10	二元符,左右两带符号表达式中,若左小右大,则返回 1,否则为 0
<=	小于或等于	10	二元符,其两侧带符号表达式若左小于或等于右,则返回 1,否则返回 0
>	大　于	10	二元符,其两侧带符号表达式若左大右小,则返回 1,否则返回 0
>=	大于或等于	10	二元符,其左侧带符号表达式大于或等于右侧,则返回 1,否则返回 0
==	等　号	9	二元符,若左右两侧表达式值相等,则返回 1,否则返回 0
!=	不等号	9	二元符,若左右两侧表达式值不等,则返回 1,否则返回 0
&	按位与	8	二元符,返回值等于两表达式按位相与之结果
^	按位异或	7	二元符,返回值等于两表达式按位异或之结果
|	按位或	6	二元符,返回值等于两表达式按位相或之结果
&&	逻辑与	5	二元符,两表达式均不为 0,返回 1,否则返回 0
‖	逻辑或	4	二元符,两表达式中至少一个不为 0,则返回 1,否则返回 0

3.2 BCD 运算

大家知道,计算机在其内部是以二进制进行运算的。欲令其进行十进制运算,必须在软件方面采取措施。有些 CPU 实现这一点比较容易,因为它们都有 DAA 指令,可用来对二进制运算结果进行十进制调整。但 AVR MCU 则因 DAA 指令使用频率不是太高而未将其纳入指令系统。在研究过二进制和二-十进制(BCD)的互相转换之后,本节尚将介绍在 BCD 的加减运算中如何用软件来实现 DAA 指令的功能。

3.2.1 数制转换

这里主要讲述无符号二进制形式和 BCD 形式间的相互转换程序。

1. 8 位无符号二进制数据转换成 3 位 BCD 形式

若有除法指令,则这样的任务很容易完成——被转换数除以 100 所得之商便为 BCD 形式的百位数;余数除以 10 得十位数;此时的余数则为个位数。

AVR MCU 无除法指令,但可按上述思路进行这样的转换——被转换数与 100 比较,若大于 100,则百位数加 1 的同时,应从被比较数中扣除 100。这样一直比较到小于 100 后再与 10 比较,以同样的方法得十位数与个位数。

[例 3.37] 把某工作寄存器中的 8 位无符号二进制数转换成相应的 3 位 BCD 形式,百位数存入 HUND 寄存器中,十位和个位存入 TENONE 寄存器中。

```
        .DEF    BIN=R16          ; 待转换二进制数
        .DEF    TENONE=R17       ; BCD 结果十位与个位数
        .DEF    HUND=R18         ; BCD 结果百位数
B8BCD:  CLR     HUND             ; 转换结果初值为 0
        CLR     TENONE
LOOP:   CPI     BIN,100          ; 待转换数与 100 比较
        BRCS    LESS             ; 小于则跳转
        INC     HUND             ; 不小于,百位数加 1
        SUBI    BIN,100          ; 被转换数减 100
        RJMP    LOOP             ; 循环,再比较
LESS:   CPI     BIN,10           ; 小于 100,与 10 比较
        BRCS    LESS10           ; 小于 10,跳转至 LESS10
        SUBI    TENONE,-$10      ; 不小于,十位数加 1
        SUBI    BIN,10           ; 被转换数减 10
        RJMP    LESS             ; 循环,再比较
LESS10: ADD     TENONE,BIN       ; 十位数加个位数
        RET
```

应当指出,AVR 指令系统中没有寄存器与 8 位立即数相加的指令,而只有这样的减法指令。故+$10 只能通过-(-$10)来实现。

2. 无符号 16 位二进制数据转换成 5 位 BCD 形式

为简化问题,我们首先研究一下把 8 位二进制数转换成 3 位 BCD 形式的另一算法。如图

3.1 所示，三个寄存器串联起来，右面 BIN 寄存器中存有待转换的 8 位二进制数，而左面两寄存器预先被清 0，最后用来保留 BCD 结果。这种转换算法的主要思想概括起来有下面几点：

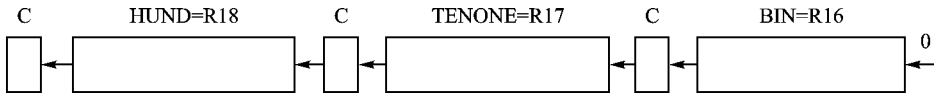


图 3.1　8 位二进制数转换成 3 位 BCD 形式算法之一

(1) BIN 寄存器中的待转换 8 位二进制数左移 8 次后，全部进入 TENONE 寄存器，这等于该数乘了 2^8。最后把 TENONE 寄存器视作转换结果的最低两 BCD 位，这又等于进行了除以 2^8 的运算。

(2) 每次左移之后，都对 HUND 和 TENONE 两寄存器中的 4 个 BCD 位进行加 3 预调整。调整后大于或等于 8 的各位保留调整结果；而小于 8 者无须调整，减 3 恢复原值。

(3) 这样 8 次左移后，在 HUND 寄存器中得 BCD 结果之百位数，而 TENONE 寄存器中则为其十位和个位数。

按 BCD 运算规则，每个 BCD 位若大于或等于 0101，则当其乘 2 时即应向上位进位。但机器按二进制运算，只有在半字节≥1000 的情况下，乘 2 时才向上半字节进位。这样，若某 BCD 位≥0101，则加 3 调整后，它≥1000，保证左移时能向上位进位。用十进制的术语讲，就是若某位≥5，则乘 2 时应向其高位进位。但是判断各 BCD 位是否≥0101 并非易事。所以我们在程序中应用了间接判断法，即加 3 后若该 BCD 位≥1000，即说明预调整前曾经≥0101，这时预调整算数，否则应减 3 恢复原值。

[例 3.38]　把 BINH:BINL 寄存器对中的 16 位无符号二进制数转换成 5 位 BCD 形式，其万位存放在 BCD2，千位百位和十位个位分别存放在 BCD1 和 BCD0 两个寄存器中。根据前述三点思想，这段子程序可设计成如下形式。

```
.DEF    BCD0=R13            ; BCD 结果十位和个位
.DEF    BCD1=R14            ; BCD 结果千位和百位
.DEF    BCD2=R15            ; BCD 结果万位
.DEF    BINL=R16            ; 待转换二进制数低字节
.DEF    BINH=R17            ; 待转换二进制数高字节
.DEF    CNT=R18             ; 循环计数器
.DEF    TEMP=R19            ; 暂存器
.EQU    ADBCD2=15           ; BCD2 地址
B16BCD: LDI   CNT,16        ; 循环计数初值
        CLR   BCD2          ; BCD 结果初值清 0
        CLR   BCD1
        CLR   BCD0
        CLR   ZH            ; 地址指针高字节为 0
SHIFT:  LSL   BINL          ; 16 位二进制数整体左移
        ROL   BINH
        ROL   BCD0
        ROL   BCD1
        ROL   BCD2
```

```
        DEC     CNT             ; 循环计数值减 1
        BRNE    ADJUST          ; 计数值不为 0,进行调整
        RET                     ; 计数值为 0,转换完成
ADJUST: LDI     ZL,ADBCD2+1     ; 建立 Z 指针
        LD      TEMP,-Z         ; Z←Z-1,读 BCD 位
        SUBI    TEMP,-$33       ; 加 $33 预调整
        SBRS    TEMP,3          ; 原低位<5?
        SUBI    TEMP,$03        ; 是,-3 恢复原值
        SBRS    TEMP,7          ; 否,高位<5?
        SUBI    TEMP,$30        ; 是,无须调整恢复原值
        ST      Z,TEMP          ; 否,存中间结果
        CPI     ZL,ADBCD2-2     ; 三字节均调整完毕?
        BRNE    ADJUST+1        ; 否,继续预调整
        RJMP    SHIFT           ; 是,继续移位
```

3. 二位 BCD 数转换成 8 位二进制数

这种转换比较简单,可以采用"一退六二五"的办法,即 BCD 的十位数减 1,若无借位,则二进制结果加 $0A,再减 1,再加 $0A,直至减到有借位时为止,个位数无须转换。

[例 3.39] BCDVAL 寄存器中有两位 BCD 数,要求把它转换成相应的 8 位二进制数,存入 BINRES 寄存器中。

```
        .DEF    BINRES=R16      ; 二进制结果
        .DEF    BCDVAL=R17      ; 待转换 BCD 值
BCDB8:  MOV     BINRES,BCDVAL   ; 二进制结果初始化
        ANDI    BINRES,$0F      ; 结果初值暂为 BCD 个位数
        SUBI    BCDVAL,$10      ; BCD 十位数减 1
        BRCS    RETURN          ; 若有借位则返回
        SUBI    BINRES,-$0A     ; 无借位,二进制结果加 10
        RJMP    BCDB8+2         ; 十位数继续减 1
RETURN: RET                     ; 转换结束
```

4. 五位 BCD 数转换成 16 位二进制数

假设这 5 位 BCD 数自万位至个位分别为 a、b、c、d 和 e。不言而喻,二进制结果应等于

$$10000a+1000b+100c+10d+e=10\{10[10(10a+b)+c]+d\}+e$$

下面一个转换子程序正是按这个公式编写的。

[例 3.40] 把 BCDH:BCDM:BCDL 三个寄存器中的 5 位 BCD 数转换成 16 位二进制数,并把结果存入 HBIN:LBIN 两个寄存器中。

从上面的计算公式中可以看出,该公式的最基本结构是某 BCD 位乘 10 后再加下一个 BCD 位,或是某 BCD 中间值乘 10 后再加下一个 BCD 位。根据这个基本结构就可以编写出一个某数乘 10 再加一个 BCD 位的子程序来。但是,下一个 BCD 位可能是千位、百位、十位,它们有的是高 4 位,有的是低 4 位。所以此子程序应考虑到通用性。

```
        .DEF    COPYL=R12       ; 暂存器
        .DEF    COPYH=R13       ; 暂存器
```

```
        .DEF    MP10L=R14          ;待×10某数之低字节
        .DEF    MP10H=R15          ;待×10某数之高字节
        .DEF    ADDER=R19          ;乘后待加值
MUL10:  SWAP    ADDER              ;下一个BCD位为高4位
        MOV     COPYL,MP10L        ;复制
        MOV     COPYH,MP10H
        LSL     MP10L              ;原值×2
        ROL     MP10H
        LSL     COPYL              ;复制值×2
        ROL     COPYH
        LSL     COPYL              ;复制值×4
        ROL     COPYH
        LSL     COPYL              ;复制值×8
        ROL     COPYH
        ADD     MP10L,COPYL        ;2倍值+8倍值
        ADC     MP10H,COPYH        ;等于×10
        ANDI    ADDER,0x0F         ;屏蔽下个字节高4位
        ADD     MP10L,ADDER        ;10倍值+下一位
        BRCC    TOMAIN             ;无进位,跳转
        INC     MP10H              ;有进位,高位加1
TOMAIN: RET
```

这段子程序的功能是把MP10H:MP10L值乘以10,再加下一个BCD位。若下一个BCD位为一寄存器的高4位,则子程序入口地址为MUL10;但如果是低4位,那么入口地址则是MUL10+1。以下是主程序段。

```
        .DEF    LBIN=R14           ;二进制结果低字节(同MP10L)
        .DEF    HBIN=R15           ;二进制结果高字节(同MP10H)
        .DEF    BCDL=R16           ;BCD十位和个位(记作d, e)
        .DEF    BCDM=R17           ;BCD千位和百位(b,c)
        .DEF    BCDH=R18           ;BCD万位(a)
BCDB16:ANDI     BCDH,$0F           ;保留万位
        CLR     MP10H
        MOV     MP10L,BCDH         ;MP10H:MP10L=a
        MOV     ADDER,BCDM
        RCALL   MUL10              ;10a+b
        MOV     ADDER, BCDM
        RCALL   MUL10+1            ;10(10a+b)+c
        MOV     ADDER, BCDL
        RCALL   MUL10              ;10[10(10a+b)+c]+d
        MOV     ADDER, BCDL
        RCALL   MUL10+1            ;10{10[10(10a+b)+c]+d}+e
        RET
```

3.2.2 BCD 加减运算

这一节要解决的主要问题是在 BCD 加减法运算中如何进行十进制调整。

1. 两个 2 位 BCD 数相加

在做 BCD 加法运算时，进行十进制调整的依据有如下两点：

- 在 BCD 加法过程中，是否有进位和半进位产生；
- 所得之和的两 BCD 位中，是否有大于 9 者。

若有进位产生，或和的十位数＞9，则应对该位进行加 6 调整。如果有半进位产生，或和的个位数＞9，那么就应对此个位数进行加 6 调整。相或之两条件同时成立是不可能的。

按 BCD 运算规则，两 BCD 位相加之和≥10，就应进位。但实际二进制运算中，两 4 位数之和≥16 时才进位。这样，若遇进位或半进位，则只有在相应位上加 6，才能得到十进制结果。两 BCD 位之和≥10，但尚不足进位时，对其进行加 6 调整，促其产生进位，以符合 BCD 运行之规则。

[例 3.41] BCD21 和 BCD20 两寄存器中各含有一个两位 BCD 数，要求把它们加起来，和之百位数置于 BCD21 中，十位和个位存入 BCD20 中。

```
        .DEF    BCD20=R16          ; 某 2 位 BCD 数
        .DEF    BCD21=R17          ; 另一 2 位 BCD 数
        .DEF    DA=R18             ; 十进制调整值
        .DEF    STATUS=R19         ; SREG 备份
ADDBCD: LDI     DA,6               ; 准备做加 6 调整
        ADD     BCD20,BCD21        ; 求 BCD 和
        IN      STATUS,SREG        ; 保存标志位
        ADD     BCD20,DA           ; 加 6 预调整
        BRHS    GRTER9             ; 和之个位数＞9，保留调整结果
        SBRS    STATUS,5           ; ≤9，但原 H=1，保留调整结果
        SUB     BCD20,DA           ; ≤9，且原 H=0，撤消预调整
GRTER9: SWAP    DA
        ADD     BCD20,DA           ; 加 $60 预调整
        BRCC    NOGR90             ; 和之十位数≤9，跳转
CSET:   LDI     BCD21,1            ; ＞9，调整后产生百位
        RET
NOGR90: SBRC    STATUS,0           ; 十位数≤9，且原 C=0
        RJMP    CSET               ; 原 C=1，产生百位
        SUB     BCD20,DA           ; 原 C=0，且十位数≤9，撤消预调整
        CLR     BCD21              ; 百位数为 0
        RET
```

2. 两 2 位 BCD 数相减

BCD 减法运算相对简单一些，因为调整依据只是借位和半借位的有无，而不存在差值某位大于 9 的问题。在 BCD 减法运算中，若有借位产生，则差之十位数要做减 6 调整。如有半借位，就应将差之个位数减 6。

［例 3.42］ BCDA 和 BCDB 两寄存器中各含一 2 位 BCD 数，要求进行 BCDA←BCDA－BCDB 的减法运算，若有借位，则将 BCDB 寄存器置成 $01，否则将其清 0。

```
        .DEF    BCDA=R16        ; BCD 被减数
        .DEF    BCDB=R17        ; BCD 减数
        .DEF    STCOPY=R18      ; SREG 备份
SUBBCD: SUB     BCDA,BCDB       ; 两 BCD 数相减
        IN      STCOPY,SREG     ; 保存状态标志
        CLR     BCDB            ; 借位标志清 0
        BRCC    CCLR            ; 无借位，跳转
        LDI     BCDB,1          ; 有借位，置借位标志
        SUBI    BCDA,$60        ; 十位数减 6 调整
CCLR:   SBRC    STCOPY,5        ; 原 H 为 0，跨越
        SUBI    BCDA,$06        ; 原 H=1，个位数减 6
        RET
```

3.2.3 BCD 运算测试程序

［例 3.43］ 试编写一段程序，令其调用所有上述子程序，一则作为一个使用实例，再则用来验证上述诸子程序运行的正确性。

```
RESET:LDI       R16,LOW(RAMEND)         ; 堆栈从片内 RAM
      OUT       SPL,R16                 ; 最高地址开始
      LDI       R16,HIGH(RAMEND)        ; AT90S8515 RAMEND=$025F
      OUT       SPH,R16                 ; 即 SP=$025F
  ; 把 $80 转换成 3 位 BCD 形式
      LDI       BIN,$80
      RCALL     B8BCD                   ; 结果 HUND:TENONE=$0128
  ; 把 54321 转换成 5 位 BCD 形式
      LDI       BINL,LOW(54321)
      LDI       BINH,HIGH(54321)
      RCALL     B16BCD                  ; 结果 BCD2:BCD1:BCD0=$54321
  ; 把 2 位 BCD 数 98 转换成 8 位二进制数
      LDI       BCDVAL,$98
      RCALL     BCDB8                   ; 结果 BINRES=$62(即 98)
  ; 把 5 位 BCD 数 65 535 转换成 16 位二进制形式
      LDI       BCDH,$06
      LDI       BCDM,$55
      LDI       BCDL,$35
      RCALL     BCDB16                  ; 结果 HBIN:LBIN=$FFFF
  ; BCD 数 51 和 79 相加
      LDI       BCD20,$51
      LDI       BCD21,$79
      RCALL     ADDBCD                  ; 结果 BCD21:BCD20=$0130
```

```
    ; BCD 数 72 减 28
        LDI     BCDA, $72
        LDI     BCDB, $28
        RCALL   SUBBCD              ; 结果 BCDB= $00,BCDA= $44
    ; BCD 数 0 减 90
        LDI     BCDA, $00
        LDI     BCDB, $90
        RCALL   SUBBCD              ; 结果 BCDB=1(负结果),BCDA= $10
HERE:   RJMP    HERE
```

3.3 多倍精度运算

多倍精度运算即多字节操作数间的数据运算。在处理高位字节时,应注意低位字节的进位或借位。但对于最低位字节来说,不存在向它进位或借位的问题。

ADC 和 SBC 指令计入了前步运算所产生的进位或借位,因此恰好可用来进行多倍精度运算。但在进行最低字节的加减运算时,应预先把进位标志清成 0。

下面将分别讨论多字节和双字节运算问题。

3.3.1 多字节加减运算

[例 3.44] 做两个带符号多字节数的加法运算,两数之字节数相同,被存放在 LENGTH 寄存器中,被加数和加数最低字节的地址分别为 ADDEND 和 ADDER,所得之和存回到被加数原位置中去。

```
        .EQU    ADDEND= $0110       ; 被加数起始地址
        .EQU    ADDER= $0100        ; 加数起始地址
        .DEF    FLAGS=R14           ; SREG 备份
        .DEF    LENGTH=R15          ; 长度寄存器
        .DEF    TMP1=R16            ; 暂存器 1
        .DEF    TMP2=R17            ; 暂存器 2
ADDSTR: LDI     ZL,LOW(ADDEND)      ; Z 作被加数指针
        LDI     ZH,HIGH(ADDEND)
        LDI     XL,LOW(ADDER)       ; X 作加数指针
        LDI     XH,HIGH(ADDER)
        CLC                         ; 预清进位标志
ADDST1: LD      TMP1,Z              ; 读被加数
        LD      TMP2,X+             ; 读加数,X←X+1
        ADC     TMP1,TMP2           ; 两数相加
        IN      FLAGS,SREG          ; 保存状态标志
        ST      Z+,TMP1             ; 和存回 SRAM,Z←Z+1
        DEC     LENGTH              ; 字节数减 1
        BRNE    ADDST1              ; 尚未加完,,再加
        SBRS    FLAGS,3             ; 已加完,有溢出?
```

```
        RET                                    ; 无,返回
        ……                                     ; 有,益出处理,略
```

这里应当指出,DEC 指令对 S、V、N 和 Z 标志有影响。故 ADC 指令后必须把 SREG 寄存器中的各状态位保存起来。待最高字节相加完毕而检查有无溢出时,应当检测 FLAGS 寄存器中保存的 V 位,而不能检测 SREG 的 V 位,因为后者是 DEC 指令所产生的标志,而非 ADC 的。

读者不难理解,若将上段程序中的 ADC 指令换成 SBC 指令,则这将是一段多字节操作数的减法程序。

3.3.2　16 位运算

这里我们将做一个全面的例题,包括 16 位数的加法、减法、比较和取补等运算。

[例 3.45.1]　为本例的 16 位运算准备具体数据。

```
.CSEG
.DEF    DATA1L=R16
.DEF    DATA1H=R17
.DEF    DATA2L=R18
.DEF    DATA2H=R19
LDI     DATA1L,0X12     ; 数据 1 低字节
LDI     DATA1H,0X34     ; 数据 1 高字节
LDI     DATA2L,0X56     ; 数据 2 低字节
LDI     DATA2H,0X78     ; 数据 2 高字节
```

[例 3.45.2]　做 16 位数的加法运算。

```
        .DEF    ADD1L=R2
        ,DEF    ADD1H=R3
        .DEF    ADD2L=R4
        .DEF    ADD2H=R5
ADD16:  MOV     ADD1L, DATA1L           ; 取数据 1 低字节
        MOV     ADD1H,DATA1H            ; 取数据 1 高字节
        MOV     ADD2L,DATA2L            ; 取数据 2 低字节
        MOV     ADD2H,DATA2H            ; 取数据 2 高字节
        ADD     ADD1L,ADD2L             ; 两数低字节相加
        ADC     ADD1H,ADD2H             ; 两高字节带进位相加
    ; ADD1H:ADD1L 的内容应为 0XAC68
```

[例 3.45.3]　16 位寄存器变量与一 16 位立即数相加,和存入该 16 位寄存器中。

```
        .DEF    ADDIL=R20
        .DEF    ADDIH=R21
        .EQU    ADDI2=0X1234
ADDI16:MOV      ADDIL,DATA1L            ; 取数据 1
        MOV     ADDIH, DATA1H
```

```
        SUBI    ADDIL, LOW(-ADDI2)      ; 两低字节相加
        SBCI    ADDIH,HIGH(-ADDI2)      ; 两高字节相加
    ; ADDIH:ADDIL 的内容应等于 0X4646
```

[例 3.45.4] 做 16 位数的减法运算。

```
        .DEF    SUB1L=R6
        .DEF    SUB1H=R7
        .DEF    SUB2L=R8
        .DEF    SUB2H=R9
SUB16:  MOV     SUB1L, DATA1L           ; 取被减数低字节
        MOV     SUB1H,DATA1H            ; 取被减数高字节
        MOV     SUB2L,DATA2L            ; 取减数低字节
        MOV     SUB2H,DATA2H            ; 取减数高字节
        SUB     SUB1L,SUB2L             ; 低字节相减
        SBC     SUB1H,SUB2H             ; 高字节带借位相减
    ; SUB1H:SUB1L 应等于 $BBBC
```

[例 3.45.5] 16 位寄存器变量减一 16 位立即数,差放在该 16 位寄存器中。

```
        .DEF    SUBIL=R22
        .DEF    SUBIH=R23
        .EQU    SUBI2=0X1234
SUBI16: MOV     SUBIL,DATA1L            ; 取被减数低字节
        MOV     SUBIH,DATA1H            ; 取被减数高字节
        SUBI    SUBIL,LOW(SUBI2)        ; 低字节相减
        SBCI    SUBIH,HIGH(SUBI2)       ; 高字节相减
    ; SUBIH:SUBIL 的内容应等于 0X21DE
```

[例 3.45.6] 将两寄存器对 CP1H:CP1L 与 CP2H:CP2L 之内容进行比较,若两者相等,则 Z 标志置 1,否则它将被清 0。

```
        .DEF    CP1L=R16
        .DEF    CP1H=R17
        .DEF    CP2L=R18
        .DEF    CP2H=R19
CP16:   CP      CP1L,CP2L               ; 低字节比较
        CPC     CP1H,CP2H               ; 高字节比较
    ; 比较后 Z=0,C=1
```

[例 3.45.7] 将寄存器对 CP1H:CP1L 之内容与一 16 位立即数 CP2 进行比较。

```
        .DEF    CP1L=R16
        .DEF    CP1H=R17
        .DEF    CPTMP=R24
        .EQU    CP2=0X3412
CPI16:  CPI     CP1L,LOW(CP2)           ; 低字节比较
        LDI     CPTMP,HIGH(CP2)         ; CP2 高字节送暂存器
        CPC     CPIH,CPTMP              ; 高字节比较
```

```
        ; 比较后 Z=1,C=0
```

[例 3.45.8]　把寄存器对 NGH:NGL 的内容取补。

```
        .DEF    NGL=R24
        .DEF    NGH=R25
NG16:   MOV     NGL,DATA1L
        MOV     NGH,DATA1H
        COM     NGL
        COM     NGH
        ADIW    NGL,1
        ; NGH:NGL 的内容应等于 $CBEE
HERE:   RJMP    HERE
```

3.4　数据块传送

这一节将讨论数据块复制问题，包括从程序存储器到 SRAM 以及由 SRAM 某区域到 SRAM 另一区域的数据块复制。对片内 EEPROM 的访问将在下一章详细讲述。

3.4.1　程序存储器至数据存储器的数据传送

大家知道，LPM 指令读程序存储器中的常数，要以 Z 寄存器作地址指针，且常数由 Flash 存储器固定读到 R0 寄存器中。因此，欲将程序存储器 Flash 中的数据块复制到数据存储器 SRAM 中去，应设置两个地址指针。一是 Z 寄存器，用来指示源数据块的首地址；一是 Y 或 X 寄存器，用以指示目的数据块的首地址。当然数据块的大小也应是一个已知数。

[例 3.46]　写出自程序存储器向数据存储器进行数据块传送的子程序。在调用这段子程序前，源块、目的块的地址指针以及块的大小等参数都是设置好了的。

```
.include  "8515def.inc"
      rjmp   reset
.def   fsize=r16              ; 数据块规模
fram:lpm                      ; 读一字节常数
      st      Y+, r0          ; 将之存入 SRAM,Y←Y+1
      adiw    Z,1             ; Z←Z+1
      dec     fsize           ; 数据块字节数减 1
      brne    fram            ; 未完,继续
      ret                     ; 已完,返回
```

3.4.2　数据存储器两区域间的数据块传送

SRAM 两区域间的数据块传送也需要三个参数，即源和目的块的首地址及数据块的规模。

[例 3.47]　写出 SRAM 两区域间数据块传送子程序，调用该子程序前，源和目的地址指针及数据块含多少字节等参数都是设置好了的。

```
        .def      ramtmp=R1           ; 暂存器
```

```
    .def    ramsize=R16          ; 数据块规模
ramram:ld   ramtmp,Z+            ; 读源块一字节,Z←Z+1
       st   Y+, ramtmp           ; 写入目的块,Y←Y+1
       dec  ramsize              ; 字节数减 1
       brne ramram               ; 未完,继续传送
       ret                       ; 已完,返回
```

3.4.3 数据块传送测试程序

[例 3.48] 把程序存储器中自 FTABLE 地址单元开始的 20 个字节常数复制到 SRAM 区,起始地址为 BLOCK1,然后再复制到 BLOCK2 地址单元开始的 SRAM 区。

```
    .equ    BLOCK1= $60
    .equ    BLOCK2= $80
    .def    TEMP=r16
RESET: ldi  TEMP, low(ramend)      ; 设置 SP 低字节
       out  SPL,TEMP
       ldi  TEMP, high(ramend)     ; 设置 SP 高字节
       out  SPH, TEMP
       ldi  zh, high(FTABLE*2)     ; 源块地址指针高字节
       ldi  zl, low(FTABLE*2)      ; 源块地址指针低字节
       ldi  yh, high(BLOCK1)       ; 目的块地址指针高字节
       ldi  yl, low(BLOCK1)        ; 目的块地址指针低字节
       ldi  fsize, 20              ; 块规模
       rcall FRAM                  ; 复制 20 个字节
       ldi  zh, high(BLOCK1)
       ldi  zl, low(BLOCK1)
       ldi  yh, high(BLOCK2)
       ldi  yl, low(BLOCK2)
       ldi  ramsize, 20
       rcall ramram
HERE:  rjmp  HERE
FTABLE:.DB  0,1,2,3,4,5,6,7,8,9
       .DB  10,11,12,13,14,15,16,17,18,19
```

这里再次提醒读者注意,用 LPM 指令从程序存储器中读数据时,地址指针寄存器 Z 的内容应等于字地址乘以 2,即 FTABLE*2。这一点在例 3.4 中已清楚地阐述过。

3.5 乘除法运算

AVR 微控制器系列中,除少数几个品牌有乘法指令外,大多数 MCU 都没有乘法指令。这一节我们将讨论如何用子程序来实现乘除法运算。

3.5.1 8 位无符号数乘法

为简化问题的讨论,我们先研究两个 4 位数的乘法。从下面一个例子的传统乘法算草中

可以总结出如下几点：

- 乘数最低位为 1，将被乘数照写下来，视作第一步的部分积；
- 乘数之次低位亦为 1，被乘数左移一位后与部分积相加，得新的部分积；
- 乘数次高位为 0，0000 左移一位后与部分积相加，得新的部分积；
- 乘数最高位为 1，被乘数左移一位后与部分积相加，得最终乘积。

```
   1010
 ×1011
------
   1010
  1010
 0000
1010
-------
1101110
```

现在把部分积和乘数分别装在两个 4 位寄存器中，部分积初始值为 0000。下面几步与传统算法应该是等效的。

- 将乘数寄存器逻辑右移一位，最低位 1 移出寄存器进入 C 标志；
- 因移出位为 1，故部分积 0000 加被乘数 1010，其和作为新的部分积；
- 部分积寄存器和乘数寄存器一起闭环右移一位，乘数原次低位 1 移入 C 标志；
- 移出位为 1，故部分积再加被乘数 1010，又得新的部分积；
- 部分积寄存器和乘数寄存器串联闭环右移一位，乘数原次高位 0 移入 C 标志；
- 移出位为 0，故部分积加 0000，以得一新的部分积；
- 部分积寄存器与乘数寄存器串联闭环右移一位，乘数原最高位 1 移入 C 标志；
- 移出位为 1，故部分积和被乘数 1010 相加，得新的部分积；
- 部分积和乘数寄存器最后一次右移，部分积寄存器为乘积之高 4 位，而原乘数寄存器最终变成乘积低 4 位。

不难看出，这里与传统算法表面上的不同点在于：

- 将传统算法的被乘数左移与部分积相加改成了部分积右移与被乘数相加；
- 把传统算法的先移位后做加法改成了先做加法而后移位。

第一条中两种情况的等效性是很显然的。第二条的目的就在于，通过先移位将乘数最低位移出，使之进入 C 标志，便于判断并决定该加被乘数，还是加 0。因为部分积初值为 0，所以先将被乘数左移一位再与 0 相加，和先与 0 相加再将部分积右移是等效的。

下面可以讨论两 8 位无符号二进制数的乘法运算子程序了。

[例 3.49]　MC8U 和 MP8U 两寄存器中各含一 8 位无符号二进制数，求两者之乘积，并将积之高低字节分别存入 M8UH 和 M8UL 两寄存器中。

```
        .DEF    MC8U=R16          ; 被乘数
        .DEF    MP8U=R17          ; 乘数/积低字节
        .DEF    M8UL=R17          ; 乘积低字节/乘数
        .DEF    M8UH=R18          ; 乘积高字节
        .DEF    CNT8U=R19         ; 位计数器
MPY8U:  CLR     M8UH              ; 乘积高字节初值为 $00
        LDI     CNT8U,8           ; 位计数初值为 8
        LSR     MP8U              ; 乘数逻辑右移一位
M8U1:   BRCC    M8U2              ; C=0,跳转
        ADD     M8UH,MC8U         ; C=1,部分积←部分积+被乘数
M8U2:   ROR     M8UH              ; 部分积与乘数闭环
        ROR     M8UL              ; 右移一位
```

```
        DEC     CNT8U           ；位计数值减 1
        BRNE    M8U1            ；位计数值非 0 则继续
        RET
```

3.5.2 8 位带符号数乘法

带符号数乘法运算用比较法(亦称 Booth 算法)来进行。所谓比较法就是对乘数每相邻两位进行比较,根据比较结果决定部分积该加 0、加被乘数或减被乘数。具体做法如下:

- 部分积初值为 0,开始时乘数右侧补充一位 0;
- 乘数最低两位若相同,则部分积加 0 后与乘数一起右移一位,构成新的部分积;
- 乘数最低两位若为 01,则部分积加被乘数后与乘数一起右移一位,得新部分积;
- 乘数最低两位若为 10,则部分积减被乘数后与乘数一起右移一位,得新部分积;
- 这样 8 位处理完后,即得最后乘积。

应当指出,开始时乘数的最低两位系指其原最低位与右侧补充位而言。

[例 3.50] MC8S 和 MP8S 两寄存器中各含有一 8 位带符号二进制数,求两数之乘积,将结果之高低字节分别存入 M8SH 和 M8SL 两寄存器中。

```
        .DEF    MC8S=R16        ；被乘数
        .DEF    MP8S=R17        ；乘数/乘积低字节
        .DEF    M8SL=R17        ；乘积低字节/乘数
        .DEF    M8SH=R18        ；乘积高字节
        .DEF    CNT8S=R19       ；位计数器
MPY8S:  SUB     M8SH,M8SH       ；积高字节和 C 清 0
        LDI     CNT8S,8         ；位计数初值为 8
M8S1:   BRCC    M8S2            ；C=0,跳转
        ADD     M8SH,MC8S       ；C=1,部分积+被乘数
M8S2:   SBRC    MP8S,0          ；乘数最低位为 0,跨越
        SUB     M8SH,MC8S       ；乘数最低位为 1,部分积-被乘数
        ASR     M8SH            ；算术右移部分积高字节
        ROR     M8SL            ；循环右移部分积低字节/乘数
        DEC     CNT8S           ；位计数值减 1
        BRNE    M8S1            ；位计数值≠0,继续
        RET
```

这段子程序一开始,部分积高字节本身相减,结果当然为 0,进位标志亦被清掉。第二条指令设置位计数器初值为 8。执行 BRCC 指令时,因 C 为 0,故将跳转到 SBRC 指令去。这条指令判断乘数之最低位是否为 0,若是,即最低两位均为 0,则越过 SUB 指令去执行 ASR 指令。也就是说,乘数最低两位为 00,部分积加 0 后与乘数一起右移一位,得新部分积。如果执行 SBRC 指令时,乘数最低位为 1,即最低两位为 10,按规则这时部分积应减被乘数。此后部分积与乘数一起右移一位,得新部分积。

如果第一次右移由乘数移至 C 的位为 1,那么当第二次执行 BRCC 指令时,将不发生跳转,而顺序执行 ADD 指令,即部分积加被乘数。但这里将有两种情况:

第一,当 SBRC 指令判断乘数最低位为 0,即最低两位为 01 时,则 CPU 将跨越 SUB 指

令，直接去执行移位指令，这就是说，此前所做部分积与被乘数的加法是算数的。

第二，若 SBRC 指令判断乘数最低位为 1，即最低两位为 11，则 CPU 将顺序执行下面的 SUB 指令。这就是说，前面所做部分积与被乘数的加法运算被撤消了，等于部分积加 0，然后再去移位，得新部分积。

如果上次由乘数移至 C 的位为 0，这与开始时的情况一样，不再赘述。

3.5.3　8 位无符号数除法

无符号数除法运算的主要思路是，首先将被除数之最高位作为最低位（前面的高位均为 0），减除数，若够减，则商 1，若不够减，则商 0 并加除数以恢复原值。第二步，将被除数左移一位，原次高位作最低位，再减除数，够减商 1，不够减则商 0，并恢复原值。这样把被除数所有位都处理完毕时，就可得到商和余数。

无符号数除法运算的具体做法如下：

- 被除数、除数和余数各占用一个工作寄存器，商将与被除数共用一个寄存器；
- 用余数寄存器本身相减的办法将余数及进位标志 C 清 0；
- 余数和被除数两寄存器闭环左移一位，被除数最高位移至余数寄存器最低位；
- 余数减除数，够减则商 1，得新余数，不够减则商 0，并恢复原余数；
- 继续左移、做减法、得新余数或恢复原余数。这样所有位均处理完毕时，被除数寄存器中将得整数商，而余数寄存器则含未除尽之余数。

［例 3.51］　DD8U 和 DV8U 两寄存器中分别含一 8 位无符号二进制数，以 DD8U 作被除数，DV8U 作除数，进行除法运算。所得之整数商存入 DD8U 寄存器中，余数放在 REM8U 寄存器内。

```
        .DEF    REM8U=R15           ; 余数
        .DEF    Q8U=R16             ; 商
        .DEF    DD8U=R16            ; 被除数
        .DEF    DV8U=R17            ; 除数
        .DEF    DCNT8U=R18          ; 位计数器
DIV8U:  SUB     REM8U,REM8U         ; 余数和 C 清 0
        LDI     DCNT8U,9            ; 位计数初值
D8U1:   ROL     DD8U                ; 被除数左移一位
        DEC     DCNT8U              ; 位计数减 1
        BRNE    D8U2                ; 位计数值不为 0，跳转
        RET                         ; 为 0，返回主程序
D8U2:   ROL     REM8U               ; 被除数最高位移至余数最低位
        SUB     REM8U,DV8U          ; 余数←余数－除数
        BRCC    D8U3                ; 若无借位则跳转
        ADD     REM8U,DV8U          ; 有借位，恢复余数
        CLC                         ; 清进位，即商 0
        RJMP    D8U1                ; 将 0 商移至被除数寄存器
D8U3:   SEC                         ; C 置 1，即商 1
        RJMP    D8U1                ; 并移入被除数寄存器
```

从这段子程序中可以看到，处理被除数最高位时所得之商，在第二次移位时才进入被除数寄存器的最低位。而这位商，或为 0，或是 1，最终应占据被除数寄存器的最高位。因此，位计数寄存器初值应为 9，而不是 8。

3.5.4 8 位带符号数除法

带符号数的除法运算可采用下述步骤进行：

- 测试被除数和除数的符号位，以确定商之符号，两者异号商为负，同号商为正；
- 将被除数和除数改成无符号数，进行无符号数的除法运算；
- 若商应为正数，则上述除法所得即为最终结果；
- 若商应为负数，则将上述无符号除法所得之商化成负数。

[例 3.52] DD8S 和 DV8S 两寄存器中各含一 8 位有符号数，将 DD8S 除以 DV8S，商存放在 DD8S 寄存器中。而余数则保留在 REM8S 寄存器中。

```
        .DEF    D8S=R14             ;商的符号
        .DEF    REM8S=R15           ;余数
        .DEF    Q8S=R16             ;商
        .DEF    DD8S=R16            ;被除数
        .DEF    DV8S=R17            ;除数
        .DEF    DCNT8S=R18          ;位计数器
DIV8S:  MOV     D8S,DD8S            ;被除数送至符号寄存器
        EOR     D8S,DV8S            ;被除数和除数相异或
        SBRC    DV8S,7              ;除数为正，跨越
        NEG     DV8S                ;除数为负，化成正数
        SBRC    DD8S,7              ;被除数为正，跨越
        NEG     DD8S                ;被除数为负，化为正数
        SUB     REM8S,REM8S         ;余数和进位清 0
        LDI     DCNT8S,9            ;位计数初值
D8S1:   ROL     DD8S                ;被除数左移一位
        DEC     DCNT8S              ;位计数值减 1
        BRNE    D8S2                ;位计数值非 0，跳转
        SBRC    D8S,7               ;商应为正数，跨越
        NEG     Q8S                 ;商应为负数，求补
        RET                         ;返回主程序
D8S2:   ROL     REM8S               ;被除数最高位移至余数最低位
        SUB     REM8S,DV8S          ;余数←余数－除数
        BRCC    D8S3                ;无借位，跳转
        ADD     REM8S,DV8S          ;有借位，余数复原
        CLC                         ;C 清 0，即商 0
        RJMP    D8S1                ;将 0 商移入被除数寄存器
D8S3:   SEC                         ;C 置 1，即商 1
        RJMP    D8S1                ;将商移入被除数寄存器
```

3.6　数据排序

现场采集来的数据，往往需要进行某种加工处理，数据排序和数字滤波就是这种加工处理的两项具体作业。数据存储器 SRAM 中某数据块的所有元素，可被要求按自小至大的升序方式，或自大至小的降序方式排列起来。这样的数据处理大多采用一种代码效率最高的所谓冒泡算法来进行。

3.6.1　冒泡排序算法

冒泡排序算法的基本操作是相邻两数值的比较。大家知道，AVR 这类指令比较丰富，两无符号数的比较可用 BRLO 和 BRSH 指令来做，而带符号数则可用 BRGE 和 BRLT 指令进行比较。相邻两数值进行比较时，若其顺序符合升序或降序的要求，则保持它们的位置不变。若有悖于这种要求，则将它们交换一下位置。这样，第一个回合，即 N 个元素进行过 $N-1$ 次比较之后，其中之最大(降序)或最小(升序)元素便像一泡沫似地浮出水面，被存放在该数据块的最低地址单元中。而第二个回合，即 $N-1$ 个元素进行过 $N-2$ 次比较后，次大和次小元素便被存入第二个地址单元中。不难想像，最后一个回合把两个最小或最大的数值也按要求存放在该块中。

[例 3.53]　SRAM 区有一无符号数数组，共 CNT1+1 个元素，其最后一元素之地址高字节为 ENDH，其地址低字节为 ENDL，要求把该数组的所有元素按降序方式排列起来。

```
        .DEF    A=R13               ; 要比较的第一个元素
        .DEF    B=R14               ; 要比较的第二个元素
        .DEF    CNT2=R15            ; 回合内比较次数计数
        .DEF    CNT1=R16            ; 回合计数器
        .DEF    ENDL=R17            ; 数组末元素地址低字节
        .DEF    ENDH=R18            ; 数组末元素地址高字节
BUBBLE: MOV     ZL,ENDL             ; Z 指向数组末地址
        MOV     ZH,ENDH
        MOV     CNT2,CNT1           ; 回合计数与回合内比较次数初值相等
ILOOP:  LD      A,Z                 ; 读一高地址元素
        LD      B,-Z                ; Z 减 1,读相邻低地址元素
        CP      A,B                 ; 两元素比较
        BRLO    L1                  ; 若 A<B,符合降序要求
        ST      Z,A                 ; A≥B,两数交换位置
        STD     Z+1,B
L1:     DEC     CNT2                ; 回合内应比较次数减 1
        BRNE    ILOOP               ; 若不为 0,则继续比较
        DEC     CNT1                ; 回合数减 1
        BRNE    BUBBLE              ; 若不为 0,则进行下一个回合
        RET                         ; 排序完成
```

这段子程序是把无符号数数组内各元素按降序排列的，即地址越高，数组元素值越小。假

如要求按升序排列，那么只要把这段子程序中的 BRLO 指令换成 BRSH 即可。若数组内各元素为带符号数，则以 BRLT 指令取代 BRLO 便可做到降序排列，用 BRGE 指令则可达到升序排列之目的。

3.6.2 数据排序测试程序

[例 3.54] 程序存储器中有一 60 个字节的常数表，其首地址为 TABLE，末地址是 TBLEND，要求将此表复制到数据存储器 ARRAY 地址单元开始的 60 个字节中去，然后再对这 60 个字节进行降序排列。

```
        .include "8515def.inc"
        .equ    SIZE=60                 ; 数据块规模
        .equ    ARRAYL=$60              ; 第一个数据元素 SRAM 地址低字节
        .equ    ARRAYH=$00              ; 高字节
          rjmp    RESET
        .def    temp=r16
RESET:  ldi     temp, low(RAMEND)       ; SP 初始化
        out     SPL, temp
        ldi     temp, high(RAMEND)
        out     SPH, temp
        clr     ZH                      ; Z 指向 ROM 表中最后一个元素
        ldi     ZL, TBLEND*2+1
        ldi     YL, low(256*ARRAYH+ARRAYL+SIZE)
        ldi     YH, high(256*ARRAYH+ARRAYL+SIZE)
LOOP:   lpm                             ; 将 ROM 常数读至 R0 中
        st      -Y,r0                   ;Y←Y-1,存入 RAM 区
        sbiw    ZL,1                    ; Z←Z-1
        cpi     YL, ARRAYL              ; 数据复制完毕？
        brne    LOOP                    ; 否,继续
        cpi     YH, ARRAYH              ; 数据复制完毕？
        brne    LOOP                    ; 否,继续
SORT:   ldi     ENDL, low(ARRAYH*256+ARRAYL+SIZE-1)
        ldi     ENDH, high(ARRAYH*256+ARRAYL+SIZE-1)
        ldi     CNT1,SIZE-1
        rcall   BUBBLE
HERE:   rjmp    HERE
TABLE:  .DB     ……
TBLEND:.DB      74,48
```

这段程序从标号 RESET 起为初始化段，它做了三件事。第一，将堆栈指针 SP 设置为 $025F，即堆栈安排在片内 RAM 最高地址处。第二，Z 指针指向 ROM 表中最后一个元素，至于为什么把表末地址 TBLEND 乘以 2 再加 1 才能指向最后一个元素，在例 3.4 中曾经做过详细说明。第三，Y 指针指向 SRAM 区数据块末地址+1。这里 ARRAYH 为数据块首地址高字节，ARRAYL 为其低字节，故 256*ARRAYH+ARRAYL 便是此首地址的 16 位表示形

式。首地址＋数据块字节数 SIZE－1 等于末地址。当然首地址＋SIZE 就是末地址＋1 了。

程序的第二段自符号地址 LOOP 开始，它的任务是把 ROM 中的表格复制到 SRAM 中去，在例 3.46 和例 3.48 的程序中都有过类似的段落。

第三段自 SORT 开始，这里向 ENDL 和 ENDH 两寄存器中装入待排序数组最后一个元素的地址低字节和高字节后，又设置了回合计数初值，即 SIZE－1。这样，调用 BUBBLE 排序子程序的参数都已齐全。通过子程序将数组排好序后，该测试程序便进入死循环，不再前进。

3.7　数字滤波

在单片机控制系统的输入信号中，往往掺杂有各种噪音干扰成分。为提高信号的可靠性，减小虚伪信息的影响，可采用软件方法实现数字滤波。

数字滤波有多种方法，可根据不同的测量参数进行选择。

程序判断滤波法：输入信号连续两次采样值间的差若超出根据经验所确定的偏差值，则表明最后一次采样值为一干扰信号，不予采信，以上次采样值作为本次采样值。若不超过上述偏差值，则本次采样值有效。当采样信号受到随机干扰或由于传感器不稳定而严重失真时，可采用这种滤波方法。

中值滤波法：对某一参数每个回合连续采样数次，然后把这些采样值按升序或降序方式排列起来，取中间值作为本回合的采样值。对于变化缓慢的参数，每个回合的采样次数可适当多些，例如可采样 7 次、9 次（一般取奇数次）等等。对于变化剧烈的参数，此法不宜采用。

算术平均值滤波法：对某一输入参数每个回合连续采样数次，取其平均值作为本回合的输入参数值。此法适用于抑制随机干扰，采样次数越多，平滑效果越好，但系统的灵敏度要降低。

下面我们将详细讨论去极值平均滤波和移动平均值滤波两种方法。

3.7.1　去极值平均滤波法

算术平均值滤波不能消除明显的干扰信号，而只是削弱其影响。去极值平均滤波法却可做到剔除干扰，从而使平均滤波结果更接近真实值。

去极值平均滤波的主要思路，就是从某输入参数连续数次采样结果中剔除一最大值和一最小值，以其余次数的采样结果平均值作为该参数本回合的采样值。

［例 3.55］　在 SRAM 存储空间自 SAMPLE 地址单元开始的 6 个字节中，存有对某输入参数的 6 次采样值。试以能剔除干扰成分的数字滤波方法求此回合接近真实的采样值。

下面的程序中，省略了堆栈指针 SP 初始化的一段。

```
.include "8515def.inc"
    rjmp    RESET                    ; 转向复位处理，下略
    ……
.EQU    SAMPLE= $0060                ; 采样数据缓冲区首地址
.DEF    CNTER=R16                    ; 采样数据计数器
.DEF    ZERO=R15                     ; 0 寄存器
.DEF    SUMH=R14                     ; 累加和高字节
.DEF    SUML=R13                     ; 累加和低字节
```

```
    .DEF    TMP=R12                 ;暂存寄存器
    .DEF    MAX=R11                 ;最大值寄存器
    .DEF    MIN=R10                 ;最小值寄存器
FILT:   LDI     ZL,LOW(SAMPLE)      ;Z指向采样数据
        LDI     ZH,HIGH(SAMPLE)     ;缓冲区首地址
        LDI     CNTER,6             ;6个采样值
        CLR     ZERO                ;0寄存器内容为0
        CLR     SUMH                ;累加和初值为0
        CLR     SUML
        LD      MAX,Z               ;最大初值与
        LD      MIN,Z               ;最小初值相等
LOOP:   LD      TMP,Z+              ;读采样值,Z←Z+1
        ADD     SUML,TMP            ;求累加和
        ADC     SUMH,ZERO
        CP      MAX,TMP             ;此采样值与最大值比较
        BRLO    NEWMAX              ;大于最大值,更新最大值
        CP      MIN,TMP             ;小于最大值,再与最小值比较
        BRLO    CNT                 ;大于最小值,跳转
        MOV     MIN,TMP             ;更新最小值
CNT:    DEC     CNTER               ;采样数据数目减1
        BRNE    LOOP                ;采样数据未处理完,继续
        SUB     SUML,MAX            ;去掉一个最大值
        SBC     SUMH,ZERO
        SUB     SUML,MIN            ;去掉一个最小值
        SBC     SUMH,ZERO
        LSR     SUMH                ;累加和除以2
        ROR     SUML
        LSR     SUMH                ;累加和再除以2
        ROR     SUML                ;SUML为滤波后采样值
        RET
NEWMAX: MOV     MAX,TMP             ;更换最大值
        RJMP    CNT                 ;继续
```

3.7.2 8点移动平均值滤波法

8点移动平均值滤波法的主要思路有如下几点：

- 每连续8个采样值为一段，以段平均值作为本段第4个采样数据的已滤波值；
- 以段外下一个采样值取代段内第一值，新段平均值便是新段第4数据的已滤波值；
- 由段外依次递补一采样值，以取代原段内第一值，形成新段，得一已滤波新值；
- 随着8值数据段的前进，已滤波值一一形成，连续排列起来，俨然在移动；
- 采样值数据块开头和结尾两端若干采样值没有相应的已滤波值。

下面我们以上述思路来研究一个例题。

[例3.56] 在SRAM中自$0060单元开始，存有对某输入参数的60个采样值，每值一

个字节，要求用 8 点移动平均值法对这些采样数据进行滤波。

大家知道，AVR 指令系统中没有 SRAM 数据的加法指令，而只有工作寄存器数据的加法指令。因此，为方便求累加和及平均值，必须选择一数据缓冲区，且该缓冲区应占据工作寄存器的位置。例如，我们选定的缓冲区地址为 $0000～$0007，即以 R0～R7 为 8 字节缓冲区。

下一步，把 $0060～$0067 的 8 值数据段读到 R0～R7 中去，求累加和及平均值，得 $0063 单元的已滤波值。

8 值数据段前进一步，即把 $0068 单元的采样值读入 R0，取代 R0 中原 $0060 单元之值，形成新的 8 值数据段。其平均值为 $0064 单元原采样值的已滤波值。

按此规律继续前进，直到整个数据块剩下最后 4 个元素不被触动为止。8 点移动平均值滤波法的数学描述公式是

$$\bar{x}_j = \frac{1}{8}\sum_{i=j-3}^{j+4} x_i$$

其中 $\bar{x}_j$ 为第 j 个已滤波值，x_i 为第 i 个采样值。

```
        .def    tmpr=r8         ; 暂存器
        .def    al=r9           ; 8 数据累加和低字节
        .def    ah=r10          ; 8 数据累加和高字节
        .def    tsize=r16       ; 数据块字节数
MAV8:   clr     yh              ; Y 指针初值为 $0000
        clr     yl
        clr     xh              ; X 指针初值为 $0000
        clr     xl
MAV81:  ld      tmpr, Z+        ; 读 SRAM 数据
        st      Y+, tmpr        ; 转存入 R0～R7 中
        cpi     yl, 8           ; 读够 8 个数据了吗？
        brne    MAV81           ; 不够，再继续
        sbiw    zl, 5           ; Z 指向第一个要滤波的值
MAV82:  clr     ah              ; 清累加和高字节
        clr     al              ; 清累加和低字节
        clr     yl              ; Y 指针指向 $0000
MAV83:  ld      tmpr,Y+         ; 将 R0～R7 值装入暂存器
        add     al, tmpr        ; 求累加和，即求
        adc     ah, xh          ; ah:al=r0+r1+…+r7
        cpi     yl, 8           ; 累加已完成？
        brne    MAV83           ; 否，继续
        lsr     ah              ; ah:al 值除以 2
        ror     al
        lsr     ah              ; ah:al 再除以 2
        ror     al
        lsr     ah              ; 第三次除以 2，即总共除以 8
        ror     al              ; 此时 al 为已滤波值
        ldd     tmpr,Z+5        ; 递补下一值进入缓冲区
```

```
        st    X+, tmpr      ; 形成新 8 值数据段
        andi  XL, $07       ; 寄存器编号只在 0～7 间变化
        st    Z+, al        ; 已滤波值存入 SRAM
        dec   tsize         ; 数据块字节数减 1
        cpi   tsize, 4      ; 数据块尚有最后 4 个值？
        brne  MAV82         ; 否，继续
        ret                 ; 是，子程序结束
```

此子程序的第一段使 X 和 Y 两指针均指向 $0000，即指向 R0 寄存器。

第二段，自 MAV81 开始，把 SRAM 数据块开头的 8 个字节分别读至 R0～R7 等 8 个寄存器中去。若 Z 指针初值为 $0060，则 8 字节读完后 Z 等于 $0068，最后减 5 得 $0063。第一个滤波值就是要放在这里的。

第三段，自 MAV82 至结尾，求 8 值数据段累加和，除以 8 求平均值，递补新值构成新的 8 值数据段，再求新段滤波后之值。应当指出，求累加和时用了 AH:AL 寄存器对。虽然每个数据只有 8 位，但若 8 个这样的数据累加起来放在一个 8 位寄存器中，恐怕就要溢出了。AH 只接受累加过程中的进位，在 ADC AH,XH 指令中 XH 始终为 0。

下面是主程序，其中堆栈初始化段从略。

```
        ldi   zl, $60       ; Z 指向 SRAM 数据块
        ldi   zh, $00       ; 首地址 $0060
        ldi   tsize, 60     ; 数据块规模 60 个字节
        rcall MAV8          ; 8 点移动平均值滤波
HERE:   rjmp  HERE          ; 就地循环
```

第四章　片内硬件资源的应用

通过前三章的学习，读者已经掌握了 AVR 单片机片内硬件资源的运作原理和程序设计的初步技巧。本章将以上述软硬件知识为基础，引导读者逐步学会如何使用这些硬件资源。

4.1　对片内 EEPROM 的访问

AVR 片内 EEPROM 可在烧录器上接受数据写入，亦可在 AVR 正常运行时响应程序对之进行的读写操作。

4.1.1　对片内 EEPROM 的随机读写

1.2.9 一节告诉我们，对片内的 EEPROM 的写操作要由 EEAR、EEDR、和 EECR 三个寄存器配合进行。首先把欲写的数据送入 EEPROM 数据寄存器 EEDR 中；把欲写往的地址告诉 EEPROM 地址寄存器 EEAR(AT90S1200)或 EEARH：EEARL(AT90S8515 的 EEAR 分成高低两个字节)；再把 EEPROM 控制寄存器 EECR 中的 EEMWE 位(EEPROM 写入允许总控位，1200 芯片无此位)写成 1；随后把 EECR 中的 EEWE 位写成 1，即给一写选通信号。这样 2.5～4 ms 之后数据将被写入到 EEPROM 指定的地址单元中。写操作完成后，EEWE 位被自动清 0，而 EEMWE 位置 1 后 4 个时钟周期即被硬件自动清 0。

但是，应当强调指出，在进行上述操作前，应搞清此前的写操作是否已经结束，即查询 EEWE 位是否为 0，不为 0 则应等待。

[例 4.1]　试编写对片内 EEPROM 的随机写入子程序，欲写数据存放于 EEdwr 寄存器中，而目的地址从 EEawrh：EEawrl 寄存器对中提取。

```
        .include "8515def.inc"
          rjmp   RESET
        .def     EEdwr=R16              ;待写入的数据字节
        .def     EEawrl=R24             ;目的地址低字节
        .def     EEawrh=R25             ;目的地址高字节
        .cseg
        .org     0x10
EEWR:   sbic     EECR,EEWE              ;若 EEWE 位不为 0，
          rjmp   EEWR                   ;则等待
        out      EEARH, EEawrh          ;EEWE=0
        out      EEARL, EEawrl          ;输出目的地址
        out      EEDR, EEdwr            ;输出待写数据
        sbi      EECR, EEMWE            ;写允许总控位置 1
        sbi      EECR, EEWE             ;写选通信号
```

;这条指令的执行时间为 2 个时钟周期，接着还要使 CPU 暂停 2 个时钟周期，此后才能执行下条指令

```
        ret
RESET   ;……
```

应当指出，若向 AT90S1200 的片内 EEPROM 进行写入，则应对上段程序做如下修改：

- 用 out EEAR，EEawr 指令取代向 EEARH 和 EEARL 两寄存器的输出指令；
- 取消 sbi EECR，EEMWE 指令。

如果被写的是 AT90S4414，那么只将 out EEARH，EEawrh 指令取消即可。

[例 4.2] 自片内 EEPROM 读一字节数据存入 EEdrd 寄存器中，源地址由 EEardh：EEard 寄存器对的内容指定。

自片内 EEPROM 中读数值前，应查询 EEWE 位是否为 0。若不为 0，则说明此刻有写操作在进行，必须等待，甚至读操作的前期准备工作亦不可开始。所谓准备工作是指向 EEAR 寄存器送入待写字节的地址。此地址安排好后，将 EECR 寄存器中的 EEPROM 读选通信号 EERE 置 1，即可读到结果。读操作只需要一条指令的执行时间，届时 EERE 位会被自动清 0。

```
        .def    EEdrd=r0                ; 存放所读数据字节
        .def    EEard=r24               ; 源地址单元低字节
        .def    EEardh=r25              ; 源地址单元高字节
EERD:   sbic    EECR, EEWE              ; 若 EEWE 位不为 0,
        rjmp    EERD                    ; 则等待
        out     EEARH, EEardh           ; 输出地址高字节
        out     EEARL, EEard            ; 输出地址低字节
        sbi     EECR, ERE               ; EEPROM 读选通信号
;此指令本身要 2 个时钟周期,接着还使 CPU 暂停两个时钟周期
        in      EEdrd, EEDR             ; 读入数据
        ret
```

4.1.2 对片内 EEPROM 的顺序读写

顺序读写的特点就在于首先要了解上次所读写的数据在片内 EEPROM 中的地址，从而确定这次应把数据写至什么地方，或该从哪个单元地址读出数据。这要求从 EEAR 寄存器中取出上次读写的地址，递增后再写回去。

[例 4.3] 在上次写操作结束后，继续向下一个地址单元中写入来自 EEdwrs 寄存器中的数据。

```
        .def    EEwtmp=r24              ; 地址低字节暂存器
        .def    EEwtmph=r25             ; 地址高字节暂存器
        .def    EEdwrs=r18              ; 待写数据暂存器
EEWRS:  sbic    EECR, EEWE              ; EEWE=0?
        rjmp    EEWRS                   ; 否,等待
        in      EEwtmp, EEARL           ; 是,读地址低字节
        in      EEwtmph,EEARH           ; 读地址高字节
        adiw    EEwtmp, 0x01            ; 地址递增一个单元
        out     EEARL, EEwtmp           ; 输出新地址低字节
        out     EEARH, EEwtmph          ; 输出新地址高字节
```

```
        out     EEDR, EEdwrs                ; 输出待写数据
        sbi     EECR, EEMWE                 ; 写允许总控位置 1
        sbi     EECR, EEWE                  ; 写允许信号置 1
;此指令本身要执行 2 个时钟周期,此后尚使 CPU 暂停 2 个时钟周期
        ret
```

[例 4.4]　顺序读出片内 EEPROM 下一个地址单元中的数据,并存入 EEdrds 寄存器中。

```
        .def    EErtmp=r24                  ; 地址低字节暂存器
        .def    EErtmph=r25                 ; 地址高字节暂存器
        .def    EEdrds=r0                   ; 所读数据结果
EERDS:  sbic    EECR, EEWE                  ; 如果此前无写操作,
        rjmp    EERDS                       ; 那么这两条指令可取消
        in      EErtmp, EEARL               ; 读地址低字节
        in      EErtmph,EEARH               ; 读地址高字节
        adiw    EErtmp, 0x01                ; 地址递增一个单元
        out     EEARL, EErtmp               ; 输出新地址低字节
        out     EEARH, EErtmph              ; 输出新地址高字节
        sbi     EECR, EERE                  ; 读选通信号置 1
        in      EEdrds, EEDR                ; 读数据
        ret
```

[例 4.5]　把 R0～R15 中的 16 个数据写到片内 EEPROM 从 TABLE 地址单元开始的 16 个字节中去。

```
        .equ    TABLE= $0100
RESET:  ldi     r16, high (RAMEND)          ; 设置堆栈
        out     SPH, r16
        ldi     r16, low (RAMEND)
        out     SPL, r16
        clr     ZH                          ; Z 指向 R0
        clr     ZL
        ldi     EEwtmp, low (TABLE-1)       ; (TABLE-1)低字节
        ldi     EEwtmph, high (TABLE-1)     ; (TABLE-1)高字节
        out     EEARL, EEwtmp               ; EEARL← $FF
        out     EEARH, EEwtmph              ; EEARH← $00
LOOP:   ld      EEdwrs, Z+                  ; 读待写数据字节
        rcall   EEWRS                       ; 顺序写入
        cpi     ZL,16                       ; 全写完毕?
        brne    LOOP                        ; 否,循环
        ……                                  ; 是,程序继续前进
```

其实,用例 4.1 中的随机写入子程序也可进行顺序写入,似乎程序还简单一点。

[例 4.6]　用 EEWR 子程序实现例 4.5 的要求。

```
.equ    TABLE= $0100
RESET:  ldi     r16, high(RAMEND)          ; 设置堆栈
        out     SPH, r16
        ldi     r16, low (RAMEND)
        out     SPL, r16
        clr     ZH                         ; Z 指向 R0
        clr     ZL
        ldi     EEawr, low (TABLE-1)       ; EEawr← $ FF
        ldi     EEawrh, high (TABLE-1)     ; EEawrh← $ 00
LOOP1:  adiw    EEawr, 0x01                ; 目的地址递增 1 单元
        ld      EEdwr, Z+                  ; 读待写数据
        rcall   EEWR                       ; 进行写入
        cpi     ZL,16                      ; 全部写完？
        brne    LOOP1                      ; 否,再写下一字节
        ……                                 ; 是,程序继续前进
```

[例 4.7]　把片内 EEPROM 最前面的 10 个字节拷贝到 SRAM $0060 单元开始的 10 个地址单元中。

如果在例 4.5 或例 4.6 的程序之后,接着通过顺序读出子程序 EERDS 来进行这项工作,那么在主程序中向 EEAR 寄存器写入地址前,应查明 EEWE 位是否为 0。因为此前启动过写操作,且眼下不知写操作是否已经完成。但若用 EERD 子程序来读。则无此问题(见例 4.8)。

```
begin:  sbic    EECR, EEWE                 ; 上次写操作已完成？
        rjmp    begin                      ; 否,等待
        ldi     EErtmp, $ ff               ; 是,EEAR← $ ffff
        out     EEARH, EErtmp
        out     EEARL, EErtmp
        clr     ZH                         ; Z← $ 0060
        ldi     ZL, $ 60
loop2:  rcall   EERDS+2                    ; 顺序读出
        st      Z+, EEdrds                 ; 拷贝到 SRAM 中去
        cpi     ZL, $ 6a                   ; 十个字节拷贝完？
        brne    loop2                      ; 否,循环
        ……                                 ; 是,下略
```

[例 4.8]　用 EERD 子程序来实现例 4.7 的要求。

```
        clr     ZH                         ; Z← $ 0060
        ldi     ZL, $ 60
        mov     EEard, ZH                  ; EEard← $ 00
        mov     EEardh, ZH                 ; EEardh← $ 00
loop3:  rcall   EERD                       ; 读出数据
        st      Z+, EEdrd                  ; 存入 SRAM 区
        adiw    EEard, 0x01                ; EEardh：EEard 值加 1
        cpi     ZL, $ 6a                   ; 10 字节全读完？
```

```
        brne    loop3                    ; 否,循环
here:   rjmp    here                     ; 是,程序就地循环
```

4.2　端口的通用 I/O 操作

AVR 的端口结构比较新颖,其每条 I/O 线均可单独设定为输出或输入,输入又有三态高阻输入和内部上拉电阻输入两种形式。

由 1.2.15 一节得知,欲将某引脚安排作输入,则必须把该口数据方向寄存器之相应位写成 0。例如,把 DDRA 寄存器的 DDA7 位写成 0,即令 PA7 作输入。此后尚面临两种选择:

第一,若该口数据寄存器相应位为 1,则此输入引脚内部接有约 30～120 kΩ 的上拉电阻。因此,空脚时会读到输入高电平。一般情况下,若通过 3 kΩ 左右的电阻将此输入引脚从外部接地,便可读到逻辑 0 电平输入值。但如果外部信号源阻抗很大,那么内部上拉电阻与信号源内阻分压,这样尽管信号为 0,但也不能使引脚达到 0 电平。这就要求把引脚设定成三态高阻输入方式。

第二,若该口数据寄存器相应位为 0,则此引脚内部上拉电阻被切断,这便是三态高阻输入方式。在这种方式下,引脚通过大阻抗,例如 1 MΩ,接地,也可读到输入 0 电平。但此时高电平输入则应接到 V_{CC},或串接上拉电阻到 V_{CC} 端。

应当注意,不管哪种输入方式下,最好不要将引脚直接接到 V_{CC} 或 GND 作高电平或低电平输入,而中间应串接电阻。

如果某端口的 8 个引脚中,有的作输入,有的作输出,那么当执行读引脚指令(例如 INRd,PINA)时,所读结果将包括输入引脚上的外来电平以及输出引脚数据锁存器,例如 PORTA 中相应位之值。

把数据方向寄存器某位写成 1,相应端口引脚便被设定成输出方式。欲通过该引脚输出的数据应写入数据寄存器的相应位。例如,把 DDA7 写成 1,则 PORTA7 位的数据便可通过 PA7 引脚输送出去。输出方式下,内部上拉电阻是切断的。这样,当输出高电平时,该引脚可向外部提供 10 mA 的电流,而当输出低电平时,可承受外来的 20 mA 之灌电流。这样的驱动能力是其他同类器件无法比拟的。

AT90S4414/8515 有 40 个引脚,与 8051 端口引脚一一对应,A 口、B 口、C 口和 D 口分别对应 8051 的 P0、P1、P2 和 P3 口。而 AT90S1200/2313 仅有 20 个引脚,其中 I/O 脚 15 个,即 PB0～PB7 和 PD0～PD6,与 AT89C2051/1051 相对应。

[例 4.9]　如图 4.1 所示,AT90S1200 的 15 条 I/O 线均作输出,其外面各有一发光二极管 LED 通过 470 Ω 电阻与电源相连。现要求各 LED 按顺序依次发光,首先由 B 口最高位 PB7 开始,延迟一段时间后,PB7 熄灭,PB6 发光。待 PD0 外的第 15 只 LED 熄灭后再周而复始。

由图 4.1 可知,输出低电平的引脚可使其 LED 发光,而输出高电平者则令其 LED 处于熄灭状态。

```
.include"1200def.inc"
    rjmp    RESET
```

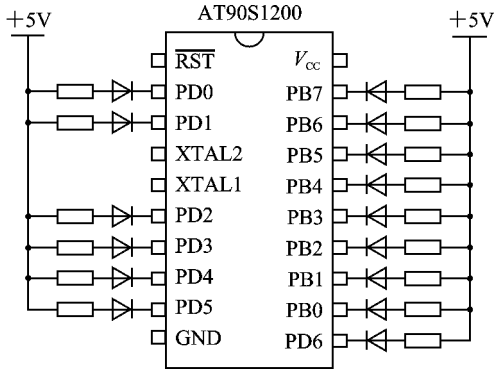


图 4.1 I/O 口作输出

```
        .org    $004
RESET:  ser     r16                 ; r16←$ff
        out     ddrb, r16           ; B 口设定为输出
        out     ddrd, r16           ; D 口设定为输出
        out     portb, r16          ; 输出高电平,关断 LED
        out     portd, r16          ; 输出高电平,关断 LED
        clc                         ; C←0
loop:   ror     r16                 ; 0 在 r16 中循环右移
        out     portb, r16          ; 点亮 B 口一 LED
        brcc    NXTPRT              ; 若 C=0,则转下一个端口
        rcall   delay               ; 延时
        rjmp    loop                ; C=1, 继续
NXTPRT: ror     r16                 ; B 口 8 只 LED 均亮过
loop1:  ror     r16                 ; PD6~PD0 依次
        out     portd, r16          ; 点亮其 LED
        brcc    loop                ; C=0, 再从 B 口开始
        rcall   delay               ; 延时
        rjmp    loop1               ; C=1, 循环
delay:  ldi     r17, $0a            ; r17 初值为 $0a
        clr     r18                 ; r18 初值为 0,即 256
        clr     r19                 ; r19 初值为 0,即 256
delay1: dec     r18                 ; 这两条
        brne    delay1              ; 为内环
        dec     r19                 ; 以下是中环
        brne    delay1
        wdr
        dec     r17                 ; 最后是外环
        brne    delay1
        ret
```

关于上段程序应当说明如下几点:

第一，ser r16 指令与 ldi r16，$ff 指令等效，它们的代码是一样的。

第二，AT90S1200 无片内 RAM，也不可外扩 SRAM，没有 PUSH 和 POP 指令。但是片内有三级硬件堆栈，供中断和子程序调用时由 CPU 自动使用。因此对于 1200 来说，不存在堆栈初始化问题。

第三，编写这样的程序时，初学者往往用寄存器来对 LED 的轮流发光进行计数。例如，计数值从 8 递减至 0 时，由 B 口 LED 切换至 D 口，而再由 7 递减至 0 时又切换回 B 口。其实，更简单的办法是每换一次 LED，便检查一下 C 值，看其是否为 0。若是，则说明 r16 的内容已为全 1，此时应从 B 口转向 D 口或相反。因为在 C 标志与寄存器 r16 的 9 位循环右移中，只有一个 0 参与，其他位均为 1，且在有关程序段中除 ror 指令外，无别的指令对 C 标志有影响。就是说，移至 C 中的数值可以保持至下次右移。

第四，在延时子程序中，dec r18（一个时钟）和 brne delay1（条件满足时两个时钟，不满足时一个时钟）两条指令为内环，共循环 256 次，总共耗时 256 * (1＋2)－1＝767 个时钟周期。dec r19 开始为中环，其执行时间为 256 * (1＋2＋767)－1＝197 119 个时钟周期。不难看出，整个子程序共延时 1 971 226 个时钟周期。若晶振频率为 4 MHz，则延迟时间约半秒钟。

[例 4.10]　AT90S1200 某系统有两项功能，其处理程序的起始地址分别为 FUNC0 和 FUNC1。上电时怎样实现此两项功能的自动选择呢？

为进行上电后的功能选择，可将 PD0 引脚设定为带内部上拉电阻的输入方式，它输入 0 时，CPU 执行 FUNC0 程序段，输入 1 时，CPU 执行 FUNC1 段。为输入 0 电平，可将引脚通过 3 kΩ 电阻由开关闭合接地。而当开关分断时，由于内部上拉电阻的作用，引脚输入为高电平。应当指出，开关的分断与闭合是在开机前进行的。开机后再动此开关并不起作用，因为程序只在上电复位后才查询引脚的输入值，其他时间不做此项工作。

```
        .include"1200def.inc"
          rjmp   RESET
        .org     $004
RESET:  ldi      r16, $7e          ; 设定 PD0
        out      ddrd, R16         ; 为输出
        ldi      r16, $01
        out      portd, r16        ; 接通内部上拉电阻
        sbic     pind,0            ; pind0 为 0，跨越
          rjmp   FUNC1             ; pind0 为 1，转 FUNC1
FUNC0:  ……                        ; pind0 为 0，执行 FUNC0 段
        ……
FUNC1:  ……
```

[例 4.11]　AT90S4414 的 B 口高 4 位用作带内部上拉电阻的输入引脚。如图 4.2 所示，它们各通过一 3 kΩ 电阻可以经开关闭合而接地。低 4 位为输出引脚，各接一 LED。要求由 4 个开关分别控制 4 个 LED 的发光与熄灭。即若 PB7 外开关闭合，则 PB3 外的 LED 发光；PB6 开关闭合，PB2 外的 LED 亮，如此等等。

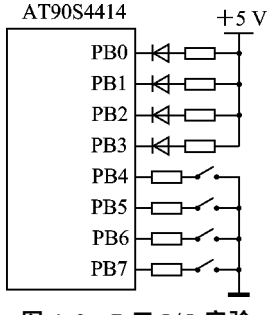


图 4.2　B 口 I/O 实验

```
        LDI     R16,$0F          ;B口高4位作输入
        OUT     DDRB,R16         ;低4位设定为输出
        LDI     R16,$FF          ;关闭LED并
        OUT     PORTB,R16        ;接通输入引脚内部上拉电阻
LOOP:   IN      R16,PINB         ;读输入引脚
        SWAP    R16              ;输入信息变成低4位
        SBR     R16,$F0          ;高4位置1
        OUT     PORTB,R16        ;按要求亮灯,维持输入上拉电阻
        RJMP    LOOP             ;循环
```

4.3 监视定时器的应用

这一节我们将利用上一节所讲述的概念做一个有关 WDT 的实验,然后再把该实验扩展成一个顺序控制的例子。

4.3.1 启用和中期管理

如 1.2.8 一节所述,WDT 是 MCU 安全运行的卫士。此外,监视定时器尚可在极需省电的掌上型仪器中大显身手。在这种情况下,启动过 WDT 后安排 MCU 进入休眠方式。待 WDT 溢出时,它将 MCU 从休眠方式下唤醒,于是后者开始检查 I/O 端口,看是否有输入信息要处理。若有,则处理;若无,则立即再度进入休眠方式。这样,可保持很低的功耗。不过,下面我们不研究这个问题,而是举例说明 WDT 如何启用,如何保证在正常情况下使之永不溢出。

[例 4.12] AT90S8515 的 4 个 I/O 口中,A 口的低 4 位为带上拉电阻的输入,其他 28 个 I/O 引脚均为输出。PA0～PA3 等 4 个输入脚各外接一 3 kΩ 电阻,且可经开关闭合而接地。28 个输出引脚各外接一 LED,并通过 470 Ω 电阻接 5 V 电源(可参见图 4.2)。现要求在监视定时器保护下 28 只 LED 轮流发光。

PA0(PINA0)引脚的作用在于根据其不同的输入值来对 WDT 进行不同的中期管理:若输入为 0,则每过约 25 ms 执行一次 WDR 指令使 WDT 复位;若输入为 1,则每过约 250 ms 执行一次 WDR 指令。WDT 的溢出时限设定为 30 ms。AT90S8515 的晶振频率等于 8 MHz。

```
        .include"8515def.inc"
        rjmp    RESET
        .cseg
        .org    0x10
RESET:  ldi     r16, low(RAMEND)     ;r16←$5f
        out     SPL, r16             ;堆栈指针低字节赋值
        ldi     r16, high (RAMEND)   ;r16←$02
        out     SPH, r16             ;堆栈指针高字节赋值
        wdr                          ;WDT使用前先复位
        ldi     r16,$09              ;WDTCR中的WDE置1,见1.2.8节
        out     WDTCR,r16            ;启动WDT,溢出时限30 ms,见表1.11
```

```
        ser     r16                 ; r16←$ff
        out     ddrb, r16           ; B口作输出
        out     ddrc, r16           ; C口作输出
        out     ddrd, r16           ; D口作输出
        ldi     r16, $f0            ; PA4～PA7作输出
        out     ddra, r16           ; PA0～PA3为输入
PA:     sec                         ; C←1, 拟输出$ef
        ldi     r16, $f7
LOOP:   rol     r16                 ; PA0～PA3带内部上拉电阻
        out     porta, r16          ; PA4～PA7轮流亮灯
        brcc    PB                  ; 若C=0,则转B口
        rcall   delay               ; C≠0,延时
        rjmp    LOOP                ; A口继续
PB:     rol     r16                 ; B口准备输出
        out     portb, r16          ; PB0～PB7轮流亮灯
        brcc    P_C                 ; 若C=0,则转C口
        rcall   delay               ; C≠0,延时
        rjmp    PB                  ; B口继续
P_C:    rol     r16                 ; C口准备输出
        out     portc, r16          ; PC0～PC7轮流亮灯
        brcc    PD                  ; 若C=0,则转D口
        rcall   delay               ; C≠0,延时
        rjmp    P_C                 ; C口继续
PD:     rol     r16                 ; D口准备输出
        out     portd, r16          ; PD0～PD7轮流亮灯
        brcc    PA                  ; 若C=0,则转A口
        rcall   delay               ; C≠0, 延时
        rjmp    PD                  ; D口继续
delay:  ldi     r17, $0a            ; 外环循环10次
        clr     r18                 ; 内环循环256次
        clr     r19                 ; 中环循环256次
delay1: dec     r18                 ; 内环时钟周期数=
        brne    delay1              ; 256*(1+2)-1=767
        dec     r19                 ; 中环延时时钟周期数=
        brne    delay1              ; 256*(1+2+767)-1
        sbis    PINA, 0             ; PINA0=1, 不执行wdr指令
        wdr                         ; PINA0=0, 执行wdr指令
        dec     r17                 ; 外环总延时
        brne    delay1              ; 约250 ms
        sbic    PINA,0              ; PINA0=0, 不执行wdr
        wdr                         ; PINA0=1, 执行wdr
        ret                         ; 子程序结束
```

当 PINA0 引脚外的开关闭合时，该引脚便通过电阻接地，于是输入为 0。在这种情况下，delay 子程序中每次中环结束时（约每过 25 ms）执行一次 wdr 指令，使 WDT 复位。但是 WDT 的溢出时限设定为 30 ms，故正常运行时，WDT 永远不会溢出，各 LED 轮流发光，情况正常。

而当开关分断时，因内部上拉电阻的作用，PINA0 引脚的输入值为 1。这样，每次 delay 子程序行将结束时（约延时 250 ms）才执行一次 wdr 指令使 WDT 复位。但在这之前，即延时到 30 ms 时，WDT 就溢出了。这要引起系统复位，使 MCU 从 $000 地址处重新执行程序。故各 LED 不会正常轮流发光。

4.3.2 顺序控制

如果各输出数据规律比较复杂，甚至无规律可循，且每步延时亦长短不一，那么，最好的办法是把这些输出值列成一表格写入程序存储器中。在程序运行过程中，用 LPM 指令一步一步地读出这些输出值，并通过端口发送给执行机构。这就是所谓顺序控制。

[例 4.13] AT90S8515 的 PA0～PA3 作带内部上拉电阻的输入引脚，其外部各通过一 3 kΩ 电阻经开关可以接地。其余 28 个 I/O 引脚均被设定为输出，它们外面各有一 LED 和 470 Ω 电阻与 5 V 电源相连。当 PINA0＝1 时，28 只 LED 一一轮流发光；PINA0＝0 时，按程序存储器中的表格进行顺序控制。晶振频率仍为 8 MHz。程序在 WDT 保护下运行。

```
        .include"8515def.inc"
        rjmp    RESET
        .cseg
        .org    0x10
RESET:  ldi     r16, low (RAMEND)
        out     SPL, r16
        ldi     r16, high(RAMEND)
        out     SPH, r16
        wdr
        ldi     r16, $09
        out     WDTCR, r16
        ser     r16
        out     ddrb, r16
        out     ddrc, r16
        out     ddrd, r16
        ldi     r16, $f0
        out     ddra, r16
```

以上程序段与例 4.12 程序开头部分完全一样。第一，它将堆栈设置在片内 RAM 的最高地址处，即栈底为 $025F。第二，启用 WDT，溢出时限设定为 30 ms。第三，B 口、C 口和 D 口作输出，A 口低 4 位为输入，高 4 位将工作于输出方式下。

```
        sbis    PINA,0                  ; PINA0＝1,跨越
        rjmp    TBLCTL                  ; PINA0＝0,转表格控制
PA:     sec
```

```
        ldi     r16,$f7          ;拟输出$ef,输入带上拉电阻
LOOP:   rol     r16              ;准备输出
        out     porta, r16       ;PA4～PA7轮流亮灯
        brcc    PB               ;若C=0,则转B口
        ldi     r17,$14          ;C≠0,延时约0.5 s
        rcall   DELAY
        rjmp    LOOP             ;A口继续
PB:     rol     r16              ;B口准备输出
        out     portb,r16        ;PB0～PB7轮流亮灯
        brcc    P_C              ;若C=0,则转C口
        ldi     r17, $14         ;C≠0, 延时约0.5 s
        rcall   DELAY
        rjmp    PB               ;B口继续
P_C:    rol     r16              ;C口准备输出
        out     portc, r16       ;PC0～PC7轮流亮灯
        brcc    PD               ;若C=0,则转D口
        ldi     r17, $14         ;C≠0, 延时约0.5 s
        rcall   DELAY
        rjmp    P_C              ;C口继续
PD:     rol     r16              ;D口准备输出
        out     portd,r16        ;PD0～PD7轮流亮灯
        brcc    PASS             ;若C=0,则转去检查输入
        ldi     r17,$14          ;C≠0,延时约0.5 s
        rcall   DELAY
PASS:   sbic    PINA, 0          ;若PINA0=0,则跨越
        rjmp    PA               ;PINA0=1,转A口
        ldi     r17,1            ;PINA0=0,延时后再查输入
        rcall   DELAY            ;延时25 ms
        sbic    PINA,0           ;再次检查输入,为0则改工作制式
        rjmp    PA               ;PINA0=1,转A口
TBLCTL: ldi     r31,$00          ;以下是顺序控制程序段
        ldi     r30,CONST*2      ;Z指向表格首地址第一个字节
GOON:   lpm                      ;读表格一字节
        out     porta, r0        ;由A口输出
        ld      r0, Z+           ;Z←Z+1, 亦可用adiw
        lpm                      ;读表格一字节
        out     portb, r0        ;由B口输出
        ld      r0, Z+           ;Z←Z+1
        lpm                      ;读表格一字节
        out     portc, r0        ;由C口输出
        ld      r0, Z+           ;Z←Z+1
        lpm                      ;读表格一字节
        out     portd, r0        ;由D口输出
```

```
        ld      r0, Z+          ; Z←Z+1
        lpm                     ; 读表中的延时参数
        mov     r17,r0          ;控制延时外环
        adiw    r30,2           ;Z←Z+2
        rcall   DELAY           ; 延时
        cpi     r30, $fe        ; Z= $fe?即表已读完?
        brne    GOON            ; 未完,继续顺序控制
        sbis    PINA,0          ; 已完,检查输入
        rjmp    TBLCTL          ; PINA0=0,顺控下个循环
        ldi     r17,1           ;PINA0=1,延时后再查输入
        rcall   DELAY           ; 延时 25 ms
        sbis    PINA,0          ; 再次检查输入
        rjmp    TBLCTL          ; PINA0=0, 顺控下个循环
        rjmp    PA              ; PINA0=1,改变控制方式
```

以上是顺序控制程序段。由控制数据列成的表格从 CONST 字地址单元开始,但为什么 Z 寄存器初始化时其内容等于 CONST * 2 呢?这在例 3.48 和例 3.4 中做过详细交代。简单说,就是把字地址变成字节地址。

从上面两段程序中可以看到,PINA0 外的开关在程序运行过程中可以闭合和分断,但程序不立即做出响应,而是等现行控制方式进行过一个完整的回合后,程序才去检查此开关的状态,即检查 PINA0 输入的是 0 还是 1。之所以延时 25 ms 后再检查此脚输入信息的原因,是要滤掉干扰信号。

下面是延时子程序,调用它之前,主程序必须为 R17 寄存器赋初值,即外环执行的次数。不同初值对应于不同的延迟时间。若 R17 初值为 1,则延时总计 25 ms 左右。

```
DELAY:  clr     r18             ; 内环循环 256 次
        clr     r19             ; 中环循环 256 次
DELAY1: dec     r18             ; 内环时钟周期数=
        brne    DELAY1          ; 256*(1+2)-1=767
        dec     r19             ; 中环延时=
        brne    DELAY1          ; 256*(1+2+767)-1
        wdr                     ; 使 WDT 复位
        dec     r17             ; 外环参数由主调程序控制
        brne    DELAY1
        ret
        .cseg
        .org    $070
CONST:  .db $ff, $81, $7e, $e7, 20
        .db $df, $7e, $81, $db, 20
        .db $bf, $bd, $db, $bd, 40
        .db $7f, $db, $bd, $7e, 40
        .db $9f, $e7, $e7, $e7, 80
```

4.4　定时/计数器 0 的应用

在 1.2.6 一节所述定时/计数器 T/C0 内部结构及运作原理的基础上，本节将例示 AVR 单片机是如何通过其 T/C0 实现定时和计数两项功能的。

4.4.1　定时功能的实现

在单片机控制系统中，往往要求每间隔一段固定时间进行一次某种操作，例如，进行一次数据采集，启停一次某个设备等等。这就要求单片机有定时功能。

[例 4.14]　AT90S1200 的晶振频率为 8 MHz，要求在 PB0～PB7 各引脚上均输出一个周期为 32 ms 的方波脉冲序列，并且相邻两引脚的脉冲波形是反相的。

前面几个例题所用延时方法的主要问题在于大量占用 CPU 的宝贵时间，使它在延时期间无暇做其他事情。但若改用定时中断法，则此类问题便可迎刃而解。

该 AT90S1200 的时钟周期为 1/8 MHz＝0.125 μs。由表 1.3 中可查得，如果向 T/C0 的控制寄存器 TCCR0 赋值 $05 的话，那么 T/C0 的时钟周期将等于 0.125 μs×1 024＝128 μs。这样，每过 16 ms/128 μs＝125 个时钟周期，让 T/C0 溢出一次，而在溢出中断服务程序中使 B 口各引脚输出反相，即可得所要求的方波脉冲序列。因此 TCNT0 的初始值应为 256－125＝131＝$83。

```
        .include"1200def.inc"
        .cseg
        .org    0X000
          rjmp  RESET
        .org    0X002                    ; 定时器 0 溢出中断向量
          rjmp  TOVF0                    ; 转向中断服务程序
        .org    0X004
RESET:  wdr                              ; 启用 WDT 前先令其复位
        ldi     r16, $09                 ; 拟启用 WDT
        out     WDTCR,r16                ; 溢出时限 30 ms
        ldi     r17, $83                 ; T/C0 初值＝－125
        out     TCNT0,r17                ; 定时间隔 16 ms
        ldi     r16, $05
        out     TCCR0, r16               ; 对 CPU 时钟进行 1 024 分频
        ldi     r16,$02                  ; 拟将 TOIE0 位置 1
        out     TIMSK,r16                ; 开放 TOV0 中断
        ldi     r16, $ff
        out     DDRB, r16                ; PB0～PB7 为输出
        ldi     r16,$55                  ; B 口输出初值
        out     PORTB,r16                ; B 口输出
        sei                              ; 开放总体中断
HERE:   rjmp    HERE                     ; 就地循环，等待中断
```

以下是中断服务程序。这里要做的事有:第一,恢复 TCNT0 的初值 $83;第二,将上次输出值取反后再次输出;第三,使 WDT 从 0 重新开始运作。

不难理解,每过 16 ms 中断一次,每次中断时输出反相,故输出方波脉冲周期为 32 ms。

```
TOVF0:  out   TCNT0,r17          ; 恢复 T/C0 初值
        com   r16                ; 输出值取反
        out   PORTB,r16          ; 输出
        wdr                      ; WDT 复位
        reti                     ; 中断返回
```

[例 4.15] AT90S1200 的晶振频率为 8 MHz,现要求 B 口外接的 8 只 LED(见图 4.1)每秒钟闪光一次,即亮暗各半秒钟,且各相邻 LED 应亮暗相间。

在 8 MHz 晶振频率之下,T/C0 的最大定时间隔为 0.125 μs×1 024×256=32.768 ms。而本例要求每 500 ms 中断一次,显然单靠 T/C0 定时是不够的。在这种情况下,用一个工作寄存器来对中断次数进行计数即可达到目的。

令 TCCR0 值等于 $03,即对 CPU 时钟进行 64 分频。故 T/C0 时钟周期为 0.125×64=8 μs。

向 TCNT0 赋初值 $06,则每过 250×8 μs=2 ms,T/C0 就要产生一次溢出中断。250 次中断当然要经历半秒钟了。

```
        .include"1200def.inc"
        rjmp    RESET
        .org    $002             ; T/C0 中断向量
        rjmp    TOVF0
        .org    $004
RESET:  wdr                      ; WDT 启用前先复位
        ldi     r16,$08          ; 拟启用 WDT
        out     WDTCR,r16        ; 溢出时限 15 ms
        ldi     r17, $06         ; T/C0 初值
        out     TCNT0, r17       ; 定时间隔 2 ms
        ldi     r16, 3           ; 拟对 CPU 时钟
        out     TCCR0, r16       ; 进行 64 分频
        ldi     r16,2            ; 拟将 TOIE0 位置 1
        out     TIMSK,r16        ;开始放 TOV0 中断
        ldi     r16,$ff          ; B 口方向设定值
        out     DDRB,r16         ; PB0~PB7 为输出
        ldi     r16,$55          ; B 口输出初值
        out     PORTB,r16        ; B 口输出
        sei                      ; 开总体中断
        ldi     r18, 250         ;中断次数初值
HERE:   rjmp    HERE             ; 就地循环等待中断
```

以下是中断服务程序。这里除恢复 TCNT0 初值和使 WDT 复位外,尚应检查是否已经中断 250 次了。若是,则应使 B 口输出反相,否则保持 B 口输出不变。这样,B 口各 LED 就可

亮半秒钟，而后暗半秒钟，如此周而复始。

```
TOVF0:  out   TCNT0,r17          ; 恢复 T/C0 初值
        wdr                      ; WDT 复位
        dec   r18                ; 中断次数减 1
        brne  TOMAIN             ; 250 次中断？
        com   r16                ; 是，B 口原输出反相
        out   PORTB,r16          ; 再次输出
        ldi   r18,250            ; 恢复中断计数初值
TOMAIN: reti                     ;中断结束
```

4.4.2　计数功能的实现

欲使 T/C0 对某外来脉冲序列进行计数，必须将此外脉冲源接至 T/C0 的输入引脚 T0 上（对 AT90S1200 来说，这是 PD4 引脚，而对 AT90S4414/8515 来说，这是 PB0 引脚），且把 T0 设定为输入方式，接不接通内部上拉电阻视情况而定。在无外脉冲源的情况下，若将 T0 设定为输出方式，则 T/C0 可对 T0 引脚上输出的脉冲序列进行计数。

［例 4.16］　AT90S8515 的 T0 引脚上有一个频率相对较低的脉冲序列输入，要求对其 10 分频并整形成方波序列，由 PD4 引脚输送出去。

借助于 T/C0 的计数功能实现对外来脉冲的分频与整形，可包含下列几个要点：

- T0 引脚设定作输入，另一引脚为输出，用来发送分频并整形后的脉冲序列；
- TCNT0 初值等于 256－分频因子/2；
- TCCR0 寄存器赋值 $06（外脉冲下降沿有效）或 $07（上升沿有效）；
- 开总体中断和 T/C0 溢出中断；
- 在中断服务程序中恢复 TCNT0 初值并使输出反相。

```
        .include"8515def.inc"
          rjmp  RESET
        .org    $007             ; TOV0 中断向量
          rjmp  TOVF0
        .org    $010
RESET:  ……                       ; 堆栈设置，略
        CBI     DDRB,0           ; PB0(T0)作输入
        SBI     DDRD,4           ; PD4 作输出
        LDI     R17,$FB          ; 256－分频因子/2
        OUT     TCNT0,R17        ; TCNT0 赋初值
        LDI     R16,$07          ; 脉冲上升沿有效
        OUT     TCCR0,R16        ; 启动计数过程
        LDI     R16,$02          ;TOIE0 位置 1
        OUT     TIMSK,R16        ; 开 TOV0 中断
        SEI                      ; 开总体中断
HERE:   RJMP    HERE             ; 就地循环
TOVF0:  OUT     TCNT0,R17        ; 恢复 TCNT0 初值
```

```
        IN      R0, PORTD          ; 读回 D 口上次输出值
        LDI     R16,1≪PD4          ; R16←0b00010000
        EOR     R0, R16            ; PD4 位取反后,
        OUT     PORTD,R0           ; 再次输出,其他位不变
        RETI                       ; 中断返回
```

从符号地址 TOVF0 开始的中断服务程序也可改成下列形式：

```
TOVF0:  out     tcnt0, r17         ; 恢复 tcnt0 初值
        sbic    portd, 4           ; 若 portd4=0,则跨越
        rjmp    one                ; portd4=1, 跳转
        sbi     portd, 4           ; 原 portd4=0, 现应置 1
        reti                       ; 中断返回
one:    cbi     portd,4            ; 原 portd4=1, 现应清 0
        reti                       ; 中断返回
```

[例 4.17] 如何使 T0 作外部中断输入引脚？

实际上这是 T/C0 计数功能的一个特例。此例的关键就是把 TCNT0 的初值设定为 $FF,这样,T0 引脚上有一个正跳变或负跳变,便可使 TCNT0 值加 1 后溢出,由 TOV0 标志去请求中断。

下面我们做这样一个例题：T0 引脚上若有信号正跳变(例如,通过电阻和开关接地,后突然将开关分断),则 PD4 引脚便发出一宽度为 4 个时钟周期的正脉冲,去启动或停止某个设备。

```
RESET:  ……
        CBI     DDRB,0             ; T0 设定为输入
        SBI     PORTB,0            ; 接通内部上拉电阻
        SBI     DDRD,4             ; PD4 为输出
        SER     R17                ; R17←$FF
        OUT     TCNT0,R17          ; TCNT0 初值为-1
        LDI     R16,$07            ; T/C0 计数
        OUT     TCCR0,R16          ; 上升沿有效
        LDI     R16,1≪TOIE0;       ; R16←$02
        OUT     TIMSK,R16          ; 开 TOV0 中断
        SEI                        ; 开总体中断
HERE:   RJMP    HERE               ; 就地循环
TOVF0:  OUT     TCNT0,R17          ; 恢复 TCNT0 初值
        SBI     PORTD,4            ; PD4 置 1
        NOP                        ; 延时一个时钟周期
        NOP                        ; 延时一个时钟周期
        CBI     PORTD,4            ; PD4 清 0
        RETI                       ; 中断结束
```

4.5　定时/计数器 1 的应用

T/C1 是 AVR 单片机中功能多样、应用颇广的一个重要部件。它除具有定时与计数功能外，尚在输出比较、输入捕捉和脉冲宽度调制等方面大有作为。

4.5.1　定时功能应用实例

T/C1 是一个 16 位定时器，其最大定时范围为 2^{26} 个 CPU 时钟周期，若晶振频率为 8 MHz，则最大定时范围可达 8 秒以上。

[例 4.18]　用 T/C0 定时，每 8 ms 中断一次，以 T/C1 对 T/C0 的中断间隔进行计时，要求分辨率为 1 μs。

对 8 MHz 晶振进行 256 分频后的信号作为 T/C0 的时钟，其周期等于 0.125 μs×256＝32 μs。由表 1.3 查得，此时 TCCR0 应赋值 $04。欲得 8 ms 定时间隔，TCNT0 初值应为 256－8 000μs/32 μs＝6。

对晶振频率 8 分频后得 1 MHz 信号作 T/C1 的时钟，从表 1.4 中查得，TCCR1B 寄存器值应等于 $02。用 T/C1 记录时间，其 TCNT1 初值最好为复位值 $0000。这样安排，其计时范围最大可达 65 ms，分辨率为 1 μs。

```
        .include"8515def.inc"
        RJMP    RESET                   ; 转复位处理
        .ORG    $007                    ; T/C0 溢出中断向量
        RJMP    TOVF0
        .ORG    $010
RESET:  ……                              ; 堆栈设置，略
        SER     R16                     ; R16←$FF
        OUT     DDRB,R16                ; B 口作输出
        OUT     DDRD,R16                ; D 口作输出
        OUT     PORTB,R16               ; B 口输出全 1
        OUT     PORTD,R16               ; D 口输出全 1
        WDR                             ; WDT 复位
        LDI     R16,$08                 ; WDE 位将置 1
        OUT     WDTCR,R16               ; 启用 WDT
        LDI     R17,$06                 ; 以 $06 作
        OUT     TCNT0,R17               ; TCNT0 初值
        LDI     R16,$04                 ; 对晶振频率 256 分频
        OUT     TCCR0,R16               ; 启动 T/C0
        LDI     R16,$02                 ; TOIE0 位将置 1
        OUT     TIMSK,R16               ; 开 T/C0 溢出中断
        SEI                             ; 开总体中断
        LDI     R18,2                   ; 准备启动 T/C1
HERE:   RJMP    HERE                    ; 就地循环
```

以下是定时器 0 的中断服务程序。第一次中断后立即启动 T/C1，使之开始计时。第二次中断后则马上使其停止运作。随即读出其计时值(先读低字节 TCNT1L)，取反后由 B 口和 D 口输出出去。B 口为高 8 位，D 口为低 8 位。LED 亮的位代表 1，暗者为 0。这样就可计算出 T/C0 两次中断间隔 8 ms 究竟有多大误差。

```
TOVF0:  OUT   TCCR1B,R18      ;8 分频,启动 T/C1,或使 T/C1 停运
        OUT   TCNT0,R17       ; 恢复 TCNT0 初值
        WDR                   ; WDT 复位
        TST   R18             ; R18=0 ?
        BRNE  NONZ            ; 否,转至 NONZ
        IN    R24,TCNT1L      ; 是,读 TCNT1 低字节
        IN    R25,TCNT1H      ; 读 TCNT1 高字节
        COM   R24             ; 低字节取反
        OUT   PORTD,R24       ; 由 D 口输出
        COM   R25             ; 高字节取反
        OUT   PORTB,R25       ; 并由 B 口输出
        RETI                  ; 中断返回
NONZ:   CLR   R18             ; 下次关闭 T/C1
        RETI                  ; 中断返回
```

[例 4.19] AT90S8515 的晶振频率为 8 MHz，要求每过 6 秒钟在其 PD4 引脚上输出一宽度约等于 1 秒钟的负脉冲。

若把晶振频率 1 024 分频后的信号作为 T/C1 的时钟，则该时钟周期为 0.125 μs×1 024 =128 μs。由表 1.4 可知，此时 TCCR1B 寄存器应赋值 $05。

欲得 6 s 的中断间隔，TCNT1=65 536−6s/128 μs=18 661；

欲得约 1 s 的中断间隔，TCNT1=65 563−1 s/128 μs≈57 724。

```
        .include"8515def.inc"
        .cseg
        .org    $000          ; 复位向量
          rjmp  RESET
        .org    $006          ; T/C1 溢出中断向量
          rjmp  TOVF1
        .org    $010
RESET:  ……                    ; 堆栈设置,略
        SBI     DDRD,4        ; PD4 作输出
        SBI     PORTD,4       ; PD4 输出高电平
        LDI     R16,HIGH(18661)  ; TCNT1 初值高字节
        OUT     TCNT1H,R16
        LDI     R16,LOW(18661)   ; TCNT1 初值低字节
        OUT     TCNT1L,R16    ; 赋值,定时间隔 6 s
        LDI     R16, $05      ; 对晶振 1 024 分频
        OUT     TCCR1B,R16    ; 选定时钟并启动 T/C1
        LDI     R16,1≪TOIE1   ; TOIE1 位置 1,
```

```
        OUT     TIMSK,R16                ; 开 TOV1 中断
        SEI                              ; 开总体中断
        CLR     R0                       ; R0 作中断特点标志
HERE:   RJMP    HERE                     ; 就地循环
```

我们用 R0 寄存器作标志，若其最低位为 1，则说明自上次中断至这次中断的间隔为 6 s，就是说至下次中断应间隔 1 s。因此，TCNT1 初值应对应 1 s 间隔，PD4 此次应输出 0。如果 R0 之最低位为 0，那么 TCNT1 应赋对应 6 s 之初值，且 PD4 应输出 1。以下是中断服务程序：

```
TOVF1:  INC     R0                       ; 特征寄存器值增 1
        SBRC    R0,0                     ; 若 R0 最低位为 0 则跨越
        RJMP    ONE                      ; 最低位为 1，转 6 s 中断
        LDI     R16,HIGH(18661)          ; 最低位为 0，
        OUT     TCNT1H,R16               ; 刚才是 1 s 中断
        LDI     R16,LOW(18661)           ; TCNT1 初值
        OUT     TCNT1L,R16               ; 应对应 6 s 中断
        SBI     PORTD,4                  ; PD4 输出 1
        RETI                             ; 中断返回
ONE:    LDI     R16,HIGH(57 724)         ; R0 最低位为 1，
        OUT     TCNT1H,R16               ; 刚才是 6 s 中断，
        LDI     R16,LOW(57 724)          ; TCNT1 初值，
        OUT     TCNT1L,R16               ; 应对应 1 s 中断
        CBI     PORTD,4                  ; PD4 输出 0
        RETI                             ; 中断返回
```

如果 PD4 外接有 LED 的话，那么执行这段程序时，就会看到 LED 暗 6 秒钟后亮 1 秒钟，再暗 6 秒，再亮 1 秒，如此循环不已。

4.5.2　输出比较功能的应用

前面例题中对输入脉冲信号进行分频和产生一定频率的方波脉冲序列问题，若用 T/C1 来解决会更简单，甚至初始化后不要中断即可达到目的。

[例 4.20]　AT90S8515 的 T1(PB1)引脚上有一输入脉冲序列，要求对其 10 分频并整形成方波序列，最后由 OC1A(PD5)引脚输送出去。

在例 4.16 中，我们曾用 T/C0 解过这个问题，使用计数溢出产生中断的办法恢复 TCNT0 初值并使输出反相。这里我们用 T/C1，只有初始化程序，而无须中断即可一劳永逸地坐等结果。

在有关初始化程序段中应包括如下几点：

- 将 T1(PB1)引脚设定为输入，OC1A(PD5)作输出；
- TCNT1 寄存器初值应等于 $0000；
- TCCR1A 寄存器应赋值 $40，即 A 匹配时 OC1A 输出反相(见表 1.5)；
- OCR1A 寄存器值应等于分频因子/2，即 $0005；
- TCCR1B 寄存器应赋值 $0F，输入脉冲上升沿触发计数，使 TCNT1 值增 1，A 匹配时

TCNT1 自动清 0(见表 1.4)。

```
cbi     ddrb, 1                 ; pb1 为输入
sbi     ddrd,5                  ; pd5 作输出
ldi     r16, $40                ; A 匹配时,
out     tccr1a, r16             ; 使 pd5 输出反相
ldi     r16,high(10/2)          ;r16← $00
out     ocr1ah,r16              ; ocr1ah← $00
ldi     r16, low(10/2)          ; r16← $05
out     ocr1al, r16             ; ocr1al←分频因子/2
ldi     r16, $0f                ; 外来脉冲上升沿有效
out     tccr1b,r16              ; A 匹配时 tcnt1 清 0
```

在这段程序中,TCNT1 初值为 $0000,可不赋值,因为 MCU 复位后 TCNT1 即等于此值。

图 4.3 所示为 T1(PB1)、OC1A(PD5)引脚上的信号波形及 TCNT1 的计数值变化情况。图中 TCNT1 值的增长应是台阶式的,这里简化成了一条斜线。T1 引脚上的输入脉冲之高低电平段都不得小于一个 CPU 时钟周期。

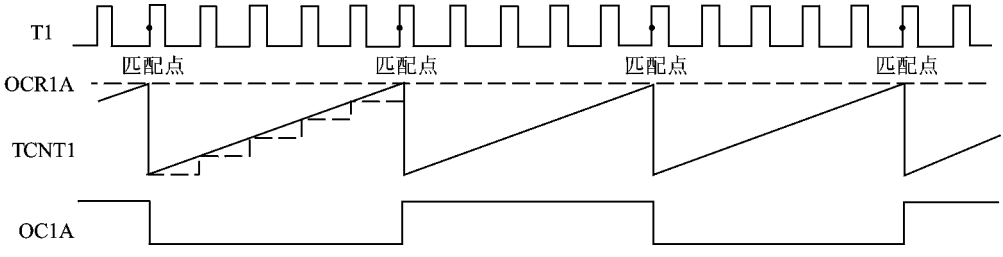


图 4.3 输出比较的分频功能

[例 4.21] AT90S8515 的晶振频率为 8 MHz,要求在 OC1A 引脚上得到一个如图 4.4 所示波形的输出信号序列。

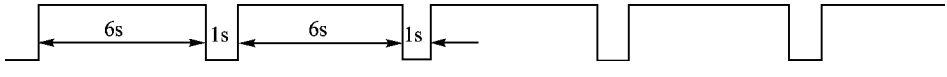


图 4.4 例 4.21 输出信号波形

在例 4.19 中我们曾经做过这样的题目,那是用 T/C1 中断时修改 TCNT1 初始值的办法来得到高低电平不同时段的。不难理解,T/C1 只能为此单一任务服务。

这里我们改用输出比较功能来做此题。在这种方式下,T/C1 可服务于以其为时基的多项任务,故不便再使用 A 匹配时 TCNT1 值自动清 0 的性能。初始化程序的设计思路归纳如下:

- PD5(OC1A)作输出引脚;
- A 匹配时输出低电平,TCCR1A 寄存器应赋值 $80(见表 1.5);
- 输出比较寄存器 OCR1A 值应等于 6 s/128 μs=46 875;
- TIMSK 寄存器之 OCIE1A 位置 1,开放 T/C1 输出比较 A 匹配中断;

● TCCR1B 寄存器值应等于 $05，对 CPU 时钟 1 024 分频，T/C1 时钟周期为 128 μs（见表 1.4）。

```
        .include"8515def.inc"
          rjmp   RESET
        .org    $004                          ; 输出比较 A 匹配中断向量
          rjmp   COMPA
        .org    $010
RESET:  ……                                   ; 堆栈设置，略
        SBI     DDRD,5                        ; PD5(OC1A)引脚作输出
        LDI     R16,$80                       ; A 匹配时，
        OUT     TCCR1A,R16                    ; OC1A 将输出低电平
        LDI     R16,HIGH(6*10^6/128)
        OUT     OCR1AH,R16                    ; 输出比较寄存器赋初值，
        LDI     R16,LOW(6*10^6/128)
        OUT     OCR1AL,R16                    ; 使 6 s 后达到 A 匹配
        LDI     R16,$05                       ; 拟对 CPU 时钟 1 024 分频，
        OUT     TCCR1B,R16                    ; 并启动 T/C1
        LDI     R16,1<<OCIE1A
        OUT     TIMSK,R16                     ; 开放 A 匹配中断
        SEI                                   ; 开放总体中断
        SET                                   ; 置 6 s 后 A 匹配标志
HERE:   RJMP    HERE                          ; 就地循环
```

在中断服务程序中，应首先判断此次中断系 6 s 后 A 匹配所致，抑或 1 s 后 A 匹配所引起。若系 6 s 后 A 匹配所致，则应在 OCR1A 寄存器原值基础上增加 1 s/128 μs，并使下次 A 匹配时 OC1A 输出高电平。如果是 1 s 后 A 匹配所引起的此次中断，那么，在中断服务程序中应将 OCR1A 值加 6 s/128 μs，且在下次 A 匹配时令 OC1A 输出低电平。

应当指出，在初始化时，我们设置了 6 秒时间，A 匹配时将输出 0 电平，并把 T 位置成了 1。下面是中断服务程序：

```
COMPA:  BRTS    TO1S                          ; 若 T=1，则转去 TO1S
        RCALL   ADD6S                         ; T=0，OCR1A 值增 6 s/128 μs
        SET                                   ; 将 T 置 1
        RETI                                  ; 中断返回
TO1S:   LDI     R24,LOW(1*10^6/128)           ; OCR1A 值将增 1 s/128 μs
        LDI     R25, HIGH(1*10^6/128)
        LDI     R16,$C0                       ; 下次 A 匹配时输出高电平
        RCALL   ADD1S                         ; T=1，OCR1A 增值 1 s/128 μs
        CLT                                   ; T 清 0
        RETI
ADD6S:  LDI     R24,LOW(6*10^6/128)
        LDI     R25,HIGH(6*10^6/128)
        LDI     R16,$80                       ; 6 s 后 A 匹配时 OC1A 输出 0
```

```
ADD1S:  OUT   TCCR1A,R16        ; 6 s 后 OC1A 输出 0,1 s 后输出 1
        IN    R1,OCR1AL         ; 读比较寄存器低字节
        IN    R2,OCR1AH         ; 读比较寄存器高字节
        ADD   R1,R24            ; 比较寄存器将增值
        ADC   R2,R25
        OUT   OCR1AH,R2         ; 向比较寄存器
        OUT   OCR1AL,R1         ; 输出新值
        RET
```

4.5.3 输入捕捉功能的应用

输入捕捉功能是进行脉冲测量的得力工具,它可用来对 ICP 引脚上输入的脉冲信号进行宽度、周期和频率等参数的测量。例如,以 ICR1 寄存器首先捕捉脉冲上升沿发生的时间,接着再捕捉下降沿发生的时间,那么两次捕捉值之差便是以 T/C1 时钟周期为单位的脉冲宽度值。若先后两次捕获到相邻两上升沿所发生的时间,则可得脉冲周期值。

[例 4.22] AT90S8515 的晶振频率为 8 MHz,要求测出 ICP 引脚上的输入脉冲宽度,分辨率为 1 μs。

根据题义要求,应对 8 MHz 晶振进行 8 分频,得 1 MHz 脉冲源作 T/C1 时钟,其周期,即对输入捕捉时间的分辨率为 1 μs。为测脉冲宽度,首先要定义上升沿作触发事件,即 TCCR1B 寄存器的 ICES1 位设定为 1。此外,为增加捕捉的可靠性,最好启用噪音消除功能,即 ICNC1 位置 1。这样,TCCR1B 应赋初值 $C2。

```
        .include"8515def.inc"
        rjmp    RESET
        .def    TMP=R16
        .def    FLAG=R0
        .def    TMP1=R1
        .def    TMP2=R2
        .def    TMP3=R3
        .def    TMP4=R4
        .cseg
        .org    $003            ; 输入捕捉中断向量
        rjmp    CAPT
        .org    $010
RESET:  ……                      ; 堆栈设置,略
        WDR                     ; WDT 启用前先清 0
        LDI     TMP,$0A         ; 溢出时限 60 ms
        OUT     WDTCR,TMP       ; 启用 WDT
        LDI     TMP,$C2         ; 除噪音,上升沿触发
        OUT     TCCR1B,TMP      ; 8 分频,启动 T/C1
        LDI     TMP,$08         ; TICIE1=1
        OUT     TIMSK,TMP       ; 开输入捕捉中断
        CLR     FLAG            ; 清标志
```

```
        CLR   ZH                ; Z 寄存器初值
        LDI   ZL,$60            ; 等于 $0060
        SEI                     ; 开总体中断
HERE:   RJMP  HERE              ; 就地循环
```

在下面的中断服务程序中，读出上升沿所发生的时间后，改设定下降沿作触发事件。若刚才的中断是由下降沿触发的，则读出捕获时间值后，再次设定上升沿作触发事件。两次捕获值之差便是所求脉冲宽度。我们做十次连续测量，将结果存入片内 SRAM 从 $0060 至 $0073 的 20 个地址单元中。此后令 WDT 和 T/C1 均停止运行。

最后应当指出，某次输入捕捉中断是由上升沿还是下降沿触发的，可通过 FLAG 寄存器的最低位来判断。若该位为 0，则系上升沿触发；反之则为下降沿触发。

```
CAPT:   WDR                     ; WDT 复位
        SBRC  FLAG,0            ; 系上升沿触发？
        RJMP  CAP2              ; 否，转下降沿触发处理
        IN    TMP1,ICR1L        ; 是，读捕获值低字节
        IN    TMP2,ICR1H        ; 读捕获值高字节
        INC   FLAG              ; 置下降沿触发标志
        LDI   TMP,$82           ; ICES1=0
        OUT   TCCR1B,TMP        ; 设定下降沿触发
        RETI
CAP2:   IN    TMP3,ICR1L        ; 下降沿触发处理
        IN    TMP4,ICR1H        ; 读捕获值
        INC   FLAG              ; 置上升沿触发处理
        LDI   TMP,$C2           ; ICES1=1
        OUT   TCCR1B,TMP        ; 设定上升沿触发
        SUB   TMP3,TMP1         ; 求脉宽低 8 位
        SBC   TMP4,TMP2         ; 求脉宽高 8 位
        ST    Z+,TMP3           ; 将测量结果
        ST    Z+,TMP4           ; 存入片内 SRAM
        CPI   ZL,$74            ; 最后一次结果？
        BRNE  TOMAIN            ; 否，准备返回
        LDI   TMP,$1A           ; 是，准备停 WDT
        OUT   WDTCR,TMP         ; WDTOE=WDE=1
        LDI   TMP,$12           ; WDTOE=1,WDE=0
        OUT   WDTCR,TMP         ; 停止 WDT 运作
        CLR   TMP               ; 准备停 T/C1
        OUT   TCCR1B,TMP        ; T/C1 停止运行
TOMAIN: RETI                    ; 中断结束
```

这段程序所测量的脉冲参数范围应如何考虑？首先，其高低电平段的持续时间均不得超过 60 ms。测量结果为 16 位，分辨率，即最小单位等于 1 μs，故测量结果最大值为 65.536 ms。但我们设置的 WDT 溢出时限等于 60 ms，且除中断服务程序外，别无他处有 WDR 指令。因此，脉冲两相邻边沿之间隔不得超过 60 ms，否则脉冲边沿尚未到来，WDT 即溢出，引起系统

复位，导致测量无法进行。

被测脉冲两相邻边沿之间隔不得小于中断服务程序的执行时间和中断响应时间之和。在我们的具体情况下，此参数约为 2.5 μs 左右。

如果要求有更高的测量分辨率，那么就应提高 T/C1 的时钟脉冲。例如，若 T/C1 的时钟频率为 8 MHz，则其分辨率等于 125 ns。

[例 4.23] AT90S8515 的晶振频率为 8 MHz，要求测出 ICP 引脚上的输入脉冲周期，分辨率不得低于 200 ns。

脉冲周期测量比宽度测量稍稍简单一点。前后两次捕捉相邻二上升沿所发生的时刻，求差值即可得到以 T/C1 时钟周期为单位的被测脉冲周期值。

根据测量分辨率的要求，选晶振作 T/C1 时钟，即分频因子为 1。此时 TCCR1B 寄存器应赋值 $C1，最大测量范围等于 0.125 μs×$2^{16}$=8.912ms。如果被测周期不超过此值，那么下段程序就是非常简明的了。

如图 4.5 所示，第二次上升沿捕获值 CAP2 与第一次捕获值 CAP1 之差 $T1$，第三次捕获值 CAP3 与 CAP2 之差 $T2$ 均为所测之周期值。这样，根据 11 个捕获值即可求得 10 个周期值，最后存入片内 SRAM 自 $0070 地址单元开始的 20 个字节中。$006E 和 $006F 两字节数据是无用的。

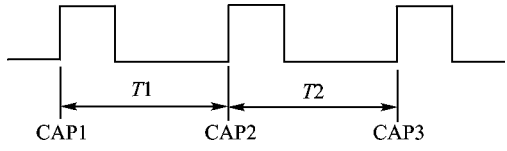


图 4.5 周期连续测量

```
        .include"8515def.inc"
        rjmp    RESET
        .org    $003
        rjmp    CAPT
        .org    $010
RESET:  ……                          ;堆栈设置，略
        WDR                         ;WDT 用前先清 0
        LDI     R16,1≪WDE           ;溢出时限 15 ms
        OUT     WDTCR,R16           ;启用 WDT
        LDI     R16,$08             ;TICIE1=1
        OUT     TIMSK,R16           ;开输入捕捉中断
        CLR     ZH                  ;建立 Z 指针
        LDI     ZL,$70-2            ;Z←$70-2
        LDI     R16,$C1             ;除噪音，上升沿触发
        OUT     TCCR1B,R16          ;时钟 8 MHz，启动 T/C1
        SEI                         ;开总体中断
HERE:   RJMP    HERE
CAPT:   IN      R2,ICR1L            ;读捕获值低字节
        IN      R3,ICR1H            ;读捕获值高字节
        SUB     R2,R0               ;求周期低字节
        SBC     R3,R1               ;求周期高字节
```

```
        IN      R0,ICR1L                ; 本次捕获值
        IN      R1,ICR1H                ; 作下次减数
        ST      Z+,R2                   ; 所得周期值
        ST      Z+,R3                   ; 存入片内 SRAM
        WDR                             ; WDT 复位
        CPI     ZL,$84                  ; 是第 11 次捕捉吗？
        BRNE    TOMAIN                  ; 否，准备返回
        LDI     R16,$18                 ; 是，准备停 WDT
        OUT     WDTCR,R16               ; WDTOE=WDE=1
        LDI     R16,$10                 ; WDTOE=1,WDE=0
        OUT     TCCR1B,R16              ; 停 WDT
        CLR     R16                     ; 准备停 T/C1
        OUT     TCCR1B,R16              ; T/C1 停止运行
TOMAIN: RETI                            ; 中断结束
```

4.5.4　频率测量

从原理上讲，脉冲频率可通过脉冲周期的测量结果计算出来。例如，测量 N 个周期所用时间为 T，那么被测脉冲频率等于 N/T。大家知道，这样的除法运算是 AVR 微控制器所不擅长的。第二个问题是，用捕捉中断的方法测量脉冲周期，如前所述，相邻两次捕捉操作间的间隔不得小于中断响应时间与中断服务程序执行时间之和。就是说，过短的周期，或者说偏高的频率很难测得出来。例如，在 8 MHz 晶振的情况下，若中断服务程序有 3～5 条指令，则高至 1 MHz 的脉冲频率就难以测得。

改用定时计数法测脉冲频率，则上述两个问题便不复存在。假如用 T/C1 对被测脉冲从 0 开始进行计数，同时 T/C0 做 1 ms 的精确定时，那么，不经任何计算，即可得到以 kHz 为单位的频率值。或者定时 2 ms、4 ms、8 ms 等，对计数结果进行简单计算便可得到频率值。用这种方法所测脉冲的最高频率可达 MCU 主频的一半。

[例 4.24]　AT90S8515 所用晶振频率为 8 MHz，要求用 T/C1 和 T/C0 配合测出 T1(PB1)引脚上的输入脉冲频率。

对 8 MHz 晶振进行 1 024 分频，得周期等于 128 μs 的脉冲信号，以其作 T/C0 的时钟。若 TCNT0 初值等于 131，则经过 125 个时钟周期，即 16 ms，T/C0 便要溢出。由表 1.3 查得，此时 TCCR0 寄存器值应等于 $05。

T/C1 设定为计数方式，上升沿有效。由表 1.4 得知，TCCR1B 寄存器应赋值 $07。

```
        .include"8515def.inc"
          rjmp  RESET
        .org    $007
          rjmp  TOVF0
        .org    $010
RESET:  ……
        clr     r16                     ; r16 清 0
        out     tcnt1h,r16              ; tcnt1 将从 0
```

```
        out     tcnt1l,r16                  ;开始计数
        cbi     ddrb,1                      ; T1 引脚作输入
        ldi     r16,256-125                 ; 过 125 个周期,
        out     tcnt0,r16                   ; T/C0 将溢出
        ldi     r16,$07                     ; T/C1 计数,上升沿有效
        ldi     r17,$05                     ; 对 MCU 时钟 1 024 分频
        out     tccr1b,r16                  ; 启动 T/C1 计数
        out     tccr0,r17                   ; 启动 T/C0 定时
        ldi     r16, 1<<TOIE0
        out     timsk, r16                  ; 开 TOV0 中断
        sei                                 ; 开总体中断
        ……
TOVF0:  clr     r16
        out     tccr0,r16                   ; T/C0 停止运行
        out     timsk,r16                   ; 关 TOV0 中断
        in      r24,tcnt1l                  ; 读计数结果低 8 位
        in      r25,tcnt1h                  ; 读计数结果高 8 位
        reti
```

寄存器对 R25：R24 最后的内容是在 16 ms 期间所记录的脉冲个数。该寄存器对的高 12 位(即 R25 的 8 位和 R24 的高 4 位)等于被测脉冲频率以 kHz 为单位的整数值,而最低 4 位(R24 的低 4 位)则是其小数值。

4.5.5 脉宽调制

AVR 的 T/C1 除具有定时、计数、输入捕捉和输出比较功能外,尚可构成两个脉冲宽度调制 PWM 输出通道。

经缓冲的 PWM 输出可用来驱动直流马达,马达转速将正比于 OCR1A 或 OCR1B 寄存器的内容。OC1A 和 OC1B 两引脚的 PWM 输出也可配置作双通道的 D/A 转换器。在这种应用场合,PWM 输出必须与常规运算放大电路相结合。如果要求输出电压结果很精确,就应该用本身带模拟供电的外部缓冲器对 PWM 输出首先进行缓冲,使矩形脉冲的高低电平接近 5 V和 0 V,然后再接运算放大器。例如,PWM 输出先经 CD4049 缓冲,再由 RC 滤波,最后通过高阻放大器输出,就可获得具有相当精度的 D/A 转换结果。

下面我们研究一个 T/C1 用于 PWM 输出的具体例子。

[例 4.25] AT90S4414 的晶振频率为 8 MHz,要求在其 OC1B 引脚上输出一个 122.3 Hz的方波脉冲序列,而在 OC1A 引脚上输出一个与此同频率但占空比阶梯式跳变的脉冲序列。占空比的跳变幅度上升和下降过程中均为 10%左右,每个占空比值保持 0.4 s。

可用 64 分频后的晶振信号(8 MHz/64=125 kHz)作 T/C1 时钟。此时 TCCR1B 寄存器应赋值 $03(见表 1.4)。若令 PWM 的分辨率为 9 位,则 PWM 频率为 125 kHz/1 022=122.3 Hz(见表 1.7)。两路 PWM 均选正向输出,从表 1.6 和表 1.8 中查得,TCCR1A 寄存器值应等于 0b10100010=$A2。

输出比较寄存器 OCR1B 赋值 $0100 即可得到占空比等于 50%的方波脉冲序列。

占空比变化的 10 个台阶通过修改输出比较寄存器 OCR1A 的内容来产生。修改 OCR1A 值可用计算法、查表法等手段进行。

计算法最好由寄存器对 R25 : R24 实现，从 $0000 值开始，每过 0.4 秒增值 $33，至 $01FE 后，每 0.4 秒再减值 $33，到 $0000 后又开始新的循环。

也可以把 10 个台阶的 11 个值存入片内 SRAM，然后每过 0.4 秒由 SRAM 读取一值，输出到 OCR1A 寄存器去。当然在初始化程序段或汇编后的烧录过程中将此 11 值写入 EEPROM 亦未尝不可。不过比较起来，还是在片内 SRAM 中列表最为方便。

PWM 周期等于 0.125 μs×64×1 022＝8.176 ms。

在 PWM 方式下，当 TCNT1 值从 $0000 点起开始改变计数方向时(由递减变成递增)，定时器 1 的溢出标志 TOV1 置 1。可利用 TOV1 标志每过 8.176 ms 请求一次中断，50 次中断(约 0.4 s)后修改 OCR1A 值，即可满足题义要求。

```
        .include"4414def.inc"
          rjmp   RESET
        .org     $006                    ; T/C1 溢出中断向量
          rjmp   TOVF1
        .org     $010
RESET:  ……
        WDR                              ; WDT 用前先清 0
        LDI      R16,1≪WDE               ; 溢出时限 15 ms
        OUT      WDTCR,R16               ; 启用 WDT
        SBI      DDRD,5                  ; OC1A 作输出
        LDI      R16,$01
        OUT      OCR1BH,R16              ; OCR1B←$0100
        CLR      R16
        OUT      OCR1BL,R16              ; OC1B 方波输出
        OUT      OCR1AH,R16              ; OCR1A 寄存器值，
        OUT      OCR1AL, R16             ; 从 $0000 开始
        LDI      R16,$03                 ; 对晶振 64 分频
        OUT      TCCR1B,R16              ; 并启动 T/C1
        LDI      R16,$A2                 ; 9 位 PWM，
        OUT      TCCR1A,R16              ; 正向输出
        LDI      R16,1≪TOIE1
        OUT      TIMSK,R16               ; 开 T/C1 溢出中断
        SEI                              ; 开总体中断
        CLR      R20                     ; 建立 EEPROM 地址指针
        LDI      R21,50                  ; 50 次中断后修改 OCR1A 值
        ……
TOVF1:  WDR                              ; WDT 复位
        DEC      R21                     ; 到 50 次中断了吗？
        BRNE     TOMAIN                  ; 否，准备返回
        LDI      R21,50                  ; 已到，恢复中断计数初值
```

```
        OUT     EEARL,R20                ; 输出 EEPROM 地址码
        SBI     EECR,EERE                ; 读选通信号置 1
        IN      R1,EEDR                  ; 读数据高字节
        INC     R20                      ; EEPROM 地址码加 1
        OUT     EEARL,R20                ; 输出地址码
        SBI     EECR,EERE                ; 读选通信号置 1
        IN      R0,EEDR                  ; 读数据低字节
        OUT     OCR1AH,R1                ; 输出比较值高字节
        OUT     OCR1AL,R0                ; 输出比较值低字节
        INC     R20                      ; EEPROM 地址码加 1
        CPI     R20,$28                  ; 表格已全读完?
        BRNE    TOMAIN                   ; 否,准备返回
        CLR     R20                      ; 是,地址指针从 0 开始
TOMAIN: RETI                             ; 中断结束
        .eseg
        .org    $00
table:  .DW $0000,$0033,$0066,$0099,$00cc,$0100
        .DW $0133, $0166, $0199, $01cc, $01FF, $01cc
        .DW $0199, $0166, $0133, $0100, $00cc, $0099
        .DW $0066, $0033
```

执行上段程序时,可用双线示波器在屏幕上观察到 AT90S4414 芯片 OC1A(PD5)和 OC1B 两引脚上的 PWM 输出波形。

4.6 模拟比较器的应用

AT89 和 AT90 系列微控制器在其内部大多配有模拟比较器,这对模拟量输入信号的处理非常方便。本节将研究模拟比较器应用方面的几个例题,藉以开拓读者的思路。

4.6.1 模拟比较器的工作设定

[例 4.26] 试写出如何等待模拟比较器输出正跳变信号的程序段。

探知模拟比较器输出正跳变信号可用查询法,亦可用中断法。用查询法可查比较器输出 ACO,也可查中断标志 ACI。

下面一段程序不断检查模拟比较器的控制状态寄存器 ACSR 中的比较器输出位 ACO,看它何时由低变高。其优点是无须预先设定,但缺点在于非常短暂的脉冲信号可能会被漏掉。这是因为在两次检查比较器的间隙中,程序要运行 3 个时钟周期,若恰在此期间来一个脉冲,则它不会被检查到。这种方法的另一个缺点是,倘若在检查开始时,输出为高电平,那么程序不得不等待输出变为低电平后再查下一个高电平的到来。

```
WAITH:  SBIC    ACSR,ACO                 ; 若输出为 0,则跨越
        RJMP    WAITH                    ; 输出为 1,等待
WAIT:   SBIS    ACSR,ACO                 ; 若输出为 1,则跨越
```

```
        RJMP    WAIT                    ; 输出为 0,等待
        ……                              ; 已等到了上升沿
```

也可以根据表 1.16 所列,把 ACSR 寄存器的最低两位 ACIS1 和 ACIS0 均置成 1,即把比较器设定成输出上升沿触发中断的运作方式,但不开中断,而只查询中断标志 ACI。这是一个比较妥善的方法,因为查询的是中断标志,而 ACI 标志能够记录比查询间隔窄的短暂脉冲。此法的另一个特点是无须首先等待负沿的到来。

关于下面一段程序尚应指出两点:

- 复位后,ACIE 位为 0,即模拟比较器中断是关闭的;
- 在修改 ACIS1 和 ACIS0 两位数值时,可能会引起 ACI 标志置位,故随后应将其清 0,清 0 方法是向其写入逻辑 1。

```
WAITH2: SBI     ACSR,ACIS0              ; 设定 ACO 上升沿,
        SBI     ACSR,ACIS1              ; 触发 ACI 标志置 1
        SBI     ACSR,ACI                ; 向 ACI 标志写 1,将其清 0
WAIT2:  ……                              ; 用户程序其他部分
        SBIS    ACSR,ACI                ; 若 ACI 为 1,则跨越
        RJMP    WAIT2                   ; ACI=0,再查
        ……                              ; 发现了上升沿
```

当然,在设定 ACO 输出上升沿触发模拟比较器中断,并随之把 ACI 标志清 0 后,将 ACIE 位置 1,再开总体中断,那么此后只要有中断到来,就说明 ACO 输出了一个上升沿。在下面的例题中即包含了这样的思想。

[例 4.27]　用 AT90S1200 微控制器监视某模拟输入信号,若其值等于或超过 5 V,则 MCU 输出低电平,其值小于 5 V 时,输出高电平。

如图 4.6 所示,在 PB0(模拟比较器正输入引脚 AIN0)引脚上外接+5 V 标准电压,而在 PB1(负输入引脚 AIN1)上接模拟输入信号。利用片内模拟比较器对两者进行比较,根据 ACO 位的变化决定 PB2 输出高电平还是低电平。

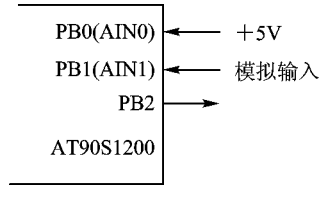


图 4.6　监视模拟输入

```
        .include"1200def.inc"
        RJMP    RESET
        .ORG    $003                    ; 模拟比较器中断向量
        RJMP    ANACMP
RESET:  LDI     R16,$FC                 ; PB0 和 PB1 为输入
        OUT     DDRB,R16                ; PB 其他引脚为输出
        LDI     R16,$00                 ; PB0 和 PB1 为三态输入
        OUT     PORTB,R16               ; 输出引脚暂输出 0
        CLR     R16                     ; ACIS1 和 ACIS0 清 0,此指令可取消
        OUT     ACSR,R16                ; 设定 ACO 任一边沿均触发中断
        SBIC    ACSR,ACO                ; 若 ACO 为 0,则跨越
        SBI     PORTB,2                 ; ACO=1,PB2 应输出 1
```

```
        SBI     ACSR,ACIE                ; 开模拟比较器中断
        SEI                              ; 开总体中断
        ……
```

在上面这个初始化程序段中，CLR 指令可以取消，因为此前的 LDI 指令已经起到了将 R16 清 0 的作用。但在修改 ACSR 寄存器最低两位值以设定 ACO 某边沿触发中断时，就会隐含一个容易被忽视的问题。那就是这样做可能导致中断标志 ACI 置位。那么开中断后此标志能否引起中断呢？不能！因为如 1.2.12 节所述，在 ACI 已经为 1 的情况下，如果用 SBI 或 CBI 指令修改 ACSR 寄存器中的其他位值，那么 ACI 位将被清 0。程序段内开模拟比较器中断的 SBI 指令即可一箭双雕地完成此项任务，既开中断，又将 ACI 位清 0。

在下面的中断服务程序中，首先检查眼下 ACO 为何值，若为 1，则说明中断是由正跳变引起的。也就是说，AIN1 引脚上的输入电压低于＋5 V，故 PB2 应输出高电平。反之，若 ACO 为 0，则 PB2 应输出低电平。

```
ANACMP: SBIC ACSR, ACO           ; ACO=0,跨越
        RJMP    ONE              ; ACO=1,转去输出 1
        CBI     PORTB,2          ; ACO=0,PB2 输出 0
        RETI                     ; 中断结束
ONE:    SBI     PORTB,2          ; ACO=1,PB2 输出 1
        RETI                     ; 中断结束
```

4.6.2 简易 A/D 转换器

除例 4.27 所述的参数超限监测外，AVR 片内模拟比较器尚可用来构成各种类型的 A/D 转换器。首先从简易的 A/D 转换谈起。

如图 4.7 所示，AT90S1200 外配两个电阻和一只电容器即可构成一个简易的 A/D 转换器。这种配置除模拟比较中断外，还使用了定时器 0 的溢出中断。使用中断可解放 MCU，使 A/D转换不过多地占用 MCU 的宝贵时间。

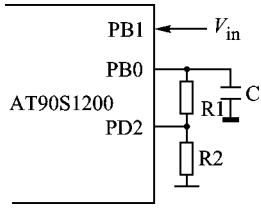


图 4.7 简易 ADC

为尽量少用片外元件，该 A/D 转换器采用了简单的阻容充电电路，由 PD2 引脚进行充放电控制。在 A/D 转换时的充电过程中，电容两端的电压 V_C 将按指数曲线上升。但若把转换器的输入电压的范围限制在 0.4 V_{DD} 以下，则电容两端电压与时间的关系就非常接近于一条直线。这样，V_C 值升至 V_{in} 值时所用的时间就可代表电压 V_{in} 的 A/D 转换结果。如果被转换电压变化范围较大，那么就可利用精确分压的办法解决这个问题。

为保证正确计时，RC 网络的时间常数必须满足下式的要求：$512(1/f)=-RC\ln(1-0.4)$

若晶振频率 $f=8$ MHz，则 RC≈125 μs。例如，C 可选容量等于 0.01 μF 高稳定性的聚苯乙烯薄膜电容器，R1 可为 13 kΩ、R2 为 1 kΩ 的金属膜电阻。

在每次转换之前，电容应充分放电，放电过程持续 200 μs 以上即可达到要求。

[例 4.28] 写出上述简易 A/D 转换器的有关程序，并说明其工作过程。

```
        .include"1200def.inc"
        .equ    preset=192              ; T/C0 预置值(256-64)
        .def    result=r16              ; 结果和中间数据
        .def    temp=r17                ; 暂存寄存器
        .cseg
        .org    $000
            rjmp   RESET
        .org    $002                    ; T/C0 溢出中断向量
            rjmp   ANACMP               ; 转 T/C0 溢出处理
            rjmp   ANACMP               ; 转模拟比较器中断处理
```

由以上程序可以看到，$002 和 $003 分别为 T/C0 溢出中断向量和模拟比较器中断向量，但两者的处理程序是同一个。

```
RESET:  ldi     temp,$fc                ; PB0 和 PB1 拟设定为输入
        out     DDRB, temp              ; PB2～PB7 为输出
        clr     temp
        out     PORTB,temp              ; PB0 和 PB1 为三态输入
        sbi     DDRD,2                  ; PD2 设定为输出
        cbi     PORTD,2                 ; PD2 输出 0,C 开始放电
delay:  clr     result                  ; 以下是延时
        ldi     temp,$f6
loop1:  inc     result
        brne    loop1
        inc     temp
        brne    loop1
        ldi     result, $0b             ; 拟启动模拟比较器
        out     ACSR,result             ; ACO 上升沿触发中断
        ldi     result,$02
        out     TIMSK,result            ; 开 T/C0 溢出中断
        sbi     ACSR,ACI                ; 清 ACI 标志
        ldi     result,PRESET
        out     TCNT0,result            ; 预置 TCNT0 初值
        clt                             ; 清 A/D 转换完成标志
        ldi     result,$02              ; 分频因子为 8
        out     TCCR0,result            ; 启动 T/C0
        sbi     PORTD,2                 ; 电容开始充电
        sei                             ; 开总体中断
wait:   brtc    wait                    ; 等待 A/D 转换完成
        ……
ANACMP: in      result, TCNT0           ; 读 TCNT0 计时结果
        clr     temp
        out     TCCR0,temp              ; T/C0 停止运行
        subi    result,PRESET+1         ; 求转换结果
```

```
cbi     PORTD,2                 ; 电容开始放电
set                             ; 置 A/D 转换完成标志
reti                            ; 中断结束
```

如果 MCU 供电电压 $V_{DD}=5$ V，那么模拟输入电压 V_{in} 应限制在 2 V 以下。这时最大 A/D转换结果为 64，因此 TCNT0 的初值设定为 256－64＝192。对这种安排来说，最大转换模拟电压不超过 8 V，即将全量程电压分压出 1/4 作为 V_{in}，全量程转换结果等于 $FF。

在初始化程序段中，设定 AIN0(PB0)和 AIN1(PB1)为高阻输入引脚，这是作模拟比较器输入所要求的。把 PD2 引脚设定为输出，且输出 0 电平，于是电容 C 便通过电阻 R1 开始放电。经过足够长的一段延时之后，启动模拟比较器，设定其输出信号的正跳变引发中断。随之启动 T/C0，令其以 192 为初值对 1 μs 周期的时钟信号进行递增计数，并将 PD2 输出置 1，使 C 开始充电，亦即开始 A/D 转换过程。

在充电过程中，当电容两端电压 V_C 超过模拟输入电压 V_{in} 时，模拟比较器的输出端 ACO 由原 0 电平跳变到 1 电平。此正跳变引起中断。在中断服务程序中读出 T/C0 的计时结果，并扣除原初值 preset＋1(中断响应时间会使 T/C0 多记录 1 μs)，即得转换结果。例如，读得 TCNT0 值为 250，则 A/D 转换结果等于 250－193＝57＝ $39。当然，若 V_{in} 系分压所得，则 A/D 转换结果可能等于 $72、$AB 或 $E4 等。

如果 V_{in} 电压过高，V_c 尚未充至 V_{in} 值时，T/C0 就已经溢出，引发溢出中断。例如，这时 TCNT0 值为 $01，扣除 preset＋1 后，得 64。注意，V_c 尚低于 V_{in} 时，结果就等于临界最大值了。故我们可以得出结论：

- 模拟比较器引发的中断表明被转换值在合理范围之内；
- T/C0 溢出引起的中断(result 值等于 64)表明模拟输入值 V_{in} 越限。

在中断服务程序中，还要使 T/C0 停止运行，令 PD2 输出 0 使 C 放电并置 A/D 转换完成标志，即为下次转换做好准备。

应当指出，上述 A/D 转换方法精确度较低，适用于一般要求不高的场合。从原理上讲，可以用 ACO 输出的跳变信号去触发 T/C1 的输入捕捉功能，以 16 位捕获值作为 A/D 转换结果。但这样做的必要性不大。

4.6.3 逐步逼近式 A/D 转换

这里我们介绍逐步逼近式 A/D 转换法，它比上述简易 A/D 转换精度要高一些。许多中档 A/D 转换器都用这种方法，例如 A/D 转换器 ADC0809 以及 8XC552 和 AT90S4434/8535 等微控制器的片内 A/D 转换器均属此种类型。

如图 4.8 所示，AT90S8515 芯片通过其 C 口外接一 D/A 转换用 R－2R 电阻网络，它把 C 口输出的数字量转换成相应的模拟量 V_{DAC}，作为片内模拟比较器正端 AIN0 的输入信号。而由比较器负端 AIN1 输入待转换的模拟信号 V_{in}。

逐步逼近法的第一步首先把 PC7 置成 1，其他位

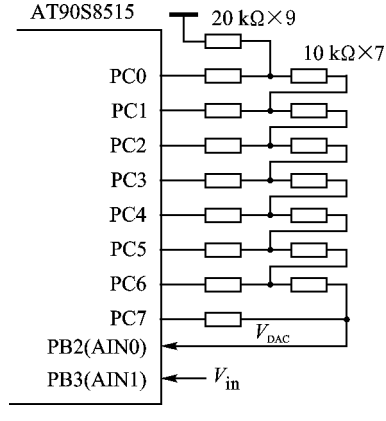


图 4.8 逐步逼近式 A/D 转换

清 0，即由 C 口输出 $80（全量程的一半），R－2R 电阻网络将此值转换成相应的模拟量 V_{DAC} 去与未知模拟量 V_{in} 进行比较。若未知电压高于或等于 V_{DAC}，则 $80 中最高位 1 保留，否则它将被清 0。

第二次把 C 口次高位置成 1，得 $C0 或 $40，$V_{DAC}$ 新值再与 V_{in} 进行比较。若 V_{in} 高于或等于 V_{DAC}，则次高位的 1 值保留下来，否则即被清掉。此过程一直持续到所有 8 位均被测试完成为止，此时所得即为最后的 A/D 转换结果。

［例 4.29］　写出上述逐步逼近式 A/D 转换的有关程序。

```
        .INCLUDE"8515DEF.INC"
            RJMP   RESET
        .DEF   TEMP=R16                 ;暂存寄存器
        .DEF   DATA=R17                 ;A/D转换结果寄存器
        .DEF   TMP1=R18                 ;逐步逼近寄存器
        .DEF   TMP2=R19                 ;延时用寄存器
        .ORG   0X10
RESET:  LDI    TEMP,$5F                 ;堆栈指针 SP=$025F
        OUT    SPL,TEMP
        LDI    TEMP,$02
        OUT    SPH,TEMP
        LDI    TEMP,$FF
        OUT    DDRC,TEMP                ;C口为输出
        CLR    TEMP
        OUT    DDRB,TEMP                ;B口作输入
        LDI    TEMP,$F3
        OUT    PORTB,TEMP               ;PB2和PB3作三态输入
        CLR    DATA                     ;结果从0开始
        LDI    TEMP,$80                 ;从半量程开始逼近
NXTD:   MOV    TMP1,TEMP                ;一位逼近值,
        OR     TMP1,DATA                ;与中间结果合并,得新逼近值
        OUT    PORTC,TMP1               ;输出逐步逼近值
        RCALL  DELAY                    ;延时750 ns
        SBIS   ACSR,ACO                 ;若ACO=1,则跨越
        MOV    DATA,TMP1                ;ACO=0,保留有效逼近值
        LSR    TEMP                     ;准备向下一位逼近
        BRCC   NXTD                     ;若C=0,则逼近下一位
        ……                              ;C=1,转换完成,下略
DELAY:  LDI    TMP2,$02                 ;延时子程序
        DEC    TMP2
        BRNE   DELAY+1
        RET
```

片内模拟比较器的延迟时间为 750 ns，故每次给它一新输入值后，必须过 750 ns 才能根据输出位 ACO 来判断比较结果。

4.7 串行口的应用

AVR 微控制器片内大多含有同步串行口 SPI 和异步串行口 UART 两个部件。AVR 串行口的位传送率颇高，甚至与某些微控制器的并行传送速度相差无几。

4.7.1 同步串行传送

AVR 的串行涉外接口 SPI，亦称同步串行口，可用来进行 AVR 与外围设备之间或若干 AVR 器件之间的高速同步数据传送。下面我们研究一个两 AVR 间数据交换的例题。

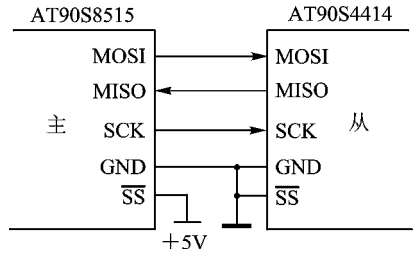


图 4.9 AVR 数据交换

[例 4.30] 将某 AT90S8515 与 AT90S4414 两微控制器片内 SRAM 中的数据块进行交换，数据块首地址均为 BLOCK，长度是 LENGTH，两机所用晶振频率皆为 8 MHz。

将 AT90S8515 设定为主控方式，AT90S4414 为从控方式。两机通过 SPI 进行 LENGTH 个字节的数据交换。其硬件连接如图 4.9 所示。

```
        .include"8515def.inc"
        rjmp    RESET
        .def    SPIdrd=r2               ;暂存从机交换来的数据
        .def    temp=r16                ;暂存寄存器
        .def    tmp1=r17                ;暂存寄存器
        .def    cnt=r18                 ;数据块字节计数
        .equ    block=$0060             ;数据块首地址
        .equ    length=50               ;数据块长度
        .cseg
        .org    $008
        rjmp    SPIint
        .org    $010
RESET:  ……                              ;建立堆栈，略
        ldi     temp,$a0                ;PB7 和 PB5 为输出
        out     ddrb,temp               ;B 口其他引脚为输入
        ldi     temp, $0f
        out     portb, temp             ;PB0～PB3 带内部上拉电阻
        ldi     temp, $d4               ;开中断，主控
        out     spcr, temp              ;正脉冲下降沿有效，4 分频
        ldi     xh, high(block)
        ldi     xl, low(block)          ;X←数据块首地址
        ldi     cnt, length             ;数据块长度计数
        ld      temp,X                  ;读数据块第一个字节
        rcall   delay                   ;延时
```

```
        out     spdr, temp          ; 发送一字节
        sei                         ; 开总体中断
here:   rjmp    here                ; 就地循环等待中断
SPIint: in      SPIdrd,spdr         ; 读从机交换来之数据
        st      X+,SPIdrd           ; 存入片内 SRAM
        dec     cnt                 ; 字节数减 1
        brne    go-on               ; 不为 0,继续
        clr     temp                ; 数据块已交换完毕
        out     spcr,temp           ; SPI 停止运行
        reti                        ; 中断结束
go-on:  rcall   delay               ; 数据块尚未交换完,延时
        ld      temp,x              ; 读数据块一字节
        out     spdr, temp          ; 发送出去
        reti                        ; 中断结束
delay:  ldi     tmp1,8              ; 延时约 3 μs
        dec     tmp1
        brne    delay+1
        ret
```

应当提醒读者注意下列两点：第一，SPI 同步串行数据传送是主从机互相交换数据的，并非单纯由主机向从机发送数据。第二，主机发出数据前之所以要稍稍延迟一点时间，是为了让从机做好数据传送的准备。不言而喻，如果复位后就进行这样的数据传送，那么开机时得先开从机。

下面是从机的初始化和中断服务程序。

```
        .include"4414def.inc"
          rjmp  RESET
        .def    SPIdrd=r2           ; 暂存主机发来的数据
        .def    temp=r16            ; 暂存寄存器
        .def    cnt=r17             ; 数据块字节计数
        .equ    block=$0060         ; 数据块首地址
        .equ    length=50           ; 数据块长度
        .org    $008                ; SPI 中断向量
          rjmp  SPIint
        .org    $010
RESET:  ……                          建立堆栈,略
        ldi     temp, $40
        out     ddrb,temp           ; MISO 为输出,其余为输入
        ldi     temp, $0f
        out     portb,temp          ; PB0～PB3 带内部上拉电阻
        ldi     temp, $c4           ; 正脉冲下降沿有效,4 分频
        out     spcr,temp           ; 从控,开中断
        ldi     xh,high(block)
        ldi     xl,low(block)       ;X←数据块首地址
```

```
        ld      temp,x          ; 读数据块第一个字节
        out     spdr,temp       ; 并输送至 SPDR
        ldi     cnt,length      ; 数据块长度计数
        sei                     ; 开总体中断
here:   rjmp    here            ; 就地循环,等待中断
SPIint: in      SPIdrd,spdr     ; 读主机发来的数据
        st      X+,SPIdrd       ; 并存至片内 SRAM
        dec     cnt             ; 字节数减 1
        brne    go-on           ; 不为 0,继续
        clr     temp            ; 数据块已交换完毕
        out     spcr,temp       ; SPI 停止运行
        reti                    ; 中断结束
go-on:  ld      temp, x         ; 读数据块一字节
        out     spdr,temp       ; 并转送至 SPDR 待与主机交换
        reti                    ; 中断结束
```

4.7.2 异步串行传送

AVR 异步串行口具有独立的波特率发生器,故不占用片内定时器。其第二个优点是发送和接收均有缓冲,因而可连续发送。但其缺点是无多机通讯功能。

[例 4.31] 某 AT90S8515 微控制器,需将从现场采集并经过数字滤波处理的数据发送给同型号的另一微控制器。两机之时钟频率均为 8 MHz,串行传送的波特率等于 19 200 bps,所传送的数据共 50 个字节,源与目的地址均为各自 SRAM 中的 $0060～$0091。

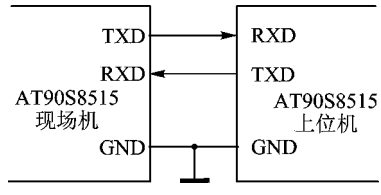


图 4.10 异步串行通讯

本题实际上是一个半双工串行通讯的例子,在图 4.10 的两条数据线中只使用一条。现场机此时只发送不接收,而上位机却相反,仅接收不发送。50 个字节的发送时间约为 26 ms。下面是现场机的初始化和中断服务程序。

```
        .include"8515def.inc"
          rjmp  RESET
        .def    temp=r16        ; 暂存器
        .def    cnt=r17         ; 字节计数
        .org    $00a
          rjmp  DREM            ; 转向 UDRE 中断服务程序
        .org    $010
RESET:  ……                      ; 建立堆栈,略
        ldi     temp, $28       ; UDRIE=1, TXEN=1
        out     ucr,temp        ; 允许 UDRE 中断,允许发送
        ldi     temp, 25
        out     ubrr,temp       ; 波特率等于 19 200 bps
        clr     zh              ; Z←$0060
```

```
        ldi     zl,$60                  ;建立数据块指针
        ldi     cnt,50                  ;字节计数初值
        sei                             ;开总体中断
        ……
DREM:   ld      temp,Z+                 ;读一字节数据
        out     udr, temp               ;启动串行发送
        dec     cnt                     ;所有字节全发完?
        brne    TOM                     ;否,准备返回
        cbi     txen                    ;是,准备结束发送
        cbi     udrie                   ;禁止 UDRE 中断
        sbiw    r30,$32                 ;恢复 Z 指针
        ldi     cnt,$32                 ;恢复字节计数初值
TOM:    reti                            ;中断结束
```

关于这段程序,应当强调下列几点。

第一,复位后,USR 寄存器中的数据寄存器空标志 UDRE=1,在中断开放的情况下(UDRIE=1,I=1),它将引起中断。

第二,中断服务程序向 UDR 中写入待发送的数据字节,此时 UDRE 标志被自动清 0。但当该字节数据由 UDR 寄存器传至发送移位寄存器行将向外串行发送时,UDRE 标志又被硬件置成 1。这样,不难理解,当某个字节数据自移位寄存器向外发送时,下一个字节又可写入 UDR 寄存器中了。而待第一个字节发送完毕时,第二个字节便由 UDR 传送到发送移位寄存器,此时又可向 UDR 写入第三个字节。总而言之,AVR 异步串行口可不间断地发送数据,无须发送一个字节中断一次,由中断服务程序启动下个数据的发送过程。

第三,在中断服务程序发送最后一个字节数据时,cbi 指令将 TXEN 位清成 0,但这不能使发送器立即进入禁止状态。而要等到移位寄存器中的数据以及 UDR 中的后续数据完全发出之后,发送器才与 TXD 引脚(PD1)脱钩,即发送器处于禁止状态。

第四,发送最后一个字节数据时,中断服务程序用 cbi 指令也将 UDRIE 位清 0,但最后这个字节由 UDR 传向发送移位寄存器时,UDR 寄存器空标志(UDRE=1)将一直保持下去。一旦需要再次进行数据传送,只将 UDRIE 和 TXEN 位置成 1 即可进行。

下面是上位机的初始化及中断服务程序。

```
        .include"8515def.inc"
          rjmp  RESET
        .org    $009
          rjmp  RCV                     ;转至接收中断服务程序
        .org    $010
RESET:  ……
        ldi     r16, $90                ;RXCIE=1, RXEN=1
        out     ucr, r16                ;开放 RXC 中断,允许接收
        ldi     r16,25
        out     ubrr,r16                ;波特率=19 200 bps
        clr     zh                      ;建立数据指针
```

```
        ldi     zl, $60                 Z←$0060
        sei                             ;开总体中断
        ……
RCV:    in      r0,udr                  ;读接收数据
        st      Z+,r0                   ;转存入片内 SRAM
        cpi     zl, $92                 ;是最后一个字节?
        brne    TOM                     ;否,准备返回
        ldi     zl, $60                 ;是,恢复指针初值
TOM:    reti                            ;中断结束
```

根据波特率可以算出,一帧数据(10 位)的传送时间约为 520 μs,即 4 160 多个时钟周期。在这么长的时间内,收方 CPU 总是可以抽暇将 UDR 中的接收数据读出的。因此产生超时错误的可能性几乎为 0。这就是为什么在中断服务程序中读出接收数据后未检查 OR 标志的原因。

最后顺便指出,发方将 TXEN 位置 1 和收方将 RXEN 位置 1 后,PD1 和 PD0 将被自动设定为 TXD 和 RXD 引脚,无须再通过 DDRD 寄存器来规定它们的数据方向。

[例 4.32] 两 AT90S8515 微控制器以异步串行通讯方式交换片内 SRAM 中的数据块。两数据块首地址分别为 block0 和 block1,每块均由 length 个字节组成。两机时钟频率都是 8 MHz。

这是一个全双工串行数据传送的例子,两机都同时进行发送和接收,接线情况可参见图 4.10。

```
        .include"8515def.inc"
            rjmp   RESET
        .def    tmp=r16                 ;暂存器
        .def    cnt=r17                 ;字节计数寄存器
        .def    cnt1=r18                ;字节计数寄存器
        .equ    block0=$0060            ;数据块首地址
        .equ    length=$32              ;数据块长度
        .org    $009
            rjmp   RCV                  ;UART 接收完成中断
            rjmp   DREM                 ;UART 数据寄存器空中断
        .org    $010
RESET:  ……                              ;建立堆栈,略
        ldi     tmp, $b8                ;RXCIE=UDRIE=1
        out     ucr,tmp                 ;RXEN=TXEN=1
        ldi     tmp,25
        out     ubrr,tmp                ;波特率等于 19 200 bps
        ldi     zh, high(block0)
        ldi     zl, low (block0)        ;Z 指向数据块首地址
        mov     yh,zh                   ;建立第二个指针
        mov     yl,zl                   ;Y 也指向数据块首地址
        ldi     cnt,length              ;发送字节计数
```

```
        mov     cnt1,cnt                ;接收字节计数
        sei                             ;开总体中断
        ……
DREM:   ld      tmp,Z+                  ;读一数据字节
        out     udr,tmp                 ;并开始串行发送
        dec     cnt                     ;字节数减 1
        brne    TOM                     ;数据已传送完毕?
        sbiw    zl, length              ;是,指针复原
        ldi     cnt,length              ;字节计数初值复原
        cbi     txen                    ;准备结束发送
        cbi     udrie                   ;禁止 udre 中断
TOM:    reti                            ;中断结束
RCV:    in      tmp,udr                 ;读接收字节
        st      Y+,tmp                  ;转存入片内 SRAM
        dec     cnt1                    ;字节数减 1
        brne    RTM                     ;数据块接收完毕?
        sbiw    yl,length               ;是,指针复原
        ldi     cnt1,length             ;字节计数复原
        cbi     rxen                    ;接收告一段落
        cbi     rxcie                   ;禁止接收中断
RTM:    reti                            ;中断结束
```

4.7.3　软件 UART

本节将详细讨论如何构成半双工软件异步串行口问题,这适用于具有 8 位定时器和外部中断功能的所有 AVR 器件。因为许多应用场合每次的信息传递只限于一个方向,所以半双工 UART 使用 MCU 的资源比较少。

[例 4.33]　某 AT90S1200 微控制器将其 R0 和 R1 中的数据串行传送给另一同型号微控制器,后者收到数据后也存入相应的寄存器中。两机时钟均为 1 MHz。

大家知道,AT90S1200 微控制器片内无 UART,故这里我们将用软件 UART 进行异步串行传送。

以晶振信号不经分频作 T/C0 的时钟,根据表 1.3 可知,TCCR0 寄存器的内容应等于 $01(CS02、CS01 和 CS00 三位为 001)。如果波特率选用 38 400 bps 的话,那么每位数据的传送时间为 $1\times10^6/38\ 400=26\ \mu s$,即 26 个时钟周期。

```
.include"1200def.inc"
.def    scopy=r15               ;状态寄存器备份
.def    tmp=r16                 ;暂存器
.def    bitcnt=r17              ;位计数器
.def    status=r18              ;状态缓冲器
.def    reload=r19              ;重装值寄存器
.def    buffer=r20              ;串行缓冲器
.equ    RDR=0                   ;接收数据就绪标志
```

```
.equ    TD=6                    ; 正在发送标志
.equ    BUSY=7                  ; 忙标志
.cseg
.org    $000                    ; 复位向量
    rjmp   start
    rjmp   extint               ; 转至外部中断 0 服务程序
    rjmp   tovf                 ; 转至定时器 0 溢出中断服务程序
    reti                        ; 模拟比较器中断(未用)
```

如图 4.11 所示,发方在其 PD4 引脚上输出一个由 1 至 0 的跳变,收方在 PD2(INT0)引脚上便检测到一个负跳变。该跳变将触发收方的外部中断,同时被收方视作起始位的到来,即异步串行传送的开始。

应当指出,收方必须用 PD2(INT0)作串行数据输入引脚,这样起始位才能引起收方中断。但发方可用任何空闲端口引脚作数据输出。

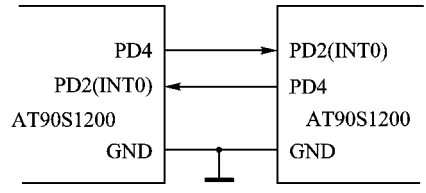


图 4.11 软件 UART 接线

以下是外部中断服务程序,其主要任务是禁止此后的外部中断,开放定时器中断,以便在定时器中断时发送数据或对输入数据位进行采样。

```
extint:  in      scopy,SREG              ; 保存 SREG 内容,1
         ldi     status,1≪BUSY           ; 置忙标志,其他位清 0,1
         ldi     tmp,(256-39+24)         ; TCNT0 首装值,1
         out     TCNT0,tmp               ; 置 TCNT0 初值,1
         ldi     tmp,1≪TOIE0             ; 拟开 T/C0 溢出中断
         out     TIFR,tmp                ; 清 T/C0 溢出标志 TOV0
         out     TIMSK,tmp               ; 开放 TOV0 中断
         clr     bitcnt                  ; 清位计数器
         out     GIMSK, bitcnt           ; 禁止外部中断
         ldi     reload, (256-26+8)      ; TCNT0 重装值
         out     SREG,SCOPY              ; SREG 内容复原
         reti                            ; 中断结束
```

TCNT0 的首装值与重装值各等于多少,在定时器中断服务程序后才能讨论清楚。现在能说的是,这些值要保证对每个数据位的采样应在该位持续时间的中点进行。

定时器 0 中断服务程序协调数据位的发送和接收。TCNT0 初值的设定保证此中断服务程序自动以等于波特率的速率执行。发送时,该程序对数据进行移位,并进而将移位后的最低数位发送出去。接收时,它对数据位进行采样,并随之将该位移入缓冲器中。

在上段程序中,我们用了一个状态缓冲器 status,其最高为 BUSY,它告知,软件 UART 正忙于工作。在下段程序中还会遇到其次高位 TD 以及最低位 RDR。TD 代表数据发送,在 UART 发送数据时置 1。status 的最低位是 RDR,即接收数据就绪标志,在新数据已经到达且可供读取时置 1。下面是定时器 0 中断服务程序。

```
tovf:    in      scopy, SREG             ; 保存 SREG 内容,1
```

```
        out     TCNT0,reload;          ; TCNT0 重装,1
        inc     bitcnt                 ; 位计数值加 1,1
        sbrs    status, TD             ; 是发送吗?1/2
        rjmp    receive                ; 否,转接收,2
        sbrc    bitcnt,3               ; 是。位计数≤7,跨越,2
        rjmp    stopb                  ; >7,转停止位
        sbrc    buffer,0               ; 最低位为 0,跨越,1/2
        sbi     PORTD,PD4              ; 最低位为 1,发送 1,2
        sbrs    buffer,0               ; 最低位为 1,跨越,1/2
        cbi     PORTD,PD4              ; 最低位为 0,发送 0,2
        lsr     buffer                 ; 待发送数据右移一位
        out     SREG,scopy             ; SREG 内容复原
        reti
stopb:  sbi     PORTD,PD4              ; 生成停止位
        sbrs    bitcnt, 0              ; 位计数=9,跨越
        rjmp    rtn                    ; =8,准备返回
txmc:   ldi     tmp, 1≪INT0;           ; 位计数=9
        out     GIMSK,tmp              ; 开 INT0 中断
        clr     tmp
        out     TIMSK,tmp              ; 关定时器中断
        cbr     status,(1≪BUSY)|(1≪TD) ; 清 BUSY 和 TD 标志
rtn:    out     SREG, scopy            ; SREG 内容复原
        reti
receive: sec                           ; C←1, 1
        sbis    PIND,2                 ; 数位采样,接收位为 1,跨越,1/2
        clc                            ; 接收位为 0,C←0
        ror     buffer                 ; 接收位移入 buffer
        in      SCOPY, SREG            ; 保存 SREG 内容
        cpi     bitcnt,9               ; 刚才接收的是停止位?
        brne    rtn                    ; 否,准备返回
        out     SREG,scopy             ; 是,SREG 内容复原
        rol     buffer                 ; 将停止位移出
        sbr     status,1≪RDR           ; 置接收数据就绪标志,清其他标志
        rjmp    txmc                   ; 转发送完成段落
```

这段定时器 0 中断服务程序是软件 UART 的核心,它由接收和发送两部分组成。

1. 接收部分

这里的接收部分是与外部中断服务程序前后衔接的。后者接收起始位,而定时器中断服务程序中的接收部分则负责接收 8 个数据位和最后的停止位。接收起始位时,bitcnt 值为 0。显然若 bitcnt 值为 9,则表明刚刚接收的是停止位。

图 4.12 所示为起始位和三个数据位接收过程中的时间关系。收方 PD2 引脚上的负跳变将引发外部中断。6 个时钟周期后,CPU 开始执行 extint 程序段,在其第 4 条指令处,为 TCNT0赋值 241。图 4.12 中 6+4 后赋值即指此而言。又过 15 个周期 TCNT0 值等于 256,

即 0,于是产生溢出中断。6 个时钟周期后,CPU 进入 tovf 程序段。在定时器 0 溢出中断服务程序 tovf 中的第 2 条指令处,TCNT0 被赋值 238。这就是图中溢出 6+2 个时钟周期后又赋值的含义。又过 6 个时钟周期 CPU 将执行 receive 段的第二条指令(各指令注释部分的数字即该指令的执行时间)对输入数位进行采样。此采样点以及以后各位的采样点,都在各位的中点处。

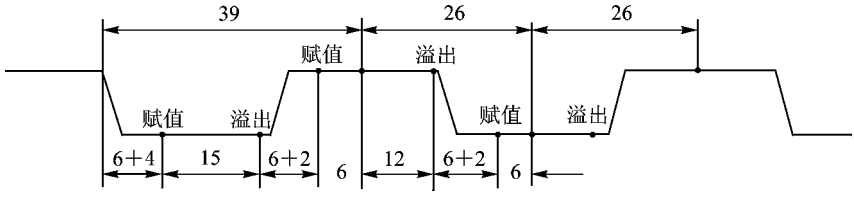


图 4.12 数据接收的时间关系

从图 4.12 中读者不难了解,为什么以后的 TCNT0 重装值为 238。

图 4.12 中的 6+4、6+2 和 6 等时钟周期数是中断响应过程和程序指令决定的,不可变动。位周期(其倒数为波特率)可选择。但若位周期太小,就可能导致 extin 或 tovf 中断服务程序尚未结束而定时器便要溢出的后果。这样,波特率就乱了。例如,在此例题中,若位周期小于 17,就会出现这种情况。而位周期过大,比如大于 170,则第一次赋值使定时器 0 过 1.5 个位周期溢出(256−190×1.5+24)就无法进行了。

最后应当指出,这样的软件 UART 未考虑滤除噪音干扰问题。

下面是初始化和发送子程序。

```
init:   ldi     tmp, 1                          ; 定时器用 CPU 时钟源
        out     TCCR0,tmp                       ; 启动定时器
        ldi     tmp,1≪ISC01                     ; 设定 INT0
        out     MCUCR,tmp                       ; 下降沿触发中断
        ldi     tmp,1≪INT0
        out     GIMSK,tmp                       ; 开放外部中断 0
        clr     status                          ; 清状态缓冲器
        ret
txm:    ldi     status, (1≪BUSY)|(1≪TD)         ; 置 BUSY 和 TD 标志
        clr     tmp
        out     GIMSK,tmp                       ; 关外部中断
        ser     bitcnt                          ; 位计数等于 $FF
        ldi     tmp,1≪TOIE0                     ; 位 1 置 1
        out     TIFR,tmp                        ; 清除 TOV0 标志
        out     TIMSK,tmp                       ; 开放 TOV0 中断
        ldi     reload, (256−26+8)              ; TCNT0 重装值
        ldi     tmp,(256−26+14)                 ; TCNT0 首装值
        out     TCNT0,tmp                       ; TCNT0 首装值就位
        cbi     PORTD,PD4                       ; 发出起始位
        ret
```

2. 发送部分

tovf 中的发送部分与发送子程序 txm 相衔接。前者的任务是发送 8 个数据位及停止位；而后者则只发送起始位。

发送起始位时，bitcnt 值等于 $FF，发送 LSB 时其值为 0。bitcnt 值等于 8 说明停止位正在发送，但尚未完成。其值为 9 则表示停止位已经发送完毕。

如图 4.13 所示，如果在产生 PD4 引脚上的负跳变时置 TCNT0 初值，那么该值应等于 256－26＋16。但从 txm 子程序的最后三条指令中可以看到，是在 TCNT0 首装值就位后才发出起始位的。cbi 指令需耗时 2 个时钟周期，故 TCNT0 此初值为 256－26＋14。

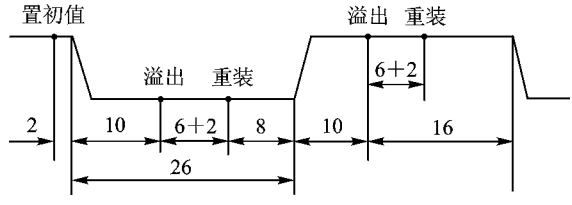


图 4.13　数据发送的时间关系

中断响应时间最少要 4 个时钟周期，执行向量地址处的跳转指令又得用 2 个周期。就是说，溢出后经 6 个周期 CPU 开始进入 tovf 程序段。tovf 的第二条指令重装 TCNT0 值，重装后又过 8 个周期，sbi 指令把 PD4 置成 1，或过 9 个周期 cbi 指令把 PD4 清成 0。重装值为 256－26＋8＝238，过 18 周期后又将溢出，如此等等。

以下是 AT90S1200 微控制器的发送子程序。

```
start:  ……
        ldi     tmp, $10
        out     DDRD, tmp          ; PD4 作输出，其他为输入
        ser     tmp
        out     PORTD, tmp         ; 输入带内部上拉电阻，PD4 输出 1
        rcall   init               ; UART 初始化
        sei                        ; 开总体中断
        mov     buffer,R0          ; 待发送数据就位
wait:   sbrc    status, BUSY       ; UART 可以发送吗？
        rjmp    wait               ; 否，等待
        rcall   txm                ; 是，开始发送
wait1:  sbrc    status, TD         ; 发送完毕？
        rjmp    wait1              ; 否，等待
        mov     buffer,R1          ; 是，待发送数据就位
        rcall   txm                ; 开始发送
wait2:  sbrc    status, TD         ; 发送完毕？
        rjmp    wait2              ; 否，等待
        ……                         ; 是，进行其他工作
```

一般说来，在调用 txm 子程序前，必须先检查 BUSY 标志，以决定眼下能否通过 txm 子程序开始数据发送。如果在发送或接收两数据位的间隙时间调用 txm 子程序，那么此前的接收

或发送过程必将夭折。

以下是接收方的主程序。阅读整节程序时,一定要注意哪段指发方,哪段指收方。

```
start:  ……
        clr     tmp
        out     DDRD, tmp                ; D 口作输入
        ser     tmp
        out     PORTD, tmp               ; 输入均有内部上拉电阻
        rcall   init                     ; UART 初始化
        sei                              ; 开总体中断
idle:   sbrs    status, RDR              ; 有接收就绪的数据吗?
        rjmp    idle                     ; 无,等待
        cbr     status, 1≪RDR            ; 清 RDR 标志
        mov     R0,buffer                ; 有,将接收数据存入 R0
idle1:  sbrs    status, RDR              ; 有接收就绪的数据吗?
        rjmp    idle1                    ; 无,等待
        cbr     status,1≪RDR             ; 有,清 RDR 标志
        mov     R1,buffer                ; 将接收数据存入 R1
```

4.8 休眠方式的应用

对电池供电的嵌入式仪表来说,低功耗的休眠方式具有非常重要的实用价值。下面我们研究一个如何进入掉电和闲置方式,以及怎样从这些方式下唤醒 MCU 的问题。顺便示范一下监视定时器的启动和关闭操作。

4.8.1 掉电方式的进入和退出

[例 4.34] 如图 4.14 所示,B 口外的 8 只 LED 发光约 1 s 后,令 AT90S8515 进入掉电方式,LED 熄灭。经过 1.9 s,WDT 溢出使系统复位,将 MCU 从掉电方式下唤醒,并再次点亮 LED,如此循环不已。

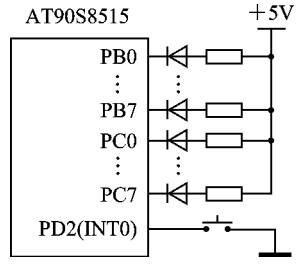


图 4.14 休眠方式实验

```
.include"8515def.inc"
    rjmp   RESET          ; 转复位处理
    rjmp   EINT0          ; 转外部中断 0 处理
.org    $007              ; 定时器 0 中断向量
  rjmp     TOVF0          ; 转定时器 0 中断处理
.org    $010
.def    tmp=r16
.def    tmp1=r17          ; 用以保存 $ff
.def    tmp2=r18          ; 用以保存 $00
.def    cnt=r19
tovf0:  wdr                ; WDT 清 0
```

```
        dec     cnt                     ; 中断次数减 1
        brne    tomain                  ; 未到 1.15 s,准备返回
        ldi     cnt, 35                 ; 已到 1.15 s,cnt 值复原
        set                             ; T 标志置 1
tomain: reti                            ; 中断结束
RESET:  ldi     tmp, $02                ; 设置堆栈
        out     SPH,tmp
        ldi     tmp, $5f
        out     SPL,tmp                 ;SP← $025f
        ser     tmp1                    ; tmp1← $ff
        out     DDRB, tmp1              ; B 口作输出
        out     DDRC, tmp1              ; C 口作输出
        out     PORTC,tmp1              ; 熄灭 C 口外 LED
        out     PORTA, tmp1             ; A 口和 D 口均作
        out     PORTD,tmp1              ; 带内部上拉电阻的输入
        ldi     tmp, $40
        out     GIMSK,tmp               ; 开 INT0 中断
        ldi     tmp,$05                 ; 8 MHz/1 024,表 1.3
        out     TCCR0, tmp              ; 启动 T/C0
        ldi     cnt, 35                 ; 中断次数计数初值
        ldi     tmp,1≪TOIE0
        out     TIMSK,tmp               ; 开 T/C0 溢出中断
        wdr                             ; WDT 启动前清 0
        ldi     tmp,$0f                 ; 准备启动 WDT
        out     WDTCR,tmp               ; WDT 启动,1.9 s 后溢出
        sei                             ; 开总体中断
        sbi     ACSR,ACD                ; 切断模拟比较器电源
here:   brtc    here                    ; 1 s 后进入掉电方式
        clt                             ; T 标志清 0
        out     PORTB,tmp1              ; 熄灭 B 口 LED
        clr     tmp2
        out     TCCR0,tmp2              ; 关闭 T/C0
        ldi     tmp,$30                 ; INT0 低电平触发
        out     MCUCR,tmp               ; 开放掉电方式
        sleep                           ; MCU 进入掉电方式
```

MCU 复位后,这段程序把 B 口和 C 口设定成输出端口。C 口输出高电平,关闭其外接 8 只 LED。而 B 口输出低电平(复位值),故其外接 LED 复位后即发光。A 口和 D 口作为带上拉电阻的输入。

8 MHz 晶振信号经 1 024 分频后作 T/C0 的输入时钟,T/C0 从 TCNT0 的复位值 $00 开始对该时钟脉冲进行计数,即作定时器运行。每过 1/8×1 024×256=32 768 μs,T/C0 溢出一次,35 次溢出延时约 1.15 s。

监视定时器 WDT 溢出时限为 1.9 s,但只要 T/C0 运行,它就永远不会溢出。因为每过

约 33 ms T/C0 中断时都将 WDT 值清 0。

MCU 正常运行(以 B 口 LED 亮为标志)1.15 s 后,关闭 T/C0,熄灭 B 口 LED,MCU 进入掉电方式。但 WDT 仍在运行。1.9 s 后 WDT 溢出,导致 MCU 复位。该复位操作唤醒掉电方式下的 MCU,使之再次重复复位后的一系列操作。

4.8.2 闲置方式的进入和退出

[例 4.35] 在例 4.34 中,当 AT90S8515 处于掉电方式下时,按动 INT0 按钮,使 MCU 从掉电方式下醒来,亮 C 口外 LED,约 1 s 后 AT90S8515 进入闲置方式。随之关闭 INT0 中断,由 T/C0 中断唤醒 MCU。

下面一段程序接在例 4.34 中的 sleep 指令后面。在 MCU 被 INT0 中断从掉电方式下唤醒并执行完 EINT0 中断服务程序后,便开始运行这段程序。

```
        ldi     tmp, $1f
        out     WDTCR, tmp              ; WDTOE=WDE=1
        ldi     tmp, $17                ; 准备关闭 WDT
        out     WDTCR,tmp               ; 关闭 WDT
```

若希望 WDT 仍继续运行,则以上四条指令应当撤消。

```
        out     TCNT0,tmp2              ; T/C0 清 0
        ldi     tmp,$05
        out     TCCR0,tmp               ; 再次启动 T/C0
        ldi     cnt, 35                 ; 中断次数记录初值
        out     PORTC,tmp2              ; 亮 C 口外 LED
here1:  brtc    here1                   ; 1.15 s 后进入闲置方式
        clt                             ; T 标志清 0
        out     PORTC,tmp1              ; 熄灭 C 口外 LED
        ldi     tmp,$20
        out     MCUCR, tmp              ; 开放闲置方式
idle:   sleep                           ; 令 MCU 进入闲置方式
        brtc    idle                    ; 不到 1.15 s,继续休眠
        clt                             ; T 标志清 0
        rjmp    here1-1                 ; 转入工作状态
EINT0:  out     GIMSK,tmp2              ; 关 INT0 中断
        reti
```

从这段程序中不难得出结论,INT0 中断以后,MCU 运作 1.15 s,又闲置 1.15 s,再运作,再闲置,周而复始循环不已。MCU 的动作和闲置分别以 C 口外 LED 的亮和暗作标志。

第五章　接口技术

本章主要研究 AVR 微控制器与显示器、键盘、打印机以及 A/D 转换器的配合使用问题。

5.1　与显示器接口

单片机系统中使用最多的显示器有发光二极管显示器 LED 和液晶显示器 LCD 两种。

5.1.1　发光二极管显示器

1. 结构及原理

欲显示十进制或十六进制数字及某些其他简单字符，可选用七段 LED 显示器。图 5.1 所示为其显示字符形状。a～g 七段和十进制小数点 dp 均为一发光二极管。若系共阳极结构，则它们的阳极为一公共点，接电源正极。八只发光二极管的阴极各自独立，哪一个阴极接地，哪一段就发光，阴极也接高电平者便呈暗状。如果是共阴极结构，那么阴极公共点接地，各阳极独立，接高电平者发光，阳极接地者呈暗状。

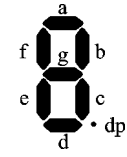


图 5.1　LED

根据以上讨论可知，欲在一 LED 显示器上显示某特定字符，必须向其公共点及各段施加正确的电压。对公共点的施压操作称位选；对各段的操作为段选。段选码亦称字形码。表 5.1 中列出了显示字符与其字形码的对应关系。应当注意，在一个字节的字形码中，一般各段的顺序由低至高依次是 a、b、c、d、e、f、g 和 dp。

表 5.1　七段 LED 显示器字形码

显示字符	共阴极字形码	共阳极字形码	显示字符	共阴极字形码	共阳极字形码
0	$3F	$C0	C	$39	$C6
1	$06	$F9	d	$5E	$A1
2	$5B	$A4	E	$79	$86
3	$4F	$B0	F	$71	$8E
4	$66	$99	P	$73	$8C
5	$6D	$92	U	$3E	$C1
6	$7D	$82	r	$31	$CE
7	$07	$F8	y	$6E	$91
8	$7F	$80	H	$76	$89
9	$6F	$90	L	$38	$C7
A	$77	$88	不显示	$00	$FF
b	$7C	$83			

2. 与 AVR 单片机接口

AVR 的 I/O 线有较大的驱动能力，故可直接与 LED 显示器相连，无须加缓冲芯片。

对 LED 显示器的控制，无非是按时向它们提供具有一定驱动能力的段选和位选信号。段选信号，即字形码，可由硬件产生，也可用软件法获得。LED 显示有动态扫描显示和静态显示之分。在单片机系统中，为节省硬件资源，多采用动态扫描显示法，且字形码由软件产生。

[例 5.1] 为某 AT90S8515 微控制器装配六只 LED，令其中四只显示地址，而另两只则显示该地址单元的内容。

如图 5.2 所示，AT90S8515 的 B 口同时接到 6 只 LED 上，该口的 8 条 I/O 线均通过一 470 Ω 电阻各接每个 LED 的一段。6 只 LED 的阴极 ka 分别由 AVR 的 PD5～PD0 输出引脚控制。

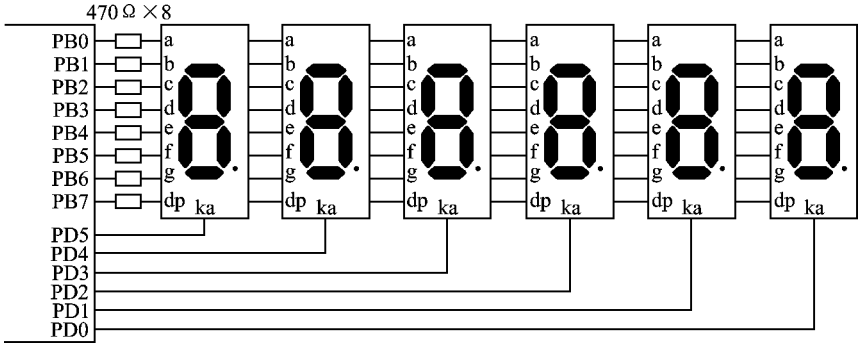


图 5.2 AT90S8515 与 6 只 LED 的接口

由图 5.2 可见，AT90S8515 通过其 B 口控制着六只 LED 显示器段选信号的产生。显然，若在这些显示器上各显示不同的字符，必须采用动态扫描法。即六只显示器逐一轮流显示，每位持续若干 ms(例如 1 ms、10 ms 或 20 ms)循环一遍，如此周而复始。这样，由于视觉暂留现象的作用，我们看到的便是六只 LED 同时在显示了。

MCU 的 B 口和 D 口都设定成输出方式，B 口负责 LED 的段选，D 口则进行位选。D 口的 8 位位选信号中，只输出一个 0，其他位均为 1。例如，从 PD5 为 0 开始，选中最左边的 LED。1 ms 后，PD4 输出 0，点亮左起第二只显示器，如此等等。

假如要显示 8515 片内 SRAM 某地址单元的内容，以 X 为地址指针，晶振频率等于 8 MHz，则有关程序段如下。

```
        .EQU    PATN= $100
        LDI     ZH,HIGH(PATN*2)          ; 建立字形码表指针
        LDI     R16,1≪TOIE0
        OUT     TIMSK,R16                ; 开定时器 0 中断
        SEI                              ; 开总体中断
START:  LDI     R17,0B11011111           ; 扫描字，从 PD5 开始
        MOV     R16,XH                   ; 取地址高字节
        SWAP    R16                      ; 其高低 4 位交换位置
        CBR     R16, $F0                 ; 原低 4 位清 0
        MOV     R1,R16                   ; R1 含 16 位地址最高 4 位
```

```
        MOV     R16,XH          ;再取地址高字节
        CBR     R16,$F0         ;高4位清0,保留低4位
        MOV     R2,R16          ;R2含16位地址次高4位
        MOV     R16,XL          ;取地址低字节
        SWAP    R16             ;高低4位交换位置
        CBR     R16,$F0         ;原低4位清0
        MOV     R3,R16          ;R3含16位地址次低4位
        MOV     R16,XL          ;再取地址低字节
        CBR     R16,$F0         ;高4位清0,保留低4位
        MOV     R4,R16          ;R4含16位地址最低4位
        LD      R16,X           ;读SRAM内容
        SWAP    R16             ;高低4位交换位置
        CBR     R16,$F0         ;清原低4位,保留原高4位
        MOV     R5,R16          ;R5含SRAM内容高4位
        LD      R16,X           ;再读SRAM内容
        CBR     R16,$F0         ;清其高4位,保留低4位
        MOV     R6,R16          ;R6含SRAM内容低4位
        MOV     ZL,R1           ;读十六进制地址码最高位
LOOP:   LPM                     ;查其字形码
        OUT     PORTB,R0        ;输出显示
        OUT     PORTD,R17       ;位选
        LDI     R16,(256-125)   ;125个T/C0周期后,
        OUT     TCNT0,R16       ;令T/C0溢出
        LDI     R16,$03         ;对晶振64分频
        OUT     TCCR0,R16       ;并启动T/C0
        LDI     R16,$20
        OUT     MCUCR,R16       ;SE=1,SM=0
        SLEEP                   ;MCU进入闲置方式
        SEC                     ;C←1
        ROR     R17             ;扫描字右旋转
        BRCC    START           ;6位已显示一遍,再重来
        SBRS    R17,4           ;左1位刚显示过?否,跨越
        MOV     ZL,R2           ;是,准备显示左2位
        SBRS    R17,3           ;左2位刚显示过?否,跨越
        MOV     ZL,R3           ;是,准备显示左3位
        SBRS    R17,2           ;左3位刚显示过?否,跨越
        MOV     ZL,R4           ;是,准备显示左4位
        SBRS    R17,1           ;左4位刚显示过,否,跨越
        MOV     ZL,R5           ;是,准备显示左5位
        SBRS    R17,0           ;左5位刚显示过,否跨越
        MOV     ZL,R6           ;是,准备显示左6位
        RJMP    LOOP            ;继续循环
TOVF0:  CLR     R16             ;定时器0中断服务程序
```

```
OUT     TCCR0,R16                    ; 令 T/C0 停止运行
RETI
.CSEG
.ORG    PATN
.DB   $3F, $06, $5B, $4F, $66, $6D, $7D, $07
.DB   $7F, $6F, $77, $7C, $39, $5E, $79, $71
.DB   $73, $3E, $31, $6E, $76, $38, $00
```

从这段程序中不难看出,0～F 等 16 个字符的字形码是通过 LPM 指令从程序存储器里存放的表格中查得的。表首地址 PATN 等于 $100,以待显示的十六进制数字作偏移量。至于表首地址高字节为什么要乘以 2 后才送至 ZH 寄存器的问题,前面已多次谈到(例 3.4 和例 3.48)。表首地址低字节为 $00,这样就省却了许多麻烦。否则低字节也得乘以 2 后再送往 ZL 寄存器。以整个 16 位地址码×2+偏移量作指针才可查到 ROM 表格中的任何一个字节。

查得字形码并输出给相应的 LED 显示器后,立即启动 T/C0,并随之令 MCU 进入闲置方式。尽管 MCU 在休眠,但 LED 显示仍在进行。1 ms 后,T/C0 溢出产生中断,MCU 被此中断从闲置状态下唤醒。MCU 执行完 TOVF0 中断服务程序,便从 SLEEP 指令后面恢复程序的运行。于是继续显示下一位,或开始新的一轮循环。

3. 与 AVR 单片机的接口之二——字形码的硬件译码法

图 5.3 所示为 BCD 七段锁存/译码/驱动器 MC14513 芯片的外形。BCD 码 0B0000～0B1001 由 D、C、B、A 四个引脚输入进来,当 LE 输入端上有一高电平信号时,所输入的 BCD 码即被锁存在芯片内部。内部译码后,由 a～g 七个引脚输出字形码。

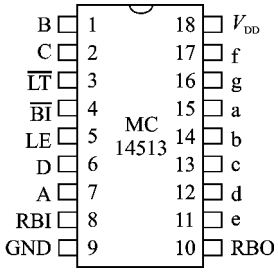


图 5.3 MC14513 外形

$\overline{LT}$为测试输入引脚。若由此输入低电平,则可使 LED 各段全亮,与 ABCD 输入信号无关。

$\overline{BI}$系熄灭输入端,若输入低电平,则各段均呈暗态。

RBI 是高位 0 消隐输入引脚,高电平有效。若此脚接高电平,则当 BCD 码为 0 时,产生消隐信号,使有效数字前的 0 不显示。

RBO 为高位 0 消隐输出端。当高位 0 消隐时,这里输出高电平,可作多位 0 消隐用。

[例 5.2] 为某 AT90S1200 系统设计一个能显示 3 位十进制数的 LED 显示器。具体要求如下:

- CPU 特别繁忙,用在显示上的时间越少越好;
- 有效数字前的 0 不显示。

根据题义要求,我们选用三片 MC14513 作为 LED 显示器的锁存/译码/驱动器,采用静态显示法,启用消隐功能。

图 5.4 中,U1、U2 和 U3 为三个 MC14513 芯片。其中 U1 和 U2 用了高位 0 消隐功能:U1 的 RBI 接高电平,故百位显示上不显示 0;此时 RBO 向 U2 的 RBI 输出高电平,故在百位消隐 0 时,十位显示器也不显示 0。

现在编写一段子程序,把某工作寄存器中的 8 位无符号二进制数以 3 位十进数的形式显示出来。读者从这段子程序中可领悟到什么是静态显示——待显示的字符被锁存入各

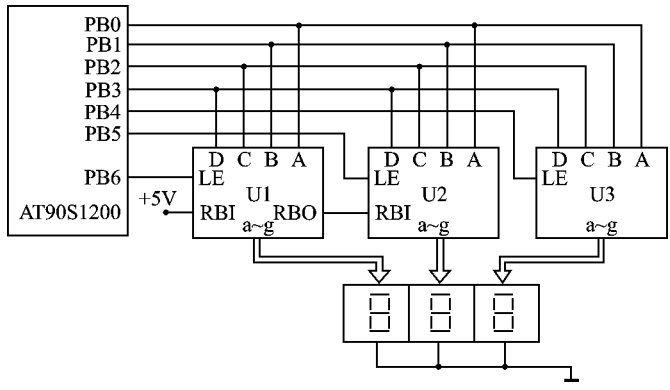


图 5.4　三位静态显示器

MC14513 芯片中，无须周而复始地逐位扫描。在重新锁入其他字符前，原显示内容是不会发生变化的。不难理解，静态显示要比动态扫描显示亮度好些。

下面是一段显示子程序 DISP。其入口条件是在主程序中已将 B 口设定成了输出方式，待显示的数据在 BIN 寄存器中。

```
        .DEF    BIN=R16                 ;待显示二进制数
        .DEF    HUND=R17                ;存放百位数
        .DEF    TENONE=R18              ;存放十位与个位数
        .DEF    TMP=R19                 ;暂存器
DISP:   RCALL   B8BCD                   ;见例 3.37
        OUT     PORTB, HUND             ;输出百位数
        SBI     PORTB,6                 ;锁入 U1 中
        MOV     TMP,TENONE              ;准备处理十位数
        SWAP    TMP                     ;十位数和个位数交换位置
        CBI     PORTB,6                 ;撤消 U1 的 LE 信号
        CBR     TMP, $F0                ;个位数清 0,保留十位数
        OUT     PORTB,TMP               ;输出十位数
        SBI     PORTB,5                 ;锁入 U2 中
        CBR     TENONE, $F0             ;十位数清 0
        CBI     PORTB,5                 ;撤消 U2 的 LE 信号
        OUT     PORTB,TENONE            ;输出个位数
        SBI     PORTB,4                 ;锁入 U3 中
        NOP                             ;延时一个时钟周期
        CBI     PORTB,4                 ;撤消 U3 的 LE 信号
        RET                             ;子程序结束
```

5.1.2　液晶显示器

液晶显示器，简称 LCD，是一种功耗很低的显示器件。其应用领域非常广泛，从电子表到计算器，从袖珍仪表到便携式微型计算机，几乎随处可见。这一节着重讨论 AVR 与 LCD 的配套使用问题。

1. AVR 直接驱动液晶显示器

LCD 是由背板(背极)和一定数量的段或点构成的,这些段和点用以形成各种显示形象。在任何段与背极间施加一电压,通常为 4 V 或 5 V,即可使该段呈现出黑色。惟一有点烦人的问题是,所施加的电压必须周期性地改变极性,否则 LCD 中就要发生化学反应,并最终导致液晶的损坏。

为防止液晶损坏,背板和各段须用 AC 信号驱动,后者通常由方波电压派生而来。如果某段应当关闭,那么该段与背板用同一波形电压驱动。这样,这段始终与背板同电位。若某段应显示,则该段用与背板反相的电压驱动。这就是说,该段与背板之间有一周期改变极性的 5 V 电压。

在少许软件的支持下,AVR 即可胜任驱动 LCD 的这项工作,无须外加 LCD 驱动器。

[例 5.3] AT90S8515 直接控制两位 LCD 显示器,显示工作寄存器 bcdval 中的两位 BCD 数。

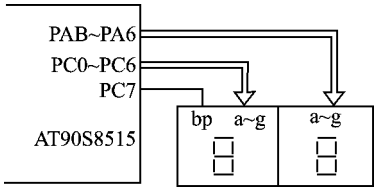


图 5.5 AVR 直接驱动 LCD

如图 5.5 所示,AT90S8515 的 C 口控制显示十位数的七段 a～g 和背板 bp;B 口控制显示个位数的七段 a～g。定时器 0 用来划分驱动电压的半周期。LCD 驱动电压的重复频率应介于 30～100 Hz 之间。若以 12 ms 作为半周期,则频率约等于 42 Hz。如果晶振频率为 8 MHz,经 1 024 分频后作 T/C0 时钟,那么 TCNT0 初值将等于 256－94＝162。式中 94 是由 12 ms 除以 T/C0 时钟周期得来的。

AT90S8515 通常处于闲置方式下,T/C0 每过 12 ms 产生一次溢出中断,使 8515 退出闲置方式。定时器 0 中断服务程序显然应包括下列两项任务:

- 重置 TCNT0 初始值;
- 使 LCD 各段之电位反相。

下面是有关程序段落。这段程序未涉及待显示值 bcdval 的更新问题,这可能在其他中断服务程序中进行。不过其值每更新一次,便要调用一次 DISUP 子程序。至于 LCD 内部的电压反相工作,则由 tovf0 中断服务程序去完成。

```
        .include"8515def.inc"
          rjmp    RESET
        .def    bcdval=r1          ;存待显示的 BCD 数
        .def    tens=r16           ;存待显示的十位数字形码
        .def    ones=r17           ;存待显示的个位数字形码
        .def    tmp=r18            ;暂存器
        .equ    PATN=$100          ;字形码表首地址
        .org    $007               ;定时器 0 溢出中断向量
          rjmp    tovf0            ;转 T/C0 溢出中断处理
        .org    $010
RESET:  ……
        wdr                        ;WDT 启动前先清 0
        ldi     tmp,$08            ;溢出时限 15 ms
```

```
        out     wdtcr, tmp          ; 启动 WDT
        ser     tmp                 ; tmp←$ff
        out     ddra,tmp            ; A 口作输出
        out     ddrc,tmp            ; C 口作输出
        ldi     zh, high (PATN*2)   ; 建立字形码指针
        ldi     tmp,1≪TOIE0
        out     timsk,tmp           ; 允许 T/C0 溢出中断
        sei                         ; 开总体中断
        ldi     tmp(256-94)         ; TCNT0 初值
        out     tcnt0,tmp           ; T/C0 溢出周期 12 ms
        ldi     tmp, $05            ; 晶振频率经 1 024 分频后,
        out     tccr0, tmp          ; 作 T/C0 时钟,并启动 T/C0
        rcall   disup               ; 调用显示更新子程序
        ldi     tmp, $20
        out     mcucr, tmp          ; 准备令 MCU
loop:   sleep                       ; 进入闲置方式
        rjmp    loop                ; 再入闲置方式
tovf0:  wdr                         ; WDT 清 0
        ldi     tmp,(256-94)        ; 每 12 ms 溢出一次
        out     tcnt0,tmp           ; TCNT0 恢复初值
        com     tens                ; 十位数字形码取反
        com     ones                ; 个位数字形码取反
        out     portc, tens         ; 显示十位数
        out     porta, ones         ; 显示个位数
        reti                        ; 中断结束
disup:  cli                         ; 暂关中断
        mov     tens, bcdval        ; 取待显示 BCD 数
        swap    tens                ; 个位数与十位数交换位置
        andi    tens, $0f           ; 屏蔽掉个位数
        mov     zl,tens             ; 准备查表
        lpm                         ; 查字形码表
        out     portc,r0            ; 显示十位数
        mov     tens,r0             ; 保存十位数字形码
        mov     ones,bcdval         ; 取待显示 BCD 数
        andi    ones, $0f           ; 屏蔽掉十位数
        mov     zl,ones             ; 准备查表
        lpm                         ; 查字形码
        out     porta,r0            ; 显示个位数
        mov     ones,r0             ; 保存个位数字形码
        sei                         ; 重开中断
        ret                         ; 子程序结束
PATN:   ……                          ; 字形码表,略
```

为了更新显示内容，要通过查表法获得字形码。在字形码表中，显示段用 1 表示，不显示段为 0，背板也为 0。待显示的数字以 BCD 形式存放在工作寄存器 bcdval 中。查表操作根据 BCD 值的低半字节进行。在向 LCD 写入新字形码前，应关闭中断，以防止恰巧在写过十位数的字形码后发生驱动电压极性的变化。因 LCD 背板电位和十位数之字形码同时给出，故若此后电压反相，则个位数字形码将适得其反。

2. 液晶显示模块 LCDM

由以上讨论可知，欲用 LCD 进行字符或图形显示，必须有下列硬件资源：

- 一定容量的 RAM，用来存放待显示的数据；
- 一定容量的 ROM，用以存放点阵字形码表；
- LCD 控制器，控制读写数据、生成点阵字形码、将液晶驱动信号转换成 AC 信号等项操作。

现在有关厂家已将上述资源及点阵式 LCD 显示器用印刷电路板连成了一个整体，这就是液晶显示模块 LCDM。用户只要向它送出相应命令和数据便可实现所需要的显示。LCDM 与各种 CPU 接口非常简单，使用起来特别方便灵活。LCDM 产品有字符型和图像型两种。在图像方式下，RAM 中的一位数据控制着 LCD 上一个点的亮与暗；在字符方式下，LCD 控制器根据存放在 DDRAM 中的字符代码从字符发生器 CG ROM 中查得其相应点阵数据，即字形码。

(1) 字符型 LCDM 的基本结构：LCD 控制器控制着下列各部件的运作。

① LCD 显示板：字符型液晶板上排列有若干个 5×7 或 5×10 点阵的字符显示位，每个显示位可显示一个字符。LCDM 有一行、两行和四个显示行的三种规格，每行最多可显示 8、16、24、40 和 80 个字符不等。例如 UC404－01 即为 40×4 规格的 LCDM，它有 4 个显示行，每行包括 40 个字符。

② 字符生成器 CG ROM：内部存放着 192 个字符的 5×7 和 5×10 点阵字形码。这 192 个字符包括英文大小写字母、标点符号和数学符号等。

③ 字符生成器 CG RAM：LCDM 允许用户自定义 4 个 5×10 或 8 个 5×7 点阵的字符。CG RAM 就用来存放这些字符的点阵字形码。这些自定义字符的 ASCII 码为 $00～$07 或 $08～$0F。每个字符的 5×7 点阵字形由 8 个字节构成，第 8 个字节为 $00，系光标所在位置(详见表 5.2)。

④ 显示数据 DD RAM：实际上这是一个 80 个字节的显示缓冲区。欲在 LCD 板某位置上显示某个字符，必须首先把该字符的 ASCII 码送往 DD RAM 与显示位置相对应的地址单元。此后液晶显示控制器根据该字符代码来确定其点阵字形码在 CG ROM 中的地址。这样就能找到任何一个字符的点阵字形，于是便可在 LCD 板上显示出它的形象来了。

DD RAM 地址与 LCDM 显示面板上字形位置的对应关系如图 5.6 所示，图中涵盖了 16×2、20×2 和 40×2 等三种规格。

顺便指出，对 20×4 规格的 LCDM 来说，其第一行的最右面一个显示位置与第三行最左面的一个显示位置所对应的 DD RAM 地址是连续的，偶数行也是如此。

DD RAM 的 80 个地址单元被划分成上半区(地址从 $00 至 $27)和下半区($40～$67)。如果显示不移位，那么显示板上下两行各显示相应半区的前 16 个(20 或 40)单元的内容。若显示连续左移或右移，则每半区 40 个单元的内容都可在相应显示行上移动地显示

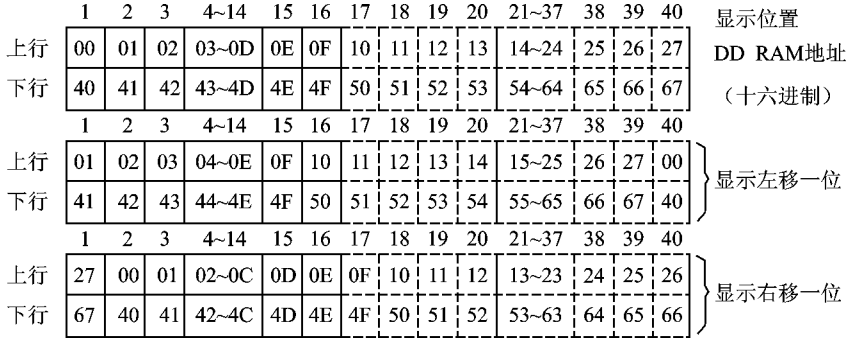


图 5.6　DD RAM 地址与字符显示位置的对应关系

出来。

例如，DD RAM 某地址单元的内容为 $41，这是字符 A 的 ASCII 码，LCD 控制器根据此字符码确定 CG ROM 中的若干个地址。这些 CG ROM 地址的内容便是字符 A 的 5×7 或 5×10 点阵字形码。如果上述 DD RAM 的单元地址为 $0F，那么 16×2 规格的 LCDM 显示板上行最右面一个位置上就要显示出字母 A 来。

但若 DD RAM 某单元的内容为 $00～$07 或 $08～$0F，则 LCDM 控制器知道这是用户自定义字符，要到 CG RAM 中去找此字符的点阵字形码。如表 5.2 所列，DD RAM 某单元 8 位内容为 0000xaaa，即 $00～$0F 中的任一值，前已谈及，这是自定义字符，其 5×7 点阵字形码位于 CG RAM 的 8 个地址单元中。CG RAM 6 位地址码中的高 3 位即为字符码的最低 3 位，表格中以 aaa 三位代表。假如 DD RAM 某单元内容为 $07，则它所代表字符的 5×7 点阵字形码将位于 CG RAM $38～$3F 等 8 个地址单元中。读者把相邻的“1”用线连起来，就会看到这是一个字符￥。表中“x”为无关位，它是 0 或 1 均无影响。

表 5.2　自定义字形码的生成

DD RAM 数据	CG RAM 地址		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
0 0 0 0 x a a a	a a a	0 0 0	x x x 1 0 0 0 1
		0 0 1	x x x 0 1 0 1 0
		0 1 0	x x x 1 1 1 1 1
		0 1 1	x x x 0 0 1 0 0
		1 0 0	x x x 1 1 1 1 1
		1 0 1	x x x 0 0 1 0 0
		1 1 0	x x x 0 0 1 0 0
		1 1 1	x x x 0 0 0 0 0

⑤ 寄存器和地址计数器：LCDM 内部有指令寄存器 IR 和数据寄存器 DR 两个寄存器及一个地址计数器 AC。后者用来指示 DD RAM 和 CG RAM 的地址。

⑥ LCDM 引线：LCDM 印制板上有 14 条引出线，其中数据线 8 条，控制线 3 条及电源线 3 条，如表 5.3 所列。

(2) 指令功能：指令代码通过 DB0～DB7 等 8 条数据线传递，由 RS 和 R/$\overline{W}$ 两信号线控

制(见表 5.4)。数据线所传递的 8 位指令码决定指令功能——清屏、返回、输入方式设置、显示开关控制、移位控制、功能设置、CG RAM 地址设置、DD RAM 地址设置、数据的读写及标志的读出等。

表 5.3 LCDM 引出线

引线号	符 号	名 称 及 功 能
1	V_{SS}	电源地线,0V
2	V_{DD}	电源线 5 V±5%
3	V_{LCD}	液晶驱动电压输入线
4	RS	寄存器选择信号线,1 选 DR,0 选 IR
5	R/$\overline{W}$	读/写控制信号线,1 为读信号,0 为写信号
6	E	使能信号线,高电平读数据,下降沿写数据
7～14	DB0～DB7	8 位数据线,双向,三态

表 5.4 LCDM 寄存器操作控制

RS	R/$\overline{W}$	操 作
0	0	指令寄存器写入
0	1	忙标志和地址计数器内容读出
1	0	数据寄存器写入
1	1	数据寄存器读出

表 5.5 中综合了 LCDM 的 11 条指令。表中的执行时间对应工作频率 250 kHz,若 LCDM 工作频率有变,则执行时间可按周期数换算出来。例如 250 kHz 下为 40 μs,在 270 kHz 下就应等于 37 μs。

下面我们将根据该表内容对这些指令逐一给出详细说明。

表 5.5 LCDM 指令表

序 号	操作选择		指令功能	指 令 控 制 字								执行时间
	RS	R/$\overline{W}$		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	0	0	清显示	0	0	0	0	0	0	0	1	1.64 ms
2	0	0	软复位	0	0	0	0	0	0	1	x	1.64 ms
3	0	0	内部方式设置	0	0	0	0	0	1	I/D	S	40 μs
4	0	0	显示开关控制	0	0	0	0	1	D	C	B	40 μs
5	0	0	移位控制	0	0	0	1	S/C	R/L	x	x	40 μs
6	0	0	系统方式设置	0	0	1	DL	N	F	x	x	40 μs
7	0	0	CG RAM 地址设置	0	1	CG RAM 地址码						40 μs
8	0	0	DD RAM 地址设置	1	DD RAM 地址码							40 μs
9	0	1	忙状态检查	BF	AC							40 μs
10	1	0	RAM 写入	DATA								40 μs
11	1	1	RAM 读出	DATA								40 μs

① 清屏指令：清除屏幕显示，即把空格字符 ASCII 码 $20 写入所有相应 DD RAM 单元，并将 DD RAM 地址指针复位成 AC=$00。

② 光标复位指令：置 DD RAM 地址为 0，光标返回至原始位置，DD RAM 内容不变。

③ 输入方式设置指令：设置数据输入时光标移动方向，确定随着新数据的输入是显示整体移动，还是光标移动。若 I/D 位为 1，则指定光标或整体显示沿 DD RAM 地址递增方向移动。I/D=0，光标或显示沿 DD RAM 地址递减方向移动。S=1，随着新数据的输入，显示将整体移动；S=0 则指定光标将随新数据的输入而移动。光标和显示的移动，均在 DD RAM 的一次读或写操作后自动完成。

④ 显示开关控制指令：其中 D、C 和 B 三位分别控制着显示的开启与关闭、要否光标配合以及光标处的字符闪烁与否。D=1，开放显示，DD RAM 的内容显示在屏幕的相应位置上。D=0 则关闭显示，DD RAM 内容保持不变。C=1 代表有光标配合，光标出现在 5×7 字符点阵的第 8 行上。C=0 则关闭光标功能。B=1 指定光标处字符要闪烁，B=0 则不要闪烁。

⑤ 光标或显示移位指令：这是一条单纯的移位指令，它不改变 DD RAM 各单元的内容。S/C=1 为显示移位命令；S/C=0 则是光标移位命令。R/L=1 代表右移；R/L=0 则系左移。

⑥ 系统设置指令：负责设定接口数据总线的宽度、显示行数及字符点阵的大小。DL=1 设定接口数据总线为 8 位，即 DB7～DB0 均投入使用。DL=0 则指定用 4 位总线接口，仅 DB7～DB4 连接，DB3～DB0 不用。这时指令或数据要分两次传送，先高 4 位，后低 4 位。N=0 设定为单行显示；N=1 则要进行双行显示。F=0 限用 5×7 字符点阵；而 F=1 则指定用 5×10 点阵。

⑦ 字符生成器 CG RAM 地址设置指令：用来给出 CG RAM 的地址。但应注意，CG RAM 地址为 $00～$3F，而按指令要求控制字应写成 $40～$7F。

⑧ DD RAM 地址设置指令：用来给出显示数据 RAM 的地址。DD RAM 的地址范围为 $00～$7F，但控制字得相应写成 $80～$FF，因为对这条指令来说，其 DB7 必须为 1。DD RAM 地址设定之后，对 DD RAM 的读写操作便是针对此地址的。

⑨ 状态及地址读出指令：用来读出忙标志 BF 及地址计数器 AC 的内容。所读结果之最高位为 BF，若为 1，则表示 LCDM 正忙于内部操作，无暇接受命令和数据的写入。BF=0 说明 LCDM 不忙，可以接受写入操作。所读结果的低 7 位系地址计数器 AC 之值。

⑩ RAM 写入指令：向 CG RAM 或 DD RAM 写入数据应与相应的地址设置指令相结合。在向 DD RAM 某地址单元写入显示字符的 ASCII 码后，AC 的内容和光标的位置将根据 I/D 位的设置而自动增减和移动。

⑪ RAM 数据读出指令：从 CG RAM 或 DD RAM 中读出数据，读后 AC 内容及光标位置将根据 I/D 位的设置而自动增减和移位。

[例 5.4]　在 DMC16230 LCDM 屏幕上分两行显示“WELCOME TO THE AVR WORLD !”，然后向左移出屏幕，但要求此显示过程循环进行。

如图 5.7 所示，AT90S8515 的 B 口作数据总线，用以在 AVR 与 LCDM 间传递命令和数据；PD2～PD4 用来向 LCDM 输出使能信号、读/写选择和寄存器选择信号。LCDM 的 V_{LCD} 引脚接 20 kΩ 电位器的滑动端，便于调节显示的辉度。下面是有关程序段落。

```
.include"8515def.inc"
rjmp        RESET
```

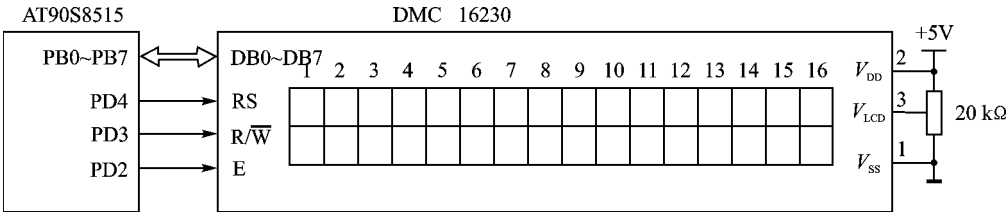


图 5.7 AVR 与 LCDM 的接口

```
        .def    tmpff=r16           ; 存放常数 $ff
        .def    tmp00=r17           ; 存放常数 $00
        .def    tmp8=r18            ; 存放常数 $08
        .def    tmprs=r19           ; 控制 RS 位进行寄存器选择
        .def    tmp=r20             ; 暂存器
        .def    cnt=r21             ; 字节计数
        .def    cnt1=r22            ; 延时用暂存器
        .def    cnt2=r23            ; 延时用暂存器
        .def    cnt3=r24            ; 延时用暂存器
        .org    $010
RESET:  ……
        ser     tmpff               ; tmpff←$ff
        out     ddrd, tmpff         ; D 口作输出
        clr     tmp00               ; tmp00←$00
        out     ddrb,tmp00          ; B 口暂作输入
        out     portb,tmp00         ; B 口作三态输入
        ldi     tmp8,0b00001000     ; RS=0,R/W=1,E=0
cycle:  out     portd, tmp8         ; 发出 RS、R/W 和 E 信号
        ldi     tmp, $01            ; 准备清屏
        ldi     tmprs, $00          ; 准备向 LCDM 写入命令
        rcall   WRITE               ; 输出清屏命令
        ldi     tmp, $38            ; 8 位接口,双行显示,5×7 点阵
        rcall   WRITE               ; 发出上述命令
        ldi     tmp, $0c            ; D=1, C=0, B=0
        rcall   WRITE               ; 开显示,关光标,不闪烁
        ldi     tmp, $06            ; I/D=1,S=0
        rcall   WRITE               ; RAM 读写按地址递增方式
        ldi     tmp, $80            ; DD RAM 首地址为 $00
        rcall   WRITE               ; 指定 DD RAM 首地址
        ldi     zl, low (CHATBL * 2)    ; 设置显示字符
        ldi     zh, high (CHATBL * 2)   ; 在 ROM 中的地址指针
        ldi     tmprs, 0b00010000   ; 准备写数据
        ldi     cnt, 16             ; 字符计数初值
        rcall   DISPL               ; 向 DD RAM 写数据
        clr     tmprs               ; 准备写命令
```

```
        ldi     tmp, $c0            ; DDRAM 地址 $40
        rcall   WRITE               ; 写第二行显示 DD RAM 地址
        ldi     tmprs, 0b00010000   ; 准备写数据
        ldi     cnt, 16             ; 字符计数初值
        rcall   DISPL               ; 向 DD RAM 写数据
        rcall   DELAY               ; 延时约 0.6 s
        ldi     cnt, 16             ; 左移位数初值
        clr     tmprs               ; 准备发出命令
go_on:  ldi     tmp, $18            ; 显示左移命令
        rcall   WRITE               ; 发出显示左移命令
        rcall   DELAY               ; 延时约 0.6 s
        dec     cnt                 ; 左移位数减 1
        brne    go_on               ; 未到 16 位,继续左移
        rjmp    cycle               ; 已移 16 位,循环
WRITE:  sbi     portd, 2            ; E 信号上升沿
        rcall   D1us                ; 延时 1 μs
busy:   wdr                         ; WDT 清 0
        sbic    pinb, 7             ; BF=0 ?
        rjmp    busy                ; 否,再读 BF
        out     portd, tmprs        ; 发出 RS、R/W 和 E 信号
        out     ddrb, tmpff         ; B 口改作输出
        sbi     portd, 2            ; E 信号上升沿
        out     portb, tmp          ; 输出控制字或命令
        rcall   D1us                ; 延时 1 μs
        cbi     portd, 2            ; E 信号下降沿
        out     ddrb, tmp00         ; B 口再作输入
        out     portb, tmp00        ; B 口作三态输入
        out     portd, tmp8         ; RS=0, R/W=1, E=0
        ret
DISPL:  lpm                         ; 自 ROM 中读数据
        mov     tmp, r0             ; 并传送给 tmp
        rcall   write               ; 写入 DD RAM
        adiw    zl, 1               ; Z 指针内容加 1
        dec     cnt                 ; 字节数减 1
        brne    DISPL               ; 不为 0,再读 ROM
        ret                         ; 所有字节均已处理
D1us:   ldi     cnt1, 2             ; 延时约 1 μs
        dec     cnt1
        brne    D1us+1
        ret
DELAY:  ldi     cnt1, $64           ; 延时约 0.6 s
        ldi     cnt2, $80
        wdr
```

```
        ldi     cnt3，$80
        dec     cnt3
        brne    DELAY+4
        dec     cnt2
        brne    DELAY+3
        dec     cnt1
        brne    DELAY+1
        ret
CHATBL：.DB $57，$45，$4C，$43，$4F，$4D，$45，$20
        .DB $54，$4F，$20，$54，$48，$45，$20，$20
        .DB $41，$56，$52，$20，$57，$4F，$52，$4C
        .DB $44，$20，$21，$20，$20，$20，$20，$20
```

(3) LCDM 的复位：LCDM 有内部复位电路，能实现上电复位。复位期间忙标志 BF 为 1，电源电压升至 4.5 V 后，此状态可维持 10 ms。复位时 LCD 控制器执行下列命令：

① 清除显示。

② 输入方式设置指令中的 I/D=1，选用地址递增方式；S=0，关显示移位功能。

③ 显示开关控制中的 D=C=B=0，即显示、光标和字符闪烁功能均被关闭。

④ 功能设置指令中的 DL=1，选 8 位数据接口；N=0，单行显示；F=0，选 5×7 点阵字符。

(4) LCDM 的时序：从上段程序中可以看到，在对 LCDM 进行读写时，AVR 以软件方式产生控制信号，通过 PD4～PD2 等三条引脚分别传递给 LCDM 的 RS、R/$\overline{W}$ 和 E 等三引线，作寄存器选择、读写选择和使能控制。这样，在编写程序时，就必须要注意这三个信号间的相互关系，即按照 LCDM 工作时序的要求给出这些控制信号。

图 5.8 所示为 LCDM 读写时序，读者阅读前段程序时应与这里的时序要求相对照。

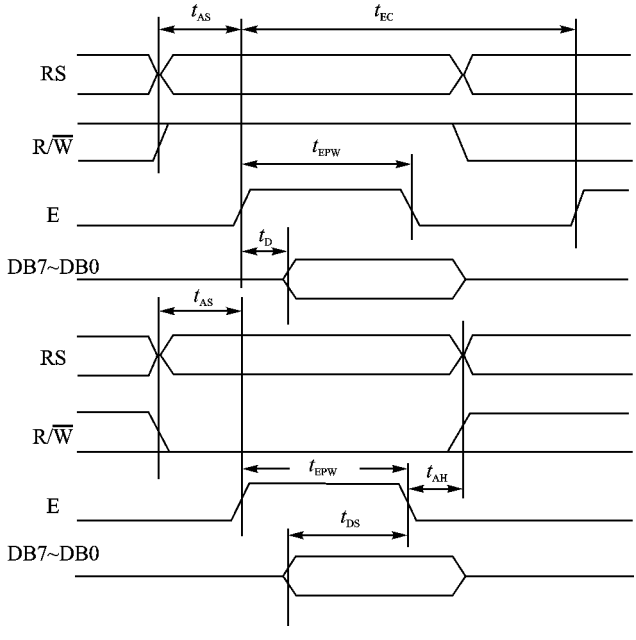


图 5.8 LCDM 工作时序

图中：t_{EC}：使能信号周期，最小值为 1 μs；

t_{AS}：地址建立时间，最小值为 140 ns；

t_{EPW}：使能信号宽度，最小值为 450 ns；

t_{DS}：数据建立时间，最小值为 195 ns；

t_{D}：数据延迟时间，最大值为 320 ns；

t_{AH}：地址保持时间，最小值为 10 ns。

5.2　与键盘接口

键盘是一种最常见的输入设置，用户可用以向计算机输入数据或命令。根据按键的识别方法分类，有编码键盘和非编码键盘两种。靠硬件识别按键者称编码键盘；通过软件识别者为非编码键盘。单片机系统中多用非编码键盘。下面我们只讨论这种键盘。

5.2.1　独立按键接口

在单片机系统中，若所需按键数量较少，则多采用独立式键盘。这种键盘结构简单，每只按键接单片机的一条 I/O 线，通过对输入线的查询，即可识别出各按键的状态来。

[例 5.5]　设计一个 AT90S8515 单片机所用的含 8 只按键的独立式键盘。

如图 5.9 所示，8 只按键分别接到 AT90S8515 A 口的 8 条 I/O 线上。无键按下时，由于上拉电阻的作用，PA0～PA7 线上均输入高电平。当某键按下时，与其相连的 I/O 线将得到低电平输入，其他未按之键的输入线上仍维持高电平输入。

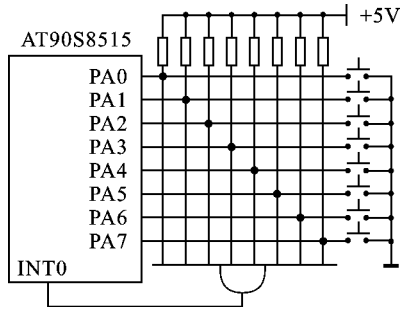


图 5.9　独立式按键

从图 5.9 中可以看到，A 口的 8 条 I/O 线逻辑相与后，所得信号加到 AT90S8515 的 INT0 引脚上。这样，每当有键按下时，ITN0 引脚上便有一个 1 至 0 的负跳变产生，从而引发中断。在中断服务程序中，首先延时 20 ms 左右，待按键抖动过后，再对各键进行查询。找到所按之键时，MCU 便转去执行相应的处理程序。

```
        .include "8515def.inc"
          rjmp    RESET
          rjmp    EXTINT
        .org      $010
RESET:  ……
        CLR       R16                    ; R16 清 0
        OUT       DDRA,R16               ; A 口作输入
        OUT       PORTA,R16              ; 无须接内部上拉电阻
        CBI       DDRD,2                 ; PD2 作输入(INT0)
        LDI       R16,$02
        OUT       MCUCR,R16              ; INT0 下降沿触发
```

```
        LDI     R17,$40          ;清除由于工作制式设定
        OUT     GIFR,R17         ;所产生的中断标志 INTF0
        OUT     GIMSK,R17        ;开 INT0 中断
        SEI                      ;开总体中断
        ……
EXTINT: RCALL   D20ms            ;延时去抖动
        SBIS    PINA,0           ;0 号键按下?否,跨越
        RJMP    PROC0            ;是,转 0 号键处理程序
        SBIS    PINA,1           ;1 号键按下?否,查 2 号键
        RJMP    PROC1            ;是,转 1 号键处理程序
        SBIS    PINA,2           ;2 号键按下?否,查 3 号键
        RJMP    PROC2            ;是,转 2 号键处理程序
        SBIC    PINA,3           ;3 号键按下?否,查 4 号键
        RJMP    PROC3            ;是,转 3 号键处理程序
        SBIS    PINA,4           ;4 号键按下?否,查 5 号键
        RJMP    PROC4            ;是,转 4 号键处理程序
        SBIS    PINA,5           ;5 号键按下?否,查 6 号键
        RJMP    PROC5            ;是,转 5 号键处理程序
        SBIS    PINA,6           ;6 号键按下?否,查 7 号键
        RJMP    PROC6            ;是,转 6 号键处理程序
        SBIS    PINA,7           ;7 号键按下?否,跨越
        RJMP    PROC7            ;是,转 7 号处理程序
        RETI                     ;中断结束
PROC0:  ……                       ;0 号键处理,略
AGAIN:  SBIS    PINA,0           ;0 号键已放开?是,跨越
        RJMP    AGAIN            ;否,再查 0 号键状态
        RCALL   D20ms            ;延时去抖动
        OUT     GIFR,R17         ;清抖动所引起的 INTF0
        RETI                     ;中断结束
PROC1:  ……                       ;其他键处理程序
D20ms:  ……                       ;延时子程序,略
```

5.2.2 矩阵式键盘接口

在单片机系统中需要安排较多的按键时,通常把键排列成矩阵形式,这样可以合理地利用硬件资源。例如,可把 16 只按键排列成 4×4 矩阵形式,用一个 8 位 I/O 口控制;把 64 只按键安排成 8×8 矩阵形式,以两个 8 位 I/O 口来控制。

[例 5.6] AT90S1200 外接一 4×4 非编码键盘。通常 MCU 处于休眠方式下,但当有键按下时,AVR 将被唤醒。此后 AVR 便对各键进行扫描,并令 LED 闪烁,用闪烁次数表示所按键的键号。AT90S1200 所用时钟为 4 MHz。

图 5.10 所示为本例题的外部硬件接线。系统复位之后,PB0～PB3 等四条行线被设定为带内部上拉电阻的输入方式;PB4～PB7 四条列线为输出方式,且输出低电平;PD2 被设定成

低电平触发的外部中断输入；PD0 和 PD1 为输出，且输出高电平，使红、绿两 LED 处于暗态。

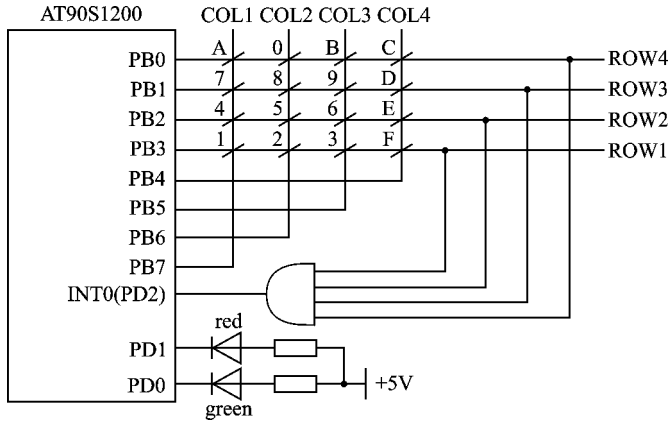


图 5.10　矩阵式键盘

通常 MCU 处于掉电方式下。若无键按下，则四条行线由于内部上拉电阻的作用而均呈高电平状态，故 INT0 处无中断请求信号。但若有键按下，则与该键相连的行列线接通，于是列线的低电平经该键所在之行线传递至与门，后者输出 0，产生 INT0 中断请求。外部中断唤醒处于掉电方式下的 MCU，此后 MCU 便对键盘进行扫描，找到所按之键，并令绿色 LED 闪烁，闪烁次数等于该键的键号。如果所按下的是 0 号键，则红色 LED 闪烁 10 次。

```
.include "1200def.inc"
.equ    ROW1=3                  ; 键的行线
.equ    ROW2=2
.equ    ROW3=1
.equ    ROW4=0
.equ    COL1=7                  ; 键的列线
.equ    COL2=6
.equ    COL3=5
.equ    COL4=4
.equ    green=0                 ; PD0 接绿色 LED
.equ    red=1                   ; PD1 接红色 LED
.equ    int=2                   ; PD2 为 INT0 输入
.def    tmp=r16                 ; 通用暂存器
.def    key=r17                 ; 寻键值时的 EEPROM 指针
.def    status=r21              ; SREG 的暂存寄存器
.def    fine=r18                ; 延时子程序用
.def    medium=r19              ; 循环计数器
.def    coarse=r20
.eseg                           ; EEPROM 段
.org    0
.db     1,2,3,15,4,5,6,14,7,8,9,13,10,0,11,12
.cseg                           ; 代码段
```

```
        .org    0
          rjmp  reset                    ; 转向 reset 段
          rjmp  scan                     ; 转向 INT0 中断服务程序
        reti                             ; 定时器中断(未用)
        reti                             ; 模拟比较器中断(未用)
reset:  ldi     tmp, 0xfb                ; D 口作输出,
        out     ddrd, tmp                ; 只有 PD2 作 INT0 输入
        ldi     tmp, 0x30                ; 允许进入掉电方式
        out     mcucr, tmp               ; INT0 低电平触发
        ldi     tmp, 0x40
        out     gimsk, tmp               ; 开 INT0 中断
        sbi     acsr, ACD                ; 关断模拟比较器
main:   cli                              ; 关总体中断
        ldi     tmp, 0xf0                ; 行线 PB0～PB3 作输入
        out     ddrb, tmp                ; 列线 PB4～PB7 作输出
        ldi     tmp, 0x0f                ; 列线输出 0
        out     portb, tmp               ; 行线带内部上拉电阻
        ldi     tmp, 0x07                ; PD2 带内部上拉电阻
        out     portd, tmp               ; 红绿 LED 熄灭
        sei                              ; 开总体中断
        sleep                            ; MCU 进入掉电方式
        rcall   flash                    ; LED 闪烁
        ldi     tmp,0x40                 ; 准备再次开放
        out     gimsk, tmp               ; INT0 外部中断
          rjmp  main                     ; 键盘扫描后 MCU 再次休眠
```

以下是中断服务程序。这里首先对四条行线进行扫描,确定所按之键位于哪条行线上。有键按下的行线输入低电平,无键按下之行线为高电平。找到行线后,在 key 寄存器中给出该行最左面一个键之键号在 EEPROM 中的地址,即行指针。例如 ROW1～ROW4 左起第一键分别为 1 号、4 号、7 号和 10 号(A)键,它们在 EEPROM 中的地址是 0、4、8 和 12。

第二步,把行线 ROW1～ROW4 改设成输出,列线 COL1～COL4 改作输入。同理可确定按键所在列的列指针。COL1～COL4 的列指针分别为 0～3。读者不难发现,列指针实际上是某键键号在 EEPROM 中相对同一行左起第一键的偏移量。这样,行指针与列指针之和便是所按键键号在 EEPROM 中的地址。例如,若行列指针和为 3,则从 eseg 段中可以看到,地址为 $03 的内容等于 15,即这是 F 键。

```
scan:   in      status,sreg              ; 保留 SREG 寄存器内容
        sbis    pinb, ROW1               ; ROW1 无键按下,查下一行
        ldi     key, 0                   ; 有,行指针为 0
        sbis    pinb, ROW2               ;ROW2 无键按下,查下一行
        ldi     key, 4                   ; 有,行指针为 4
        sbis    pinb, ROW3               ; ROW3 无键按下,查下一行
        ldi     key, 8                   ; 有,行指针为 8
```

```
        sbis    pinb, ROW4      ; ROW4 无键按下,查列线
        ldi     key, 12         ; 有,行指针为 12
        ldi     tmp, 0x0f       ; 行线 PB0～PB3 改为输出
        out     ddrb, tmp       ; 列线 PB4～PB7 改作输入
        ldi     tmp, 0xf0       ; 行线输出低电平
        out     portb, tmp      ; 列线输入带内部上拉电阻
        rcall   settle          ; 给端口一改制时间
        sbis    pinb, COL1      ; COL1 无键按下,查下一列
        ldi     tmp, 0          ; 有,列指针为 0
        sbis    pinb, COL2      ; COL2 无键按下,查下一列
        ldi     tmp, 1          ; 有,列指针为 1
        sbis    pinb, COL3      ; COL3 无键按下,查下一列
        ldi     tmp, 2          ; 有,列指针为 2
        sbis    pinb, COL4      ; COL4 无键按下,跨越
        ldi     tmp, 3          ; 有,列指针为 3
        add     key, tmp        ; 行列指针合并
        ldi     tmp, 0xf0       ; 行线又作输入
        out     ddrb, tmp       ; 列线又作输入
        ldi     tmp, 0x0f       ; 列线输出低电平
        out     portb, tmp      ; 行线输入带内部上拉电阻
        out     sreg, status    ; 恢复 SREG 寄存器内容
        clr     tmp             ; 暂关 INT0 中断
        out     gimsk, tmp      ; 因按键所引起的 ITN0 低电平
        reti                    ; 可能要维持相当长一段时间
flash:  out     eear, key       ; 键号在 EEPROM 中的地址
        sbi     eecr, eere      ; EEPROM 读选通
        in      tmp, eedr       ; 读所按键号
        tst     tmp             ; 是 0 号键吗？
        breq    zero            ; 是,转去亮红灯
grflash: cbi    portd, green    ; 非 0 键,亮绿灯
        rcall   delay           ; 延时 0.5 s
        sbi     portd,green     ; 熄绿灯
        rcall   delay           ; 延时 0.5 s
        dec     tmp             ; 键号减 1
        brne    grflash         ; 闪烁次数不足,循环
        ret                     ; 闪烁次数已完成
zero:   ldi     tmp, 10         ; 红灯将闪烁 10 次
        cbi     portd, red      ; 红灯亮
        rcall   delay           ; 延时 0.5 s
        sbi     portd, red      ; 红灯灭
        rcall   delay           ; 延时 0.5 s
        dec     tmp             ; 闪烁次数减 1
        brne    zero+1          ; 闪烁次数不足,循环
```

```
        ret                                    ; 闪烁次数已完成
delay:  ldi     coarse, 10                     ; 延时子程序,见例 4.9
        clr     medium
        clr     fine
        dec     fine
        brne    delay+3
        dec     medium
        brne    delay+2
        dec     coarse
        brne    delay+1
        ret
settle: ldi     tmp, 255
        dec     tmp
        brne    settle+1
        ret
```

5.2.3 键盘和显示器复合接口

[例 5.7] 用晶振频率为 4.096 MHz 的 AT90S1200 设计一个 24 小时工业计时器,或家政实时时钟 RTC。该计时器可按预先设定的时间对两路负荷进行开关控制。当通过键盘输入负荷的开关时间时,4 位 LED 将显示所设定的时和分值。

图 5.11 所示为该计时器的硬件结构。这里以 B 口高 4 位作键盘行线 ROW4～ROW1;以低 4 位作列线 COL1～COL4。B 口多路切换,又传递 4 位共阳极 LED 的各段显示信号 a～dp。D 口的低 4 位各通过一反相器控制一 LED 的阳极;其高 3 位控制两路负荷 LD1 和 LD2 以及按键的压电发声器 PZ。

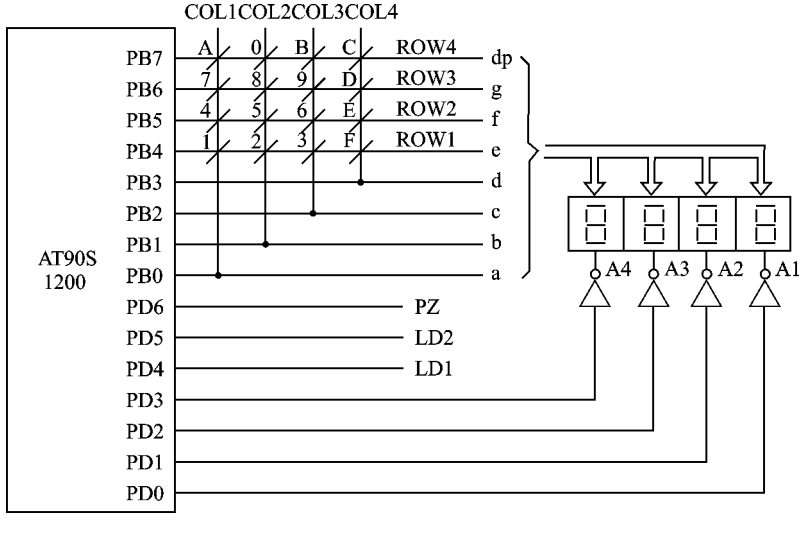


图 5.11 AVR 与 4×4 键盘和 4 位 LED 显示器接口

数字键 0～9 用来输入实时时钟和两负荷应接通与分断的时间值,A～F 六个键作命令

键。表 5.6 所列为这些键的键名、键值以及它们的功能。

表 5.6　例 5.7 按键名称、键值及功能表

键　名	键　值	功　能	键　名	键　值	功　能	键　名	键　值	功　能
1	0	数字键 1	6	6	数字键 6	A	12	SET,设定时间
2	1	数字键 2	E	7	LD1OFF,1 号负荷断	0	13	数字键 0
3	2	数字键 3	7	8	数字键 7	B	14	CLEAR,清除
F	3	LD1ON,1 号负荷通	8	9	数字键 8	C	15	LD2OFF,2 号负荷断
4	4	数字键 4	9	10	数字键 9	无	16	
5	5	数字键 5	D	11	LD2ON,2 号负荷通			

```
.def    loset=r1                ;分设定值
.def    hiset=r2                ;时设定值
.def    ld1minon=r3             ;负荷 1 接通时间(分)
.def    ld1hron=r4              ;负荷 1 接通时间(时)
.def    ld1minoff=r5            ;负荷 1 分断时间(分)
.def    ld1hroff=r6             ;负荷 1 分断时间(时)
.def    ld2minon=r7             ;负荷 2 接通时间(分)
.def    ld2hron=r8              ;负荷 2 接通时间(时)
.def    ld2minoff=r9            ;负荷 2 分断时间(分)
.def    ld2hroff=r10            ;负荷 2 分断时间(时)
.def    temp=r16                ;暂存器
.def    second=r17              ;RTC 秒计数
.def    minute=r18              ;RTC 分计数
.def    hour=r19                ;RTC 时计数
.def    mask=r20                ;数位闪烁屏蔽字
.def    blink=r21               ;点闪烁计时器
.def    bounce=r22              ;按键去抖动计时器
.def    flash=r23               ;闪烁延时计时器
.def    lobyte=r24              ;分值显示缓冲寄存器
.def    hibyte=r25              ;时值显示缓冲寄存器
.def    key=r26                 ;扫描所得键号
.def    tock=r27                ;5 ms 脉冲
.def    flags=r28               ;命令键标志字节
.equ    ms5=7                   ;显示时间 5 ms 标志
.equ    keyok=6                 ;键去抖动后置 1,必须再次清 0
.equ    ld2off=5                ;负荷 2 分断,C 键按下标记
.equ    ld2on=4                 ;负荷 2 接通,D 键按下标记
.equ    ld1off=3                ;负荷 1 分断,E 键按下标记
.equ    ld1on=2                 ;负荷 1 接通,F 键按下标记
.equ    ld2=1                   ;置 1 时告知家政程序,
.equ    ld1=0                   ;去检查两负荷的开关时间
.equ    col1=0                  ;LED a 段/键盘 col1
.equ    col2=1                  ;LED b 段/键盘 col2
```

```
        .equ    col3=2                  ; LED c 段/键盘 col3
        .equ    col4=3                  ; LED d 段/键盘 col4
        .equ    row1=4                  ; LED e 段/键盘 row1
        .equ    row2=5                  ; LED f 段/键盘 row2
        .equ    row3=6                  ; LED g 段/键盘 row3
        .equ    row4=7                  ; LED dp 段/键盘 row4
        .equ    A1=0                    ; 共阳极驱动信号,
        .equ    A2=1                    ; 均为低电平有效
        .equ    A3=2
        .equ    A4=3
        .equ    LOAD1=4                 ; 输出至负荷 1(低电平有效)
        .equ    LOAD2=5                 ; 输出至负荷 2(低电平有效)
        .equ    PZ=6                    ; 输出至压电发声器(低电平有效)
        .include"1200def.inc"
        .eseg                           ; EEPROM 段
        .org    0
table1: .db $c0, $f9, $a4, $b0, $99, $92, $82, $f8, $80
        ; 数字  0   1    2    3    4    5    6    7    8
        .db $90, $86, $8e, $a3, $ab, $ff, $ff
        ; 数字  9   E    F    O    n   空   空
table2: .db 1, 2, 3, 15, 4, 5, 6, 14, 7, 8, 9, 13, 10, 0, 11, 12
        .cseg                           ; 程序代码段
        .org    0
        rjmp    reset                   ; 转复位处理
        reti                            ; 外部中断处理(未用)
        rjmp    tick                    ; 转定时器溢出(5 ms)处理
        reti                            ; 模拟比较器中断(未用)
reset:  ser     temp                    ; temp←$ff
        out     DDRB, temp              ; B 口作输出
        out     DDRD, temp              ; D 口作输出
        out     PORTB, temp             ; 键盘行列均为高电平,关 LED 显示
        out     PORTD, temp             ; 关 LED 显示及负荷
        ldi     temp, $04               ; 对 4.096 MHz 晶振 256 分频
        out     TCCR0, temp             ; 定时器时钟周期为:
        ldi     temp, 176               ; 0.244 1 μs×256=62.5 μs
        out     TCNT0, temp             ; 定时间隔 62.5 μ×(256-176)=5 ms
        ldi     temp, $02
        out     TIMSK, temp             ; 开定时器中断
        clr     flags                   ; 清控制标志
        clr     tock                    ; 清 5 ms 脉冲
        clr     bounce                  ; 清按键去抖动计时器
        clr     flash                   ; 清闪烁延时计时器
        clr     blink                   ; 清点闪烁计时器
```

```
        sei                                    ; 开总体中断
```

以上的复位初始化程序段做了三件事。第一，将 B 口和 D 口都设置成输出方式，且它们的各条线均输出高电平。这样便关断了 4 位 LED 显示器及两路负荷。第二，将定时器设定为定时方式，并开放其中断，使之每过 5 ms 请求一次中断。第三，对某些寄存器做了清 0 处理。

下面是时间设定程序段。按下 SET 键(即 A 键，键值为 12，见表 5.6)之前，在 4 位 LED 显示器上闪烁显示“EEEE”字符。

```
timeset:  ldi     hibyte, $aa          ; 在 4 位 LED 上，
          ldi     lobyte, $aa          ; 显示“EEEE”
          ser     mask                 ; 并令其闪烁
notyet:   rcall   display              ; 调用显示子程序
          brtc    notyet               ; 重复，直至 SET 键按下
          rcall   setrtc               ; 设定时钟值
          mov     hour, hiset          ; 时值
          mov     minute, loset        ; 分值
          clt                          ; 清 T 标志
```

在上述时间设定程序段中，SET 键按下之前，反复调用显示子程序 display。该子程序的主要任务在于穿插进行显示与键盘扫描。从 EEPROM 段的 table1 中可以看到，地址为 $0a 的字节单元之内容等于 $86，即字符 E 的字形码。就是说，在 SET 键按下之前，display 子程序在每位 LED 上闪烁显示字符 E。

在键盘扫描过程中，若发现 SET 键按下，则 CPU 将去调用时钟值设定子程序 setrtc。时、分值设定后，便执行下面一段家政服务子程序。

```
do:       clr     mask                 ; 显示不再闪烁
          cpi     blink, 100           ; 半秒钟已到？
          brne    nohalf               ; 否，跳转
          clr     blink                ; 是，清 dp 闪烁计时器
          com     flash                ; 闪烁延时计时值取反
nohalf:   cpi     second, 60           ; 60 秒已到？
          brne    nochange             ; 否，跳转
          clr     second               ; 是，秒值清 0
          inc     minute               ; 分值加 1
          mov     temp, minute         ; 准备处理分值
          andi    temp, $0f            ; 分值十位数清 0
          cpi     temp, 10             ; 分值个位数＝$0a？
          brne    nochange             ; 否，跳转
          andi    minute, $f0          ; 是，分值个位数清 0
          ldi     temp, $10
          add     minute, temp         ; 分值十位数加 1
          cpi     minute, $60          ; 是 60 分吗？
          brne    nochange             ; 否，跳转
          clr     minute               ; 是，分值清 0
          inc     hour                 ; 时值加 1
```

```
        mov     temp, hour
        andi    temp, $0f
        cpi     temp, 10                ; 时值个位数= $0a ?
        brne    nochange                ; 否,跳转
        andi    hour, $f0               ; 是,个位数清 0
        ldi     temp, $10
        add     hour, temp              ; 时值十位数加 1
nochange:cpi    hour, $24               ; 是 24 时吗 ?
        brne    sameday                 ; 否,跳转
        clr     hour                    ; 是,时值清 0
        clr     minute                  ; 分值清 0
        clr     second                  ; 秒值清 0
sameday: mov    lobyte, minute          ; 分值送至显示缓冲单元
        mov     hibyte, hour            ; 时值送至显示缓冲单元
        rcall   display                 ; 调用显示子程序
        brtc    case1                   ; 无 SET 键按下,跳转
        rcall   setrtc                  ; 有,调用 rtc 设定子程序
        mov     hour, hiset             ; 设定时值就位
        mov     minute, loset           ; 设定分值就位
        clt                             ; 清 T 标志
case1:  sbrc    flags, ld1              ; ld1 标志为 0,转 case2
        rjmp    chkload1                ; ld1=1, 跳转去查负荷 1
case2:  sbrc    flags, ld2              ; ld2=0, 转 case3
        rjmp    chkload2                ; ld2=1, 去查负荷 2
case3:  sbrc    flags, ld1on            ; 负荷 1 接通时间重新设定 ?
        rjmp    setld1on                ; 是,重设接通时间
case4:  sbrc    flags, ld1off           ; 负荷 1 分断时间重新设定 ?
        rjmp    setld1off               ; 是,重设分断时间
case5:  sbrc    flags, ld2on            ; 负荷 2 接通时间重设 ?
        rjmp    setld2on                ; 是,重设负荷 2 接通时间
case6:  sbrc    flags, ld2off           ; 负荷 2 分断时间重设 ?
        rjmp    setld2off               ; 是,重设分断时间
case7:  rjmp    do                      ; 重复执行本段程序
```

以上这段家务活动程序不断重复执行。这里,首先看是否要对分和时计数值进行更新;其次再进行显示和键盘扫描;最后,若无 SET 键按下,或有 SET 键按下并接受了新的时间设定值后,程序便去检查两路负荷的情况,控制它们的开关操作。这样三步轮流执行,周而复始。以下是负荷处理程序的续篇。

```
chkload1:cp     hour, ld1hroff          ; 负荷 1 关断时间(时)已到 ?
        brne    onload1                 ; 否,跳转
        cp      minute, ld1minoff       ; 负荷 1 关断时间(分)已到 ?
        brne    onload1                 ; 否,跳转
        sbi     PORTD, LOAD1            ; 是,关断负荷 1
```

```
onload1: cp      hour, ld1hron          ; 负荷 1 接通时值到？
         brne    case2                  ; 否,跳转
         cp      minute, ld1minon       ; 负荷 1 接通分值到？
         brne    case2                  ; 否,跳转
         cbi     PORTD, LOAD1           ; 是,接通负荷 1
         rjmp    case2
chkload2:cp      hour, ld2hroff         ; 负荷 2 关断时值到？
         brne    onload2                ; 否,跳转
         cp      minute, ld2minoff      ; 负荷 2 关断分值到？
         brne    onload2                ; 否,跳转
         sbi     PORTD, LOAD2           ; 是,关断负荷 2
onload2: cp      hour, ld2hron          ; 负荷 2 接通时值到？
         brne    case3                  ; 否,跳转
         cp      minute, ld2minon       ; 负荷 2 接通分值到？
         brne    case3                  ; 否,跳转
         cbi     PORTD, LOAD2           ; 是,接通负荷 2
         rjmp    case3
setld1on:sbr     flags, $01             ; 置负荷 1 有效标志 ld1
         rcall   setrtc                 ; 调用 rtc 时间设定子程序
         mov     ld1hron, hiset         ; 负荷 1 接通时间设定值(时)
         mov     ld1minon, loset        ; 负荷 1 接通时间设定值(分)
         cbr     flags, $04             ; 清 ld1on 标志
         rjmp    case4
setld1off:rcall  setrtc                 ; 调用 rtc 时间设定子程序
         mov     ld1hroff, hiset        ; 负荷 1 分断时间设定值(时)
         mov     ld1minoff, loset       ; 负荷 1 分断时间设定值(分)
         cbr     flags, $08             ; 清 ld1off 标志
         rjmp    case5
setld2on:sbr     flags, $02             ; 置负荷 2 有效标志 ld2
         rcall   setrtc                 ; 调用 rtc 时间设定子程序
         mov     ld2hron, hiset         ; 负荷 2 接通时间设定值(时)
         mov     ld2minon, loset        ; 负荷 2 接通时间设定值(分)
         cbr     flags, $10             ; 清 ld2on 标志
         rjmp    case6
setld2off:rcall  setrtc                 ; 调用 rtc 时间设定子程序
         mov     ld2hroff, hiset        ; 负荷 2 分断时间设定值(时)
         mov     ld2minoff, loset       ; 负荷 2 分断时间设定值(分)
         cbr     flags, $20             ; 清 ld2off 标志
         rjmp    case7
```

case1 和 case2 处的两条指令，根据 ld1 和 ld2 两标志的情况决定下一步的行动。例如，若 ld1 为 1，则要跳转到 chkload1 处去对负荷 1 进行检查，看预定的接通或分断时间是否已到，若是，则执行相应操作。

case3～case6 处的四条指令检查此前是否有 C、D、E 和 F 键按下，即有无两负荷开关时间重新设定的要求。若有，则分别转至 setld1on 等四小段程序去把新设定值存放起来。

以下是显示及键盘扫描子程序。这里前面一段进行键盘扫描；中间一段为按键后的反馈信号——压电音响处理；最后为数码显示。

```
display:  cbr     flags, $40      ; 按键去抖动标志清 0
          ldi     key, $10        ; 预置无键按下值(表 5.6)
          ser     temp            ; B 口输出高电平
          out     PORTB, temp     ; 暂关显示
          in      temp, PORTD
          ori     temp, $0f       ; 保持负荷状态不变
          out     PORTD, temp
          ldi     temp, $0f       ; COL1～COL4 为输出
          out     DDRB, temp      ; ROW1～ROW4 为输入
          ldi     temp, $f0       ; COL1～COL4 输出低电平
          out     PORTB, temp     ; ROW1～ROW4 带内部上拉电阻
          ldi     temp, 20        ; 给以短暂延迟时间，
tagain1:  dec     temp            ; 让端口工作方式稳定下来
          brne    tagain1         ; 以下开始键盘扫描
          sbis    PINB, ROW1      ; ROW1 上无键按下，查 ROW2
          ldi     key, 0          ; 有，置 ROW1 最低键值
          sbis    PINB, ROW2      ; ROW2 上无键按下，查 ROW3
          ldi     key, 4          ; 有，置 ROW2 最低键值
          sbis    PINB, ROW3      ; ROW3 上无键按下，查 ROW4
          ldi     key, 8          ; 有，置 ROW3 最低键值
          sbis    PINB, ROW4      ; ROW4 上无键按下，转查列线
          ldi     key, 12         ; 有，置 ROW4 最低键值
          ldi     temp, $f0       ; COL1～COL4 作输入
          out     DDRB, temp      ; ROW1～ROW4 作输出
          ldi     temp, $0f       ; ROW1～ROW4 输出低电平
          out     PORTB, temp     ; 接通 COL1～COL4 内部上拉电阻
          ldi     temp, 20        ; 给以短暂延迟时间
tagain2:  dec     temp            ; 使端口工作方式，
          brne    tagain2         ; 稳定下来
          clr     temp            ; 开始扫描列线
          sbis    PINB, COL1      ; COL1 上无键按下，查下一列
          ldi     temp, 0         ; 有，列偏移量为 0
          sbis    PINB, COL2      ; COL2 上无键按下，查下一列
          ldi     temp, 1         ; 有，列偏移量为 1
          sbis    PINB, COL3      ; COL3 上无键按下，查下一列
          ldi     temp, 2         ; 有，列偏移量为 2
          sbis    PINB, COL4      ; COL4 上无键按下，转求键值
          ldi     temp, 3         ; 有，列偏移量为 3
```

```
        add     key, temp                ; 求键值
        cpi     key, $10                 ; 键值= $10
        breq    nokey                    ; 是,无键按下,转去显示
        ldi     temp, $10                ; 由 table1 转向 table2
        add     key, temp                ; 得键号在 EEPROM 中的地址
        out     EEAR, key                ; 向 EEPROM 发出地址
        sbi     EECR, ERE                ; EEPROM 读选通
        in      key, EEDR                ; 读出键号
        cpi     key, 10                  ; 是 A 键(SET)吗?
        brne    notset                   ; 否,查下一个键
        set                              ; 是,将 T 标志置 1
notset: cpi     key,11                   ; 是 B 键(CLEAR)吗?
        brne    notclear                 ; 否,查下一个键
        sbi     PORTD, LOAD1             ; 是,关断所有负荷
        sbi     PORTD, LOAD2
        cbr     flags, $03               ; LD1 和 LD2 两标志清 0
notclear: cpi   key, 15                  ; 是 F 键(LD1ON)吗?
        brne    notld1on                 ; 否,查下一个键
        sbr     flags, $04               ; 是,置 LD1ON 标志
notld1on: cpi   key, 14                  ; 是 E 键(LD1OFF)吗?
        brne    notld1off                ; 否,查下一个键
        sbr     flags, $08               ; 是,置 LD1OFF 标志
notld1off: cpi  key, 13                  ; 是 D 键(LD2ON)吗?
        brne    notld2on                 ; 否,查下一个键
        sbr     flags, $10               ; 是,置 LD2ON 标志
notld2on: cpi   key, 12                  ; 是 C 键(LD2OFF)吗?
        brne    tactile                  ; 否,转 PZ 处理
        sbr     flags, $20               ; 是,置 LD2OFF 标志
```

以上是键盘扫描段,其思路与例 5.6 无异。读者从这段程序中可以看到,它只处理了 A～F 等六个键按下时所要求的事项,而对数字的处理则由 setrtc 子程序去完成。不过,只要有键按下,就应有一个响声作为反馈信息。

下面是压电发声器处理程序段,每次有键按下时,都将发出 5 ms 长频率约 4 kHz 的音响。

```
tactile: cbr    flags, $80               ; 清 5 ms 标志 ms 5
        cbi     PORTD, PZ                ; 接通压电发声器 PZ
        ldi     temp, 170
t1again: dec    temp                     ; 125 μs 后,
        brne    t1again
        sbi     PORTD, PZ                ; 再关断 PZ
        ldi     temp, 170
t2again: dec    temp                     ; 延时 125 μs
        brne    t2again
```

```
        sbrs    flags, ms5          ; 已到 5 ms
        rjmp    tactile+1           ; 否,再次接通 PZ
notok:  cpi     bounce, 40          ; 是,200 ms 已到?
        brlo    nokey               ; 否,跳转
        sbr     flags, $40          ; 是,置去抖动标志
nokey:  ser     temp
        out     DDRB, temp          ; B 口作输出
        out     PORTB, temp         ; 输出高电平,暂清显示
```

键盘扫描时若未发现有键按下,则子程序将直接转到下面一段去进行数码显示。或者命令键按下已经做了处理,或者数字键按下尚未进行处理,两种情况都有压电音响做出反馈,此后该子程序也要过渡到下面的显示段去。

从显示段中可以看到,若需闪烁显示(即 mask 寄存器内容为 $ff),则每过 0.25 s,flash 寄存器内容取反一次。若其最低位为 1,则 4 位 LED 都有数码显示;为 0,则应显示内容被 mask 内容屏蔽,即不见数码发光。就是说,各 LED 亮暗周期各为 0.25 s,即每 0.5 s 闪烁一次。

```
        cbi     PORTD, A1           ; 接通最低显示位
        mov     temp, lobyte        ; 读分值
digit1: cbr     flags, $80          ; 清 5 ms 时段标志
        andi    temp, $0f           ; 屏蔽分值十位数
        out     EEAR, temp          ; 向 EEPROM 传送地址
        sbi     EECR, EERE          ; 发读选通信号
        in      temp, EEDR          ; 读字形码
        sbrs    flash, 0            ; 每 0.5 秒闪烁一次
        or      temp, mask          ; 若有必要则该位闪烁
        out     PORTB, temp         ; 该位 LED 显示 5 ms
led1:   sbrs    flags, ms5          ; 5 ms 时段已到?
        rjmp    led1                ; 否,等待
        sbi     PORTD, A1           ; 是,关断最低位 LED
        ser     temp                ; temp←$ff
        out     PORTB, temp         ; 暂清除显示
        cbi     PORTD, A2           ; 接通次低位 LED
        mov     temp, lobyte        ; 再读分值
        swap    temp                ; 分值个位与十位交换位置
digit2: cbr     flags, $80          ; 清 5 ms 时段标志
        andi    temp, $0f           ; 屏蔽分值个位数
        out     EEAR, temp          ; 向 EEPROM 传送地址
        sbi     EECR, EERE          ; 发读选通信号
        in      temp, EEDR          ; 读字形码
        sbrs    flash, 0            ; 每 0.5 秒闪烁一次
        or      temp, mask          ; 若有必要,则该位闪烁
        out     PORTB, temp         ; 次低位 LED 显示 5 ms
```

```
led2:   sbrs    flags, ms5          ; 5 ms 时段已到？
        rjmp    led2                ; 否,等待
        sbi     PORTD, A2           ; 是,关断次低位 LED
        ser     temp
        out     PORTB, temp         ; 暂清除显示
        cbi     PORTD,A3            ; 接通次高位 LED
        mov     temp, hibyte        ; 读时值
digit3: cbr     flags, $80          ; 清 5 ms 时段标志
        andi    temp, $0f           ; 屏蔽时值十位数
        out     EEAR, temp          ; 向 EEPROM 发送地址
        sbi     EECR, EERE          ; 发读选通信号
        in      temp, EEDR          ; 读时值个位数字形码
        sbrs    second, 0           ; 将使 dp 闪烁
        andi    temp, $7f           ; 将使 dp 发光
        sbrs    flash, 0            ; 要闪烁吗？
        or      temp, mask          ; 要,加屏蔽
        out     PORTB, temp         ; 次高位 LED 显示 5 ms
led3:   sbrs    flags, ms5          ; 5 ms 时段已到？
        rjmp    led3                ; 否,等待
        sbi     PORTD, A3           ; 是,关断次高位 LED
        ser     temp
        out     PORTB, temp         ; 暂清除显示
        cbi     PORTD, A4           ; 接通最高位 LED
        mov     temp, hibyte        ; 再读时值
        swap    temp                ; 其十位与个位交换位置
        andi    temp, $0f           ; 屏蔽时值个位数
        brne    digit4              ; 十位数≠0, 转显示
        ldi     temp, $ff           ; 十位数=0,不显示
digit4: cbr     flags, $80          ; 清 5 ms 时段标志
        andi    temp, $0f           ; 屏蔽高 4 位
        out     EEAR, temp          ; 向 EEPROM 发地址
        sbi     EECR, EERE          ; 发读选通信号
        in      temp, EEDR          ; 读时值十位数字形码
        sbrs    flash, 0            ; 要闪吗？
        or      temp, mask          ; 要,加屏蔽
        out     PORTB, temp         ; 最高位 LED 显示 5 ms
led4:   sbrs    flags, ms5          ; 5 ms 时段已到？
        rjmp    led4                ; 否,等待
        sbi     PORTD, A4           ; 是,关闭最高位 LED
        ser     temp
        out     PORTB, temp         ; 清除显示
        tst     mask                ; mask=0？
        breq    outlet              ; 是,子程序结束
```

```
        cpi     blink, 50          ; 否,已到 250 ms?
        brlo    outlet             ; 否,子程序结束
        clr     blink              ; 是,清除闪烁计时
        com     flash              ; 闪烁字节取反
outlet: ret
```

下面是实时时钟或负荷开关时间设定子程序,分和时设定值分别存于 loset 和 hiset 两寄存器中。

```
setrtc: ser     mask               ; 设定闪烁显示
        ldi     hibyte, $df        ; 时值十位数上显示"n"
        ser     lobyte             ; 其他位均不显示
hihrus: clr     bounce             ; 去抖动计时器清 0
bounce1: rcall  display            ; 键盘扫描及显示
        sbrs    flags, keyok       ; 键已稳定?
        rjmp    bounce1            ; 否,再去显示
        cbr     flags, $40         ; 是,清除键稳定标志
        cpi     key, $03           ; 时值十位数≥3?
        brsh    hihrus             ; 是,拒绝接受,再读键盘输入
hihrok: swap    key                ; 否,将其移至字节高 4 位
        mov     hiset, key         ; 并存入时值设定寄存器中
        ldi     hibyte, $0d        ; 在时值个位数上显示"n"
        add     hibyte, hiset      ; 时钟十位数与 n 并入一个字节
lohrus: clr     bounce             ; 去抖动计数器清 0
bounce2: rcall  display            ; 键盘扫描及显示
        sbrs    flags, keyok       ; 键已稳定?
        rjmp    bounce2            ; 否,再去显示
        cbr     flags, $40         ; 是,清除键稳定标志
        mov     temp, hibyte
        andi    temp, $f0          ; 时值个位数清 0
        add     temp, key          ; 得完整的两位时值
        cpi     temp, $24          ; 时值≥24?
        brsh    lohrus             ; 是,拒绝,再读键盘输入
        add     hiset, key         ; 否,时值十位数和个位数合成,
lohrok: mov     hibyte, hiset      ; 一个字节并送入显示缓冲单元
        ldi     lobyte, $df        ; 在分值高位数上显示"n"
himinus: clr    bounce             ; 去抖动计数器清 0
bounce3: rcall  display            ; 键盘扫描并显示
        sbrs    flags, keyok       ; 键已稳定?
        rjmp    bounce3            ; 否,再去显示
        cbr     flags, $40         ; 是,清键稳定标志
        cpi     key, 6             ; 分值高位≥6?
        brsh    himinus            ; 是,拒绝,再读键盘输入
lominok: swap   key                ; 否,分值高位数移至高 4 位
```

```
        mov     loset, key          ; 并存入分设定值单元
        ldi     lobyte, $0d         ; "n"字形码在 EEPROM 中的地址
        add     lobyte, loset       ; 在分值低位数上显示"n"
lominus: clr    bounce              ; 清去抖动计时器
bounce4: rcall  display             ; 键盘扫描及显示
        sbrs    flags, keyok        ; 键已稳定？
        rjmp    bounce4             ; 否,再去显示
        cbr     flags, $40          ; 是,清键稳定标志
        cpi     key, 10             ; 键号>9 吗？
        brsh    lominus             ; 是,再读键盘输入
        add     loset, key          ; 否,分值个位与十位数合成一个字节
        clr     mask                ; 显示不再闪烁
        ret                         ; 子程序结束
```

请读者注意,按下 SET 键后,setrtc 子程序所接受的是实时时钟的时间设定值;而按下 C、D、E 或 F 键后,它所接受的则是负荷的开关时间。

SET 键按下后,各位 LED 上闪烁"E"的现象停止。setrtc 子程序开始以"*n*"作光标等待用户输入时间值,每输入一位,*n* 向右移动一位,并和已输入的数值一起闪烁显示在相应的 LED 位上。

最后是每 5 ms 执行一次的定时器中断服务程序 tick。

```
tick:   in      R0, SREG            ; 保存状态寄存器的内容
        inc     tock                ; 中断次数计数值增 1
        inc     blink               ; 闪烁计时器值增 1
        inc     bounce              ; 抖动计时器值增 1
        sbr     flags, $80          ; 置 5 ms 标志以控制显示时间
        cpi     tock, 200           ; 1 秒钟已到？
        breq    onesec              ; 是,转秒值加 1 指令
        nop                         ; 平衡中断时间
        rjmp    nosecond            ; 否,准备退出中断服务
onesec: inc     second              ; 秒值加 1
        clr     tock                ; 中断次数计数重新从 0 开始
nosecond: ldi   R31, 176            ; 恢复定时器初值
        out     TCNT0, R31
        out     SREG, R0            ; 恢复状态寄存器原内容
        reti                        ; 中断结束
```

这段程序中 nop 指令的作用在于使整个中断服务程序的执行时间固定,不受分支的影响。具体地说,一秒钟已到,则由 breq 指令到 nosecond 要用 4 个时钟周期。而未到一秒钟,由 breq(条件满足时执行时间为 2 个时钟周期,而此时条件不满足则为 1 个周期)到 nosecond 也为 4 个时钟周期,若无 nop 指令,就是 3 个周期了。

最后应当指出,这个例题是从 Atmel 公司网上下载的,其主要思路很好,但有些细节却不甚合理。其中的印刷错误和多余指令我们都做了处理,但未做大的修改。例如两键同时按下

以及输入时值个位数＞9 时应拒绝接受等问题，程序均未能解决。不过我们还是可以从中学到不少东西的。

5.3 与 BCD 码拨盘接口

数字拨码盘是为单片机系统设定控制参数最合适的输入器件，它具有使用方便、接口简单等优点。BCD 码拨盘开关是数字拨码盘的一种，可用来输入一位十进制数。图 5.12 所示为其结构示意。其齿轮型圆盘可在 0～9 十个位置间拨动，每个位置都对应一位十进制数，且拨盘旁有相应数字指示。如图 5.12 所示，拨盘后面有 5 条引出线，其中 A 为输入控制线，标号为 8、4、2、1 的是数据线。齿轮拨盘的每个位置可使 0 条、1 条、2 条或 3 条数据线与输入控制线接通。例如，拨盘处于位置 7 时，A 与标号为 4、2 和 1 的三条数据线接通。若拨盘处于位置 4，则 A 只与数据线 4 接通。

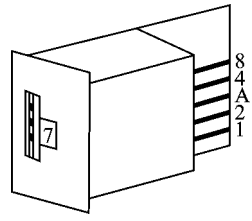


图 5.12 BCD 码拨盘开关

5.3.1 与端口直接接口

如果只需连接一、二个 BCD 码拨盘开关，那么用 AVR 一个端口即可。

[例 5.8] 某 AT90S8515 单片机系统有 10 项功能，其各相应程序段首地址为 FUNC0～FUNC9。要求为此系统设计一个功能选择开关，并写出功能的切换程序。功能选择只在复位后进行，正常运行期间不考虑功能切换。

如图 5.13 所示，把 BCD 码拨盘开关的输入控制线 A 接 5 V 电源，而其 4 条数据线分别连到 AT90S8515 各接一下拉电阻的 PA0～PA3 上。这样，当拨盘开关处于 0～9 十位置上时，8515 从 PA0～PA3 上恰好读到 0～9 等十个相应的数字。例如，拨盘开关被拨至位置 2 时，则图 5.13 中的 A 与 2 之间的开关接通，即 PA1 接 5 V，而 PA0、PA2 和 PA3 通过下拉电阻接地。所以 A 口低 4 位读到的输入值为 0b0010。

但是，对于 AVR 来说，可充分利用其端口内部上拉电阻的作用。在图 5.14 中，PA0～PA3 直接连到拨盘开关的 4 条数据线上，而开关的输入控制线 A 则接地。在这种接口情况下，应把 A 口设置成带上拉电阻的输入方式。若拨盘开关处于 0 位，则无数据线与 A 接通。这时由于内部上拉电阻的作用，PA0～PA3 输入值均为 1。但若开关处于其他位置，例如处于位 8，则数据线 8 通过 A 接地，于是 PA3～PA0 输入值为 0b0111，取反后等于 0b1000，即 BCD 数 8。

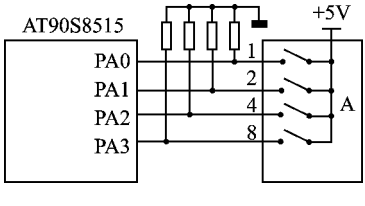


图 5.13 BCD 码拨盘与 AVR 接口

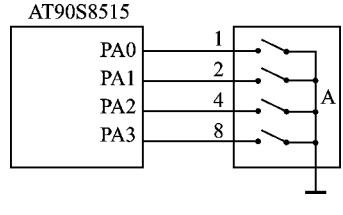


图 5.14 BCD 码拨盘与 AVR 接口之二

下面是功能选择程序。

```
        .include"8515def.inc"
        rjmp    RESET
        .org    $010                    ;跳转指令表
        rjmp    FUNC0
        rjmp    FUNC1
        ……
        rjmp    FUNC9
RESET:  ……
        clr     r16                     ;r16←0
        out     DDRA, r16               ;A口作输入
        ser     R16                     ;r16←$ff
        out     PORTA, r16              ;接通内部上拉电阻
        in      r16, PINA               ;读拨盘开关输入值
        com     r16                     ;取反
        clr     ZH                      ;设置间接跳转指针
        ldi     ZL, $10                 ;Z指向跳转指令表首地址
        add     ZL, r16                 ;以拨盘输入作偏移量
        ijmp                            ;跳转至rjmp指令表中,
FUNC0:  ……                              ;再由那里跳转到相应功能段
        ……
FUNC9:  ……
```

5.3.2　通过门电路接口

如果单片机系统内要接入多个BCD码拨盘开关为它输入参数,那么为节约I/O口资源,可通过门电路将端口与拨盘开关连接起来。

[例5.9]　为某实时时钟系统设计一个复位后提供时值和分值的子系统。

不难理解,欲输入时值和分值4位十进制数,得用四个BCD码拨盘开关。但究竟如何把它们组织起来,才能最大限度地节约单片机的I/O口资源呢?若按上例的连接方法,显然要用两个8位端口。但若增加四个4输入的与非门电路,如图5.15所示,就可以用一个8位端口来控制四个拨盘开关了。每个与非门的输入均与四个BCD码拨盘编号相同的数据线接通。例如,四个拨盘中编号为8的数据线都接到了PA3外与非门的输入端上,编号为4的四条数据线作为PA2外与非门的输入,如此等等。四个拨盘自左至右分别代表时值十位数、个位数和分值十位数和个位数。

按这种接口安排,显然PA3～PA0应设定成输入方式,而PA7～PA4应为输出方式。读时值十位数时,PA7输出低电平,PA6～PA4输出高电平;读时值个位数时,PA6输出0,而其他三引脚则输出1。假设时值个位数为9,则A2将与编号为8和1的两数据线接通。于是PA6的低电平输出就可加到PA3和PA0外的与非门之输入端上。而PA2和PA1外两个与非门的四个输入因上拉电阻的作用而均为1,故从PA3～PA0引脚上可读得0b1001,即9。

下面是复位后读时、分设定值的程序段,时值和分值分别存入hour和minute寄存器中。

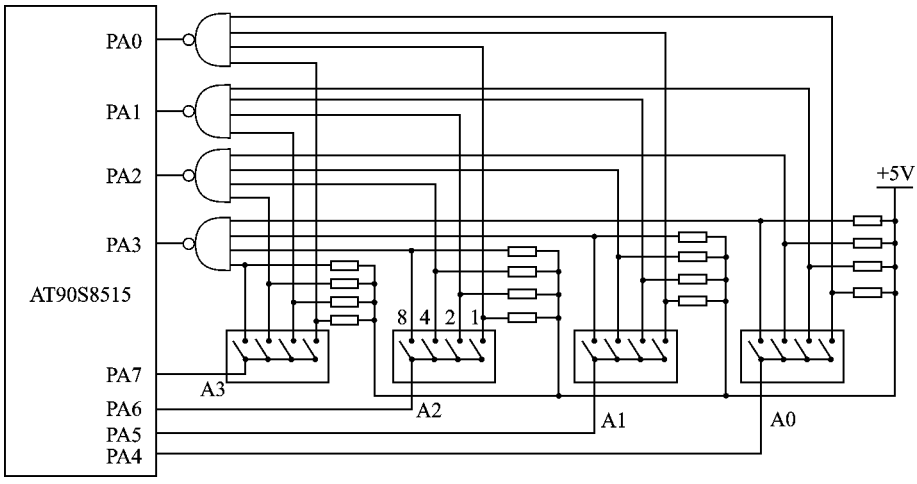


图 5.15　4 位 BCD 码拨盘开关接口电路

```
        .include"8515def.inc"
          rjmp    RESET
        .def      hour=r16
        .def      minute=r17
        .def      temp=r18
        .org      $010
RESET:  ……                          ;堆栈设置,略
        ldi       temp, $f0         ;PA7～PA4 作输出
        out       DDRA, temp        ;PA3～PA0 作输入
        ldi       temp, $70         ;PA7 输出 0
        out       PORTA, temp       ;PA6～PA4 输出 1
        in        hour, PINA        ;读时值十位数
        andi      hour, $0f         ;屏蔽高 4 位
        swap      hour              ;十位数就位
        ldi       temp, $b0         ;PA6 输出 0
        out       PORTA, temp       ;PA7、PA5、PA4 输出 1
        in        temp, PINA        ;读时值个位数
        andi      temp, $0f         ;屏蔽高 4 位
        add       hour, temp        ;得时值两位数
        ldi       temp, $d0         ;PA5 输出 0
        out       PORTA, temp       ;PA7、PA6 和 PA4 输出 1
        in        minute, PINA      ;读分值十位数
        andi      minute, $0f       ;屏蔽高 4 位
        swap      minute            ;分值十位数移至高 4 位
        ldi       temp, $e0         ;PA7～PA5 输出 1
        out       PORTA, temp       ;PA4 输出 0
        in        temp, PINA        ;读分值个位数
        andi      temp, $0f         ;屏蔽高 4 位
```

```
add     minute, temp                    ; 得分值两位数
……
```

5.4　与微型打印机接口

单片机应用系统多以微型打印机作其输出设备。本节将主要讨论微型打印机与 AVR 单片机的配合使用问题。

5.4.1　TPμP 微型打印机

TPμP 是一个由单片机控制的点阵式微型打印机系列。其接口电路与 centronics 标准打印机接口兼容，可与各种计算机及智能仪表连接。TPμP 有 C 系列卧式、T 系列台式和 A 系列面板式三个主要类型。

TPμP—40A 是一种每行可打印 40 个字符的面板式微型打印机，其接口信号线主要有下列几条。

DB7～DB0：数据线，单向，从计算机发出的命令和数据均通过数据线传送给打印机。

$\overline{\text{STB}}$：数据选通输入信号，低电平有效。DB7～DB0 上的数据只有在有效的$\overline{\text{STB}}$信号的作用下才能被锁入打印机内部锁存器中。

BUSY：打印机的输出信号，高电平有效。此信号若为高电平，则表示打印机正忙于内部事务，无暇接受外来数据。

$\overline{\text{ACK}}$：打印机的应答信号，输出，低电平有效。若为低电平，则表示打印机已处理完所接受的数据，以此告知计算机可以送新数据来。

图 5.16 所示为上述各信号间的时序关系。从这里可以看到，当打印机的 BUSY 信号为低电平时，计算机把数据送至数据线 DB7～DB0 上，并接着发出一个宽度大于 0.5 μs 的选通信号$\overline{\text{STB}}$，把数据线上的数据锁入打印机中。$\overline{\text{STB}}$的下降沿使 BUSY 信号升为高电平。这意味着打印机此刻正忙于处理刚刚收到的数据，这时计算机切勿送数据来。待打印机处理完此数据，或执行完打印任务时，BUSY 信号变为低电平，并触发内部单稳态触发器，使之发出一应答信号$\overline{\text{ACK}}$，以此告知计算机可以送新数据来了。

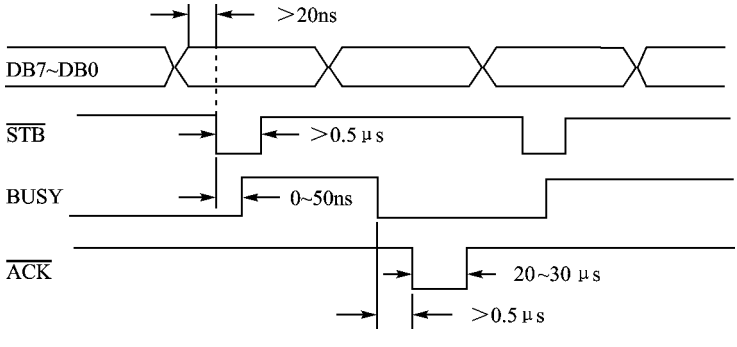


图 5.16　TPμP—40A 接口信号时序

5.4.2 AVR 与 TPμP—40A 接口

AVR 单片机系列齐全，片内资源丰富，端口驱动能力强。因此基本无须外扩存储器或其他诸如 8255 之类的接口芯片，与外部设备可以实现无粘合接口，即直接连接。

AVR 单片机可通过查询法了解微型打印机的状态信息，亦可用中断法与其沟通信息。这样就有两种大同小异的接口方法。

[例 5.10] 时钟频率为 4 MHz 的 AT90S1200 直接与 TPμP—40A 接口，以查询方式控制单片机与打印机间的数据传送。要求画出硬件接口图，并编写一段程序，令打印机在打印纸某行中间位置打印出 GOOD LUCK 字样。

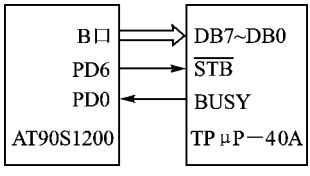


图 5.17 AVR 与微型打印机直接接口

这种接口十分简单，如图 5.17 所示。AT90S1200 用其 B 口作数据线，向打印机输出字符和命令代码；PD6 输出宽度大于 0.5 μs 的选通脉冲；而 PD0 则用来查询打印机的状态。

下面是 AT90S1200 对 TPμP—40A 的数据传送程序。这里以汇编指令 DB 在片内 EEPROM 中给了行将打印出的字符之 ASCII 码，以及 $07 和 $0D 两个命令代码。$07 是水平制表命令码，随后的 $0F 为其参数，两者的意思是让打印头向右空走 15 个字符的位置。$0D 为回车换行码，即打印结束符。

```
        .include "1200def.inc"
        .def    temp=r16                ; 暂存器
        .def    table=r17               ; 存放 EEPROM 表格首地址
        .eseg
        .org    0
        .db     $07,$0f,$47,$4f,$4f,$44
        .db     $20,$4c,$55,$43,$4b,$0d
        .cseg
        .org    0
        rjmp    RESET
        .org    $004
RESET:  ldi     temp, $f0               ; PD6～PD4 作输出
        out     DDRD, temp              ; PD3～PD0 为输入
        clr     table                   ; 表格寄存器清 0
        sbi     PORTD, 6                ; PD6 输出 1
        ldi     temp, $ff               ; temp←$ff
        out     DDRB, temp              ; B 口作输出
wait:   sbic    PIND, 0                 ; BUSY=0,跨越
        rjmp    wait                    ; 否,等待
        out     EEAR, table             ; 向 EEPROM 发送表格地址
        sbi     EECR, EERE              ; 向 EEPROM 发读选通信号
        in      temp, EEDR              ; 从表中读待打印字符
        out     PORTB, temp             ; 向打印机输出字符
```

```
cbi     PORTD, 6        ; 将STB降为低电平
nop                     ; 延时 0.25 μs
nop                     ; 延时 0.25 μs
sbi     PORTD, 6        ; 宽度为 0.75 μs 的STB结束
inc     table           ; 表格寄存器值增 1
cpi     table, $0c      ; 字符全部传送完毕？
brne    wait            ; 否，继续发送字符
……                      ; 是，字符发送结束
```

[例 5.11]　AT90S8515 直接与 TPμP—40A 接口，以中断方式控制单片机与打印机间的数据传送。

欲将查询方式改成中断方式，只须把前例中 BUSY 信号和 PD0 换成$\overline{\text{ACK}}$和 PD2(INT0)即可达到要求(见图 5.18)。不难理解，这两种情况下的数据传送本身并无任何差异，只是得知打印机何时可接受数据的方式不同而已。

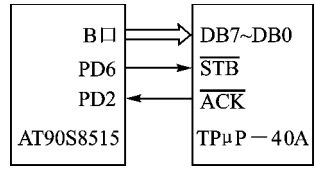


图 5.18　AVR 与微型打印机接口之二

在中断方式下，AT90S8515 用其 B 口向 TPμP—40A 发送数据，通过 PD6 产生宽度大于 0.5 μs 的$\overline{\text{STB}}$负脉冲选通信号。打印机收到数据后，以$\overline{\text{ACK}}$信号向 AVR 单片机发出中断请求，通知它送新数据来。

在这种方式下，第一个字符在主程序中发送，其余字符每次中断时在中断服务程序中发送一个。从对图 5.16 的时序分析中可知，INT0 应选下降沿触发方式。如果选用低电平触发方式，恐怕中断服务程序已经结束，而长度为 20～30 μs 的 $\overline{\text{ACK}}$低电平依然保持着。若将 INT0 设定为上升沿触发方式，则时间拖拉太长，下个字符不能及时传送。

5.5　片内 A/D 转换器

对于 AVR 单片机来说，与其舍近求远去讲它与 A/D 转换器的接口问题，不如将其片内 A/D 转换器向读者交代清楚。如果简易 A/D 转换即可满足要求，那么可用片内模拟比较器构成 A/D 转换器，如 4.6.2 和 4.6.3 两节所述。若需精度及分辨率较高的转换，则可选用片内含 A/D 转换器的 MCU，例如 AT90S4434/8535 等。

5.5.1　内部结构及运作原理

AT90S4434/8535 以其片内 10 位逐步逼近式模/数转换器 ADC 为其特色。如图 5.19 所示，该 ADC 的输入部分是一个 8 通道模拟多路开关，它接受来自 A 口每个引脚上的模拟量输入。此模拟量经多路开关输送到采样保持电路上，后者保证在转换期间模拟输入量是恒定的。

片内 ADC 由 AV_{CC}和 AGND 两个模拟电源引脚单独供电。模拟地 AGND 必须与数字地 GND 连接起来，而 AV_{CC}上的电压与 V_{CC}之差不得超过±0.3 V。

逐步逼近式 ADC 内含一个数/模转换器 DAC，它把 10 位逐步逼近寄存器 SAR 的内容转换成内部电压，去与模拟输入电压在采样/保持比较器上进行比较。转换逻辑根据比较器的输出通知 SAR，下次要增值还是减值。这样比较十次之后，转换告成，这时 SAR 的内容便是转

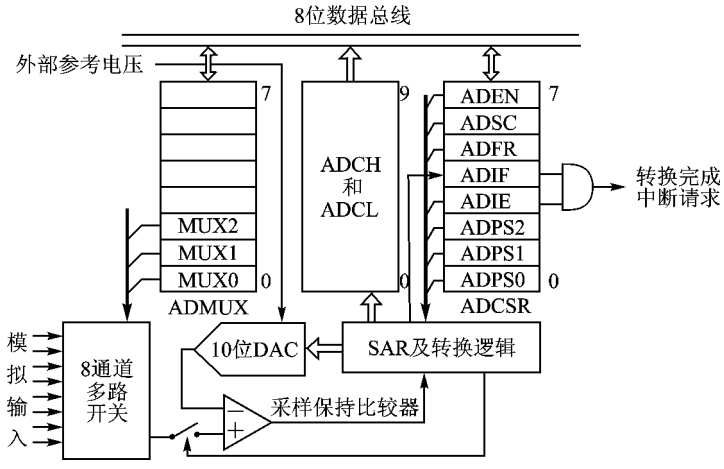


图 5.19 AVR 片内 A/D 转换器

换结果。

1. A/D 转换的启动

AT90S4434/8535 的片内 A/D 转换器有两种运作方式——单次转换和自由运行方式。在单次转换方式下，每次转换都得由用户启动。而在自由运行方式下，ADC 不断采样，不断更新 ADC 数据寄存器 ADCH 和 ADCL 的内容。

欲启动 A/D 转换，必须先向 ADC 控制和状态寄存器 ADCSR 中的开放位 ADEN 写入逻辑 1，即接通其时钟源。片内 ADC 的时钟分频器，如图 5.20 所示，对系统时钟 CK 进行分频以得到合适的 ADC 时钟频率。ADC 可接受频率范围为 50～200 kHz 的输入时钟信号，时钟频率过高会导致转换精度的降低。将 ADCSR 寄存器中的 ADEN 位置 1，就可使分频器按该寄存器最低 3 位 ADPS2～ADPS0 的设定对系统时钟进行分频（见表 5.7）。若把 ADEN 位清 0，则分频器将一直处于复位状态，此时无法启动 A/D 转换。

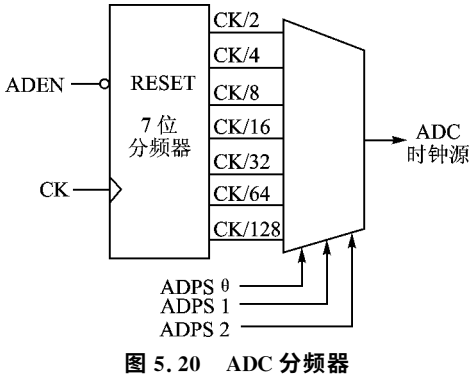


图 5.20 ADC 分频器

表 5.7 ADC 时钟分频选择

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

在 ADEN 为 1 的情况下，再把 ADCSR 寄存器中的启动转换位 ADSC 写成逻辑 1，则转换将在 ADC 时钟脉冲的下一个上升沿时启动。但真正的采样/保持操作要在转换启动后1.5 个 ADC 时钟周期处开始。在转换进行过程中，ADSC 位一直保持为 1；转换完成时，它被硬件自动清 0。

ADC 开放后第一次启动的 A/D 转换，将有一个使 ADC 初始化的哑转换作先导。不过，

对于用户来说，惟一的差别就是这次转换要比通常转换多用 12 个 ADC 时钟周期。

2. A/D 转换过程

从原理上讲，采样之后，转换逻辑首先把逐步逼近寄存器 SAR 的最高位置成 1，其他位均清成 0。此值为 $200，即全量程的一半。内部 10 位 DAC 于是将此值转换成相应的模拟电压，去与来自多路开关的输入电压进行比较。若外部输入电压高于或等于内部 DAC 转换出的电压值，则 SAR 中最高位的"1"保留，否则它将被清 0。

转换逻辑下次把 SAR 的次高位置成 1，得 $300 或 $100，此值被转换成模拟电压后，将再次与外来模拟输入电压进行比较。若后者高于或等于前者，则相应位的"1"值保留下来，否则即被清掉。此过程一直继续到所有 10 位均被测试完为止。此时 SAR 寄存器值便是最终转换结果。

从转换时间上看，表 5.8 告诉我们，单次和自由运行两种方式第一次启动后，均需 14 个 ADC 周期采样模拟输入信号。其实，这段时间，如图 5.21 所示，含 13 周期的哑转换及真正转换过程中的第一个周期。由图 5.22 和图 5.23 可以看到，两种方式下的一次转换时间均为 13 个 ADC 周期，其中前两个周期为采样时间。在单次转换方式下，一次转换完成后，如图 5.21 所示，尚需过一个时钟周期才能开始新的一轮转换。如果在这个期间 ADSC 被置成高电平，那么新的一轮转换将在下个周期立即开始。

表 5.8　ADC 转换时间

条　件	采样周期数	结果就绪/周期数	总转换时间/周期	总转换时间/μs
第一次转换，自由运行	14	25	25	125～500
第一次转换，单次方式	14	25	26	130～520
自由运行转换	2	13	13	65～260
单次方式转换	2	13	14	70～280

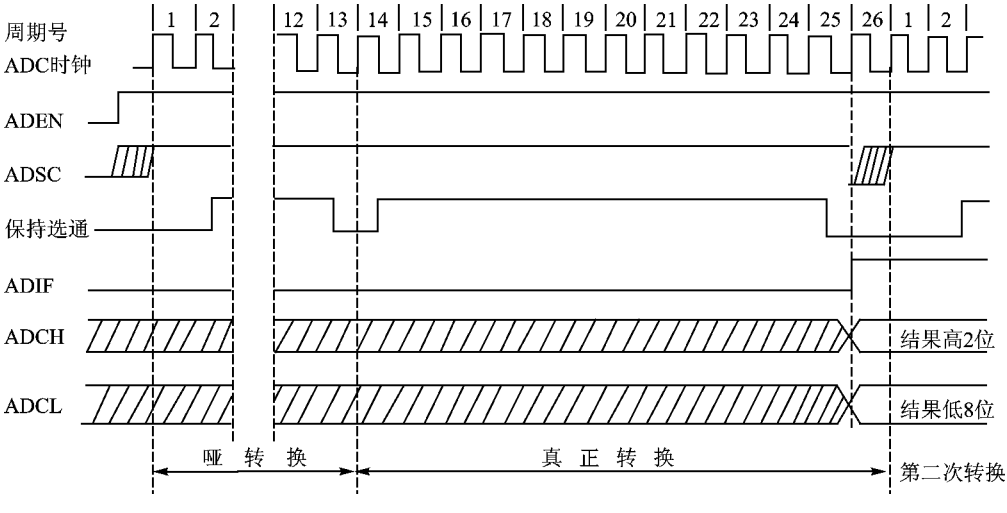


图 5.21　单次方式第一次 A/D 转换时序

在自由运行方式下，如图 5.23 所示，在转换结果被写到 ADC 结果寄存器中去之后，新的

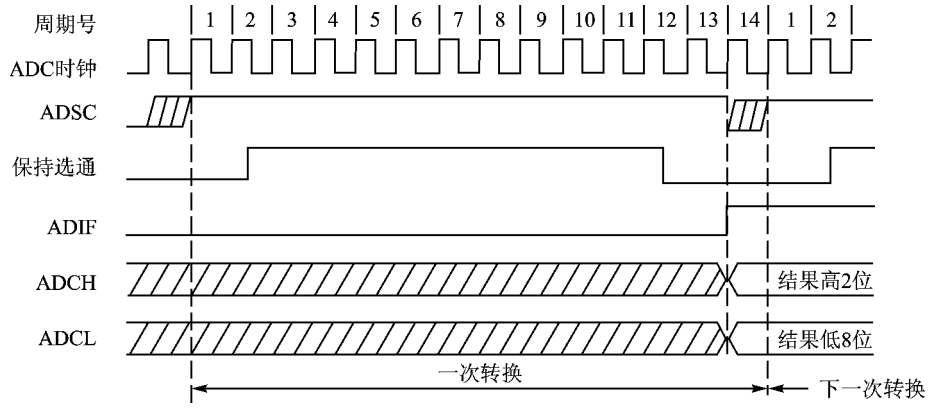


图 5.22 单次方式 A/D 转换时序

一轮转换便立即开始。若 ADC 时钟频率为 200 kHz，则自由运行方式的最低转换时间等于 65 μs，这相当于 15.4 ksps(15.4 k 采样/秒)。

应当指出，如果在转换过程中，有另外一个模拟通道被选中，那么 ADC 将在完成本次转换之后才进行通道切换。

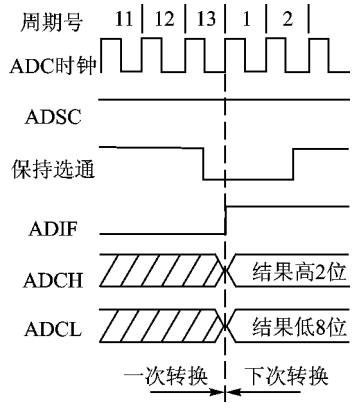


图 5.23 自由运行方式 A/D 转换时序

3. A/D 转换结果

转换完成后，10 位转换结果由 SAR 寄存器传送到数据寄存器 ADCH 和 ADCL 中去。为读转换结果而访问这两个寄存器时，专用的数据保护逻辑能保证前后两次读到的是同一次转换的结果，两者内容不是失配的。这就要求读数据时，先读低 8 位 ADCL。一旦 ADCL 接受 MCU 的读操作，ADC 对数据寄存器的访问即被阻断。这意味着，假如 ADCL 已被读出，但在读 ADCH 前，一次转换完成了，那么这两个寄存器的内容均不会得到更新。因此最后这次转换结果便因不准 ADC 向 ADCL 和 ADCH 两寄存器写入而丢失。ADCH 也被读出后，ADC 对此两寄存器的访问才重新获准。

ADC 有其自身的中断标志 ADIF，在每次中断完成时置 1 去请求中断(A/D 转换完成中断向量地址为 $00E)。在 MCU 对 ADCL 和 ADCH 寄存器所进行的两次读操作间，ADC 对这两个寄存器的访问是被禁止的。如果此期间有转换完成，尽管其结果要丢失，但 ADIF 标志还是要触发 ADC 中断的。

4. ADC 寄存器

AT90S4434/8535 的 I/O 空间中，自 $07($27)至 $04($24)的 4 个地址单元分别是 ADMUX、ADCSR、ADCH 和 ADCL 四个寄存器。

(1) ADC 模拟输入选择寄存器 ADMUX：地址 $07($27)，其高 5 位为保留位，只读，且恒读作 0，整个 8 位复位值为 $00。其低 3 位 MUX2～MUX0 是模拟通道选择位。若 3 位自高至低为 000，则选中 PA0 引脚上的模拟输入；如果等于 101，那么要转换的乃是来自 PA5 引脚上的模拟输入电压。

$07	7	6	5	4	3	2	1	0	
($27)	—	—	—	—	—	MUX2	MUX1	MUX0	ADMUX

(2) ADC 控制和状态寄存器 ADCSR：地址 $06($26)，可读亦可写，复位值等于 $00。

$06	7	6	5	4	3	2	1	0	
($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSR

位 7——ADC 开放位 ADEN：向这里写逻辑 1 即将开放 ADC，写入 0，则将关掉 ADC。在转换过程中关断 ADC 将终止此次转换。

位 6——ADC 转换启动位 ADSC：在单次转换方式下，欲启动每次转换均必须将此位写成 1。在自由运行方式下，只须在第一次启动时向该位写入逻辑 1。在 ADC 开放后，第一次将 ADSC 位写成 1 时，或 ADC 开放与 ADSC 位置 1 同时进行时，这初次转换将以一次哑转换为其先导。此哑转换要对 ADC 进行初始化。

在转换过程中，ADSC 位一直保持为高电平；在转换完成后但在转换结果被写入 ADC 数据寄存器之前，ADSC 被自动清成 0。这使用户可在现行转换完成之前，如图 5.22 所示，启动新的一次转换。待本次转换完成后，新的一轮转换便立即开始。由图 5.21 可见，哑转换引导真正转换时，ADSC 位在真正转换完成前将一直保持为高电平。

请注意，向此位写 0 是无效的，ADSC 不会因之清 0，转换也不因此而停止。

位 5——ADC 自由运行选择位 ADFR：在此位置 1 时，ADC 将运作于自由运行方式下。此方式下，ADC 不断对模拟输入进行采样，不断更新数据寄存器的内容。向此位写 0 即终止自由运行方式。

位 4——ADC 中断标志 ADIF：此标志在一次 A/D 转换完成且数据寄存器得到更新时置 1。若 ADIE 位和 SREG 寄存器中的 I 位均为 1，则 ADC 转换完成中断将获得响应。在中断响应过程中，ADIF 标志被硬件自动清 0。或者，向该标志位写逻辑 1 也可将之清 0。

请注意，若对 ADCSR 寄存器进行读-修改-写操作，则挂起的中断可能会被禁止。使用 CBI 和 SBI 指令对 ADCSR 进行操作时，情况也是如此。

位 3——ADC 中断允许位 ADIE：当此位与 I 同时为 1 时，ADC 转换完成中断便可获得响应。

位 2～位 0——ADC 分频选择位 ADPS2～ADPS0：这三位用来选择 ADC 分频器对 XTAL 信号频率的分频因子，以便得到合适的 ADC 输入时钟频率(见表 5.7)。

(3) ADC 数据寄存器 ADCH 和 ADCL：地址分别为 $05($25)和 $04($24)，只读，复位值两者均为 $00。ADCH，8 位，其中高 6 位无用，低 2 位用来存放转换结果的高 2 位。ADCL 存放转换结果的低 8 位。当 ADC 转换完成时，转换结果被硬件自动写入此两寄存器中。在自由运行方式下，很重要的是两寄存器都要读，且先读 ADCL，后读 ADCH。

5. ADC 噪音消除功能

AT90S4434/8535 片内 ADC 的最大特色在于它的噪音消除器。此噪音消除器在 MCU 的闲置方式下开放 A/D 转换，以降低由 CPU 内核感应来的噪音。按下述步骤去做，即启用了这项噪音消除功能。

(1) 确保 ADC 已开放，但未进行转换。必须选用单次转换方式，必须开放 ADC 转换完成

中断，即 ADEN＝1，ADSC＝0，ADFR＝0，ADIE＝1。

（2）进入闲置方式。一旦 CPU 停止运作，ADC 便将启动一次转换。

（3）在 ADC 转换完成前若无其他中断发生，则 ADC 中断将唤醒 MCU 去执行 ADC 转换完成中断服务程序。

6. ADC 噪音消除技巧

AT90S4434/8535 片内外的数字电路会产生可能影响模拟测量精度的电磁干扰。若对转换精度要求严格，则可用下述技巧降低噪音水平。

（1）两芯片之模拟部分及应用电路中的所有模拟部件，在印刷电路板上应有独立的地平面。此地平面与数字地平面在印刷电路板上通过单一点连接。

（2）模拟信号传递路径应尽可能短。应确保模拟信号途经该模拟地平面传送，并使这些传送通路远离高速切换的数字通路。

（3）AV_{CC} 引脚应与数字电路供电电压引脚 V_{CC} 通过阻容网络相连，如图 5.24 所示。

（4）使用前述 ADC 噪音消除功能以降低来自 CPU 的感应噪音。

（5）若 A 口某些引脚用作数字输出，则在 A/D 转换期间勿使其改变状态。

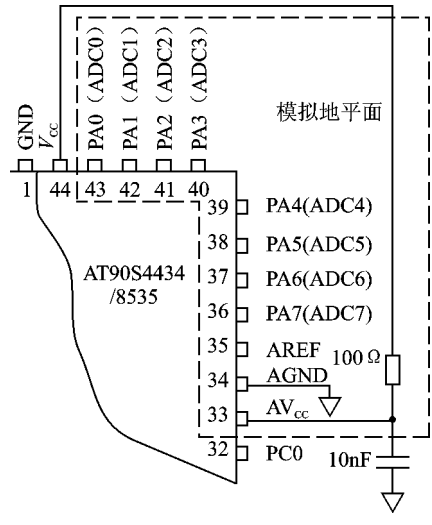


图 5.24 ADC 电源连接

最后应当指出，因为 AV_{CC} 要为 A 口输出驱动器供电，所以若有 A 口某引脚作输出，则上述阻容网络不该使用。

5.5.2 ADC 应用程序

[例 5.12] 利用 AT90S8535 设计一个实验系统，在其 PA0 引脚上通过电位器输入 0～5 V电压，观察 A/D 转换结果在 LED 显示器上的变化。系统时钟频率为 8 MHz。

图 5.25 所示为该系统的硬件构成情况。调节电位器的滑动端，使 PA0 的输入电压等于 0 V、2.5 V 和 5 V，将在 LED 显示器上分别看到转换结果 $000、$200 和 $3FF。下面是实验程序。

```
.include "8535def.inc"
      rjmp    reset
.org    $00e                    ; ADC 转换完成中断向量
      rjmp    adcint
.def    resulth=r2              ; 存放 ADCH 内容
.def    resultl=r1              ; 存放 ADCL 内容
.def    outer=r19               ; 延时外环计数器
.def    inner=r18               ; 延时内环计数器
.def    round=r17               ; 显示回合计数器
.def    temp=r16                ; 暂存寄存器
```

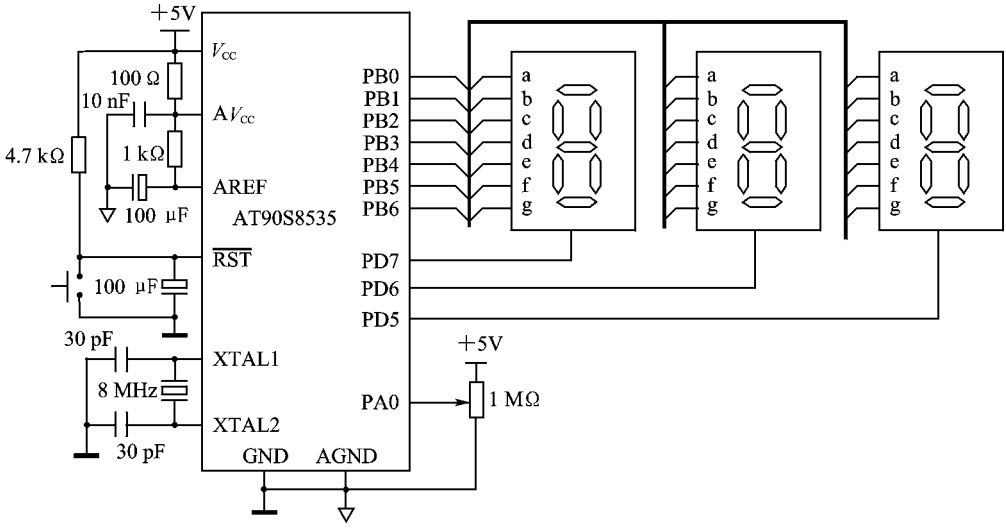


图 5.25　AT90S8535 A/D 转换实验系统

```
        .equ    label= $0100                ; 字形码首地址
        .org    $011
reset:  ldi     temp, $02                   ; 设置堆栈
        out     sph, temp
        ldi     temp, $5f
        out     spl, temp                   ; sp= $025f
        ser     temp
        out     ddrb, temp                  ; B 口为输出
        out     ddrd, temp                  ; D 口也作输出
        out     portd, temp                 ; D 口输出全 1
        clr     temp
        out     ddra, temp                  ; A 口作输入
        out     porta, temp                 ; 不接内部上拉电阻
        clt                                 ; 清 T 标志
        sei                                 ; 开总体中断
        ldi     temp, $8e                   ; ADEN=ADIE=1,单次方式
        out     adcsr, temp                 ; ADC 时钟频率=8 MHz/64=125 kHz
        clr     temp
        out     admux, temp                 ; 选 PA0 作模拟输入
        sbi     adcsr,6                     ; 启动转换
here:   brtc    here                        ; 等待转换完成
        clt                                 ; T 标志清 0
        ldi     round, 160                  ; 显示时间≈0.7 s
display: ldi    zh, high(label * 2)         ; 准备查表
        mov     temp, resulth               ; 取转换结果高位
        rcall   pattern                     ; 查表读高位字形码
        cbi     portd, 7                    ; 显示高位
```

```
        rcall   delay                   ; 延时约 1.5 ms
        mov     temp, resultl           ; 取转换结果中位和低位
        swap    temp                    ; 中低位交换位置
        rcall   pattern                 ; 查表,读中位数字形码
        sbi     portd, 7                ; 高位和低位暂不显示
        cbi     portd, 6                ; 显示中位
        rcall   delay                   ; 延时 1.5 ms
        mov     temp, resultl           ; 再取转换结果中位和低位
        rcall   pattern                 ; 查表,读低位数字形码
        sbi     portd, 6                ; 高位和中位暂不显示
        cbi     portd, 5                ; 显示低位
        rcall   delay                   ; 延时 1.5 ms
        dec     round                   ; 显示回合数减 1
        brne    display                 ; 不为 0,继续显示
        rjmp    here-1                  ; 显示告一段落,再启动转换
pattern: andi   temp, $0f               ; 查表、读字形码子程序
        ldi     zl, low(label * 2)
        add     zl, temp                ; 以转换结果数字作偏移量
        lpm                             ; 查表
        out     portb, r0               ; 输出字形码去显示
        ret                             ; 子程序结束
delay:  ldi     outer, 20               ; 延时约 1.5 ms 子程序
        ldi     inner, 200
        dec     inner
        brne    delay+2
        dec     outer
        brne    delay+1
        ret
adcint: in      resultl, adcl           ; ADC 转换完成中断服务程序
        in      resulth, adch           ; 读转换结果
        set                             ; T 标志置 1
        reti                            ; 中断返回
    .org        $100                    ; 字形码表
    ……
```

[例 5.13] 在图 5.25 中,通过 PA0～PA7 向片内 ADC 输入不同值的模拟电压,例如 PA0 输入 0 V,PA1 输入 0.5 V,PA2 输入 1 V 等等,要求把它们并发地转换成数字量,存放在片内 SRAM $60～$6F 等十六个地址单元中。

并发转换意味着宏观上同时完成的转换,而微观上则是诸通道相继进行的转换。并发转换要涉及到多通道的扫描问题。如前所述,通道切换要等一次转换完成后方可进行,因此,ADC 的自由运行方式可用来中间无停顿地扫描多个通道。一般说来,在 ADC 转换完成中断服务程序中,可选新的模拟通道。但应注意下述事实。

ADC 转换完成中断是在转换结果就绪时触发的。在自由运行方式下,一旦中断触发,下

一轮转换便立即启动。如果在中断触发后更改 ADMUX 值，试图立即以此新值选择模拟通道进行转换，那是做不到的。因为在此次中断触发时，下轮转换同时启动，用的是旧通道设定值，新设定值晚了一步。

解决这个问题的办法其实很简单，把模拟通道的设定提前一步进行就是了。在进行 PA0 模拟量的转换时，将 ADMUX 值改成 $01，这样，PA0 转换完成中断触发时，便自然立即启动 PA1 的转换过程。而在 PA0 转换完成中断服务程序中，再把 ADMUX 值改成 $02，如此等等。这样的结果便是，PA0 转换完成之际硬件立即自动启动 PA1 模拟量的转换，软件设定 PA2 通道。即 PAn 转换完成，自动启动 PA n+1通道的转换，软件设定 PA n+2通道。请看下面的程序。

```
        .include "8535def.inc"
            rjmp    reset
        .org    $00e
            rjmp    adcint
        .def    channel=r1              ;模拟通道号
        .def    lresult=r2              ;转换结果低字节
        .def    hresult=r3              ;转换结果高字节
        .org    $011
reset:  ……
        clr     channel                 ;从 0 通道(PA0)开始
        out     admux, channel
        ldi     r16, $ee                ;自由运行方式
        out     adcsr, r16              ;启动转换
        sei                             ;开总体中断
        clr     xh                      ;建立 SRAM 指针
        ldi     xl, $60
        inc     channel                 ;通道号增 1
        out     admux, channel          ;选通道 1
here:   rjmp    here                    ;等待中断
adcint: in      lresult, adcl           ;读转换结果
        in      hresult, adch
        st      x+, hresult             ;存入 SRAM
        st      x+, lresult
        inc     channel                 ;通道号加 1
        out     admux, channel          ;选下一个通道
        cpi     xl, $70                 ;刚转换的是通道 7 吗?
        brne    tomain                  ;否,返回
        cbi     adcsr, 7                ;是,停止转换
tomain: reti                            ;中断结束
```

第六章　开发工具及应用

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

6.1　AVR 实时在线仿真器 ICE200

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,及时引进了美国原装 AVR 实时在线仿真器 ICE200,如图 6.1 所示。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。因为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 6.1　ICE200

ICE200 的仿真软件最新版集成环境(IDE)为 Studio3.X,除支持以上 11 种 AVR 以外,还可模拟其他 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 2 KB 限制版。该软件及其升级版均可从互联网(www.imagecraft.com)上免费获得。

6.2　JTAG ICE 仿真器

AVR 高档单片机 ATmega 系列 ATmega16、ATmega32、ATmega323、ATmega128 等具

有 JTAG 仿真接口，如图 6.2 所示，可以实现在线实时仿真，这在 8 位单片机中属于领先技术。JTAG 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。

JTAG ICE 仿真器与 AVR 的连线，如图 6.3 所示。



图 6.2　JTAG ICE

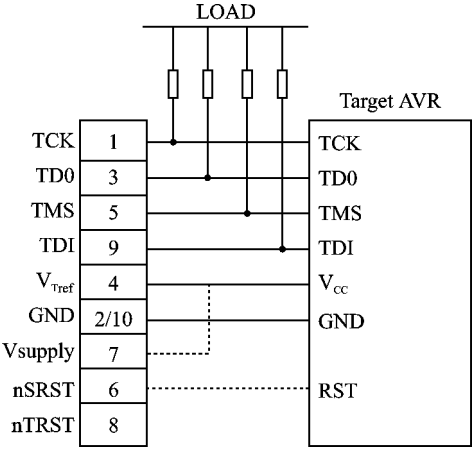


图 6.3　JTAG ICE 与 AVR 连线

6.3　开发下载实验器 SL－AVR＋

“四合一”增强型开发实验器 SL—AVR＋是配合《AVR 单片机应用技术》一书的出版研制的新产品。“四合一”等于 AVR 编程器＋模拟仿真器＋实验器＋科研样机，见图 6.4。SL—AVR＋硬件采用模块化设计，便于用户灵活组成科研项目所需的硬件结构。硬件有 RS232 通信接口；串行下载监控；DIP8～DIP40 通用插座，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他用途；有 6 位 LED 数码管作显示；有 2X16 点阵 LCD 液晶显示器；有 17 键的键盘；有 PC 机键盘接口插座；模拟比较输入电路；音响电路；单片机通用（AVR/MCS—51）复位电路；模拟电压输入电路等，随机附 120 mm×170 mm 万通实验板及一片 AT90S8515 器件。它适用于所有具有串行 ISP 下载编程功能的 AVR/AT89S 单片机，用户板上的器件无需拆下即可编程，同时还可做 AVR/AT89S 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；提供功能强大的 WIN 版汇编软件 AVRASM、模拟仿真调试软件 AVR Studio3. X 及串行下载软件 AVR PROG，同时也提供限时版的 Icc AVR C 编译器，Keil C51 6. XX 编译器；对初学 AVR/MCS—51 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。本机工作晶振可插拔更换，便于用户超频、降频实验。特别适用仪器仪表专业、科技人员开发之用。SL—AVR＋开发实验器提供了几十个实用实验程序，也可改变硬件接口，修改程序，实现原程序的功能。这对大专院校学生发挥其创造性思维及动手能力的培养特别有用。本开发实验器也可当科研样机使用。

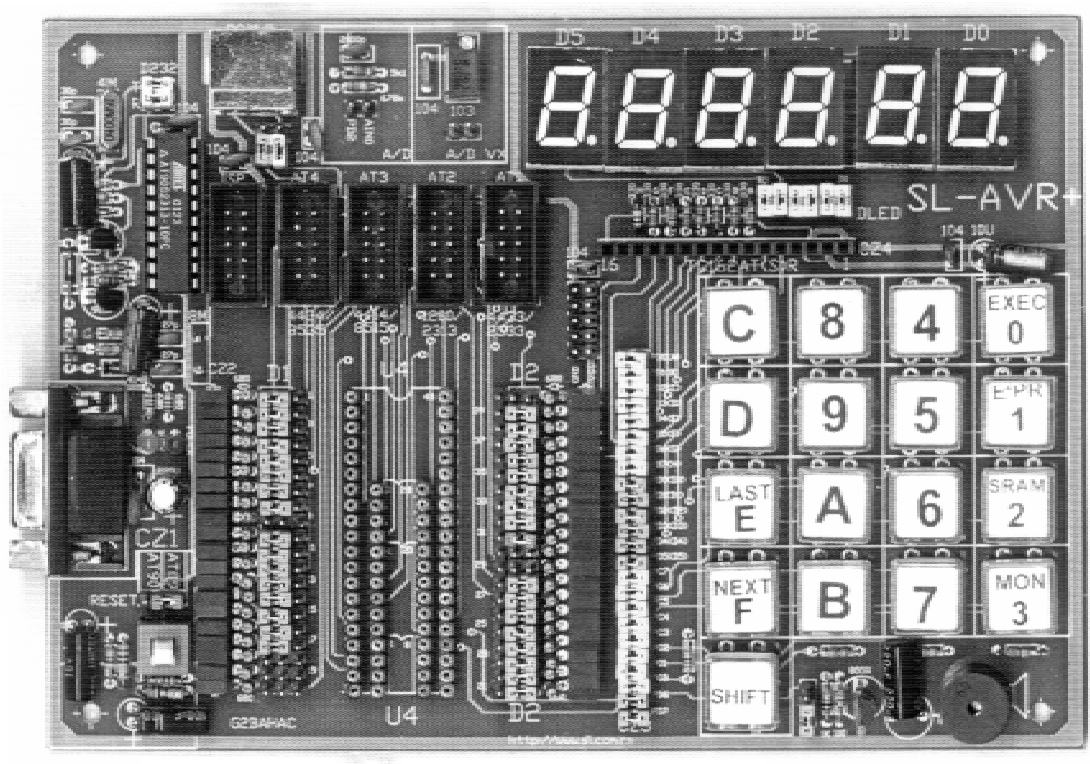


图 6.4 SL—AVR＋图

SL—AVR＋**电原理图，见图** 6.5。

SL—AVR＋**硬件接口电路见图** 6.6。**各器件说明如下**：

1. D232

D232 **通信线短路块**：**插上短路块，接到** AT90S2313；**拔出短路块，可从** T、R **端用插针线接到单片机通信口，可做单片机异步通信** UART **或主从同步通信** SPI **或** ISP **下载通信，见图** 6.7。

2. AT90S2313

AVR **开发实验器监控芯片**，AT90S2313 **为增强型开发实验器通用单片机**（AVR/AT89S）**监控**。

3. ISP

该列 DC10 **的**（ISP）**插座，即** AVR **单片机的下载信号插座，见图** 6.8。

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，其他器件作为选购件。

ISP **下载插座引脚功能分别为**：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。**

4. CZ1（双龙标准）

电源及通讯下载插座，电源线为地及＋5V，**通讯电缆一头接** CZ1，**另一头接计算机** RS232 **九针插座。**

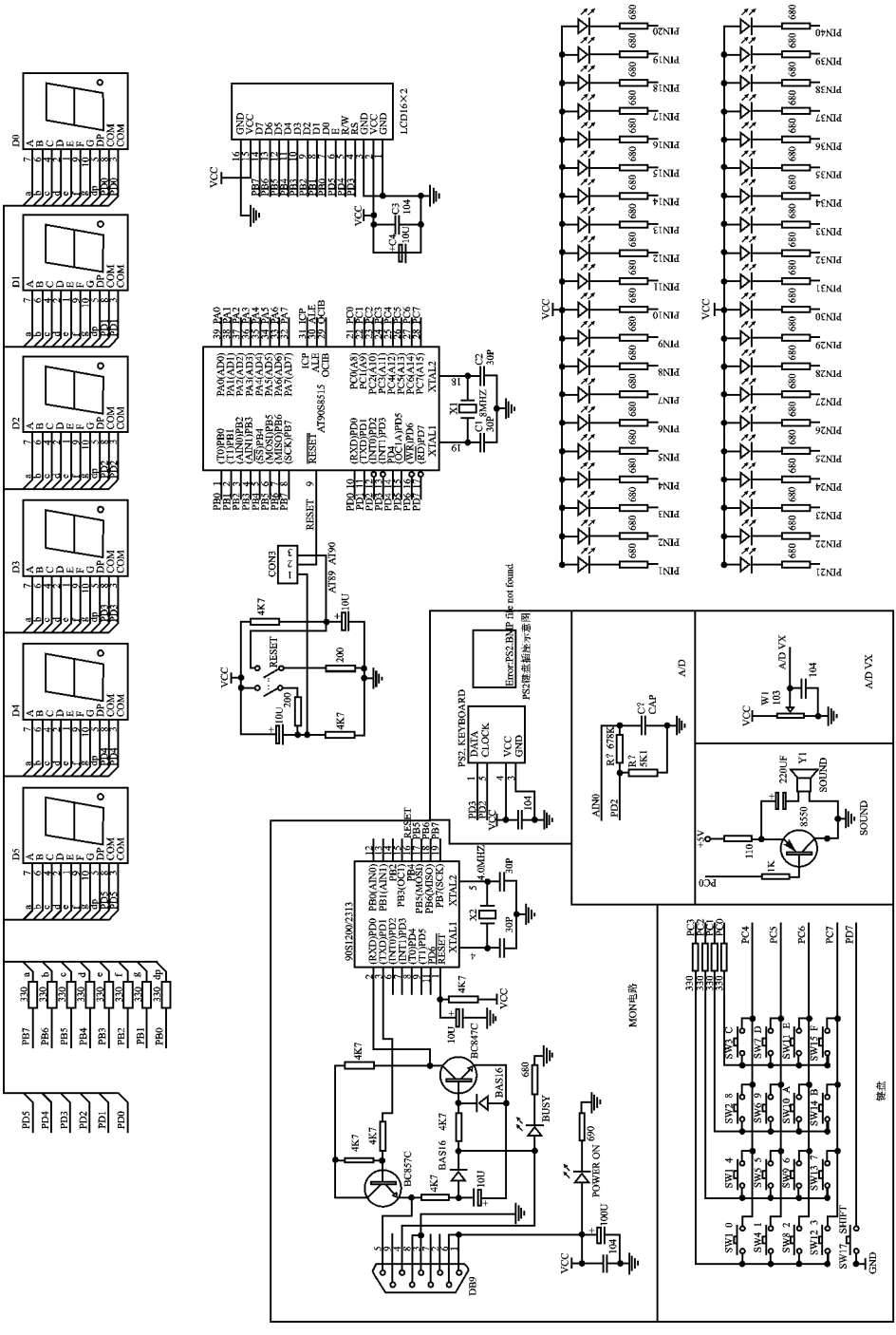


图6.5 SL—AVR+电原理图

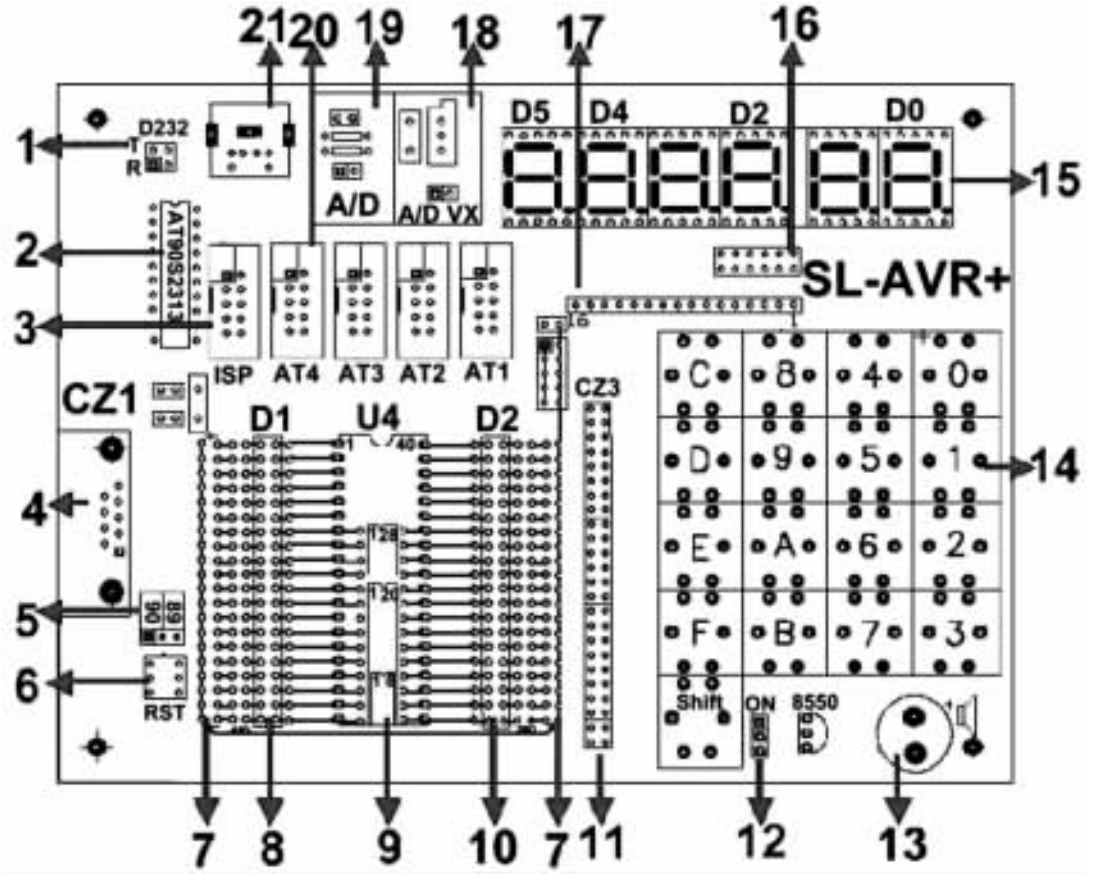


图 6.6 SL—AVR+硬件接口电路

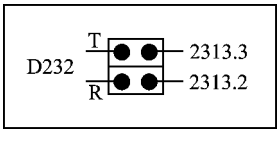


图 6.7 通信线短路块

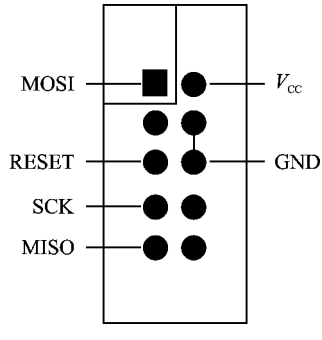


图 6.8 ISP 信号座引脚功能

POWER：**红色** LED，**电源指示**；BUSY：**绿色** LED，**下载通讯工作忙指示**。

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. (8 与 10) D1、D2 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他用途，如电源引线等。

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。

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他 I/O 口，也可改用其他器件(如 AT90S2313/AT90S8535 等)接到器件相应引脚上(D1、D2 短路块处)。本开发机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，实现原程序的功能。

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，再按数字键时，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，位线也可作其他用途。

17. CZ4 接线座

为 2 行×16 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

为片内摸拟比较器，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/AT90S2313 ISP 下载信号线座；

AT3 插针座为 AT90S4414/AT90S8515 及 AT89S8252/AT89S53 ISP 下载信号线座；

AT4 插针座为 AT90S4434/AT90S8535 ISP 下载信号线座。

如用于其他 AVR 单片机，用转接线接到其他 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工作可靠性，特别对晶振设置作了改进，在工作时，随机所附晶振接插小印板（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）。晶振数值可跟据实际工作改变（如超频、降频等）。

6.4 AVR 集成开发环境（IDE）

ATMEL AVR Studio3.53 集成开发环境（IDE），包括：(1) AVR Assembler 汇编器；(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 Studio3.53 集成开发环境窗口，见图 6.9。

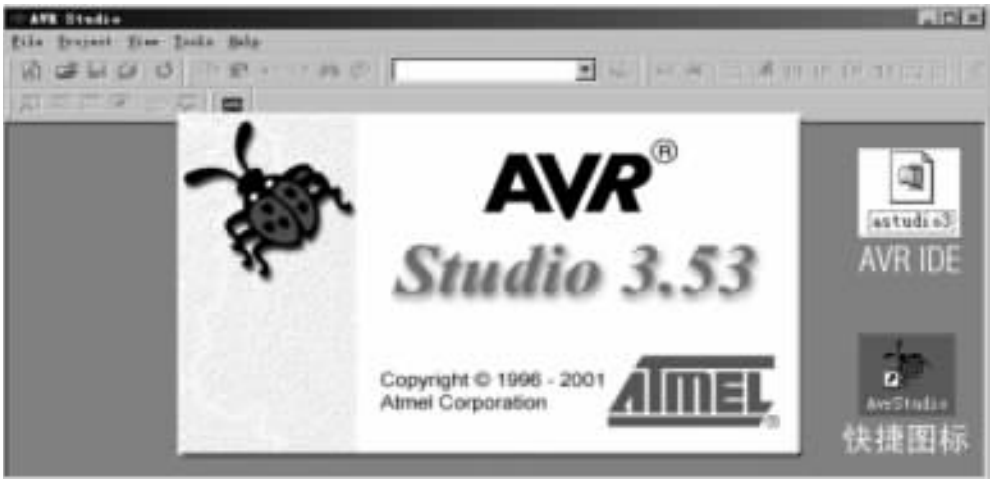


图 6.9 AVR studio 3.53 集成开发环境窗口

6.4.1 AVR Assembler 汇编器

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，汇编出错有错误定位、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，如 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；⑥ 即自动新建工程项目，见图 6.10。

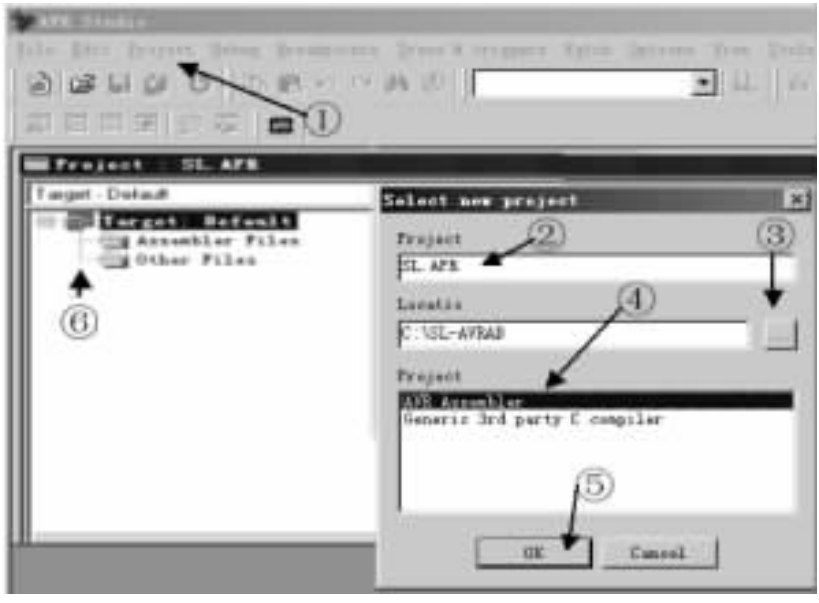


图 6.10　建立工程项目

2. 打开已保存的工程项目

File→Open **选择路径，打开已存在的工程项目**。

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，如** SL.ASM；③ **选择存放路径**；④ **按** OK **键**；⑤ **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** 6.11。

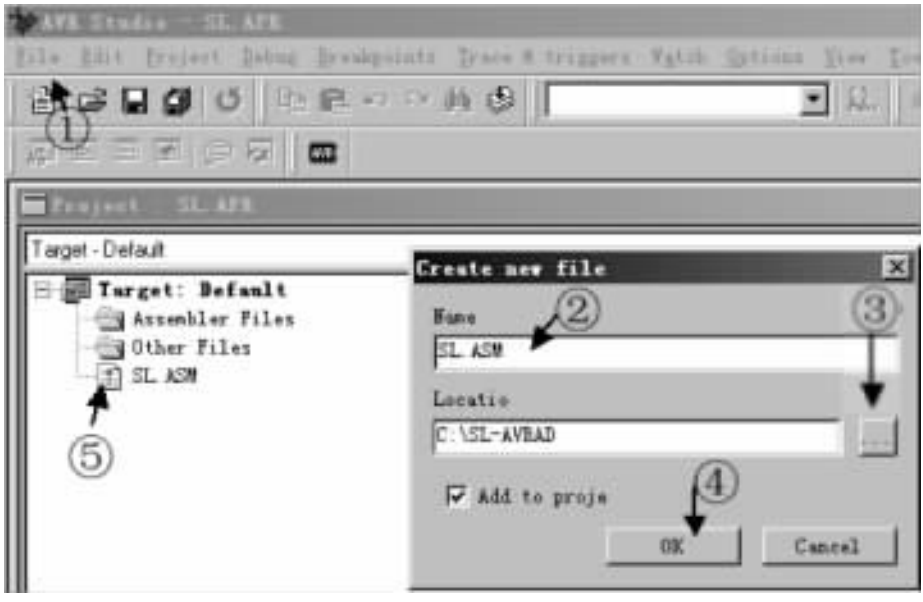


图 6.11　新建汇编文件名

4. 打开已保存的汇编文件

File→Open 选择路径，打开已存在的汇编文件。

5. 源文件汇编选项

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 6.12。

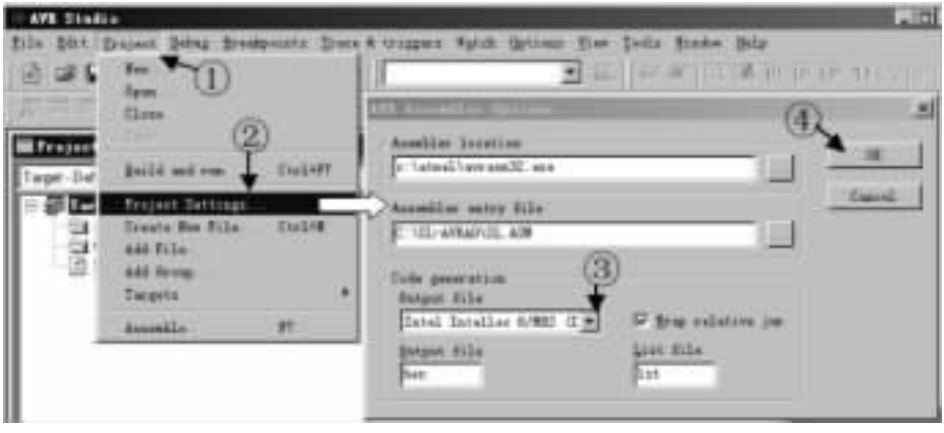


图 6.12 源文件编译选项

6. 新文件的编辑与汇编

① 窗口输入源文件 SL.ASM；② 选 Project 菜单；③ 选 Project 下拉菜单的 Assemble 汇编选项，进行汇编；④ 汇编通过，显示汇编提示，见图 6.13。



图 6.13 新文件的编辑与汇编

6.4.2　AVR Studio

对源文件 debug 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"图示。

选菜单 Debug→Go 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,根据源程序调试要求,还可打开更多的观察窗口, 见图 6.14。



图 6.14　AVR studio 调试窗口

打开已有文件进入调试: ① 选择菜单 File; ② 选择下拉菜单 Open; ③ 在一定路径下找到已有文件,文件类型可以是 *. obj; *. d90; *. cof; *. hex;也可以是源文件、工程项目文件等, 见图 6.15。

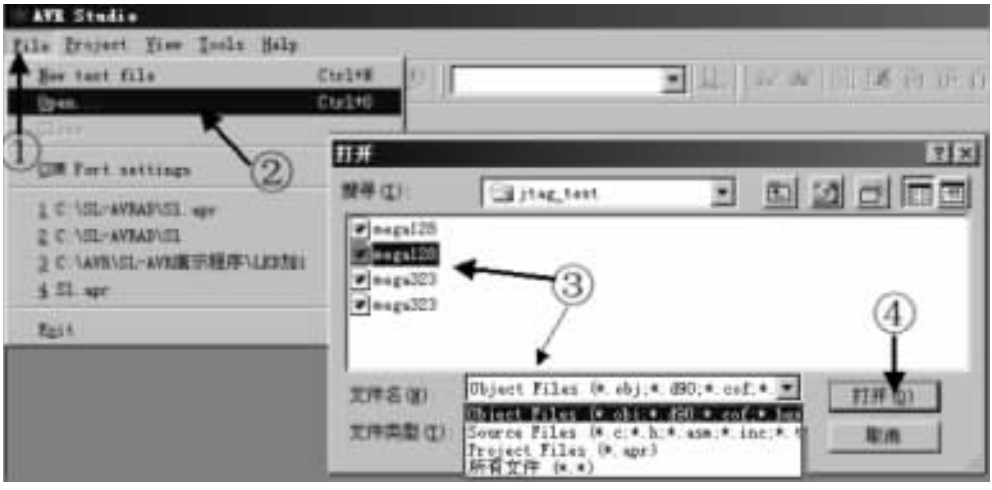


图 6.15　打开已有文件窗口

6.4.3　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)，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 串行编程

启动 AVR Studio3.53 **程序。如果对芯片编程，可以启动** TOOLS **菜单中的** STK500/AVR ISP/JTAG ICE **或** Alt＋9 **按钮，如图** 6.16 **所示，根据窗口提示操作。**

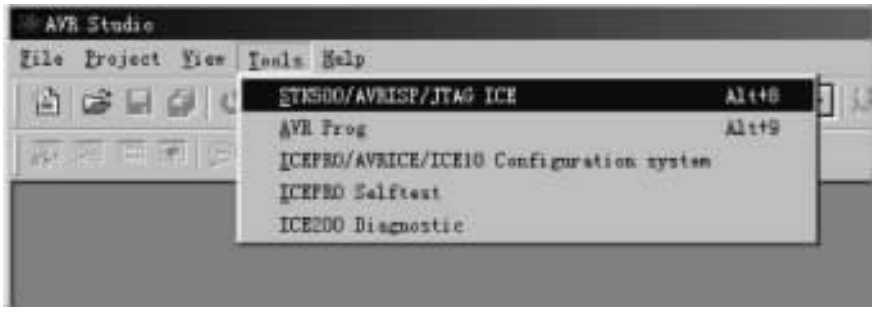


图 6.16　进入 AVR ISP 下载操作

2. AVR Prog 串行编程

启动 AVR Studio3.53 **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，根据窗口提示操作。**

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** 6.17 **所示，根据窗口提示操作。**

(2) **按图** 6.16 **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 6.17 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 下载窗口！

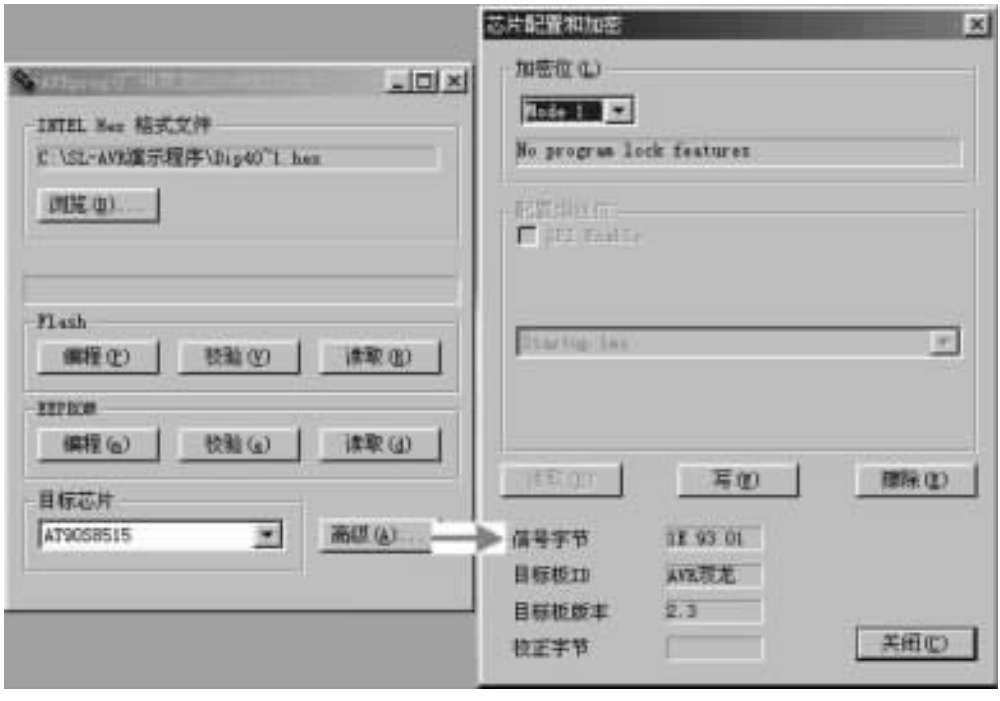


图 6.17　AVR ISP 下载操作窗口

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，也适用于 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—AVRAD\ *.ASM（实验程序）；

（6）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。

参考文献

[1] ATMEL,AVR RISC Microcontroller Data Book, 1999

[2] WWW. Atmel. com

[3] 林容益编著. AVR 高速 16 位元 PD 单晶片微控制器应用基础篇. 台北:全华科技图书股份有限公司,1999

北京航空航天大学出版社单片机与嵌入式系统图书

书　名	作　者	定价
嵌入式系统综合类		
嵌入式系统原理及应用开发技术	桑　楠	23.0
ARM 嵌入式处理器结构与应用基础(含光盘)	马忠梅	36.0
DSP 技术从书		
DSP 基本理论与应用技术	李哲英	32.0
DSP 基础与应用系统设计	王念旭	62.0
TMS320C54x DSP 结构、原理及应用	戴明桢	25.0
TMS320C54x DSP 应用程序设计与开发	刘益成	39.0
TMS320C54x DSP 应用系统设计	郑　红	18.0
TMS320LF240x DSP 结构、原理及应用(含光盘)	刘和平	36.0
TMS320F206 DSP 结构、原理及应用	李　刚	24.0
单片机教材系列		
单片机实验与实践教程(二)	夏继强	17.0
单片机初级教程——单片机基础	张迎新	24.0
单片机中级教程——单片机原理及应用	张俊谟	23.0
单片机高级教程——单片机应用与设计	何立民	28.5
单片机实用教程（高校计算机教学系列教材）	李　勋	17.5
MCS-51 系列单片机应用系统设计	何立民	35.0
单片机微型计算机大学读本	李　勋	18.0
单片机基础	李广弟	20.0
单片机基础(修订版)	李广弟	29.0
单片机原理及接口技术(简明修订版)	李朝青	26.0
单片机的 C 语言应用程序设计(修订版)	马忠梅	24.5
MCS-51/96 系列单片机原理及应用（修订版）	孙涵芳	35.0
单片机应用技术选编系列		
单片机应用技术选编（1）～（8）	何立民	
（1）～（8）的价格分别为：46.0、32.0、52.0、48.0、49.0、52.0、55.0、59.5		
单片机外围器件实用手册系列		
单片机外围器件实用手册——电源器件分册	关德新	57.0
单片机外围器件实用手册——存储器分册	窦振中	55.0
单片机外围器件实用手册——数据传输接口器件分册	邬宽明	54.0
单片机外围器件实用手册——输入通道器件分册	纪宗南	48.0
单片机应用技术从书		
单片机应用系统的功率接口技术	余永权	10.8
低功耗单片微机系统设计	何为民	9.5
I^2C 总线应用系统设计	何立民	15.5
单片机的 C 语言应用程序设计	马忠梅	23.5
单片机程序设计基础	周航慈	23.5
单片机应用系统抗干扰技术	王幸之	38.5

书　名	作　者	定价
PIC 单片机系列		
PIC 系列单片机原理和程序设计	窦振中	29.0
PIC 系列单片机应用设计与实例	窦振中	45.0
PIC 系列单片机的开发应用技术	武　锋	23.0
PIC 系列单片机接口技术及应用系统设计	王有绪	36.0
PIC16F87x 数据手册——28/40 脚 8 位 FLASH 单片机	刘和平	22.0
PIC16F87x 单片机实用软件与接口技术——汇编语言及其应用（含光盘）	刘和平	39.0
PIC16F87x 单片机实用软件与接口技术——C 语言及其应用（含光盘）	刘和平	32.0
PIC 单片机实用教程——基础篇	李学海	29.5
PIC 单片机实用教程——提高篇	李学海	即将出版
Intel 公司 MCS－51/96/196 单片机系列		
MCS-51/196 单片机浮点程序和实用程序	张克彦	21.0
Intel 80C196 单片机应用实践与 C 语言开发	程　军	32.0
MCS-51 系列单片机接口电路与应用程序实例	沈德金	13.0
MCS-51/96 系列单片机原理及应用	孙涵芳	16.0
MCS-51/96 系列单片机原理及应用（修订版）	孙涵芳	35.0
MCS-51 系列单片机实用接口技术	李　华	40.0
Intel 16 位单片机	孙涵芳	53.0
MCS-51 系列单片机应用系统设计	何立民	35.0
8098 单片机原理及实用接口技术	李杏春	38.0
嵌入式单片机 8XC251 用户指南	王　勇	22.0
MOTOROLA 单片机系列		
Motorola 单片机及专用集成芯片应用系统设计	陈粤初	29.0
MC68332 单片机结构与应用	张　宁	28.0
Motorola M68HC05 系列单片机及其应用基础	刚寒冰	25.0
Motorola J,K 系列廉价单片机原理及应用	余永权	19.5
Motorola 单片机应用技术	张　辉	15.0
Motorola 单片机原理及应用系统开发技术	谢宋和	26.0
PHILIPS 单片机系列		
80C51 系列微处理器系统原理、功能集成与应用	邬宽明	17.5
80C51 系列小型微控制器原理、应用和开发	邬宽明	19.0
80C51、XA 十六位微控制器系统设计、器件和应用	邬宽明	27.5
PHILIPS 51LPC 系列单片机原理及应用设计	周航慈	19.0
EPSON 单片机系列		
EPSON 单片机原理与应用	宋建国	33.0
EPSON 8 位单片机原理与应用	薛宗祥	20.0
EPSON 图形液晶显示控制器系列 SED135X 和 SED1374	郭　强	20.0
EPSON 单片系列液晶显示器及其应用	李继是	20.0

书　　名	作　者	定价
ATMEL 单片机系列(AVR,AT89)		
AVR 高速嵌入式单片机原理与应用	耿德根	40.0
AVR 单片机应用技术	李　勋	23.0
AVR 单片机应用设计	丁化成	22.0
ATMEL 89 系列单片机应用技术	余永权	32.0
其它公司单片机系列		
C8051Fxxx 高速 SOC 单片机原理及应用	潘琢金	27.0
EM78447B 单片机应用研究与制作（含光盘）	黄　勇	23.0
ADμC8xx 系列单片机原理与应用技术	李　刚	18.0
亿恒(西门子)C166 系列 16 位单片机原理与开发(含光盘)	程　军	40.0
MSP430 系列 FLASH 型超低功耗 16 位单片机	胡大可	30.0
MSP430 系列超低功耗 16 位单片机原理与应用	胡大可	29.5
三菱 3850 系列单片机原理和应用	陆延丰	25.0
三菱 M16C/62 单片机原理与应用	陆延丰	39.0
东芝单片机原理及其在家用电器中的应用	谌清平	29.5
日立 H8/3048 系列单片机应用技术	李　勋	29.5
青少年单片机系列丛书		
青少年机器人初级教程—太空穿梭机 LOGO,QBASIC 语言编程	卢燕林	10.0
少儿单片机工程师	盛焕鸣	7.0
DP801 单片机系统入门	车金相	7.0
当代美猴王——少年单片机应用与开发	胥筱汀	17.0
单片机模糊技术应用选编系列		
模糊技术与神经网络技术选编(1)～(5)	刘增良	
(1)～(5)的价格分别为:50.0、56.0、50.0、50.0、50.0		
计算机模糊控制原理及应用	戎月莉	7.5
模糊信息优化处理技术及其应用	黄崇福	16.0

书　　名	作　者	定价
模糊专家系统原理与设计	刘有才	18.0
单片机模糊逻辑开发软件	余永权	16.8
模糊逻辑控制技术及应用	窦振中	23.0
模糊逻辑与神经网络——理论研究与探索	刘增良	33.0
单片机模糊逻辑与模糊控制	余永权	40.0
神经模糊系统及其应用	王士同	28.0
模糊控制技术与模糊家用电器	余永权	23.0
单片机综合类		
单片机微型计算机应用和开发系统	徐爱卿	12.0
单片机应用程序设计技术	周航慈	7.3
单片机总线扩展技术	尤一鸣	10.5
CAN 总线原理和应用系统设计	邬宽明	25.0
单片机现场可编程外围芯片 PSD 的原理及应用	孙涵芳	39.5
单片机人机接口实例集	公茂法	17.0
智能化测量控制仪表原理与设计	徐爱钧	32.0
单片微机测控系统设计大全	王福瑞	38.0
从 C 到嵌入式 C 编程语言——入门、实用、深入	梁合庆	35.0
PC 机及单片机数据通信技术	李朝青	28.0
《单片机与嵌入式系统应用》杂志 2001 年(1～6)合订本		42.0
《单片机与嵌入式系统应用》杂志 2001 年(7～12)合订本		42.0
单片机 C 高级语言程序设计及其应用	袁　涛	18.0
2001 嵌入式及单片机国际学术交流会议论文集(中文)	沈绪榜	150
从神经元芯片到控制网络	凌志浩	21.0
计算机语音技术(修订版)	朱民雄	37.0
电动机的单片机控制	王晓明	24.0
单片机 &DSP 外围——数字 IC 技术手册(上册)	李朝青	即将出版
单片机 &DSP 外围——数字 IC 技术手册(下册)	李朝青	即将出版

可直接向出版社邮购以上图书。通信及汇款地址:北京航空航天大学出版社邮购组(邮编 100083)
网址:http://www.buaapress.com.cn　电话:010-82317024,传真:010-82328026。另加 2 元挂号费。(表中反白者为新书)

投稿单片机与嵌入式系统图书请联系:北航出版社　马广云　电话:010-82317022　传真:010-82317043
Email:pressb@public3.bta.net.cn

《单片机与嵌入式系统应用》月刊

- 北京航空航天大学出版社承办
- 何立民教授主编
- 中央级科技期刊
- 单片机与嵌入式系统专业期刊
- 特色:专业期刊、专家办刊、着眼世界、面向国内、应用为主、读者第一。
- 已开辟有 8 个栏目:业界论坛、专题论述、技术纵横、新器件新技术、应用天地、经验交流、学习园地、编读往来
- 杂志社网址:http://www.microcontroller.com.cn,http://www.dpj.com.cn

诚挚欢迎业界人士向本刊投稿,欢迎广大读者订阅本刊。

联系方式

通信或汇款地址:北京市海淀区学院路 37 号《单片机与嵌入式系统应用》杂志社
邮编:100083　　电话:(010)82317029,82313656　　传真:010-82317043
E-mail:mcu@publica.bj.cninfo.net(投稿亦请用此 E-mail 地址)
广告业务联系人:王金萍　魏洪亮　　开户行:北京市商业银行学院路支行
户名:《单片机与嵌入式系统应用》杂志社　　账号:01090339100120111029936

订阅方式

国内统一刊号:CN 11-4530/V　　国际标准刊号:ISSN 1009-623X
每期定价:8 元,全年定价:96 元,每月 1 日出版。
第一种订阅方式:通过邮局订阅,邮发代号:2-765。
第二种订阅方式:直接向《单片机与嵌入式系统应用》杂志社订阅,另加 15%邮资。