主编　薛钧义　张彦斌

凌阳16位单片机原理及应用

编著　薛钧义　张彦斌
虞鹤松　樊　波

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

内 容 简 介

本书介绍了中国台湾凌阳科技股份有限公司(Sunplus Technology CO. LTD)近年推出的基于SOC的16位μ'nSP™CPU单片机原理及应用。主要内容为:μ'nSP™CPU系列单片机的基本结构与组成原理;寻址方式和指令系统;片内的外设部件,如并行口、串行口、定时器、计数器、ADC和DAC(PWM)等;中断系统;汇编语言程序设计;集成开发环境IDE及较多的应用实例。书中重点介绍了凌阳公司最新推出的以SPCE061系列为主的各单片机的特点,以便用户根据需要选用所需的单片机,组成应用系统和嵌入式计算机系统。此类单片机特别是在数字信号处理及语音处理方面有着明显的优越性。书中提供了较多的既用汇编语言又用C语言编程的应用程序实例。

本书既可作为本科生教材,也可作为高年级学生科研实践的参考资料和工程技术人员自学及开发产品的参考资料。

图书在版编目(CIP)数据

凌阳16位单片机原理及应用/薛钧义等编著. —北京:北京航空航天大学出版社,2003.1

ISBN 7-81077-261-9

Ⅰ.凌… Ⅱ.薛… Ⅲ.单片微型计算机,凌阳基础知识 Ⅳ.TP368.1

中国版本图书馆CIP数据核字(2002)第090948号

凌阳16位单片机原理及应用

主编 薛钧义 张彦斌

编著 薛钧义 张彦斌 虞鹤松 樊 波

责任编辑 金友泉

责任校对 陈 坤

*

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(邮编100083) 发行部电话:(010)82317024 传真:(010)82328026

http://www.buaapress.com.cn

E-mail:bhpress@263.net

河北省涿州市新华印刷厂印装 各地书店经销

*

开本:787×1 092 1/16 印张:20 字数:512千字

2003年2月第1版 2003年2月第1次印刷 印数:5 000册

ISBN 7-81077-261-9 定价:26.00元

前　言

随着计算机新技术不断地涌现和 VLSI 设计技术的迅猛发展，计算机无论在系统体系结构还是在性能上都发生了巨大的变化。计算机一方面向着高速、智能化的巨型机方向发展，另一方面向着嵌入式微型机的方向发展。其中，微型计算机以其性能和容量不断提高，而价格不断下降的趋势，使其在社会各个领域乃至家庭生活中发挥着越来越大的作用。

当今微型计算机技术的发展形成两大技术分支：一是以微处理器 MPU 为核心组成的通用微机系统；另一分支是发展面向对象的用于实时测控领域的微控制器 MCU，亦称单片微计算机(单片机)。随着超大规模高速集成电路的发展，现代电子技术的设计与应用进入了片上系统 SOC(System On a Chip)阶段，从而使单片机的设计与应用技术发生了深刻的变化。SOC 的设计要从整个系统的功能及性能出发，把微处理器(MPU)、芯片结构、数字/模拟等外围器件全部放置在一块芯片中，完成整个系统的功能，真正实现"系统单片机"。故 SOC 为单片机的应用提供了更为广阔的应用前景及更强的生命力。

台湾凌阳科技股份有限公司近年推出的 μ'nSP™单片机就是基于 SOC 的新型的数/模混合的系统级芯片。在一个芯片内集成了单片机数据采集或控制系统所需的模拟和数字外设及其它功能部件。凌阳公司推出的单片机有 8 位、16 位共 50 多种不同的 IC 芯片型号及其产品，以适用于不同的应用领域，用户可根据需要选择相应的单片机系统。凌阳单片机顺应了单片机技术的发展趋势，其系统芯片具有集成度高、数/模混合、功能全、低功耗、低电压和易于开发等特点。另外，凌阳单片机还增加了适合于 DSP 的某些特殊指令；有些系列的单片机还嵌入了 LCD 控制/驱动和双音多频发生器功能。这些都进一步扩大了单片机的应用范围。

凌阳公司为 μ'nSP™单片机的应用开发提供了较为完善的开发支持工具：硬件工具和软件工具。硬件工具即为 μ'nSP™CPU 系统仿真板，用于程序的二进制代码的下载调试；软件工具按其操作系统分为 DOS 环境下的分立工具和 Windows 环境下操作的集成工具。开发系统提供了高性能的 C 语言编译器(Gcc，它采用符合 ANSI 标准的 C 语言编程，并提供了 C 语言的编程函数库(C－Lib)。开发系统中还提供了凌阳公司推出的按照凌阳音频编码方法进行语音压缩算法的 SCAM－Lib 算法库，用于开发和实现芯片的语音功能，从而使得凌阳单片机在数字信号及语音识别应用领域中显示出更大的优越性。

为了及时向读者介绍凌阳公司生产的基于 SOC 的 μ'nSP™系列单片机，在凌阳公司及北京北阳电子技术有限公司提供资料的基础上，我们组织编写了本教材，作为首次出版并介绍关于凌阳 μ'nSP™系列单片机的书籍。本书在内容和体系安排上注意到可作为教材使用的特点，从通用的单片机组成原理的框架结构基础讲述 μ'nSP™单片机，力求摆脱使用手册的形式，既讲述原理又配合例子加以说明。有些主要章节还留有习题与思考题，以便读者自学与掌握重点。本书注重实际应用，列举了一些应用实例，读者可通过实例掌握编程方法并掌握系统的综合应用技术，以利于读者在此基础上开发高水准的 μ'nSP™单片机应用系统。

本书由薛钧义、张彦斌老师任主编。第一章、第二章由张彦斌老师负责组织编写；第三章、第四章、第六章、第八章由樊波老师负责组织编写；第五章、第七章由虞鹤松老师负责组织编

写；第八章、第九章由薛钧义、张彦斌老师负责组织编写。全书由薛钧义老师统稿。张永华、范宇老师，研究生罗旭、皮海峰、钟寒梅、庞淑敏，本科毕业班同学陶磊、陈璐、张佳、王征宇和余政等为本书提供了一些应用程序实例，并参与了部分编写工作。

我们还要感谢台湾凌阳公司、北京北阳公司在产品资料和开发工具方面提供的各种帮助与支持。有关 μ'nSP™ 系列单片机进一步的资料及最新动态、应用信息，请访问下列网站：

凌阳公司：http://www.sunplus.com.tw；

北阳公司：http://www.unsp.com.cn。

本书的及时出版得到北京航空航天大学出版社责任编辑金友泉同志及北阳公司的大力支持与关心。在此，一并表示衷心的感谢。

由于时间仓促，书中肯定存在很多错误和不妥之处，恳请读者批评指正。

编者 2002．6 月

于西安交通大学工业自动化系

现将部分电子、测控、计算机、仪器、机械等类高等学校教材登录如下，望高等学校教师、学生及相关工程人员参阅与征订，欢迎大家对教材内容提出批评意见，欢迎来稿出版。

联系电话 010　82317037，邮编 100083，地址：北京航空航天大学出版社，责任编辑：金友泉。

2002 年已出版的高等院校教材

书　号	书　　名	编著者	定　价
873－6	电子工艺及电子工程设计	汤元信等	20.00
121－3	测控电路及装置	孙传友等	26.50
184－1	测控系统原理与设计	孙传友等	28.50
154－X	现代电信交换技术与通信网	尤克　等	22.00
009－8	现代数字移动通信原理及应用技术	尤克　等	26.00
212－0	现代电气及可编程控制技术	王永华等	27.00
012－0	光纤通信及网络技术	徐宝强等	19.00
083－7	电视原理与组网技术	徐宝强等	27.00
864－7	微机原理与接口技术	邵鸿余	30.00
074－8	微型计算机技术——系统、接口与通信	田辉　等	21.00
185－X	PC 电脑的组成、组装调试与维护	盛鸿宇等	16.00
012－8	计算机网络技术及网络集成	马桂　等	26.00
186－8	电子电路学习要点与例题精解	何希才	21.00
957－0	新型电子电路应用实例精选	易文平	25.00
204－X	智能控制与 Lon 网络开发技术	马　莉	30.00
261－9	凌阳 16 位单片机原理及应用	薛钧义等	26.00
	能力成熟度模型(CMM)与软件开发技术	黎连业等	24.00
205－8	工程图学基础(少学时)	王农　等	24.50
236－8	工程图学基础习题集	王农　等	11.00
206－8	工程图学与计算机绘图(多学时)	于颖　等	29.00
237－6	工程图学与计算机绘图习题集	于颖　等	13.00
978－3	现代工程制图	于颖　等	27.00

2003 年待出版的高等院校教材

序　号	书　　名	编著者	定　价
1	微弱信号检测技术	熊晓东等	
2	测控软件开发技术	吴正平等	
3	自动控制原理	冯巧玲等	
4	虚拟仪器实用教程	周求湛等	
5	虚拟仪器及现代测控系统	曹玲芝等	
6	数字系统 Verilog 设计教程(上)	夏宇闻	
7	数字系统 Verilog 设计教程(下)	夏宇闻	
8	现代数字逻辑设计教程	邵鸿余	
9	机械电子工程学	李钟慎等	
10	实用电工技术教程	张永红等	
11	电子工艺实习及电子产品制作	张翠霞等	

目　　录

第一章　凌阳单片机简介

1.1　单片计算机概述

电子计算机的发明是20世纪人类最辉煌的成就之一。自从1946年第一台计算机问世以来，计算机的发展日新月异。短短几十年间，已由电子管数字计算机发展到今天的超大规模集成电路计算机。近年来，计算机一方面向着高速、智能化的超级巨型机方向发展，另一方面向着微型机的方向发展。

在计算机科学的发展过程中，微型计算机的出现是划时代的进步，它为计算机的发展和普及开辟了一条崭新的途径。当今微型计算机技术的发展形成了两大分支，一是以微处理器(Micro Processor Unit，简称MPU)为核心所构成的通用微机系统。它主要用于科学计算、数据处理、图形图像处理、数据库管理、人工智能、数字模拟与仿真等领域。另一分支是微控制器(Micro Controller Unit，简称MCU)，亦称单片机。单片机主要用于工业测控，如家用电器、电子娱乐玩具、计算机外围设备、工业智能化仪表、机器人、生产过程的自动控制、农业、化工、军事和航空航天等领域。作为通用微机核心的微处理器MPU和工业测控领域使用的微控制器MCU，在微型计算机技术的发展过程中既互相区别，又互相融合，互相促进。它们虽各具特色，但在原理和技术上是紧密联系的。

通用微处理器把计算机的运算器和控制器集成在一块芯片上，因而又称之为中央处理器CPU(Central Processor Unit)。由CPU、存储器(RAM和ROM)、I/O(Input/Output)接口和I/O设备就构成了通用的微型计算机系统。可见通用微机系统由多个IC(Intergraded Circuit)芯片组成，CPU只是其中的一个组成部分。如果把CPU、RAM、ROM、中断系统、定时/计数器、以及I/O口等主要微型机部件集成在一块芯片上，就称该芯片为单片微型计算机SCMC(Single Chip Micro Computer)，简称单片机。虽然单片机只是一个芯片，但从组成和功能上看，它已具备了计算机系统的属性，因而是一个简单的微型计算机。

单片微型计算机主要应用于控制领域，用以实现各种测试和控制功能，所以国际上更通用的叫法为微控制器MCU(Micro Controller Unit)。事实上，单片机在发展过程中也不断完善和增加其控制方面的性能，如硬件上增加A/D、PWM部件及高速I/O口，软件上增加乘、除及位处理指令等。但在我国仍习惯于单片机这一名称，因此本书使用“单片机”一词。

在单片机诞生之前，为了满足工控对象的嵌入式应用要求，只能将通用计算机进行机械加固、电气加固后嵌入到对象体系(如舰船)中构成诸如自动驾驶仪和轮机监控系统等。由于通用计算机的巨大体积和高成本，无法嵌入到大多数对象体系(如家用电器、汽车、机器人、仪器仪表等)中，由此单片机便应运而生。单片机的单芯片的微小体积和极低的成本，可广泛地嵌入到如玩具、家用电器、机器人、仪器仪表、汽车电子系统、工业控制单元、办公自动化设备、金融电子系统、舰船、个人信息终端及通信产品中，成为现代电子系统中最重要的智能化工具。所以，也常常将单片机称为嵌入式微控制器EMCU(Embedded Micro Controller Unit)。正是

由于单片机在嵌入式系统中的成功应用，使单片机成为发展最快，品种最多，在嵌入式系统中获得最广泛应用的机型。

就单片机本身而言，还可分为通用型和专用型两种类型。所谓通用型单片机，其内部资源比较丰富，性能全面，适应性强，能满足多种应用需求。但用户使用时需要进一步设计，才能组建成一个以通用单片机为核心再配以其它外围电路的应用控制系统。如 MCS—51、MCS—96 和凌阳 16 位等单片机。专用型单片机是针对一种产品或一种控制应用而专门设计的，设计时已经对系统结构的最简化、软硬件资源利用的最优化、可靠性和成本的最佳化等方面都作了通盘的考虑和论证，所以专用单片机具有十分明显的综合优势。如电度表和 IC 卡读写器的单片机和凌阳 8 位单片机中某些 IC 芯片都具备上述特色。显然，随着单片机应用的广泛深入，各种专用单片机芯片将会越来越多，并且必将成为今后单片机发展的重要方向。

如上所述，单片机只是一个芯片，而在实际应用中，常常需要扩展外围电路和外围芯片构成具有一定应用功能的单片机系统。由于软硬件资源所限，单片机系统本身不能实现自我开发，要进行系统开发设计，必须使用专门的单片机开发系统。对于复杂系统，可借助于微型机开发系统 MDS(Microcomputer Development System)进行开发。而对于一般系统可用专门的单片机开发系统，称为单板单片机开发系统或在线仿真器 ICE(In Circuit Emulation)，通过它们可以进行单片机的软硬件开发和 EPROM 的写入。目前国内市场上可提供各种类型和型号的单片机开发系统，为单片机的开发应用提供了得力的工具，使单片机用户有了很大的选择余地。本书在其后某些章节中，将用专门篇幅讨论凌阳单片机的开发特点及其颇具特色的开发系统。

通过对单片机的回顾和展望，可以使我们更加理解单片机发展的前景。

1971 年微处理器研制成功后不久，出现了一位的单片机。1976 年 Intel 公司首先推出了 MCS—48 系列 8 位单片机。它以体积小、功能全、价格低等特点，赢得了广泛地应用，这为单片机的发展奠定了基础，成为单片机发展进程中的一个重要的阶段。其后，在 MCS—48 成功的刺激下，许多半导体芯片生产厂商竞相研制和发展自己的单片机系列。到目前为止，世界各地已相继研制出大约 50 多个系列 300 多个品种的单片机产品。在众多单片机产品中，以 1983 年 Intel 公司推出的 MCS—96 系列单片机为典型代表，涌现出一批性能更高、功能更强的 16 位单片微型计算机，在单片机市场中与 8 位机“平分秋色”。尽管现在已经有了 32 位的单片机，但由于 8 位机、16 位机不断更新换代，性能不断提高，功能日趋完善，可以满足绝大部分控制应用的要求，因此在一段较长的时期内，8 位、16 位机仍是单片机的主要机型。

综观单片机近 30 年的发展历程，单片机今后将向多功能、高性能、高速度、低电压、低功耗、低价格的方向发展。另外，单片机从单片微型计算机向微控制器(MCU)发展，它体现了单片机向片上系统(SOC)的发展方向。按系统要求片内不断扩展外围电路功能，既有数字逻辑电路部分，又有数据采集和控制系统中常用的模拟部件、外围接口和其它一些功能部件，实现应用系统在片内集成。新近推出的凌阳 μ'nSP™ 单片机及美国 Cygnal 公司的高速 C8051F 系列单片机均为适应这种发展要求的、基于 SOC 的单片机。

1.2 凌阳单片机简介

凌阳单片机是由台湾凌阳科技股份有限公司生产的系列产品。该公司是从事 IC 电路设

计和生产的专业化公司，目前主要产品包括：液晶 IC、微控制器 IC、多媒体 IC、语音 IC、音乐 IC 及各种专用 IC。多年来，该公司致力于研发、制造、经销高品质和高附加值的消费性集成电路芯片，使凌阳科技在消费性 IC 国际市场已占有一席之地。

凌阳单片机目前主要有 8 位机和 16 位机两个系列。其中，8 位机根据不同用途分别带有 LCD(Liquid Crystal Demonstrator)驱动或带有单通道、双通道或多通道发声功能，因而可适合制作各种款式的计算器、数据库、游戏机及各种档次的电子琴以及高级电子玩具或电子宠物等，也可用于嵌入式计算机系统。

如果说凌阳 8 位单片机充分体现了该公司的优势和特色，更适用于开发消费性产品，而此后推出的以 μ'nSP™CPU 为内核的 16 位单片机，则是适用面更广的通用型单片机。它以 μ'nSP™ 为内核，集成不同规模的 ROM、RAM、ADC 和 PWM(DAC)、并行/串行接口等片内资源，如图 1－1 所示。

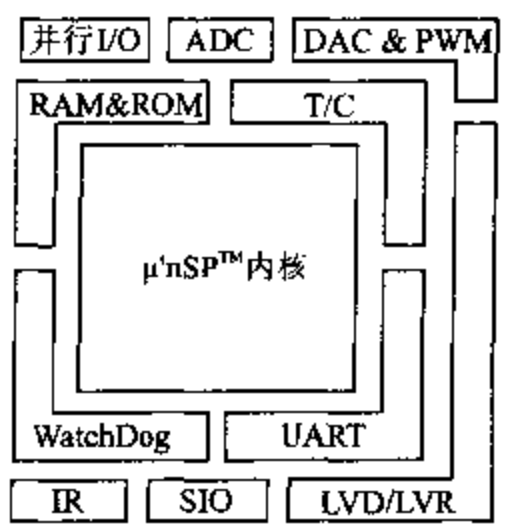


图 1－1　μ'nSP™的模块结构

图中以 μ'nSP™ CPU 为内核，其它功能模块根据应用需要为可选结构。这就使得以 μ'nSP™ 为核心组成的单片机系统可大可小，某些功能模块在系统组成时可有可无。借助这种积木式结构，可对不同的对象、不同的应用领域，形成不同的系列，以适应不同的应用场合。而用这种方法设计的每一种派生产品无疑会具有更强的功能和更低的成本。这也正是凌阳 16 位单片机的基于 SOC 的结构特点。

目前凌阳 8 位单片机已有 SPL 系列、SPC 系列、SPF 系列以及其它系列共 50 多种不同 IC 芯片型号的产品。以 μ'nSP™ 为内核的 16 位单片机也有 SPCExxx 系列、SPT660x 系列、SPMC903 系列共 7～8 种 IC 型号的产品。这些产品既顺应了单片机技术发展的趋势，又充分考虑到市场和用户的需求，具有集成度高、功能全、低功耗、低电压、可靠性高且易于开发等特点。特别是凌阳 16 位单片机提供的某些特殊指令，为其应用添加了 DSP(Digital Signal Processing)功能，且其指令结构提供了对高级语言的支持，这就使凌阳 16 位单片机在目前众多的 16 位单片机中独具特色，必将受到用户的普遍欢迎。

1.3　凌阳 8 位单片机

凌阳 8 位单片机按 CPU 内核不同可分为两大族：一族以 6502CPU 作为其内核，主要有 65N02 系列、65R02 系列以及 65S02 系列；另一族以凌阳 CPU 为其内核，主要有 CPU12 系列和 CPU8 系列。表 1－1 列出了凌阳 8 位单片机系列中 CPU 内核、IC 芯片型号及其各自的主要用途。

从使用的角度讲，表中诸多 IC 芯片按功能又可大致分为 4 种：SPL 系列、SPC 系列、SPF 系列以及其它系列。SPL 系列基本上都带有 LCD 驱动，而某些 SPL 系列还带有发声功能，可用来制作各种款式的计算器、数据库及游戏机等；SPC 系列则是带有双声道发声功能的单片机，可用来制作各种高级电子玩具或电子宠物等；SPF 系列是带有多通道(4 通道、8 通道等)发声功能的单片机，可用来研制各种档次的电子琴类产品。

表 1-1 凌阳的8位单片机系列

CPU内核类型	IC芯片型号	用 途
65N02(6502全套指令)	SPL61A、SPL130A、SPL191A、SPLB10A、SPDC256A、SPDC512A、SPDC512B、SPDC1000A、SPDC1000B	计算器、数据库、游戏机和嵌入式计算机系统等
65R02 (6502减指令系统—部分扩展指令)	SPF02A、SPL02C、SPL02D、SPL03B、SPL03C、SPL05A、SPL05B、SPL06A、SPL06B、SPL128A、SPLB20A、SPLB20A1、SPLB21A、SPLB22A、SPLB23A、SPLB24A、SPLB25A、SPLB26A、SPLG01A	电子琴计算器、数据库、游戏机和嵌入式计算机系统等
65S02(6502减指令系统)	SPF06A1、SPF18A1、SPF20A、SPF30A1、SPF30B、SPL02A	各种档次的电子琴、玩具、计算器、数据库和游戏机等
凌阳CPU12,CPU8 (6502减指令系统+部分扩展指令)	SPCxxx、SPCRxx、SPMCxx、SPFA64A、SPFA128A、SPL08A、SPL08A1、SPL081A、SPL10A、SPL15A、SPL15B、SPL25B、SPL25C、SPL30A、SPL31A、SPL60A、SPL190A	各种档次的电子琴、高级电子玩具、计算器、数据库、游戏机和嵌入式计算机系统等

事实上,由于凌阳8位单片机普遍具有体积小、功耗低、性能好和可靠性高且易于开发等特点,所以,除可用于上述专门用途外,也均可用来研制开发具有特殊功能的各种嵌入式计算机系统。

1.4 凌阳16位单片机

一、概 述

随着单片机功能集成的发展,其应用也逐渐地由单纯的控制扩展为控制处理、数据处理以及数字信号处理(Digital Signal Processing,DSP)等领域。凌阳16位单片机就是为顺应这种发展需求而推出的。

凌阳16位单片机的内核围绕 μ'nSP™ 所形成的16位 μ'nSP™ 系列单片机(或称为 μ'nSP™ 家族),它具有如下技术性能:

* 较宽的电源电压范围:2.6~5.5 V;
* 较宽的系统时钟频率范围:0.375~24.576 MHz;
* 16位数据总线提高了工作速度;
* 大容量的ROM及静态RAM;
* 红外通信接口;
* RS—232通用异步全双工串行接口;
* 10位A/D及D/A;
* 内置式带自动增益控制的扩音机输入通道;
* 32 768 Hz的实时时钟;
* 低电压复位/低电压监测系统。

另外,凌阳16位单片机的某些系列还嵌入了LCD控制/驱动和双音多频发生器功能。

表 1-2 列出了 16 位单片机的各主要系列、IC 芯片型号及其各自的主要用途。

表 1　2　凌阳 16 位单片机一览表

系列类型	IC 型 号	主 要 用 途
SPCExxx	SPCE500A、SPCE060、SPCE061	发声和语音识别
SPT660x	SPT6601、SPT6602	通信领域中带 LCD 驱动的来电识别
SPMC903	处于研发阶段	一般控制领域及教学实验

表中的 SPCExxx 系列具有全双工异步通信的串行接口，可实现多机通信，方便地组成分布式控制系统；红外收发通信接口可用于近距离的双机通信或制作红外遥控装置；A/D、D/A 转换接口可方便地用于各种数据的采集、处理和控制输出；若把 A/D、D/A 转换接口与 μ'nSP™中的 DSP 运算功能配合使用，就可方便地实现语音识别功能，从而使其可应用于语音识别领域。

表中的 SPT660x 系列所嵌入的双音频(Dual Tone Multi Frequency，DTMF)发生器可实现电话拨号功能；内置的 LCD 控制器/驱动器能实现最多 224 点(8COM * 28SEG)的显示；同时，SPT660x 系列还内置了具有 4～8 级自动增益的 ADC 通道及可用于播放乐曲/语音的 DAC 通道，若将其与 μ'nSP™的 DSP 运算功能配合使用，就可方便地实现来电辨识和语音拨号之功能。

表中的 SPMC903 系列是凌阳公司最新推出的，它是一种带多路 A/D 转换且其 ROM 为闪存的微控制器，它与 EPROM 写入器配合使用，非常适合于产品的研制、开发或教学实验等。

不难看出，凌阳 16 位单片机由于采用了模块式集成结构，很容易以 μ'nSP™为内核派生出各种具有专门用途的系列产品，这正是凌阳单片机设计中的一大特色。

二、凌阳 16 位单片机的主要特点

1. 集成度高，易于扩展

凌阳 16 位单片机以 μ'nSP™为内核，内部采用总线结构，把各功能部件模块化地集成在一个芯片里，有效地减少了各功能部件之间的连线，提高了集成度，增强了芯片的可靠性和抗干扰能力。此单片机属于非总线型单片机，省去了片外的地址、数据及控制并行总线，而外部封装引脚可明显减少，使芯片成本下降。它既可通过 I/O 端口与外设器件通信，也可采用串行扩展方式。目前，由于串行传输速度的不断提高，用串行口扩展外围器件，特别在 SOC 中，已得到了较多的应用。

凌阳 16 位单片机采用 CMOS 制造工艺，有较好的低功耗和功耗管理功能，它提供了软件激发的弱振方式、空闲方式和掉电方式三种低功耗工作方式，大大降低了芯片功耗。另外，该单片机工作电压范围大，可在低电压供电时正常工作，并可用电池供电，从而可大幅度地减少器件的功耗。低功耗是一切电子应用系统所追求的目标，特别是在野外作业、便携式仪器仪表的开发等方面具有特殊的意义。

2. 较强的中断处理能力

凌阳 16 位单片机中断系统可支持三种类型的中断：异常中断、事件中断和软件中断。中

断系统共有10个中断向量，可响应14个中断源的中断请求，由于对外界突发事件具有快速反应能力，使其适合用于实时控制领域。

3. 高效的指令系统

凌阳16位单片机的指令系统以字为单位，从而格式紧凑、灵活，执行时间迅速，并提供了对高级语言和数字信号处理运算能力的良好支持，从而可有效缩短基于凌阳单片机的各种产品的研发周期。

同时，凌阳16位单片机片内还集成了大容量的ROM、静态RAM；全双工异步通信的串行接口、红外收发通信接口；A/D、D/A转换接口和多功能的I/O口。其指令系统提供的具有较高运算速度的16位×16位的乘法运算和内积运算指令。这就使凌阳16位单片机可方便地用于复杂的数字信号处理，却又比常用的DSP芯片便宜，因而具有较高的性能价格比。另外，在SPCE061及SPCE060单片机中还采用了Flash ROM技术，有32K Flash ROM单元。Flash ROM的使用使得基于SOC的单片机技术得到进一步发展，极大地改变了单片机应用系统的结构模式、开发与测试手段以及运行条件。

第二章　μ'nSP™单片机的组成原理及功能介绍

本章以 SPCE 系列为例，首先介绍 μ'nSP™单片机的基本结构，然后分别介绍其内核、存储器、系统时钟以及系统维护功能，最后以 SPCE061 为例，介绍 μ'nSP™单片机的引脚功能及最小系统组成。

2.1　μ'nSP™单片机的基本结构及功能

SPCE 系列单片机是以 μ'nSP™为内核的第一个系列产品，也是应用最为典型的产品。目前推出的包括 SPCE 仿真器芯片、SPCE060、SPCE061 和 SPCE500A。下面以 SPCE 为例介绍 μ'nSP™单片机内核的基本结构和功能。

SPCE 系列单片机的结构框图如图 2－1 所示。

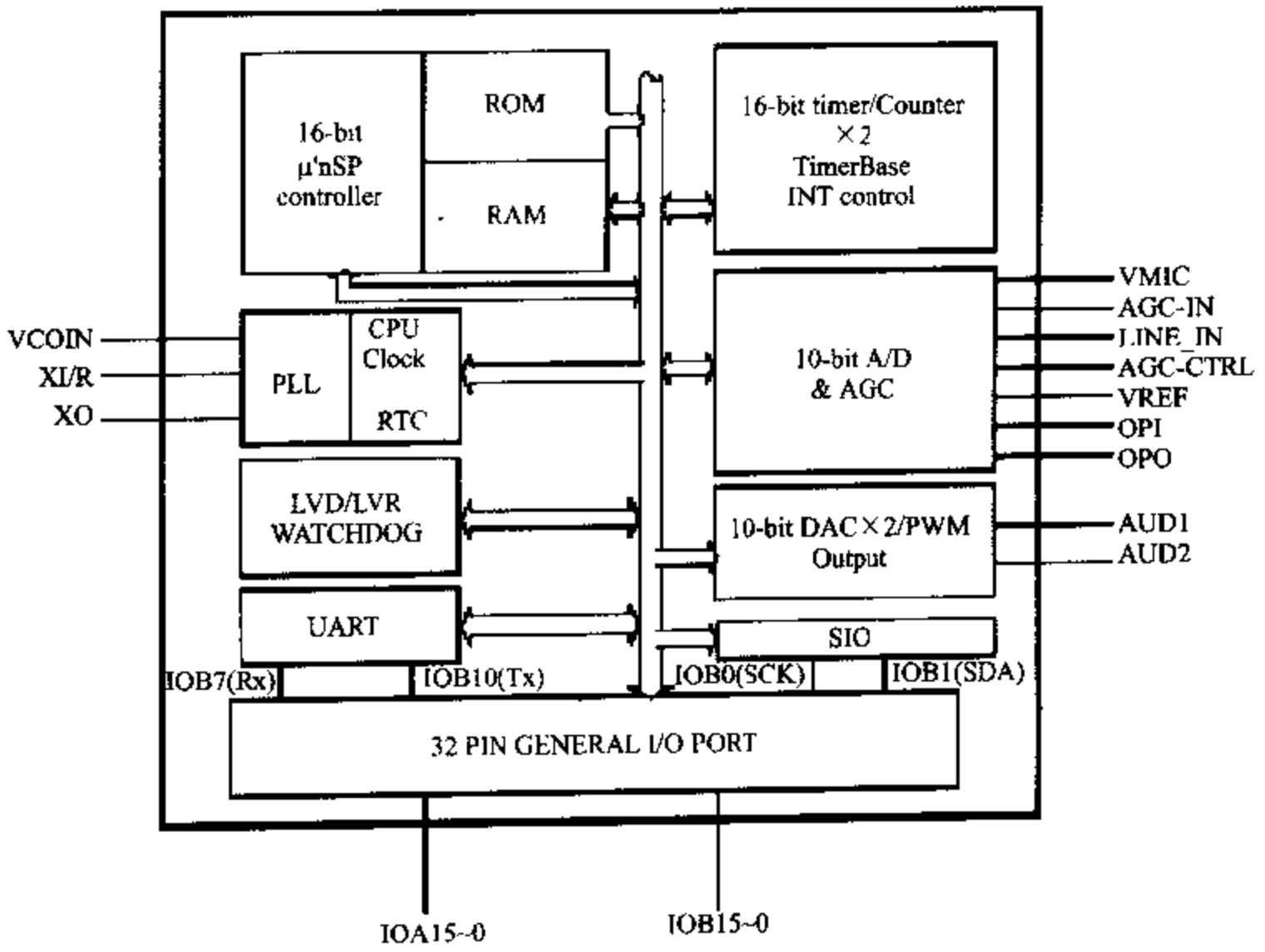


图 2－1　SPCE 系列单片机的总体结构框图

它包含以下主要部件：

- 内置 2 K 字的静态 RAM；

● 内置ROM或闪存ROM：零页中32 KB的快速ROM和非零页中256 KB的常速ROM(对于μ'nSP™单片机而言，最大ROM空间为4 MB，这是由其16位地址线和6位代码/数据段的总线结构所决定的)；

● 32位可编程并行I/O接口；

● 通用异步全双工串行接口UART，具有RS—232标准的发送/接收波形；

● 串行设备接口：可与串行设备进行数据传输；

● 单或8通道10位模/数转换ADC，并在一个通道内置有自动增益控制的扩音器；

● 双通道10位数/模转换DAC方式的音频输出，每个通道的输出能力为3 mA；

● 两个16位可编程定时器.计数器，可自动预置计数初值；

● 两个数字式采样/比较端口；

● 内置32 768 Hz实时时钟；

● 锁相环晶体振荡器或RC振荡器，为系统提供20.480 MHz/24.576 MHz的时钟信号；

SPCE系列单片机的技术特性如下：

● 工作电压范围：2.6～5.5 V；

● 系统工作频率范围：0.375～49.152 MHz；

● 具有红外通信功能：可对红外信号进行接收或发送；

● 14个中断源来自系统时钟、定时器/计数器、时间基准发生器、外部时钟源输入、触键唤醒以及通用异步串行接口等；

● 掉电方式下的系统运行可将功耗降至5 V电源电压下的2 μA；

● 具有运行/睡眠方式下的看门狗维护功能；

● 具有低电压复位/低电压监测功能。

关于其各项功能将在本书后面的相关章节作详细说明。

μ'nSP™单片机的其它系列包括：用于带LCD显示的来电辨识或语音拨号的SPT660系列，用于一般用途并可进行教学实验的SPMC903系列。它们的结构和技术特性与SPCE系列略有不同。有SPCE系列作为基础，其它系列的应用与结构可参考相关的技术手册。

2.2 μ'nSP™单片机的内核结构

图2-2为μ'nSP™的内核结构，它基本是由总线、算术逻辑运算单元、寄存器组和堆栈所组成的。

一、数据总线和地址总线

μ'nSP™具有16位数据线和22位地址线。因此，其基本数据类型是16位的"字"，而22位地址线A0～A21最多可寻访4 M字的存储空间，用于存放指令代码和数据。地址线中的高6位A16～A21来自状态寄存器SR中的6位代码段(CS)或6位数据段(DS)，低16位A0～A15则来自相应的16位寄存器。通常，地址线的高6位成为存储器的页索引码，简称页码(page)；而低16位则称为存储器的地址偏移(offset)。高6位与低16位并在一起组成22位寻址地址线。

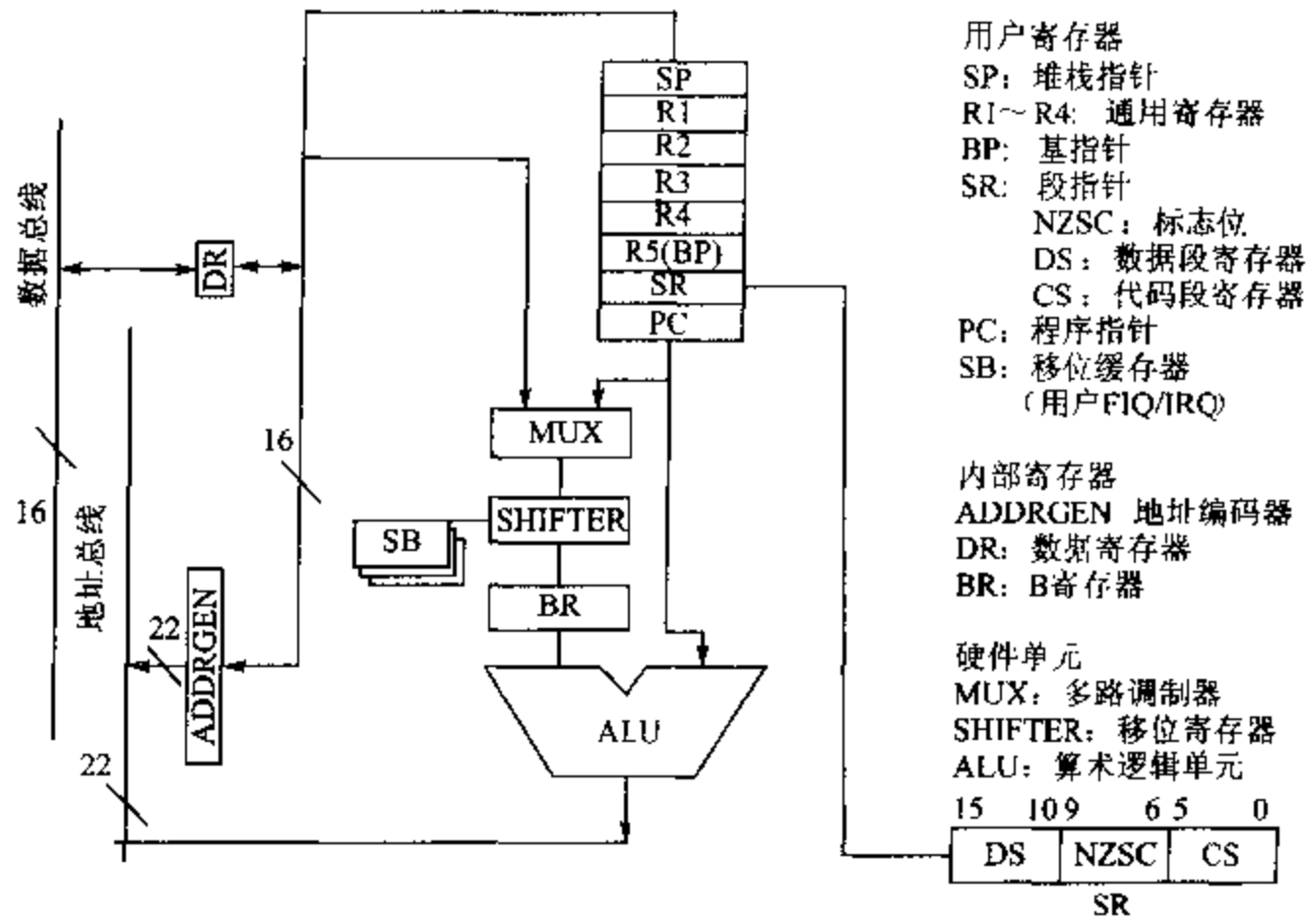


图 2－2　μ'nSP™的内核结构

二、算数逻辑运算单元 ALU

ALU 是运算器的核心部件，它与用户寄存器、内部寄存器及移位器间通过内部总线连接，组成 CPU 的运算部分，完成各种运算功能。μ'nSP™的 ALU 在运算能力上不仅用于 16 位基本的算术逻辑运算，同时也能用于数字信号处理的 16 位×16 位的内积运算。从而能较方便地应用于语言识别等领域。

1. 16 位基本算术逻辑运算

表 2－1 列出了 μ'nSP™能完成的 16 位基本算术逻辑运算内容和相应的助记符。

2. 带移位操作的 16 位算术逻辑运算

μ'nSP™ 的 ALU 前串接一移位器 SHIFTER，这意味着操作数在经过 ALU 的算术逻辑运算前可进行各种移位处理。所以，μ'nSP™的指令系统中专门有一组复合式的“移位逻辑操作”指令。程序设计者可以利用这些复合式指令，编写出更为精练的程序源代码。

3. 16 位×16 位的乘法运算和内积运算

除了普通的 16 位的算术逻辑运算指令外，μ'nSP™的指令系统还提供出速度较高的 16 位×16 位的乘法运算指令 Mul 和内积运算指令 Muls。在 8088 的 CPU 内，16 位×16 位的乘法运算要花费 128～154 个时钟

表 2－1　μ'nSP™的基本算术逻辑运算

助记符	操作功能描述	操作类型
ADD	加	A+B
ADC	带进位加	A+B+C
SUB	减	A+～B+1
SBC	带进位减	A+～B+C
CMP	比　较	A－～B+1
NEG	取　补	～B+1
NOR	异　或	A^B
LOAD	写　入	A=B
OR	或	A | B
AND	与	A & B
TEST	测　试	测试 A,B
STORE	读　出	B=A

周期，具体取决于不同的寻址方式。而 Mul 指令只需花费 12 个时钟周期，Muls 指令花费 $10n+8$ 个时钟周期(其中 n 为乘积求和的项数)。这两条指令为 μ'nSP™单片机应用于复杂的数字信号处理运算方面提供了便利的条件。

三、用户寄存器组

μ'nSP™的 CPU 寄存器组里有 8 个 16 位的用户寄存器，其各自的功能描述在表 2-2 中列出。

表 2-2 CPU 寄存器及其功能

寄存器 ID 号	寄存器符号	功能名称
0(000)	SP	堆栈指针寄存器
1(001)	R1	般寄存器
2(010)	R2	般寄存器
3(011)	R3	一般寄存器
4(100)	R4	般寄存器
5(101)	BP(R5)	基址指针寄存器
6(110)	SR(R6)	状态寄存器
7(111)	PC(R7)	程序指针寄存器

(1) 堆栈指针寄存器 SP(Stack Pointer)：SP 中的内容指示当前堆栈段内的栈顶地址。CPU 执行压栈/出栈指令(push/pop)、子程序调用/返回指令(call/retf)以及进入中断服务子程序 ISR(Interrupt Service Routine)或从 ISR 返回指令(reti)时自动减少/增加，以示堆栈指针的移动。堆栈的最大容量范围限制在 64 K 字的 RAM 内，即地址为 0x000000～0x00FFFF 的存储器中。

(2) 一般寄存器 R1～R4：通常可分别用于数据运算或传送的源及目标寄存器。其中 R3、R4 还可组成一个 32 位的结果寄存器 MR，用于存放乘法运算和内积运算之结果(R4 存放高位字)。

(3) 基址指针寄存器 BP(Base Pointer)：BP 可用来存放当前堆栈段内数据区的基地址偏移量，它可和一个 6 位的立即数组成 22 位数据地址，从堆栈中存取数据。除此之外，BP 也可作为一般用户寄存器 R5 使用，完成与其它一般寄存器相同的工作。

(4) 状态寄存器 SR：SR 的低 6 位和高 6 位分别为代码段选择字段(CS)和数据段选择字段(DS)。它们可分别与 16 位的寄存器合在一起形成 22 位地址码，用来寻址 4M 字容量的存储器。除了调用指令 Call 可改变 CS 值外，对 CS，DS 的赋值只能通过对 SR 的赋值完成。SR 中间的 4 位(B6～B9)为状态标志位。CPU 在执行条件跳转指令(Branch)时需测试这些标志以控制程序的流向。这些标志是：

① 位标志 C：当 C=0 时，表示运算过程中无进位或借位产生；而 C=1 时，表示运算过程中有进位或借位产生。在无符号数运算中，16 位数可以表示的数值范围是 0x0000～0xFFFF，即 0～65 535。如果运算结果大于 0xFFFF，则标志位 C 被置 1。

② 符号标志 S：对于有符号数运算，当 S=0 时，表示实际运算结果非负(包括正数和零)；

当 S=1 时，表示实际运算结果为负数。对于有符号数运算，16 位数所表示的数值范围为 0x8000～0x7FFF，即 -32 768～32 767。若运算结果为负数，则标志位 S 置 1。

③ 零标志 Z：当 Z=0 时，表示运算结果不为 0；当 Z=1 时，表示运算结果为 0。

④ 负标志 N：当 N=0 时，表示运算结果的最高位(B15)为 0；当 N=1 时，表示运算结果的最高位(B15)为 1。

根据标志位 N 和 S，可判断运算结果是否溢出。若标志位 N 和 S 不同，即当 S=0，N=1 或 S=1，N=0 时，则说明有溢出发生。下面举几个例子说明：

```
[例 1]  R1=32 767;      //编译后的 R1=0x7FFF
        R2=32 767;      //编译后的 R2=0x7FFF
        R1+=R2;         //运算后的 R1=0xFFFE
                        //因为运算结果的 B15=1，所以 N=1
                        //因为 R1、R2 都为正，R1+R2 的结果也为正，所以 S=0
                        //当 N=1，S=0 说明有溢出
[例 2]  R1=10;          //编译后的 R1=0x000A
        R2=-15;         //编译后的 R2=0xFFF1
        R1+=R2;         //运算后的 R1=0xFFFB
                        //因为运算结果的 B15=1，所以 N=1
                        //因为 R1+R2 的结果为负，所以 S=1
                        //当 N=1，S=1 说明无溢出
[例 3]  R1=-32 766;     //编译后的 R1=0x8002
        R2=-10;         //编译后的 R2=0xFFF6
        R1+=R2;         //运算后的 R1=0x7FF8
                        //因为运算结果的 B15=0，所以 N=0
                        //因为 R1+R2 的结果为负，所以 S=1
                        //当 N=0，S=1 说明有溢出
```

在运算操作过程中，若目标寄存器是 PC，则所有的标志位均不受影响。对 SR 的赋值仅影响标志位 N 和 Z。

(5) 程序计数器 PC(Program Counter)：它的作用与其它微控制器(MCU)中的 PC 作用相同，是作为程序的地址指针来控制程序流程的专用寄存器。当 CPU 完成一条指令后，PC 会累加这条指令所占的字节数，以指向下一条指令地址。在 μ'nSP™ 中，PC 通常与 SR 寄存器中的 CS 字段共同组成 22 位的程序代码地址。

四、移位器 SHIFTER 和移位缓存器 SB

移位器 SHIFTER 串接在 ALU 前面，可进行多种移位操作，对 ALU 中的算术逻辑运算完成预处理的功能。SHIFTER 在进行移位操作时需要有移位缓存器 SB (Shifter Buffer) 的配合操作，SB 是专用于移位或乘法，除法操作的移位缓存单元，在操作完成后，缓存单元中的值不是固定的，因此用户可不必理会。需要说明的是：SB 实际上是一个三层缓存器，分别对应用户程序、FIQ 程序和 IRQ 程序。CPU 会依据当前程序的运行模式自动进行切换。μ'nSP™ 指令系统中设置有一组能实现“移位算术逻辑操作”的指令，程序员可利用此类指令编写出更

为精练的程序指令。

2.3 μ'nSP™的片内存储器结构

μ'nSP™的片内存储器地址映射如图2-3所示。4 M字的存储器地址可映射成64页，每一页有64 K字的存储容量，页内地址取决于相关16位寄存器或存储器的值。页的地址映射值0x00～0x3F由地址线A16～A21编址形成。而A16～A21的编址即页的选择，通过对SR寄存器中6位的代码段选择字段(CS或数据段选择字段DS的赋值完成。

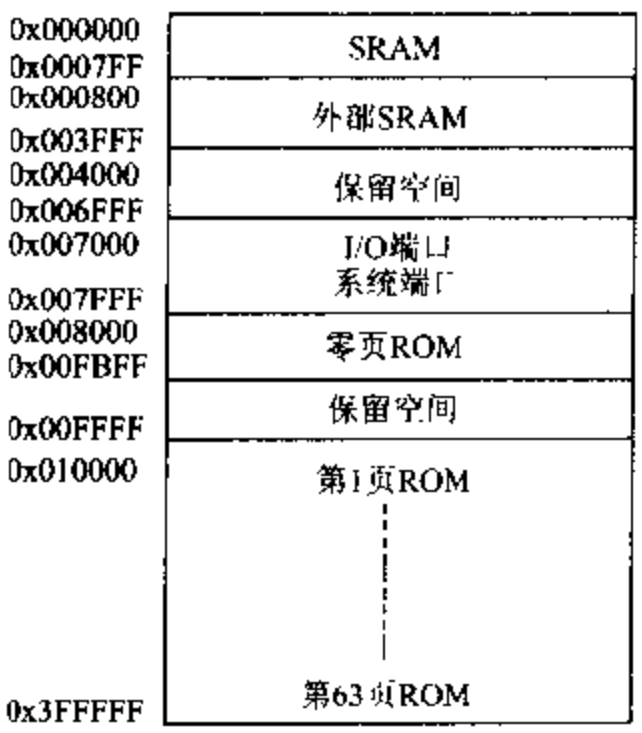


图2-3 μ'nSP™片内存储器地址映射

存储器映射中的首页(即零页)的地址映射值为0x00。零页的专门设计用于需要频繁访问的数据存储单元，例如用户定义的变量存储器或一些外部设备接口等。其它非零页是由高密度巨大容量的掩膜ROM设计制成，用于存储程序或固定的数据代码。整个片内存储器可分成静态数据存储器SRAM和程序存储器ROM两块区域，采用统一编址方式。

一、静态数据存储器SRAM

SRAM的地址区域在000000H～003FFFH范围内。前64 K个字，即000000H～00003FH地址范围内的存储器寻访速度为两个时钟周期；其余000040H～003FFF地址范围内(2 KB)存储器的寻访速度则为三个时钟周期。

此外，μ'nSP™单片机还可扩展14 KB的外部SRAM(000800H～003FFFH)

二、程序存储器ROM

μ'nSP™中的ROM具有两种寻访速度。零页中地址范围为008000H～00FFFFH的32 KB ROM(其中00FC00H～00FFF4H为系统保留空间，00FFF5～00FFFF存放中断向量)，其读写寻访速度为三个时钟周期。其余非零页中的ROM，即地址在010000H～3FFFFFH时，当CPU要读出其中的数据时亦为三个时钟周期；但要对其写入数据时会在三个写周期里的每个时钟周期上再加一个时钟周期，形成双时钟周期的写周期。故非零页ROM的整个写操作要花费六个时钟周期。

三、堆　栈

μ'nSP™的堆栈遵循后进先出的原则，其结构如图2-4所示。

堆栈由上而下生成，栈底为高地址，栈顶为低地址。堆栈指针SP总是指向位于栈顶的第一个空项。在压入(push)一个字数据后SP减1。将多个寄存器压栈写入时总是让指令中序号n最高的寄存器先入栈，直至序号最低的寄存器最后入栈(序号n由低到高为1～7)。因

此，执行指令 push r1，r4 to [sp] 与指令 push r4，r1 to [sp] 是等效的。

弹栈(pop)操作前，SP 总是指向栈顶的第一个空项，在弹栈拷贝数据之前 SP 要加1，且总是将先弹出拷贝的数据置入指令中序号最低的寄存器，直至最后一个拷贝数据置入序号最高的寄存器为止。

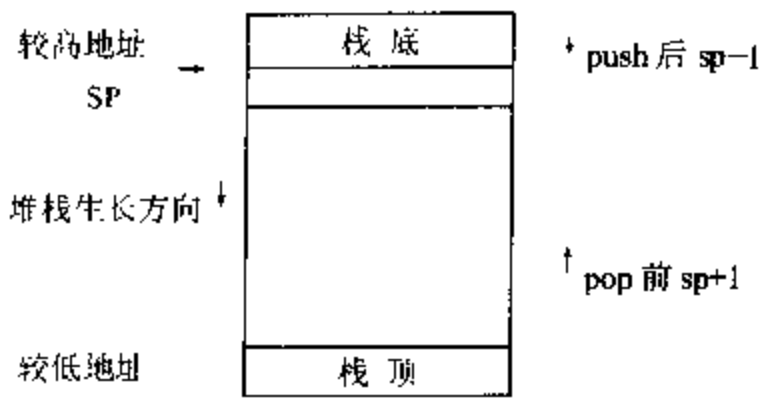


图 2-4　μ'nSP™的堆栈结构

另外，存储器空间的 007000H ～ 007FFFH 段为 I/O 端口与系统端口保留空间，其具体配置与功能将在各相关部分详细介绍。

2.4　系统时钟与低功耗工作方式

一、系统的时钟

μ'nSP™的系统时钟信号 f_{osc} 和 CPU 工作时钟信号 CPUCLK 均来自其时钟系统。

时钟系统基本上由三部分组成：锁相环晶体振荡器 PLL、可编程分频计数器以及时基发生器。通过 PLL 对外接 32 768 Hz 晶振信号进行倍频处理，产生出 f_{osc} 信号，作为系统的时钟源。f_{osc} 信号经过分频产生出 CPUCLK 信号；控制寄存器 P_SystemClock 用来控制 f_{osc} 的频率选择及 CPUCLK 的分频倍数。同时，32 768 Hz 经时基发生器的分频处理，为定时器/计数器提供时钟源信号并为中断系统提供不同的时基中断源信号。系统时钟的结构如图 2-5 所示。

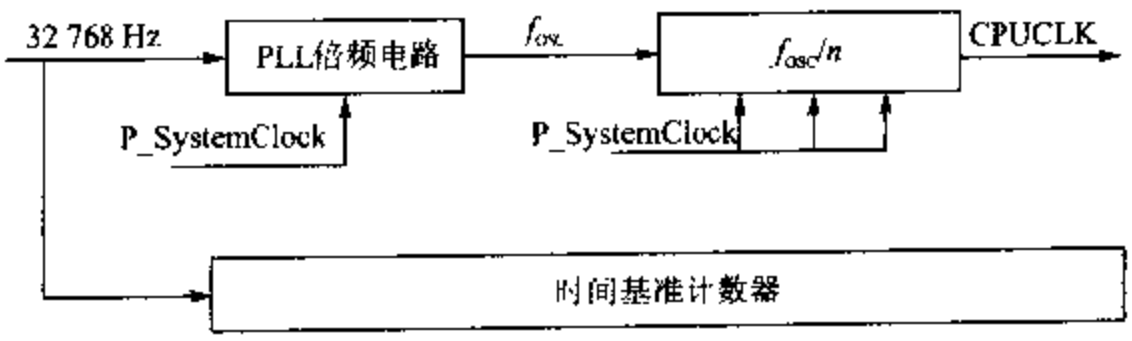


图 2-5　μ'nSP™时钟系统的结构

二、系统的低功耗工作方式

通常，μ'nSP™可以在两种状态下运行：唤醒状态和睡眠状态。所谓唤醒状态就是指μ'nSP™的 CPU 处在工作状态下，通过执行指令来指挥整个系统的动作。而睡眠状态则是指 CPU 处在休息状态，停止执行任何指令。系统靠外设的触发来结束 CPU 的"休眠"，使其重新进入工作状态。这里，"外设的触发"泛指唤醒源，譬如键唤醒和实时时钟定时唤醒等。

μ'nSP™支持三种低功耗工作方式，即弱振方式、空闲方式和掉电方式。这三种方式都是靠指令控制完成的。

弱振方式是 μ'nSP™运行在唤醒状态时振荡源自动进入弱振工作状态而降低系统的功耗。

空闲方式则是除了保持一些片内外设和外部中断源功能外，将关闭所有处理器功能，但振荡源仍在工作。任何一个中断源产生的中断都将使 μ'nSP™ 恢复到进入空闲方式前的状态运行。

掉电方式则将进一步关闭振荡源和受其影响的其它部件，使 μ'nSP™ 的功耗减至极小。

空闲方式和掉电方式均使 μ'nSP™ 运行在睡眠状态下的低功耗方式。

关于时钟系统的配置及 μ'nSP™ 工作方式的控制指令将在后续章节中做详细讨论。

2.5 系统维护

为了增加整个单片机系统的安全性和可靠性，μ'nSP™ 单片机设置了两项系统维护功能：程序维护和系统电源维护。前者可通过看门狗计数器实现；后者则通过低电压复位/监测电路实现。二者均可通过 μ'nSP™ 系列产品定制的屏蔽任选设置或取消来实现。

一、看门狗计数器(WatchDog)功能

设置 WatchDog 功能可监视系统的程序运行有无异常，并在异常时自动采取恢复措施。如果 WatchDog 的计数在固定的时间间隔内被程序指令不断地清除而不产生溢出，表明程序运行正常。但若在规定的时间间隔里 WatchDog 的计数未被清除而产生溢出，则表示程序运行异常，CPU 对此会进行系统复位处理，即重新运行程序而使系统恢复初始状态，以防止系统程序的“跑飞”。

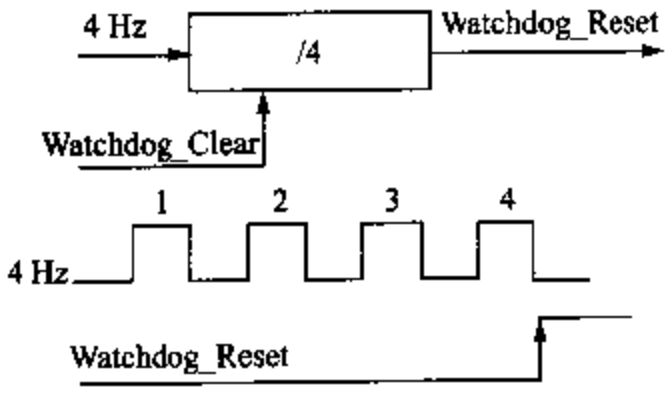


图 2-6 WatchDog 的结构和信号时序

μ'nSP™ 单片机的 WatchDog 的清除时间周期为 0.75 s。因为 WatchDog 的溢出复位信号 WatchDog_Reset 是由 4Hz 时基信号经四分频之后产生的，即每四个 4 Hz 时基信号(下降沿记数)将会产生一个 WatchDog_Reset 信号，如图 2-6 所示。而清除 WatchDog 的 WatchDog_Clear 信号却可以发生在 4 Hz 信号(0.25 s)之间的任意一个时刻点上。假如 WatchDog_Clear 信号发生在第四个 4 Hz 信号尾端的 0.01 s 即第 0.25 s 时刻，此时虽然 WatchDog 被清掉，但由于它发生在第四个脉冲的下降沿之前，再经三个 4 Hz 信号即 0.75 s(即累计 4 个脉冲下降沿)后，如果一直没有 WatchDog_Clear 信号，便会产生出一个 WatchDog_Reset 信号。

当然，如果 WatchDog_Clear 信号发生在 4 Hz 信号始端的 0.01 s，则经过 0.99 s 后若无 WatchDog_Clear 信号便会产生 WatchDog_Reset 信号。因此，清除 WatchDog 的时间周期为 0.75 s。

表 2-3 列出了 WatchDog 配置单元 P_WatchDog_Clear 及其内 B0 对 WatchDog 清除的控制。清除 WatchDog 只需写入 P_WatchDog_Clear 单元‘0001’即可。

二、低电压复位/低电压监测(LVR/LVD)功能

LVR/LVD 的结构如图 2-7 所示。

表 2-3 WatchDog 的配置及 WatchDog 的清除

配置单元	读写属性	存储地址	配置单元功能说明
P_WatchDog_Clear	写	7012H	清除 WatchDog 单元
B15~B1	B0	控制位功能解释	
	WatchDog_Clear		
0~0	0	不清除 WatchDog	
0~0	1	清除 WatchDog	

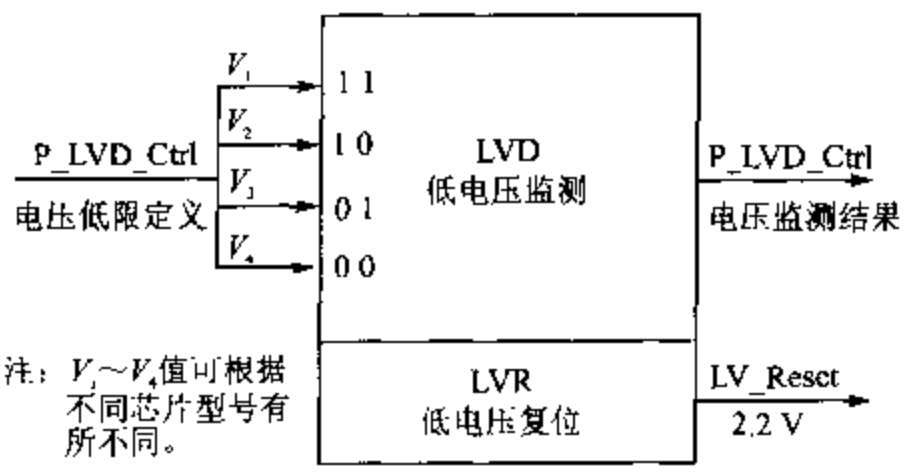


图 2-7 LVR/LVD 的结构

(1) LVR 功能：电源电压过低时可能会引起系统运行暂停。通常，这是由于电池电压在系统上电过程中的反跳或电池能量不足所致。此时，若设置了 LVR 功能，便会在四个时钟周期之后产生一个复位信号 LV_Reset，使系统复位。

(2) LVD 功能：此功能用来监测和报告系统电源电压的工作情况。通过对 P_LVD_Ctrl 单元的编程写入可以控制电压监测的低限。

表 2-4 列出了在 LVD 的配置 P_LVD_Ctrl 单元里，写入的定义电压监测了低限的控制字及读出的电压监测了结果的特征值。

表 2-4 低电压监测的配置及控制

配置单元	读写属性	存储地址	配置单元功能说明
P_LVD_Ctrl	读/写	7019H	写入选择电压监测之低限的控制字，读出电压监测结果之特征值

P_LVD_Ctrl 单元写入的控制字及读出的特征值				
B15(读)	B14~B2	B1(写)	B0(写)	控制字或特征值说明
X	X~X	0	0	选择 V_4 为电压监测低限
X	X~X	0	1	选择 V_3 为电压监测低限
X	X~X	1	0	选择 V_2 为电压监测低限
X	X~X	1	1	选择 V_1 为电压监测低限
0	X~X	X	X	读出'0'表示系统电源电压正常
1	X~X	X	X	读出'1'表示系统电源电压越过低限

注：X 表示该处的值既可为"1"又可为"0"。

由图2-7可以看出，LVD共有四级电压低限供选择。假设LVD的低限设为最低一级V_1，那么，当系统的电源电压低于V_1时，P_LVD_Ctrl单元的B15将读出'1'，编程者据此可编写电源电压超常处理的程序。

2.6 单片机引脚及最小系统

μ'nSP™单片机是基于SOC的产品，用户可根据需要选择相应系列中的单片机，不同系列的单片机其引脚排列也不尽相同。这里以SPCE061为例，介绍其引脚功能及排列，以使读者对μ'nSP™单片机有更为感性的认识。SPCE061的引脚排列如图2-8所示。

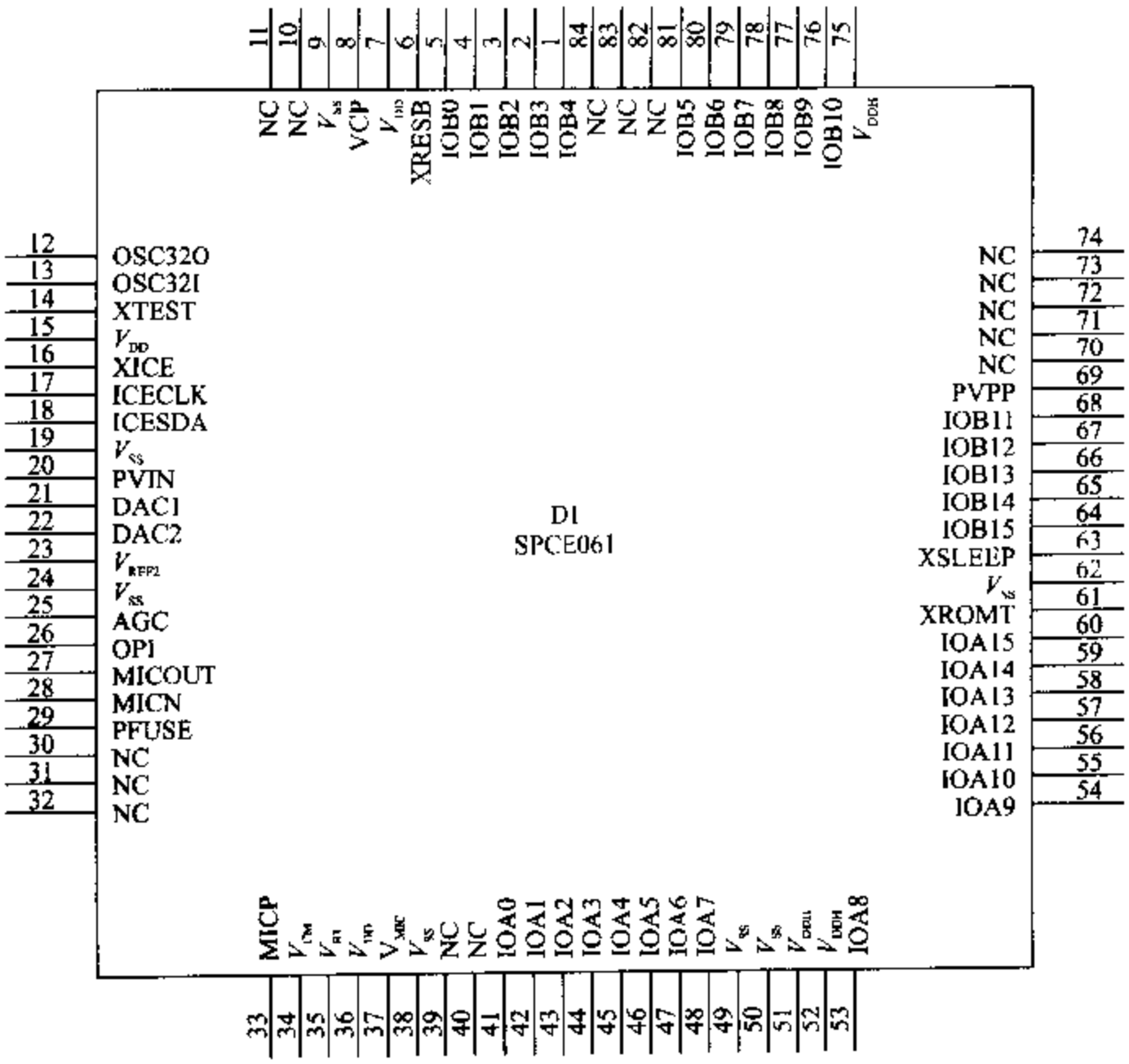


图2-8 SPCE061的引脚排列

SPCE061A共有84个引脚，各引脚功能说明如下：

IOA0～IOA15：I/O口A共16个；

IOB0～IOB15：I/O口B共16个；

OSC32I：振荡器输入。在石英晶振模式下是石英元件的一个输入脚，而在阻容模式下RC是振荡器的输入脚；

OSC32O：振荡器输出。在石英晶振模式下是石英元件的一个输入脚或是外部脉冲的输入脚；

XRESB:复位输入。若这个脚输入低电平时会使控制器被重置复位;

ICE_EN:ICE 使能端;

ICE_SCK:ICE 时钟端;

ICE_SDA:ICE 数据端;

PVIN:程序保密设定端;

PFUSE:程序保密设定端;

DAC1:音频输出通道 1;

DAC2:音频输出通道 2;

V_{REF2}:2 V 参考电压输出端;

AGC:语音输入自动增益控制端;

OPI:Microphone 的第二运放输入端;

MICOUT:Microphone 的第一运放输出端;

MICN:Microphone 的正向输入端;

MICP:Microphone 的负向输入端;

V_{RI}:外部 A/D 最高参考电压,即 A/D 转换输入电压上限值输入端;该端输入一个 5 V 的参考电压,则 A/D 转换电压输入范围为 0～5 V;

V_{CM}:ADC 参考电压输出端;

V_{MIC}:Microphone 电源;

SLEEP:睡眠状态指示端。当 CPU 进入睡眠状态时,该端输出一个高电平;

V_{DDH}:I/O 电平参考。该端输入一个 5 V 的参考电压,则 I/O 输入输出高电平为 5 V;

V_{DD}(第 7 端):锁相环电源;

V_{SS}(第 9 端):PLL 地;

V_{DD}(第 15 端):数字电源;

V_{SS}(第 24 端):模拟地;

V_{SS}(第 38 端、第 49 端、第 50 端、第 62 端):数字地;

V_{DD}(第 15 端、第 36 端):数字电源;

此外,还有 15 个引端悬空未用。

用户可充分利用片内资源,组成所需系统。SPCE061 的最小系统接线如图 2-9 所示。

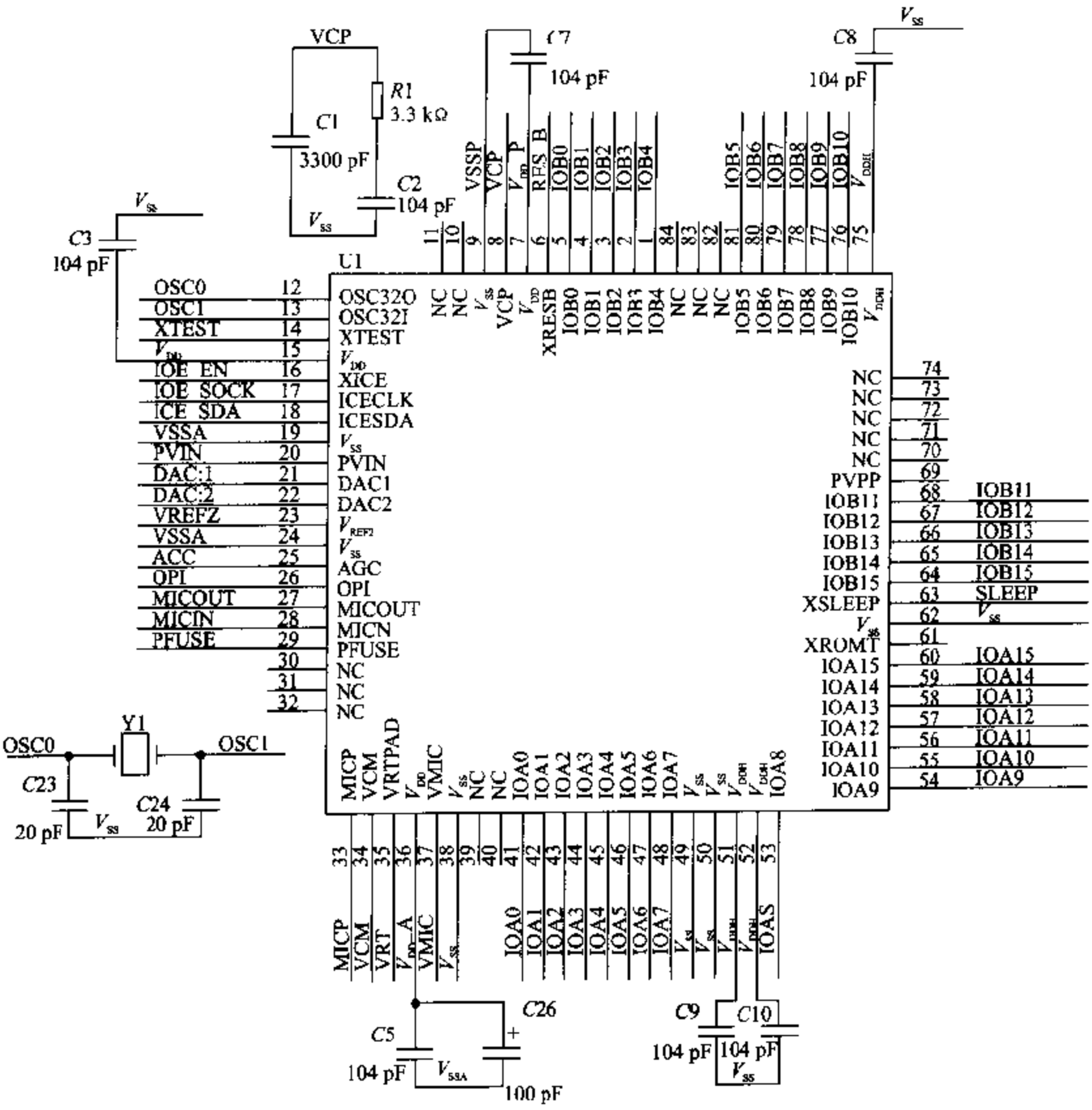


图 2·9 SPCE016 最小应用系统

第三章　寻址方式和指令系统

计算机是按照顺序一条条依次执行指令而工作的。通常将完成某项特定任务的指令的序列称为程序。用户要计算机完成各项任务，就要设计各种应用程序。程序设计语言有三种：机器语言、汇编语言和高级语言。机器语言是计算机惟一能识别的语言，而用汇编语言或高级语言编写的程序称为源程序，但源程序最终都必须翻译成机器语言，这样的程序称为目标程序。该目标程序，计算机才能识别，然后逐一执行。但是，机器语言只是一种用二进制数 0、1 组成的代码，人们不易辨识、记忆，因此使用不便、易错，很难用它来进行程序设计。

高级语言是面向问题和计算过程的语言，它通用于各种不同的计算机，用户编程时不必仔细了解所用的计算机的具体性能与指令系统；高级语言的语句功能强，常常一个语句相当于很多条计算机指令。用高级语言编制程序的速度比较快、也便于学习和交流，所以使用很多。目前，大多数单片机仿真开发装置都支持高级语言，而 μ'nSP™ 系统开发软件工具 ANSI－C 编译器，可以用 C 语言编写源程序文件，经预处理后就可生成汇编程序。

当编制程序工作量不大、规模较小时使用汇编语言编程比较方便。用汇编语言编制程序时，程序的每一条语句都与计算机的某一条具体指令相对应，因此必须熟悉机器的指令系统。有经验的程序员用汇编语言编出的程序其质量优于用高级语言编出的程序。根据统计，高级语言译成机器语言后，一般长度要增加 15 %～200 %，占用的内存空间将扩大，程序执行时间也相应增长。因此，对于工控、检测等实时控制系统以及一些灵活小巧的智能产品，采用汇编语言编程，其优越性比较明显。故汇编语言程序设计有其特定的应用范围，应用仍相当广泛。

应用汇编语言编程，必须掌握其指令系统。μ'nSP™ 的指令系统提供了多种寻址方式的指令集，主要包括了数据传送类指令、算术运算类指令、逻辑操作类指令、程序转移及设置类指令等。μ'nSP™ 指令系统的格式结构紧凑，以字为单位，指令执行速度快，并提供了对高级语言的支持和对数字信号处理 DSP(Digital Signal Processing)运算的支持，功能更加强大。

3.1　指令的格式与分类

在介绍指令格式及指令系统以前，把描述指令等一些符号的意义先做如下简单的介绍。

Rd：	目的寄存器或目的存储器指针；
Rs：	源寄存器或源存储器指针；
IM6、IM16：	6 位、16 位立即数；
A6、A16：	存储器 6 位、16 位地址常量；
Rx～Ry：	序列寄存器，x、y 为序号，1 为最低，7 为最高；
MR：	由 R4、R3 组成的 32 位结果寄存器(R4 为高字组，R3 为低字组)；
＋，－，＊，～：	加、减、乘、求补操作符记号；
&，｜，^，：	逻辑与、或、异或操作符记号；
＋＋、－－：	指针单位字增量、减量操作符；

[ ]：　寄存器间接寻址标志；

{ }：　任选项；

C：　进位位 Carry；

D：　非零页数据段寻址标志；

ss、us：　有符号数与有符号数之间、无符号数与有符号数之间的运算关系；

nn：　寄存器移位位数；

n：　内积运算项数；

FIR：　有限冲击响应(Finite Impulse Response)算法，指数字信号处理中的一种具有线性相位及任意幅度特性的数字滤波器算法；

//：　注释符；

#：　算术或逻辑操作符；

( )：　寄存器或存储单元中的数据。

一、指令的格式

汇编语言经编译后将形成机器语言——机器码。μ'nSP™的机器码有5种格式，根据指令的操作数数目又可分为0、1、2和3四种格式。在介绍指令格式以前，先对指令格式中常用到的符号说明如下：

OP——　操作类型域。用来指明指令的功能、寻址方式及操作类型的域。

OPD　　操作数域。根据不同的操作类型或寻址方式，操作数有寄存器、立即数和地址偏移量等。

OPDE——　操作数扩展域。根据不同的寻址方式，扩展操作数有16位立即数和16位地址偏移量等。

COND——条件码域。JUM P转移指令中的各条件码。

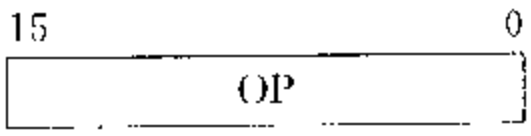
FL——标志域。用于标识操作属性的各种标志(D、@、S、F、I，详见后面各节)。

RG——　范围域。用于标识操作范围的域(如SIZE为序列号，*nn*为移位位数等)。

μ'nSP™指令的组成格式不是以字节为单位，而是以16位的字为单位。有单字指令和双字指令两种格式，结构较为紧凑。下面按操作数数目介绍4种指令格式的具体形式。

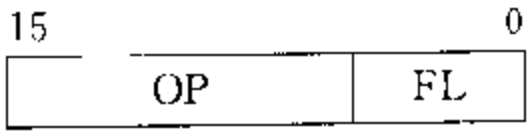
(1) 0操作数指令：0操作数指令为单字指令，有两种格式：

①



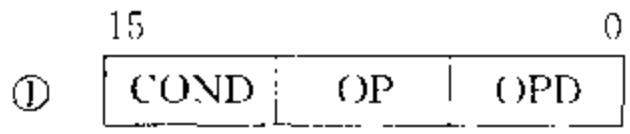
这种指令只有操作类型域OP，如RETF、RETI、NOP、BREAK、FIR－MOV ON和FIR－MOV OFF指令等。

②

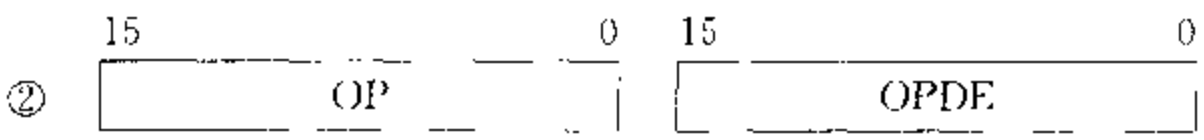


这种指令具有操作类型域OP和用于标识操作属性的标志FL。如FIQ ON、FIQ ON、IRQ ON、IRQ OFF和INT指令等。

(2) 1操作数指令：1操作数指令有单字指令和双字指令两种格式：

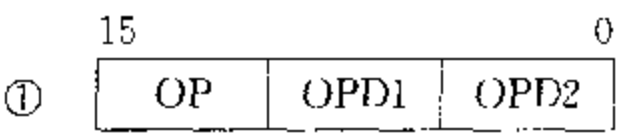


这种指令为单字指令，具有条件码 COND、操作类型域 OP 和一个操作数 OPD。如 Jcond Label、jmp Label 指令等。

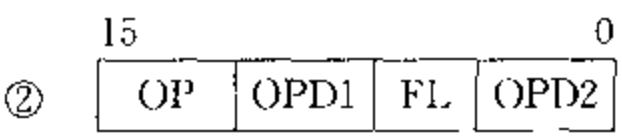


这种指令为双字指令，其中一个字为操作类型域 OP，另一个字为扩展操作数 OPDE。如 Rd=Rs#IM16、Rd=Rs#[A16]指令等。

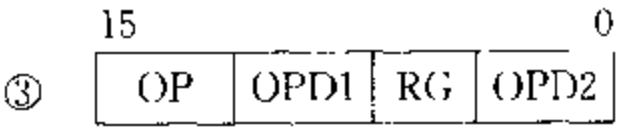
(3) 2 操作数指令：2 操作数指令有单字指令和双字指令之分，而单字指令又有四种格式：



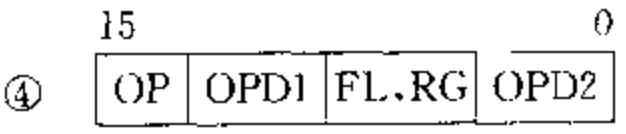
这种指令具有操作类型域 OP 和两个操作数 OPD1、OPD2。如 Rd#=IM6、Rd#=[A6]、Rd#=[BP+IM6]指令等。



这种指令具有操作类型域 OP、操作属性标志 FL 和两个操作数 OPD1、OPD2。如 MR=Rd*Rs{,ss}、MR=Rd*Rs,us 指令等。

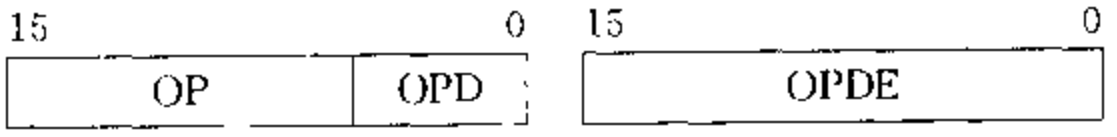


这种指令具有操作类型域 OP、操作范围 RG 和两个操作数 OPD1、OPD2。如 PUSH Rx,Ry to [Rs]、POP Rx,Ry from [Rs]指令等。

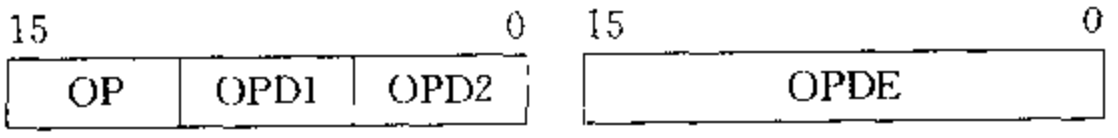


这种指令具有操作类型域 OP、操作属性标志 FL、操作范围 RG 和两个操作数 OPD1、OPD2。如 MR=[Rd]*[Rs]{,ss}{,n}、MR=[Rd]*[Rs],us{,n}指令等。

以下指令为双字指令，具有操作类型域 OP、一个操作数 OPD 和一个扩展操作数 OPDE。如 GOTO Label 指令等。



(4) 3 操作数指令：3 操作数指令为双字指令，只有一种格式：



这种指令具有操作类型域 OP，两个操作数 OPD1、OPD2 和一个扩展操作数 OPDE。如 Rd=Rs#IM16、Rd=Rs#[A16]指令等。

上述4种指令格式中，操作数的位置取决于寻址方式(见下一节)。

二、指令的分类

μ'nSP™的指令总共可分为4大类：

(1) 数据传送类；

(2) 算术运算类；

(3) 逻辑操作类；

(4) 控制转移及设置类。

这4大类指令中共有6种寻址方式，下节将予以简要介绍。

为了记忆、书写等方便，各种单片机厂家对指令系统的每一条指令常用能描述指令功能的符号(助记符)来表示指令的操作码，也可用符号来表示操作数。一般助记符都用英文的缩写来描述指令的特征，不但便于记忆，也便于理解和分类。但在学习μ'nSP™的指令系统时须注意，μ'nSP™指令系统比一般单片机指令系统更接近C语言，也就是说有些指令有助记符，有些指令无助记符，如指令R1=0x2000，执行的操作是将立即数0x2000传送到寄存器R1中，并没有特定助记符。

3.2 寻址方式

计算机传送数据，执行算术运算和逻辑操作等都要涉及到操作数。一条指令的执行，先要从操作数所在地址寻找到与本指令有关的操作数，如何得到这个操作数存放的地址就称为寻址方式。一般来说，寻址方式越多，灵活性越大，指令系统将越有效，计算机的功能也随之越强。μ'nSP™的指令系统共有六种基本寻址方式，分述如下。

一、立即数寻址

在这种寻址方式中，指令格式中直接包含了操作数，可以立即参与指令所规定的操作，故此操作数即被称为立即数。立即数有两种：6位立即数(范围为0x00～0x3F)和16位立即数(范围为0x0000～0xFFFF)，分别用IM6和IM16表示。

如指令R1=0x25，其中0x25就是立即数。这一指令的功能是将6位立即数0x25传送到寄存器R1中。如指令R1=R1+0x2400，这一指令的功能是将16位立即数0x2400与寄存器R1中的内容相加，将结果回存到寄存器R1中。

二、寄存器寻址

在这种寻址方式中，操作数存放在某工作寄存器中，也叫寄存器直接寻址。

如指令R2=R1+0x2400，这一指令是3操作数指令，一个操作数存放在R1中，另一个操作数存放在R2中，这两个操作数都属于寄存器寻址，第三个操作数为立即数0x2400。该指令的功能是把寄存器R1中的内容与立即数0x2400相加，将结果存放到寄存器R2中。

三、直接寻址

直接寻址就是在指令格式中直接给出了存放操作数的存储器地址，以供寻址取数或存放

数据。在书写指令时，直接寻址地址需用方括号“[ ]”括起来，以表明是存储器操作数。直接寻址有 3 种方式。

1. 6 位地址直接寻址

在这种寻址方式中，指令中给出的是 6 位有效地址[A6]，指令周期短。由于仅给出 6 位有效地址，只能用于零页存储器前 64 个字存储单元范围内寻址。

如指令 R1＋＝[0x30]，则[0x30]就是 6 位地址直接寻址。该指令的功能是将寄存器 R1 中的内容与零页存储器 0x30 单元中的内容相加，再将和存放到寄存器 R1 中。

2. 16 位地址直接寻址

在这种寻址方式中，指令中给出的是 16 位有效地址[A16]，指令周期较 6 位地址直接寻址长。由于给出的有效地址是 16 位，只能用于在零页存储器 64 K 字存储单元范围内寻址。

如指令 R2＝R1＋[0x3000]，则[0x3000]就是 16 位地址直接寻址。该指令的功能是将寄存器 R1 中的内容与零页存储器 0x3000 单元中的内容相加，将和存放到寄存器 R2 中，执行操作过程如图 3－1 所示。图中 0x3000 为存储器单元地址，0x0002 为 0x3000 单元中的数据，而 R1、R2 则属于前述寄存器寻址。

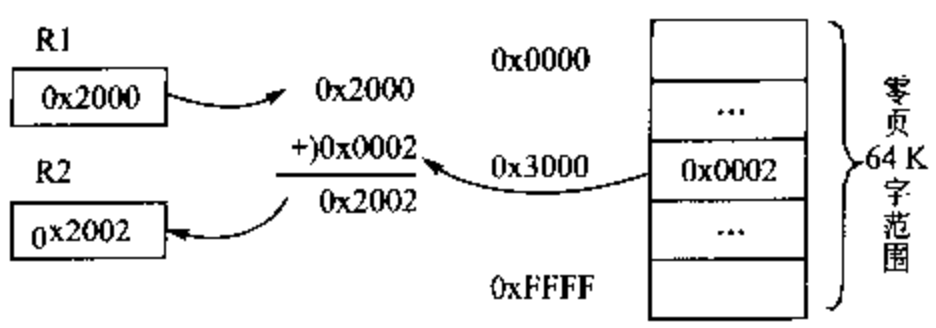


图 3－1　R2＝R1＋0x3000 指令 16 位地址直接寻址操作示意

3. 22 位地址直接寻址

在这种寻址方式中，指令中给出的是 22 位有效地址[A22]，它由高 6 位地址（CS 代码段）和页内 16 位地址拼接而成。由于给出的有效地址是 22 位，故可在 64 页代码段 4 MB 的存储单元范围内寻址。操作过程如图 3－2 所示。此类指令执行周期较长。

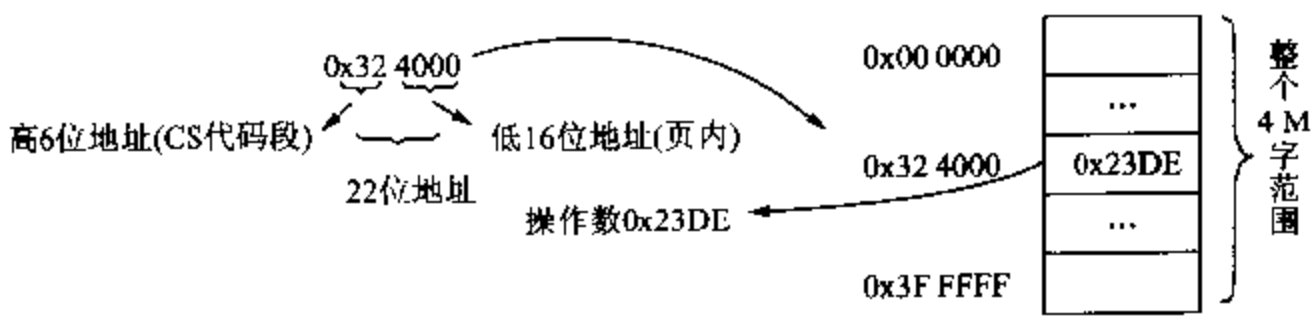


图 3－2　22 位地址直接寻址操作示意

这种寻址方式主要用于子程序的调用和返回。如指令 Call sub_1 指令，执行的功能是调用子程序 sub_1。

四、寄存器间接寻址

寄存器间接寻址就是在指令格式中指定某一寄存器，其中存放的是操作数地址的地址，即操作数的地址是由寄存器间接提供的。由于操作数是存放在存储单元中，故在书写指令时，所用寄存器需用方括号“[ ]”括起来，其格式为{D:}[R]。当省略 D 时，R 指向零页存储单元；当

D不省略时，由R和段寄存器SR中的数据段DS共同指向存储单元，其中DS的值决定了存储器的页码。这是μ'nSP™指令系统中惟一可在零页又可在非零页数据段寻址的寻址方式。寄存器间接寻址有以下5种形式。

1. 寄存器间接寻址

此时寄存器的内容为操作数存放的地址，指令执行后寄存器的内容不变。

如指令 R1＝[R5]。若 R5 的内容是 0x2000，则该指令执行的操作是将零页存储器 0x2000 单元的内容传送到寄存器 R1 中，指令执行后 R1 中的内容为 0x2A78，而 R5 的内容不变，操作过程如图 3－3 所示。

2. 寄存器前置增量间接寻址

按这种寻址方式指令执行时，先将寄存器内容加1，然后再以寄存器的内容为操作数存放的地址，故指令执行后，寄存器的内容发生了＋1的变化。

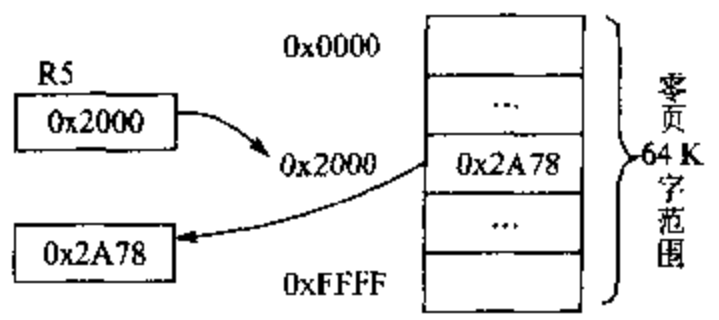


图 3－3　R1＝[R5]指令寄存器间接寻址操作示意

如指令 R1＝D:[＋＋R5]。若 R5 的内容是 0x2008，则该指令执行的操作是先将 R5 的内容加1，然后再将 DS 数据段所指向的页存储器 0x2009 单元的内容 0x380E 传送到寄存器 R1 中，指令执行后 R5 的内容为 0x2009，操作过程如图 3－4 所示。

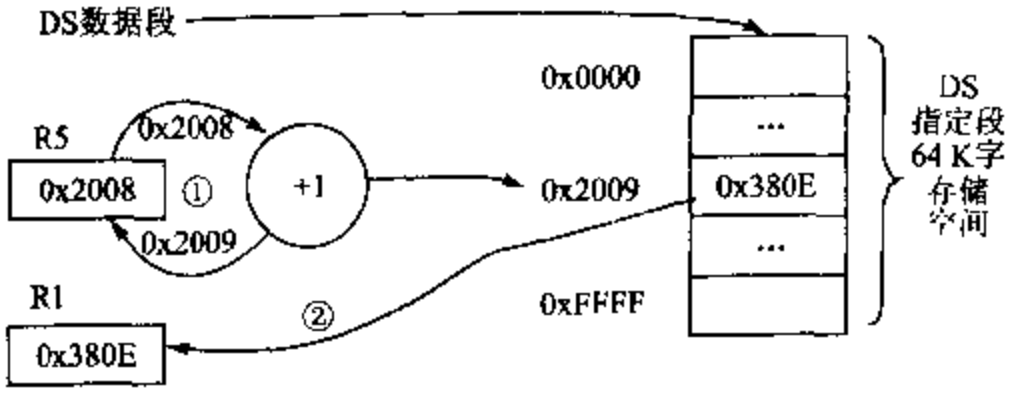


图 3－4　R1＝D:[＋＋R5]寄存器间接寻址操作示意

3. 寄存器后置增量间接寻址

按这种寻址方式指令执行时，先以寄存器的内容作为操作数存放的地址，寻找或存放操作数，执行指令规定的操作，然后再将寄存器内容加1。指令执行后，寄存器的内容同样地发生了＋1的变化。

如指令 R1＝D:[R5＋＋]。若 R5 的内容是 0x200，则该指令执行的操作是先将 DS 数据段所指向的页存储器 0x200 单元的内容传送到寄存器 R1 中，然后再将 R5 的内容加1，指令执行后 R5 的内容为 0x201。

这种寻址方式在程序设计中可方便地对一维数组元素进行检索操作，指令执行后自动指向下一个数的存储地址，为取下一个数据作准备。

4. 寄存器后置减量间接寻址

按这种寻址方式执行指令时，先将寄存器的内容作为操作数存放的地址，寻找或存放操作数，执行指令规定的操作，然后再将寄存器内容减1。指令执行后，寄存器的内容发生了－1的变化。

如指令 R1＝D:[R5－－]。若 R5 的内容是 0x200，则该指令执行的操作是先将 DS 数据

段所指向的页存储器0x200单元的内容传送到寄存器R1中，然后再将R5的内容减1，指令执行后R5的内容为0x1FF。

5. 寄存器自动增减量间接寻址

按这种寻址方式执行指令时，先将寄存器的内容作为操作数存放的地址，寻找或存放操作数，执行指令规定的操作，然后再将寄存器内容自动增1或减1。指令执行后，寄存器的内容发生了±1的变化。这种寻址方式主要用于堆栈的操作。

如指令PUSH R1，R3 to [SP]。这条指令执行的操作是将寄存器R3、R2、R1的内容依次压入堆栈指针SP所指出的存储单元，入栈过程中，SP自动减1。

五、变址寻址

在这种寻址方式中，先对基址指针寄存器BP和6位立即数求和，即BP+IM6，算出有效地址，然后再从该地址处取操作数。在书写指令时，需用方括号"[ ]"将BP和IM6括起来即[BP+IM6]。因为有效地址是16位，这种寻址方式只能用于零页64 K字存储单元范围内寻址。

如指令R1=[BP+0x20]。若BP的内容为0x1000，则该指令执行的操作是将零页存储器0x1020单元的内容传送到寄存器R1中。

六、PC相对寻址

PC相对寻址主要用于转移指令中，指令执行时检测段寄存器SR中标志段的各标志位，作为程序转移的条件。若条件满足则程序跳转到与PC相关的地址上。相对偏移量为有符号6位二进制数，即跳转的范围限制在PC±63个字，相对地址往后最多可跨过63个字，往前最多可倒退63个字。PC相对寻址用于访问程序存储区。

如指令JZ Label1，执行的操作是：若寄存器SR中的零标志位Z=1(前一条指令结果为0)，则修改程序指针PC=PC±IM6，转移到Label1标号地址上，否则继续执行下一条指令。

3.3　数据传送类指令

数据传送类指令是指令系统中使用最多的一类指令，其一般操作是把源操作数传送(复制)到目的操作数，指令执行后源操作数不变，目的操作数修改为源操作数。按其功能，又可把它们分为三类：装载数据、存储数据和堆栈操作。

一、装载数据指令

这一类传送指令主要用于给目的寄存器装载(LOAD)数据，即执行Rd=X的功能。源操作数X可以是存储器操作数据，可以是立即数，也可以是寄存器中的数据。目的操作数只能是寄存器。

根据寻址方式的不同，表3-1给出了所有装载数据类指令，同时，表中也给出了指令格式、指令周期(所包含的时钟周期)以及对标志位状态的影响。从表中可以看出，装载数据类指令只对负标志位N、零标志位Z有影响，即可根据装载的数据置位或复位N、Z，而对符号标志位S、进位位C无影响。

表3-1 数据装载类指令

指令语法	指令格式								指令周期	标志位状态			
	第一字							第二字		N	Z	S	C
Rd=IM6	xxxx	Rd	xxx	IM6				无	3	√	√	—	—
Rd=IM16	xxxx	Rd	xxxxxx			Rs		IM16	6/8				
Rd=[BP+IM6]	xxxx	Rd	xxx	IM6				无	8				
Rd=[A6]	xxxx	Rd	xxx	A6				无	6/8				
Rd=[A16]	xxxx	Rd	xxxxxx			Rs		A16	9/11				
Rd=Rs	xxxx	Rd	xxxxxx			Rs		无	3/8				
Rd={D:}[Rs] Rd={D:}[++Rs] Rd={D:}[Rs--] Rd={D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs		无	7/9				

说明：① 当Rd为PC时，执行周期是表中较长者，且数据装载后SR中的所有标志位不受影响

② √ 表示标志位受影响，— 表示标志位不受影响。以后各表中表示方法都相同，不再重复

指令格式中Rd、Rs的操作码如下：

SP	R1	R2	R3	R4	BP(R5)	SR	PC
000	001	010	011	100	101	110	111

下面根据源操作数寻址方式的不同介绍各条指令。

1. 6位立即数寻址

执行Rd=IM6指令时，自动将IM6进行零扩展为16位宽度后，存入Rd。Rd可以是R1～R4、BP、SP以及SR，但不能用PC。

如指令R1=0x28，在执行时先将立即数进行零扩展为0x0028，然后再将0x0028赋值给寄存器R1。

2. 16位立即数寻址

若指令Rd=IM16，其中，Rd可以是R1～R4、BP、SP、SR，也可以是PC。

如指令R2=0x28FF，该指令将立即数0x28FF传送给寄存器R2。

3. 变址寻址

若指令Rd=[BP+IM6]，其中，Rd可以是R1～R4、BP、SP以及SR，但不能用PC，[BP+IM6]只能在零页寻址。

如指令R3=[BP+0x08]。若当前(BP)=0x2000，则有效地址为零页存储单元0x2008，该指令的功能是将零页存储单元0x2008中的数据传送给寄存器R3。

4. 6位地址直接寻址

若指令Rd=[A6]，其寻址范围限于零页0x00～0x3F，Rd可以是R1～R4、BP、SP、SR以及PC。

如指令R4=[0x28]，若零页存储单元0x28中的数据为0x2000，则该指令功能是将存储单元0x28中的数据0x2000传送给寄存器R4。

5. 16 位地址直接寻址

若指令 Rd=[A16],其寻址范围为零页 0x0000～0xFFFF,Rd 可以是 R1～R4、BP、SP、SR 以及 PC。

如指令 R5=[0x2480],若零页存储单元 0x2480 中的数据为 0x3000,则该指令功能是将存储单元 0x2480 中的数据 0x3000 传送给寄存器 R5。

6. 寄存器寻址

若指令 Rd=Rs,其中,Rs 可以是 R1～R4、BP、SP 及 SR,而 Rd 可以是 R1～R4、BP、SP、SR 及 PC。

如指令 SR=R1,该指令功能是将寄存器 R1 的内容传送给段寄存器 SR。

7. 寄存器间接寻址

该寻址方式共有 Rd={D:}[Rs]、Rd={D:}[++Rs]、Rd={D:}[Rs--]和 Rd={D:}[Rs++]4 条指令。其中,Rd 可以是 R1～R4、BP、SP、SR 以及 PC。在表 3-1 的指令格式中,若 D=0,则为零页存储器;若 D=1,则由段寄存器 SR 中的 DS 字段决定存储器地址的页码。4 种寻址方式则由指令格式中的@决定,其对应关系如下:

00　[Rs]　　执行后 Rs 不变;

01　[Rs--]　　执行后 Rs 减 1;

10　[Rs++]　　执行后 Rs 加 1;

11　[++Rs]　　执行前 Rs 加 1。

关于 D 和@的说明,后续各指令寄存器间接寻址均相同,不再重复。

如零页存储单元 0x1001、0x1002 中的数据依次为 0x1234、0x78DE,R5 中的数据为 0x1001,DS 段为 0,则下列程序执行后结果为:

```
R4=[R5]          //将零页中 R5 指定的单元中的数据→(R4),执行后 R4 中的数据为 0x1234,R5 不
                 //变,寄存器间接寻址前省略 D,默认为零页
R4=D:[R5--]      //先将 DS 段和 R5 指定的单元中的数据→(R4),再执行(R5)-1→(R5),执行后使
                 //R4 中的数据为 0x1234,R5 中的数据为 0x1000
R4=D:[R5++]      //先将 DS 段和 R5 指定的单元中的数据→(R4),再执行(R5)+1→(R5),执行后使
                 //R4 中的数据为 0x1234,R5 中的数据为 0x1002
R4=D:[++R5]      //先将(R5)-1→(R5),再将 DS 段和 R5 指定的单元中的数据→(R4),执行后使 R4
                 //中的数据为 0x78DE,R5 中的数据为 0x1002
```

[例 1]　将 16 位立即数 0x1234 送入 R1 中,将 6 位立即数 0x3F 送入 R2 中,并且将 R1、R2 的数据分别保存于 R3、R4 中,再将 R1 清零。执行如下:

```
R1=0x1234        //16 位立即数 0x1234 送入 R1 中,标志位 N=0,Z=0
R2=0x3F          //6 位立即数 0x3F 扩展成 16 位并送入 R2 中,标志位 N=0,Z=0
R3=R1            //R1 的数据保存于 R3,标志位 N=0,Z=0
R4=R2            //R2 的数据保存于 R4,标志位 N=0,Z=0
R1=0             //将 0 赋给 R1,标志位 N=0,Z=1
```

二、存储数据指令

这一类传送指令主要用于将寄存器的数据存储(STORE)到存储单元,即执行 X=Rd 的功能,目的操作数可以采用间接寻址、直接寻址和寄存器间接寻址 3 种方式。源操作数只能是

寄存器，也就是说，给存储单元送数，只能通过寄存器传送，而不能直接将立即数送至存储单元。

根据寻址方式的不同，表3-2给出了所有存储数据类指令。从表中可以看出，存储数据类指令对所有标志位都无影响。

表3-2 存储数据类指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
[BP+IM6]=Rd	xxxx	Rd	xxx	IM6			无	8				
[A6]=Rd	xxxx	Rd	xxx	A6			无	6/8				
[A16]=Rd	xxxx	Rd	xxxxxx			Rs	A16	9/11				
{D:}[Rs]=Rd {D:}[++Rs]=Rd {D:}[Rs--]=Rd {D:}[Rs++]=Rd	xxxx	Rd	xxx	D	@	Rs	无	7/9	—	—	—	—
说明：当Rd为PC时，执行周期是表中较长者												

下面根据目的操作数寻址方式的不同介绍各条指令。

1. 变址寻址

若指令[BP+IM6]=Rd中，Rd可以是R1～R4、BP、SP以及SR，但不能用PC，[BP+IM6]只能在零页寻址。

如指令[BP+0x08]=R1，若当前BP=0x2000，R1中的数据为0x1003，则该指令功能是将数据0x1003送入零页存储单元0x2008中。

2. 6位地址直接寻址

若指令[A6]=Rd，其寻址范围限于零页0x00～0x3F，而Rd可以是R1～R4、BP、SP、SR以及PC。

如指令[0x28]=R2，若R2中的数据为0x2004，则该指令功能是将数据0x2004送入零页存储单元0x28。

3. 16位地址直接寻址

若指令[A16]=Rd，其寻址范围为零页0x0000～0xFFFF，而Rd可以是R1～R4、BP、SP、SR以及PC。

如指令[0x2480]=R3，若R3中的数据为0x3004，则该指令功能是将数据0x3004送入零页存储单元0x2480。

4. 寄存器间接寻址

该类指令共有{D:}[Rs]=Rd、{D:}[++Rs]=Rd、{D:}[Rs--]=Rd和{D:}[Rs++]=Rd 4条指令。其中，Rd可以是R1～R4、BP、SP、SR以及PC。

如：若R4中的数据为0x238F，R5中的数据为0x1001，DS段为0，则下列程序执行后结果为：

```
[R5]=R4          //将(R4)→零页R5指定的单元，执行后使0x1001单元中的数据为0x238F，
                 //R5不变
```

```
D:[R5--]=R4        //先将(R4)→DS段和R5指定的单元中，再执行(R5)－1→(R5)，执行后使
                   ./0x1001 单元中的数据为 0x238F,R5 中的数据为 0x1000
D:[R5++]=R4        //先将(R4)→DS段和R5指定的单元中，再执行(R5)＋1→(R5)，执行后使
                   //0x1001 单元中的数据为 0x238F,R5 中的数据为 0x1002
D:[++R5]=R4        //先将(R5)＋1→(R5)，再执行(R4)→DS段和R5指定的单元中，执行后使
                   //0x1002 单元中的数据为 0x238F,R5 中的数据为 0x1002
```

[例 2] 将 R3 的值保存于 0x25 存储单元中，可以采取下面三种方法实现：

```
(1) [0x25]=R3      //将 R3 的值存储于 0x25 存储单元，为直接地址寻址
(2) R2=0x25        //将立即数 0x25 送入 R2 中
    [R2]=R3        //将 R3 的值存储于 0x25 存储单元，为寄存器间接寻址
(3) BP=0x20        //将立即数 0x20 送入 BP 中
    [BP+5]=R3      //将 R3 的值存储于 0x25 存储单元，为变址寻址
```

[例 3] 将 25H、26H、27H 单元清零，可以采取下面 3 种方法实现：

```
(1) R1=0           //将 0 赋给 R1，使标志位 N=0,Z=1
    R2=0x25        //将立即数 0x25 送入 R2 中
    [R2++]=R1      //将 R1 的值存储于 0x25 单元，并执行 R2=R2+1
    [R2++]=R1      //将 R1 的值存储于 0x26 单元，并执行 R2=R2+1
    [R2]=R1        //将 R1 的值存储于 0x27 单元
(2) R1=0           //将 0 赋给 R1，使标志位 N=0,Z=1
    R2=0x27        //将立即数 0x27 送入 R2 中
    [R2--]=R1      //将 R1 的值存储于 0x27 单元，并执行 R2=R2－1
    [R2--]=R1      //将 R1 的值存储于 0x26 单元，并执行 R2=R2－1
    [R2]=R1        //将 R1 的值存储于 0x25 单元
(3) R1=0           //将 0 赋给 R1，使标志位 N=0,Z=1
    R2=0x24        //将立即数 0x24 送入 R2 中
    [++R2]=R1      //先执行 R2=R2＋1，再将 R1 的值存储于 0x25 单元中
    [++R2]=R1      //先执行 R2=R2－1，再将 R1 的值存储于 0x26 单元中
    [++R2]=R1      //先执行 R2=R2＋1，再将 R1 的值存储于 0x27 单元中
```

三、堆栈操作

堆栈操作共有两条指令，完成堆栈的两种基本操作：一种叫压入(PUSH)或入栈操作；一种叫弹出(POP)或出栈操作。堆栈中的数据是以“后进先出”的结构方式处理的。这种“后进先出”的特点是由堆栈指针 SP 来控制的，SP 自动跟踪栈顶的地址，且总是指向栈顶空的位置。

表 3－3 给出了这两条指令。从表中可以看出，压入指令对所有标志位都无影响，而弹出指令只对负标志位 N、零标志位 Z 有影响。

表 3－3　堆栈操作类指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
PUSH Rx, Ry to [Rs]	xxxx	Rd	xxx	SIZE	Rs	无	$3n+4$	－	－	－	－
POP Rx, Ry from [Rs]	xxxx	Rd	xxx	SIZE	Rs	无	$3n+4/3n+6$	√	√	--	－

1. 入栈操作

PUSH Rx, Ry to [Rs]

该条指令的功能是将 n(n=1～7,指令格式中的 SIZE,代表压入堆栈的寄存器个数)个序列寄存器 Rx～Ry(Rx～Ry≠SP)中的字数据压入 Rs 指出的一组存储器中,且总是将序号高的寄存器内容先压入,堆栈存储器地址初值由指针 Rs 指出。

由于堆栈编址采用向下生成方式,SP 指向栈顶空的位置,所以入栈操作时,先将序号高的寄存器内容压入堆栈,然后使 SP 自动减 1,即(SP)－1→(SP),指向栈顶的下一个空单元,再将下一个序号高的寄存器内容压入堆栈,依次类推,最后 SP 指向新的栈顶空位置。可以看出,μ'nSP™的堆栈操作方式与一般的 CPU 不同,通常一般的 CPU 的堆栈采用向上生成方式,且 SP 指向栈顶位置而不是空位置。所以在入栈操作时,先修改堆栈指针即(SP)＋1→(SP),然后再将寄存器内容压入堆栈,SP 总是指向栈顶。这一点在使用 μ'nSP™单片机时需特别注意。

例如指令 PUSH R3,PC to [SP],该指令执行的功能是把从 R3～PC (R7)中的字数据压入堆栈中,执行过程如图 3－5 所示,操作如下:

(1) 先把 PC 压入堆栈,即(PC)→((SP));
修改堆栈指针,即(SP)－1→(SP);

(2) 把 SR 压入堆栈,即(SR)→((SP));
修改堆栈指针,即(SP)－1→(SP)。

⋮

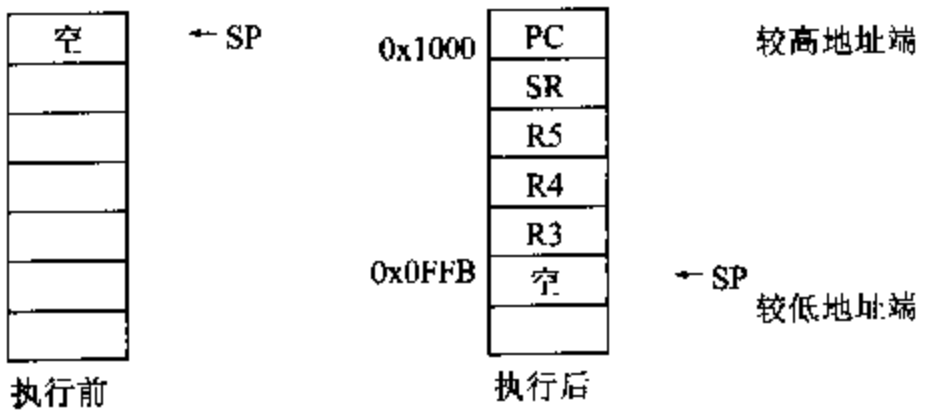


图 3－5 PUSH R3, PC to [SP] 指令示意

其余类推。若指令执行前 SP 中的数据为 0x1000,则执行后 SP 中的数据为 0x0FFFB,指向新的空栈顶。

注意:由于入栈时总是将序号高的寄存器中的数据先压入,所以指令 PUSH R3,PC to [SP]与指令 PUSH PC,R3 to [SP]是等效的,功能完全相同。指令 PUSH Rx,Ry to [Rs]是把 Rx～Ry 的所有寄存器都压入堆栈。若要压入单个寄存器,则指令形式是 PUSH Rx,Rx to [Rs]。

2. 出栈操作

POP Rx,Ry from [Rs]

该条指令的功能是将由 Rs 指出的 n(n=1～7,指令格式中的 SIZE)个存储单元中的字数据传送到序列寄存器 Rx～Ry(Rx～Ry≠SP)中,且总是先传送数据到序号低的寄存器内,堆栈存储器地址初值由指针 Rs 指出。当目的寄存器为 PC 时,执行周期是表中较长者,且传送数据后 SR 中的所有标志位不受影响。

由于堆栈编址采用向下生成方式，SP 指向栈顶空的位置，所以出栈操作时，先将 SP 自动加 1，即(SP)+1→(SP)，指向栈顶的第一个数据；然后将该数据传送到序号低的寄存器内，使 SP 自动加 1，指向堆栈内的下一个数据；再将该数据传送到下一个序号低的寄存器内。依次类推，最后将 SP 指向新的栈顶空位置。

例如指令 POP　R3，PC to [SP]，该指令执行后的功能是把 SP 指向的存储器单元中的 5 个字数据依次传送到 R3～PC (R7)寄存器中。执行过程如图 3－6 所示，操作如下：

(1) 先修改堆栈指针，即(SP)+1→(SP)；

把 R3 弹出堆栈，即((SP))→(R3)；

(2) 修改堆栈指针，即(SP)+1→(SP)；

把 R4 弹出堆栈，即((SP))→(R4)。

⋮

其余依次类推。若指令执行前 SP 中的数据为 0x0FFB，存储器 0x0FFC～0x1000 单元中的数据依次为 0x1A00、0x1B00、0x1C00、0x1D00、0x1E00，则执行后 SP 中的数据为 0x1000，指向新的空栈顶，使上述 5 个字数据依次传送到寄存器 R3～PC(R7)寄存器中。

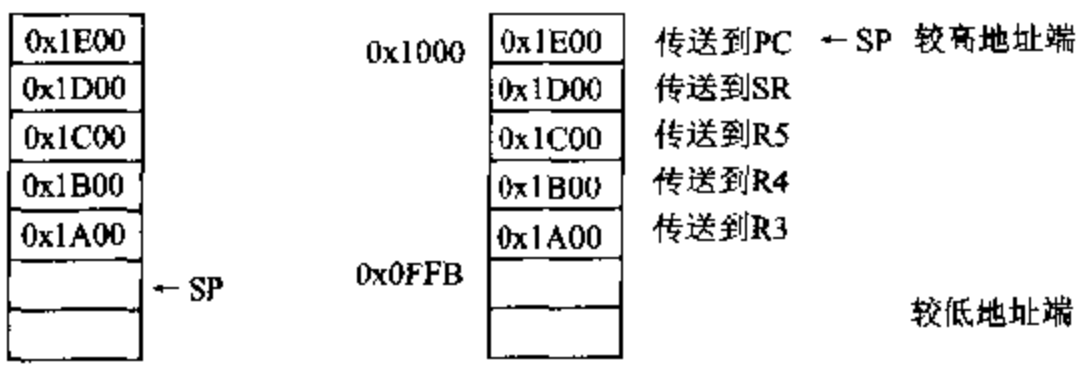


图 3－6　POP　R3，PC from [SP] 指令示意

注意：由于出栈时总是先将存储器中的字数据传送到序号低的寄存器内，所以指令 POP R3，PC　from [SP] 与指令 POP　PC，R3 from [SP] 是等效的，功能完全相同。指令 POP　Rx，Ry from [Rs] 是依次把堆栈中的数据传送到寄存器 Rx～Ry。若要把单个数据从堆栈弹出到寄存器，则指令形式是 POP　Rx，Rx from [Rs]。

由此可见，PUSH 和 POP 是两种逆传指令，一般应成对使用。它们常被用在保护现场(即把寄存器中的数据暂存到某内存区)和恢复现场的程序中，增加了编程的灵活性。

3.4　算术运算类指令

该类指令主要完成加、减、乘的算术运算，没有除法指令。另外，求补、比较指令实质上也是执行算术运算，放到算术运算类指令中一并介绍。加法、减法、比较指令对负标志位 N、零标志位 Z、符号标志位 S、进位位 C 都有影响，根据运算结果置位或复位 N、Z、S 和 C；乘法指令对状态标志位无影响；求补指令只对 N、Z 有影响。虽然算术/逻辑单元 ALU 仅执行 16 位无符号整数的算术运算，但进位标志 C 则给多字无符号整数的加、减运算和移位操作提供了方便，用软件监视溢出标志可方便地实现补码运算(带符号数的运算)。此外，状态标志位又往往可作为条件判断或条件跳转的依据。

加、减运算以及求补、比较的寻址方式与前面介绍的装载数据类完全相同，如不作特别说明外，则有关装载数据类指令的寻址说明完全适用于加、减运算，且加、减运算类似于C语言，也无特定助记符。如复合赋值运算：+=(自反加等于)、--=(自反减等于)等，下面分别予以介绍。

一、加法运算指令

在加法运算中，被加数X必须是寄存器(既可以是目的寄存器Rd，也可以是源寄存器Rs)，加数Y可以是6种寻址方式中的任何一种寻址方式，其操作结果为X+Y的和存放在目的寄存器Rd中，执行为X+Y→X。加法运算又可以分为不带进位加法和带进位加法两种。

1. 加法指令

加法指令即不带进位的加法X+Y→X，根据寻址方式的不同，表3-4给出了所有加法指令。

表3-4 加法指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=IM6 Rd=Rd+IM6	xxxx	Rd	xxx	IM6		无	3				
Rd=Rs+IM16	xxxx	Rd	xxxxxx		Rs	IM16	6/8				
Rd+=[BP+IM6] Rd=Rd+[BP+IM6]	xxxx	Rd	xxx	IM6		无	8				
Rd+=[A6] Rd=Rd+[A6]	xxxx	Rd	xxx	A6		无	6/8				
Rd=Rs+[A16]	xxxx	Rd	xxxxxx		Rs	A16	9/11	√		√	√
Rd+=Rs Rd=Rd+Rs	xxxx	Rd	xxxxxx		Rs	无	3/8				
Rd--{D:}[Rs] Rd+={D:}[++Rs] Rd-={D:}[Rs--] Rd-={D:}[Rs++]	xxxx	Rd	xxx	D	@ Rs	无	7/9				
说明：当Rd为PC时，执行周期是表中较长者，且运算后SR中的所有标志位不受影响											

加法指令的源寄存器Rs可以是R1～R4、BP、SP、SR及PC共8个寄存器中的任何一个。当加数采用变址寻址[BP+IM6]时，目的寄存器Rd可以是R1～R4、BP、SP及SR，但加数采用其它寻址方式时，则Rd可以是R1～R4、BP、SP、SR及PC共8个寄存器中的任何一个。

下面给出部分指令：

```
R1+=0x20          //((R1)+0x20)→(R1)，等效于R1=R1+0x20，即该指令有表中所示两种书
                  //写方式，执行的功能是将R1中的数据与6位立即数0x20相加，结果存放到
                  //R1中。若当前R1中的数据是0x7FFF，则执行该指令后，使R1中的数据为
                  //0x801F，各标志位的状态分别为C=0(无进位)，S=0(结果为正，无符号数)，
                  //Z=0(结果不为0)，N=1(结果最高有效位为1)
R1=R2+0x100A      //((R2)+0x100A)→(R1)，执行的功能是将R2中的数据与16位立即数
                  //0x100A相加，结果存放到R1中
```

```
PC=R3+0x2400          //((R3)+0x2400)→PC.执行的功能与上类似,但由于目的寄存器是程序指
                      //针,程序将转移,故该指令可用于快速查表跳转
R1+=[BP+0x24]         //((R1)+((BP)+0x24))→(R1),等效于R1←R1+[BP+0x24]。该指令有
                      //表中所示两种书写方式,执行的功能是将R1中的数据与(BP)+0x20指定的
                      //零页单元中的数据相加,结果存放到R1中,[BP+0x24]为变址寻址方式
BP+=[D_tabel]         //(BP+D_tabel单元中的数据)→(BP),等效于BP=BP+[D_tabel]。该指令
                      //有表中所示的两种书写方式。执行的功能是将BP中的数据与D_tabel单元
                      //中的数据相加,结果存放到BP中.D_tabel的范围在零页0x00~0x3F,
                      //[D_tabel]为6位地址[A6]直接寻址方式
BP=R1+[Q_table]       //((R1)+(Q_table))→(BP),执行的功能是将R1中的数据与Q_table单元中
                      //的数据相加,结果存放到BP中。该指令可用于查表,D_tabel的范围在零页
                      //0x0000~0xFFFF中,[Q_table]为16位地址[A16]直接寻址方式
PC+=R3                //((PC)+(R3))→(PC),等效于PC←PC+R3。该指令有表中所示的两种书
                      //写方式。执行的功能是将PC中的数据与R3中的数据相加,结果存放到PC
                      //中。可见该指令可强制程序转移
R3+=[R4]              //((R3)+零页中R4指定的单元中的数据)→(R3),即执行的功能是将R3中
                      //的数据与零页R4指定单元中的数据相加,结果存放到R3中
R3+=D:[++R4]          //①(R4)-1→(R4),②((R3)+由DS段和R4指定单元中的数据)→(R3),即
                      //该指令执行的功能是先把R4中的数据加1,然后再将R3中的数据与由DS
                      //段和R4指定单元中的数据相加,结果存放到R3中
R3+=D:[R4--]          //①((R3)+由DS段和R4指定单元中的数据)→(R3),②(R4)-1→(R4),即
                      //该指令执行的功能是先将R3中的数据与由DS段和R4指定单元中的数据
                      //相加,将结果存放到R3中,再把R4中的数据减1
R3+=D:[R4++]          //①((R3)+由DS段和R4指定单元中的数据)→(R3),②(R4)+1→(R4),即
                      //该指令执行的功能是先将R3中的数据与由DS段和R4指定单元中的数据
                      //相加,结果存放到R3中,再把R4中的数据加1
```

[例4] 求两个数0x5588、0x2219的和,若结果存放于R1中,可以采取下面两种方法实现:

```
(1) R1=0x5588         //使标志位 N=0,Z=1
    R1+=0x2219        //使标志位 N=0,Z=0,S=0,C=0
(2) R2=0x5588
    R1=R2+0x2219
```

[例5] 求0x2222、0x2223两个存储单元中的数据之和,若结果存于R1中,可以采取下面三种方法实现:

```
(1) R1=0              //R1清空,标志位 N=0,Z=1
    R2=0x2222         //R2为地址指针,指向0x2222
    R1+=[R2++]        //读取R2间接寻址的单元值,并使R2的值加1
    R1+=[R2]          //求和
(2) R1=0              //R1清空,标志位 N=0,Z=1
    R2=0x2223         //R2为地址指针,指向0x2223
    R1+=[R2--]        //读取R2间接寻址的单元值,并修改指针R2
    R1+=[R2]          //求和
(3) R1=0x2222         //R1为地址指针,指向0x22222
```

R1=[R1]

R2=0x2223 //R2为地址指针，指向0x2223

R1+=[R2] //求和

2. 带进位的加法指令

带进位的加法指令与不带进位的加法指令的惟一区别，就是在执行加法时，进位标志也参与加法运算，即执行X←Y+C→X，其余完全相同。指令格式、指令周期以及对标志位状态的影响如表3-5所列。

表3-5 带进位加法指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=IM6, Carry Rd=Rd+IM6, Carry	xxxx	Rd	xxx	IM6		无	3	√	√	√	√
Rd=Rs+IM16, Carry	xxxx	Rd	xxxxxx		Rs	IM16	6/8				
Rd+=[BP+IM6], Carry Rd=Rd+[BP+IM6], Carry	xxxx	Rd	xxx	IM6		无	8				
Rd+=[A6], Carry Rd=Rd+[A6], Carry	xxxx	Rd	xxx	A6		无	6/8				
Rd=Rs+[A16], Carry	xxxx	Rd	xxxxxx		Rs	A16	9/11				
Rd+=Rs, Carry Rd=Rd+Rs, Carry	xxxx	Rd	xxxxxx		Rs	无	3/8				
Rd+=[Rs], Carry Rd+=[--Rs], Carry Rd+=[Rs--], Carry Rd+=[Rs++], Carry	xxxx	Rd	xxx	D	@ Rs	无	7/9				
说明：当Rd为PC时，执行周期是表中较长者，且运算后SR中的所有标志位不受影响											

下面给出部分指令：

R1+=0x20, Carry //((R1)+0x20+C)→(R1)，等效于R1=R1+0x20+C，有两种书写方式

R1=R2+0x2400, Carry //((R2)+0x2400+C)→(R1)

R1+=[BP+0x24], Carry //((R1)+零页((BP)+0x24)指定的单元中的数据+C)→(R1)，它等效于R1=R1+[BP+0x24], Carry

R2+=[0x24], Carry //((R2)+零页0x24单元的中的数据+C)→(R2)，它等效于R2=R2+[0x24], Carry

BP=R1+[0x23DF], Carry //((R1)+零页0x23DF单元的中的数据+C)→(BP)

R5+=R3, Carry //((R5)+(R3)+C)→(R5)等效于R5=R5+R3, Carry，有两种书写方式

R3+=[R4], Carry //((R3)+零页中R4指定单元中的数据+C)→(R3)

R3+=DS:[++R4], Carry //先执行(R4)+1→(R4)，再执行((R3)+由DS段和R4指定单元中的数据+C)→(R3)

R3+=DS:[R4--], Carry //先执行((R3)+由DS段和R4指定单元中的数据+C)→(R3)，再执行(R4)-1→(R4)

R3+=DS:[R4++], Carry //先执行((R3)+由DS段和R4指定单元中的数据+C)→(R3)，再执行(R4)+1→(R4)

二、减法运算指令

减法运算中，被减数 X 必须是寄存器(既可以是目的寄存器 Rd，也可以是源寄存器 Rs)，减数 Y 可以是 6 种寻址方式中的任何一种寻址方式，操作结果即 X－Y 的差存放在目的寄存器 Rd 中，执行为 X－Y →X。减法运算也分为不带进位减法和带进位减法两种。

1. 减法指令

减法指令即不带进位的减法为 X- Y→X，根据寻址方式的不同，表 3－6 给出了所有减法指令。

表 3－6　减法指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd－＝IM6 Rd＝Rd- IM6	xxxx	Rd	xxx	IM6		无	3	√	√	√	√
Rd＝Rs－IM16	xxxx	Rd	xxxxx		Rs	IM16	6/8				
Rd－＝[BP－IM6] Rd＝Rd－[BP－IM6]	xxxx	Rd	xxx	IM6		无	8				
Rd－＝[A6] Rd＝Rd－[A6]	xxxx	Rd	xxx	A6		无	6/8				
Rd＝Rs－[A16]	xxxx	Rd	xxxxx		Rs	A16	9/11				
Rd- ＝Rs Rd＝Rd－Rs	xxxx	Rd	xxxxxx		Rs	无	3/8				
Rd－－{D:}[Rs] Rd ＝{D:}[＋＋Rs] Rd－＝{D:}[Rs－ －] Rd- ＝{D:}[Rs＋ ＋]	xxxx	Rd	xxx	D	@	Rs	无	7/9			
说明：当 Rd 为 PC 时，执行周期是表中较长者，且运算后 SR 中的所有标志位不受影响											

减法指令的源寄存器 Rs 可以是 R1～R4、BP、SP、SR 及 PC 等 8 个寄存器中的任何一个。当减数采用变址寻址[BP＋IM6]时，目的寄存器 Rd 可以是 R1～R4、BP、SP 及 SR，但减数采用其它寻址方式时，则 Rd 可以是 R1～R4、BP、SP、SR 及 PC 等 8 个寄存器中的任何一个。

下面给出部分指令：

```
SR－＝0x20          //((SR)－0x20)→(SR)，它等效于 SR＝SR－0x20，有两种书写方式
R1＝R2－0x3400      //((R2)－0x3400)→(R1)
R1－＝[BP＋0x24]    //((R1)－零页中((BP)＋0x24)指定的单元中的数据)→(R1)，等效于R1＝
                    //R1－[BP＋0x24]，有两种书写方式
R2－＝[0x3E]        //((R2)－零页中 0x3E 单元中的数据)→(R2)，等效于 R2＝R2＋[0x3E]，有
                    //两种书写方式
R5＝R1－[0x28FF]    //((R1)－零页 0x28FF 单元中的数据)→(R5)
BP－＝R3            //(BP)－(R3)→(BP)，等效于 BP＝BP－R3，有两种书写方式
R3－＝[R4]          //((R3)－零页中 R4 所指定的单元中的数据)→(R3)
R3－＝D:[－＋R4]    //先执行(R4)－1→(R4)，再执行((R3)－－由 DS 段和 R4 指定的单元中的数
                    //据)→(R3)
```

R3－＝D:[R4－－]　　//先执行((R3)－由DS段和R4指定的单元中的数据)→(R3)，再执行
//(R4)－1→(R4)

R3－＝D:[R4＋＋]　　//先执行((R3)－由DS段和R4指定的单元中的数据)→(R3)，再执行
//(R4)＋1→(R4)

[例6] 求0x2048、0x2049单元的差，结果存放于R1中。

R2＝0x2048　　//地址指针

R1＝[R2＋＋]　　//取0x2048单元的值

R1－＝[R2]　　//求差

2. 带进位的减法指令

带进位的减法指令与不带进位的减法指令的惟一区别，就是在执行加法时，进位标志也参与减法运算，即执行的为X－Y－C→X指令，其余完全相同。指令格式、指令周期以及对标志位状态的影响如表3－7所列。

表3－7 带进位减法指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
Rd－＝IM6，Carry Rd＝Rd－IM6，Carry	xxxx	Rd	xxx	IM6			无	3	√	√	√	√
Rd＝Rs－IM16，Carry	xxxx	Rd	xxxxxx		Rs		IM16	6/8				
Rd－＝[BP＋IM6]，Carry Rd＝Rd－[BP＋IM6]，Carry	xxxx	Rd	xxx	IM6			无	8				
Rd－＝[A6]，Carry Rd＝Rd－[A6]，Carry	xxxx	Rd	xxx	A6			无	6/8				
Rd＝Rs－[A16]，Carry	xxxx	Rd	xxxxxx		Rs		A16	9/11				
Rd－＝Rs，Carry Rd＝Rd－Rs，Carry	xxxx	Rd	xxxxxx		Rs		无	3/8				
Rd－＝{D:}[Rs]，Carry Rd－＝{D:}[＋＋Rs]，Carry Rd－＝{D:}[Rs－－]，Carry Rd－＝{D:}[Rs＋＋]，Carry	xxxx	Rd	xxx	D	@	Rs	无	7/9				
说明：当Rd为PC时，执行周期是表中较长者，且运算后SR中的所有标志位不受影响												

下面给出部分指令：

SR－＝0x20，Carry　　//((SR)－0x20－C)→(SR)，它等效于SR＝SR－0x20，Carry，有两种
//书写方式

R1＝R2－0x3400，Carry　　//((R2)－0x3400－C)→(R1)

R1－＝[BP＋0x24]，Carry　　//((R1)－((BP)＋0x24)－C)→(R1)，它等效于R1＝R1－[BP＋0x24]，
//Carry，有两种书写方式

R2－＝[0x3E]，Carry　　//((R2)－(0x3E)－C)→(R2)，它等效于R2＝R2－[0x3E]，Carry
//有两种书写方式

R5＝R1－[0x28FF]，Carry　　//((R1)－(0x28FF)－C)→(R5)

BP－＝R3，Carry　　//((BP)－(R3)－C)→(BP)，它等效于BP＝BP－R3，Carry，有两种

```
                                    //书写方式
R3－＝[R4], Carry                   //((R3)－零页中 R4 所指定的单元中的数据－C)→(R3)
R3－＝D:[＋＋R4], Carry             //先执行(R4)＋1→(R4),再执行((R3)－由 DS 段和 R4 指定的单元
                                    //中的数据－C)→(R3)
R3－＝D:[R4－－], Carry             //先执行((R3)－由DS段和R4指定的单元中的数据－C)→(R3),
                                    //再执行(R4)－1→(R4)
R3－＝D:[R4＋＋], Carry             //先执行((R3)－由DS段和R4指定的单元中的数据－C)→(R3),
                                    //再执行(R4)－1→(R4)
```

三、乘法运算指令

μ'nSP™的指令系统除了提供普通的 16 位加、减法运算指令外，还有速度较快的 16 位乘法运算指令 Mul 和内积运算指令 Muls。这 2 条指令特别适合于复杂的数字运算，特别是内积运算，使 μ'nSP™单片机具有与数字信号处理 DSP 类似的功能。可应用于具有复杂数字信号运算处理的场合，也可应用于运算速度要求较快的场合。

表 3－8 给出了乘法运算和内积运算的指令格式、指令周期，它们又可分为有符号数和无符号数两种。指令格式中的 S＝0 表示两个有符号数运算，S＝1 表示一个无符号数与一个有符号数运算。运算结果对标志位状态都无影响，下面分别介绍。

表 3－8　乘法指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
MR＝Rd＊Rs｛,ss｝ MR＝Rd＊Rs ,us	xxxx	Rd	S	xxxxx		Rs	无	3	–	–	–	–
MR＝[Rd]＊[Rs]｛,ss｝｛,n｝ MR＝[Rd]＊[Rs]⁻,us｛,n｝	xxxx	Rd	S	x	SIZE	Rs	无	8				

1. 乘法指令

乘法指令只能采用寄存器寻址，且 Rd、Rs 只能采用 R1～R4、BP(R5)。

(1) 有符号乘法：MR＝Rd＊Rs｛, ss｝。

该条指令的功能是将寄存器 Rd 中有符号数字数据和寄存器 Rs 中的有符号数字数据相乘，指令规定乘积存入 R4 与 R3 组成的 32 位寄存器中，其中 R4 为乘积的高字，R3 为乘积的低字。ss 表示为有符号数，若省略则默认为有符号数。

```
如：MR＝R3＊R1, ss      //将寄存器 R3 和 R1 中的两个有符号数相乘，结果存入 R4 与 R3 中
    MR＝R3＊R1          //与指令 MR＝R3＊R1, ss 功能相同
```

(2) 无符号和有符号数乘法：MR＝Rd＊Rs, us。

该条指令的功能是将寄存器 Rd 中无符号数字数据和寄存器 Rs 中的有符号数字数据相乘，乘积存入 R4 与 R3 组成的 32 位寄存器中，其中 R4 为乘积的高字，R3 为乘积的低字。us 表示为无符号数，不能省略，若省略则默认为有符号数。该条指令只能进行一个有符号数和一个无符号数的乘法，且指令中的第一个数据(Rd)总是无符号数，第二个数据(Rs)是有符号数。

```
如：MR＝R1＊R2, us      //将寄存器 R1 中的无符号数和 R2 中的有符号数相乘，结果存入 R4 与 R3 中
    MR＝R2＊R1, us      //将寄存器 R2 中的无符号数和 R1 中的有符号数相乘，结果存入 R4 与 R3 中
```

[例7]　计算一年(365天)共有多少小时,结果存放在R4(高位)和R3(低位)中。

```
R1=365
R2=24
MR=R1*R2,us          ;计算乘积
```

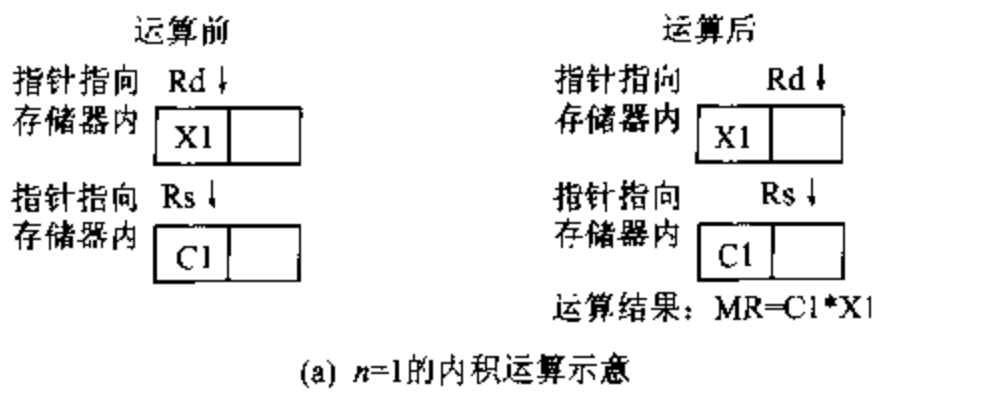
2. 内积指令

内积指令只能采用寄存器间接寻址,且Rd、Rs只能采用R1～R4、BP(R5),并指向寄存器SR中的DS字段指定的当前存储器地址页面。

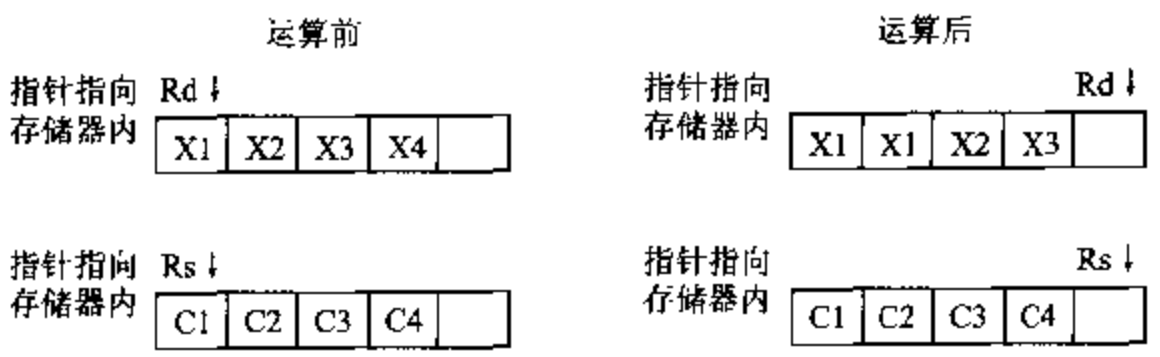
(1) 有符号内积:MR=[Rd]*[Rs]{,ss}{,n}。

该条指令的功能是将目的寄存器Rd和源寄存器Rs所指存储器地址中的有符号字数据进行n项内积运算,结果存入由R4和R3组成的32位寄存器中。其中R4为乘积的高字,R3为乘积的低字。ss表示为有符号数,若省略则默认为有符号数,n的值可设为1～16,若省略则默认为1。

内积运算操作过程如图3-7所示,其中图(a)为$n=1$的操作过程,图(b)为$n=4$的操作过程。



(a) n=1的内积运算示意



(b) n=4的内积运算示意

图3-7　内积运算操作示意

从图3-7可以看出,所谓内积运算实质是求Rd和Rs所指存储器地址中数据的n项乘积累加和。运算过程中,Rd、Rs作为指针自动调整。若FIR运算数据自动移动功能开放(FIR_MOV ON)且指令中的参数$n>1$时,由Rd指向的存储器中的数据内容会在求和运算后向前移动一个字的位置,如图3-7中的X1、X2、X3、X4自动前移,由X1、X1、X2、X3代替了原来的X1、X2、X3、X4。如果FIR运算数据自动移动功能关闭(FIR_MOV OFF),则求和后Rd指向的存储器中的数据内容不变。

内积运算主要用于数字信号处理,如n阶FIR滤波算法。需要注意:Rd指向的存储器中的数据内容会在求和运算后向前移动一个字的位置,所以Rd用于采样数据指针,Rs用于系数指针。数据移动的意义在于可以用新的采样数据代替旧的采样数据,简化程序设计,加快程序执行速度。

```
如:MR  [R2_ * [R1], 6        //将寄存器 R2 和 R1 指向的存储器中的 6 个有符号数进行内积运算,
                             //结果存入R4:R3(即MR)中。若FIR_MOVON,则R2指向的存储
                             //器内容前移一个字
     MR=[R1] * [R2], ss,8    //将寄存器 R1 和 R2 指向的存储器中的 8 个有符号数进行内积运算,
                             //结果存入R4R3中。若FIR_MOVOFF,则R1指向的存储器内容
                             //不变
```

(2) 无符号内积:MR=[Rd] * [Rs], us {,n}。

该条指令的功能是将目的寄存器 Rd 和源寄存器 Rs 所指存储器地址中的无符号字数据进行 n 项内积运算,结果存入由 R4 和 R3 组成的 32 位寄存器中。其中 R4 为乘积的高字,R3 为乘积的低字。us 表示为无符号数,不能省略,若省略则默认为有符号数,n 值可设为 1～16,若省略则默认为 1。该指令与有符号内积指令的惟一区别就是无符号,其它完全相同。

```
如:MR=[R2] * [R1], us, 6    //将寄存器 R2 和 R1 指向的存储器中的 6 个无符号数进行内积运算,
                             //结果存入R4与R3中,若FIR_MOVON,则R2指向的存储器内容
                             //前移一个字
```

使用上述两条内积指令时必须注意:

① 由于内积运算求的是乘积的累加和,结果自动存入 R4R3(MR)组成的 32 位寄存器中。所以,在执行内积指令前,必须将寄存器 R3、R4 清零,否则 R3、R4 中的数据将计入累加和,导致出错。

② 由于 R4、R3 存放结果,运算中 Rd 和 Rs 自动调整,所以 Rd、Rs 都不能使用 R3、R4,且 Rd、Rs 不能使用同一个寄存器。

四、求补指令

求补指令 NEG 用于求补码,执行(－X)＋1 的操作,即将被求数据 X 取反后加 1,操作结果存放在目的寄存器 Rd 中。求补指令可以采用 6 种寻址方式中的任何一种寻址方式,根据寻址方式的不同,表 3－9 给出了所有求补指令。

表 3－9　求补指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
Rd=－IM6	xxxx	Rd	xxx	IM6			无	3	√	√	－	－
Rd=－IM16	xxxx	Rd	xxxxxx			Rs	IM16	6/8				
Rd=－[BP＋IM6]	xxxx	Rd	xxx	IM6			无	8				
Rd=－[A6]	xxxx	Rd	xxx	A6			无	6/8				
Rd=－[A16]	xxxx	Rd	xxxxx			Rs	A16	9/11				
Rd=－Rs	xxxx	Rd	xxxxx			Rs	无	3/8				
Rd=－[Rs] Rd=－[＋＋Rs] Rd=－[Rs－－] Rd=－[Rs＋＋]	xxxx	Rd	xxx	D	@	Rs	无	7/9				
说明:当 Rd 为 PC 时,执行周期是表中较长者 7,且求补后 SR 中的所有标志位不受影响												

下面给出部分指令：

```
R1=-0x28                ;//求0x28的补码，先将0x28进行零扩展为16位数据0x0028，再求
                        ;//0x0028的补码，求补后(R1)=0xFFD8
R2=-0x28FF              ;//求0x28FF的补码，求补后使(R2)-0xD701
R3=-[BP+0x30]           ;//零页((BP)+0x30)单元中的数据求补→(R3)
R4=-[0x30]              ;//零页0x30单元中的数据求补→(R4)
BP=-[0x2800]            ;//零页0x2800单元中的数据求补→(BP)
R5=-R4                  ;//(R4)求补→(R5)
R4=-DS:[R3--]           ;//先将由DS段和R3指定单元中的数据求补→(R4)，再执行(R3)--1
                        ;//→(R3)
```

[例8] 求-600与0x2340单元的差，结果存放到R1中。

```
R1=-600|BP=0x2340       ;//地址指针
R2=-[BP]                ;//求0x2340单元中数据的补码
R1+=R2                  ;//相加，结果送R1
```

五、比较指令

比较指令CMP用于对两个数据X、Y的比较，并执行X-Y的运算，运算后X、Y的内容不变，且不保存结果，仅对4个标志位N、Z、S、C的状态有影响。

比较指令中，一个数必须是寄存器(既可以是目的寄存器Rd，也可以是源寄存器Rs)，另一个数可以是6种寻址方式中的任何一种寻址方式。根据寻址方式的不同，表3-10给出了所有比较指令。

表3-10 比较指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
CMP Rd, IM8	xxxx	Rd	xxx	IM6			无	3	√	√	√	√
CMP Rs, IM16	xxxx	Rd	xxxxxx			Rs	IM16	6/8				
CMP Rd, [BP+IM6]	xxxx	Rd	xxx	IM6			无	8				
CMP Rd, [A8]	xxxx	Rd	xxx	A6			无	6/8				
CMP Rs, [A16]	xxxx	Rd	xxxxxx			Rs	A16	9/11				
CMP Rd, Rs	xxxx	Rd	xxxxxx			Rs	无	3/8				
CMP Rd, [DS:][Rs] CMP Rd, [DS:][++Rs] CMP Rd, [DS:][Rs--] CMP Rd, [DS:][Rs++]	xxxx	Rd	xxx	D	@	Rs	无	7/9				
说明：当Rd为PC时，执行周期是表中较长者，且比较后SR中的所有标志位不受影响												

比较指令主要用于判断条件，以实现条件转移。例如：

```
CMP R1, 0x2000          //R1与立即数0x2000相比较
JE Label1               //若R1=0x2000，则跳转到标号Label1处
:
```

```
CMP R1, R2          //R1 与 R2 相比较
JBE Label2          //若 R1≤R2,则跳转到标号 Label2 处
⋮
```

3.5　逻辑操作类指令

μ'nSP™的指令系统具有丰富的逻辑操作指令，主要包括与、或、异或、测试、移位和循环等逻辑操作指令。其中，与、或、异或等指令的的寻址方式与前面介绍的装载数据类完全相同，如不作特别说明，则有关装载数据类指令的寻址说明完全适用于与、或、异或逻辑操作。而移位、循环操作只能采用寄存器直接寻址。部分逻辑操作类似于 C 语言，也无特定助记符，如复合操作运算：&=(反自与操作)、|=(反自或操作)和^=(反自异或操作)等，下面分别给以介绍。

一、逻辑与指令

逻辑与指令是将两个字数据按位进行与操作，其定义如下：

$$\begin{cases} 0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0 & \text{//有 0 即 0} \\ 1 \wedge 1 = 1 & \text{//全 1 为 1} \end{cases}$$

由上可知，利用逻辑与操作能够将一个操作数中的某些位置 0，而其余位保持原状。逻辑与操作中，一个数必须是寄存器(既可以是目的寄存器 Rd，也可以是源寄存器 Rs)，另一个数可以是 6 种寻址方式中的任何一种寻址方式，操作结果存放在目的寄存器 Rd 中。根据寻址方式的不同，表 3－11 给出了所有逻辑与指令。从表中可以看出，逻辑与指令只对负标志位 N、零标志位 Z 有影响。

表 3－11　逻辑与指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
Rd&=IM6 Rd=Rd&IM6	xxxx	Rd	xxx	IM6			无	3	√	√	—	—
Rd=Rs&IM16	xxxx	Rd	xxxxxx			Rs	IM16	6/8				
Rd&=[BP+IM6] Rd=Rd&[BP+IM6]	xxxx	Rd	xxx	IM6			无	8				
Rd&=[A6] Rd=Rd&[A6]	xxxx	Rd	xxx	A6			无	6/8				
Rd=Rs&[A16]	xxxx	Rd	xxxxxx			Rs	A16	9/11				
Rd&=Rs Rd=Rd&Rs	xxxx	Rd	xxxxxx			Rs	无	3/8				
Rd&={D:}[Rs] Rd&={D:}[++Rs] Rd&={D:}[Rs--] Rd&={D:}[Rs++]	xxxx	Rd	xxx	D	@	Rs	无	7/9				
说明：当 Rd 为 PC 时，执行周期是表中较长者，且操作后 SR 中的所有标志位不受影响												

下面给出部分指令：

```
R1&=0x3F          //(R1)^0x3F→(R1),执行时先将0x3F扩展为16位数0x003F,再和R1中的
                  //数据相与,相与后R1中的数据低6位不变,高10位全变为"0",与指令常用
                  //于屏蔽某些不用的位
R2&=[BP+0x24]     //(R2)^零页((BP)+0x24)单元中的数据→(R2)
R3&=[0x24]        //(R3)^零页0x24单元中的数据→(R3)
BP=R3&[0x24FF]    //(R3)^零页0x24FF单元中的数据→(BP)
BP&=R4            //(BP)^(R4)→(BP)
PC&=D:[R5++]      //先将(PC)^(DS,R5)指定的单元中的数据→(PC),再执行(R5)+1→(R5)
```

二、逻辑或指令

逻辑或指令是将两个字数据按位进行或操作，其定义如下：

$$\begin{cases} 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1 & \text{//有1即1} \\ 0 \vee 0 = 0 & \text{//全0为0} \end{cases}$$

由上可知，利用逻辑或操作能够将一个操作数中的某些位置1，而其余位保持原状。逻辑或操作中，一个数必须是寄存器(既可以是目的寄存器Rd，也可以是源寄存器Rs)，另一个数可以是6种寻址方式中的任何一种寻址方式，操作结果存放在目的寄存器Rd中。根据寻址方式的不同，表3-12给出了所有逻辑或指令。从表中可以看出，逻辑或指令只对负标志位N、零标志位Z有影响。

表3-12 逻辑或指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd |=IM6 Rd=Rd | IM6	xxxx	Rd	xxx	IM6		无	3				
Rd=Rs | IM16	xxxx	Rd	xxxxxx		Rs	IM16	6/8				
Rd |=[BP-IM6] Rd=Rd | [BP-IM6]	xxxx	Rd	xxx	IM6		无	8				
Rd |=[A6] Rd=Rd | [A6]	xxxx	Rd	xxx	A6		无	6/8	√	√	-	-
Rd=Rs | [A16]	xxxx	Rd	xxxxxx		Rs	A16	9/11				
Rd |=Rs Rd=Rd | Rs	xxxx	Rd	xxxxxx		Rs	无	3/8				
Rd |={D:}[Rs] Rd |={D:}[++Rs] Rd |={D:}[Rs--] Rd |={D:}[Rs++]	xxxx	Rd	xxx	D	@ Rs	无	7/9				

说明：当Rd为PC时，执行周期是表中较长者，且操作后SR中的所有标志位不受影响

下面给出部分指令：

```
R1 |=0x3F          //(R1) ∨0x3F →(R1),执行时先将 0x3F 扩展为 16 位数 0x003F,再和 R1 中的
                   //数据相或;相或后 R1 中的数据高 10 位不变,低 6 位全变为"1"。或指令常用
                   //于将某些位置 1
R2 |=[BP+0x24]     //(R2) ∨零页((BP)+0x24)单元中的数据 →(R2)
R3 =[0x24]         //(R3) ∨零页 0x24 单元中的数据 →(R3)
BP=R3 | [0x24FF]   //(R3) ∨零页 0x24FF 单元中的数据→(BP)
BP |=R4            //(BP) ∨(R4)→(BP)
PC =D:[R5--]       //先将(PC) ∨(DS,R5)指定的单元中的数据→(PC),再执行(R5)-1→(R5)
```

三、逻辑异或指令

逻辑异或是将两个字数据按位进行异或操作，其定义如下：

$$\begin{cases} 0 \oplus 1 = 1 \oplus 0 = 1 & \text{//相异为 1} \\ 0 \oplus 0 = 1 \oplus 1 = 0 & \text{//相同为 0} \end{cases}$$

由上可知，某逻辑量与"0"进行"异或"操作时保持原值不变，而与"1"相"异或"时，其结果为相反的值。逻辑异或操作中，一个数必须是寄存器(既可以是目的寄存器 Rd，也可以是源寄存器 Rs)，另一个数可以是 6 种寻址方式中的任何一种寻址方式，操作结果存放在目的寄存器 Rd 中。根据寻址方式的不同，表 3-13 给出了所有逻辑异或指令。由表中可以看出，逻辑异或指令只对负标志位 N、零标志位 Z 有影响。

表 3-13　逻辑异或指令

指令语法	指令格式							指令周期	标志位状态			
	第一字						第二字		N	Z	S	C
Rd ^ =IM6 Rd=Rd ^ IM6	xxxx	Rd	xxx	IM6			无	3	√	√	—	—
Rd=Rs ^ IM16	xxxx	Rd	xxxxxx			Rs	IM16	6/8				
Rd ^ =[BP+IM6] Rd=Rd ^ [BP+IM6]	xxxx	Rd	xxx	IM6			无	8				
Rd ^ =[A6] Rd=Rd ^ [A6]	xxxx	Rd	xxx	A6			无	6/8				
Rd=Rs ^ [A16]	xxxx	Rd	xxxxxx			Rs	A16	9/11				
Rd ^ =Rs Rd=Rd ^ Rs	xxxx	Rd	xxxxxx			Rs	无	3/8				
Rd ^ ={D:}[Rs] Rd ^ ={D:}[+ -Rs] Rd ^ ={D:}[Rs -] Rd ^ ={D:}[Rs- +]	xxxx	Rd	xxx	D	@	Rs	无	7/9				

说明：当 Rd 为 PC 时，执行周期是表中较长者，且操作后 SR 中的所有标志位不受影响

下面给出部分指令：

```
R1 ^ =0x283F          ;(R1)⊖0x283F →(R1)
R2 ^ =[BP+0x24]       ;(R2)⊕零页((BP)+0x24)单元中的数据→(R2)
R3 ^ =[0x21]          ;(R3)⊕零页0x21单元中的数据→(R3)
BP=R3 ^ [0x24FF]      ;(R3)⊙零页0x21FF单元中的数据 →(BP)
R4 ^ =R4              ;(R4)⊖(R4)→(R4),执行后R4中的数据为0,即将R4清零
PC ^ =DS:[++R5]       ;先将(R5)+1→(R5),再执行(PC)⊕(DS,R5)指定的单元中的数据 →(PC)
```

四、测试指令

测试指令TEST用于将两个数据X、Y执行逻辑与，即X ^ Y的运算，且不改变X、Y的值，结果不保存，仅对负标志位N、零标志位Z有影响。

测试指令中，一个数必须是寄存器(既可以是目的寄存器Rd，也可以是源寄存器Rs)，另一个数可以是6种寻址方式中的任何一种寻址方式。根据寻址方式的不同，表3-14给出了所有测试指令。

表3-14 测试指令

指令语法	指令格式								指令周期	标志位状态			
	第一字							第二字		N	Z	S	C
TEST Rd, IM6	xxxx	Rd	xxx	IM6				无	3	√	√		
TEST Rs, IM16	xxxx	Rd	xxxxxx		Rs			IM16	6/8				
TEST Rd, [BP+IM6]	xxxx	Rd	xxx	IM6				无	8				
TEST Rd, [A6]	xxxx	Rd	xxx	A6				无	6/8				
TEST Rs, [A16]	xxxx	Rd	xxxxxx		Rs			A16	9/11				
TEST Rd, Rs	xxxx	Rd	xxxxxx		Rs			无	3/8				
TEST Rd, [D:][Rs] TEST Rd, [D:][++Rs] TEST Rd, [D:][Rs--] TEST Rd, [D:][Rs++]	xxxx	Rd	xxx	D	@	Rs		无	7/9				

说明：当Rd为PC时，执行周期是表中较长者，且测试后SR中的所有标志位不受影响

下面给出部分指令：

```
TEST  R1, 0x25        ;(R1) ^ 0x25
TEST  R2, 0x25BD      ;(R2 ^ 0x25BD)
TEST  R3, [BP+0x20]   ;(R3) ^ 零页存储单元((BP)+0x20)中的数据
TEST  R1, [0x30]      ;(R4) ^ 零页存储单元0x30中的数据
TEST  SR, [0x78FF]    ;(SR) ^ 零页存储单元0x78FF中的数据
TEST  R2, BP          ;(R2) ^ (BP)
TEST  PC, [R1]        ;(PC) ^ 零页R1所指单元中的数据
TEST  R2, DS:[++R4]   ;先将(R1)+1→(R4),再执行(R2) ^ 由(DS,R4)指定的单元中的数据
```

五、移位指令

移位指令共有3种：逻辑/算术左移、逻辑右移和算术右移。移位指令只能采用寄存器寻址，μ'nSP™的CPU中的移位缓冲器SB参与了所有的移位操作，每条指令一次最多能移4位。该指令除了移位功能外，还具有数据传送、算术运算、逻辑操作功能，即具有双重功能。移位指令对标志位的影响与移位后的操作对标志位的影响一致。下面分别介绍各条移位指令。

1. 逻辑/算术左移指令 LSL

逻辑/算术左移指令LSL的功能是：先对源寄存器Rs和移位寄存器SB进行*nn*(可以设置为1～4)位逻辑/算术左移，然后再和目的寄存器Rd之间或进行传送数据，或进行算术运算，或进行逻辑操作，最后结果保存在Rd中。算术/逻辑左移指令LSL的操作过程如图3-8所示。

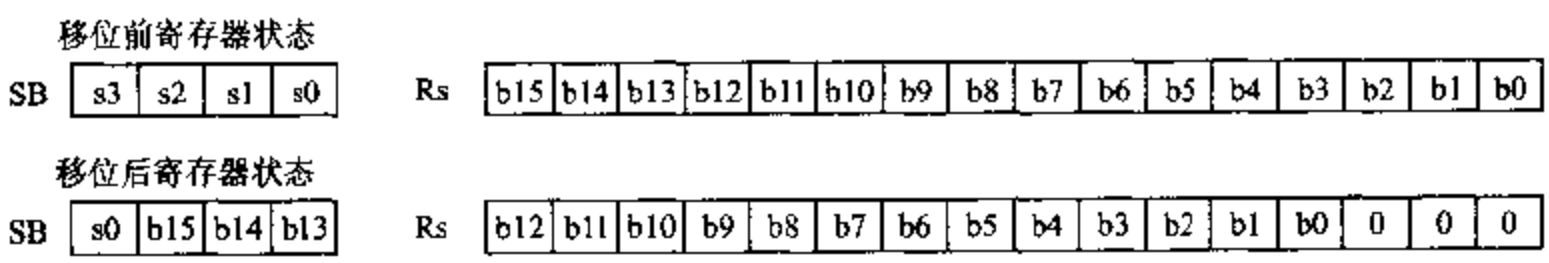


图3-8　LSL操作过程示意

上图中，Rs为源寄存器，SB为移位缓冲器。假设*nn*=3，即3位逻辑/算术左移。在移位过程中，第一次移出的位是Rs的最高位，它进入SB的最低位，Rs空出的最低位补0，SB移出的最高位丢失。随后的移位也按同样的过程进行。图中给出了3次移位结束后寄存器的状态。可以看出，每向左移动一次，实际上相当于Rs乘以2。

表3-15给出了所有逻辑/算术左移指令。

表3-15　逻辑/算术左移指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=Rs LSL *nn*,{Carry} Rd-= Rs LSL *nn*,{Carry} CMP Rd, Rs LSL *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	√	√
Rd= -Rs LSL *nn* Rd&=Rs LSL *nn* Rd |= Rs LSL *nn* Rd ^ =Rs LSL *nn* TEST Rd, Rs LSL *nn* Rd=Rs LSL *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	-	-
说明：当Rd为PC时，执行周期是表中较长者，且移位后SR中的所有标志位不受影响											

下面给出部分指令：

R1+=R2 LSL 2, Carry　　//((R1)←R2左移2位+C)→(R1)，相当于((R1)+(R2)×4+C)→(R1)

R1+=R2 LSL 4　　//((R1)+R2左移4位)→(R1)

```
R2-=R3 LSL 3, Carry      ' ((R2)-R3 左移 2 位 · C) →(R2)
R2-=R3 LSL 4             ' ((R2)  R3 左移 4 位) →(R2)
CMP R3, R4 LSL 2         ' (R3)-(R4 左移 2 位)
R2=  R3 LSL 1            ' (R3 左移 1 位求补) →(R2)
BP&=~R2 LSL 2            ' ((BP)∧(R2 左移 2 位))→(BP)
R2 |=SP LSL 1            ' ((R2)∨(SP 左移 1 位)) →(R2)
R3 ^=PC LSL 1            ' ((R3)⊕(PC 左移 1 位))→(R3)
TEST R4, PC LSL 1        ' (R4)∧(PC 左移 1 位)
R4=BP LSL 2              ' (BP 左移 2 位)→(R4)
```

2. 逻辑右移指令 LSR

逻辑右移指令 LSR 的功能是：先对源寄存器 Rs 和移位寄存器 SB 进行 *nn*(可以设置为 1～4)位逻辑右移，然后再和目的寄存器 Rd 之间或进行传送数据，或进行算术运算，或进行逻辑操作，最后结果保存在 Rd 中。逻辑右移指令 LSR 的操作过程如图 3-9 所示。图中移位次数为 3。

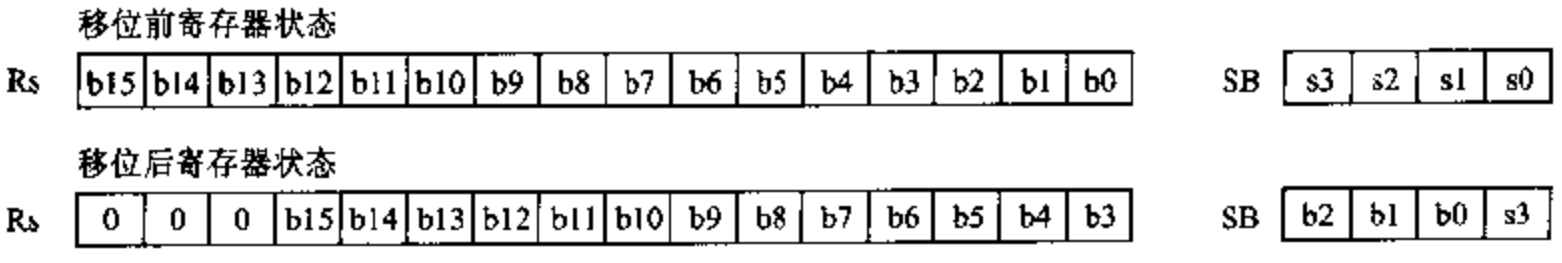


图 3-9 LSR 操作过程示意

逻辑右移指令 LSR 的服务对象是无符号数。在移位过程中，第一次移出的位是 Rs 的最低位，它进入 SB 的最高位，Rs 空出的最高位被填入 0，SB 移出的最低位丢失。随后的移位也按同样的过程进行。图中给出了 3 次移位结束后寄存器的状态。

表 3-16 给出了所有逻辑右移指令。

表 3-16 逻辑右移指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=Rs LSR *nn*,{Carry} Rd-=Rs LSR *nn*,{Carry} CMP Rd, Rs LSR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	√	√
Rd=-Rs LSR *nn* Rd&=Rs LSR *nn* Rd|=Rs LSR *nn* Rd^=Rs LSR *nn* TEST Rd, Rs LSR *nn* Rd=Rs LSR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	—	—
说明：当 Rd 为 PC 时，执行周期是表中较长者，且移位后 SR 中的所有标志位不受影响											

下面给出部分指令：

```
R1 - =R? LSR 2, Carry        //((R1)+R2 逻辑右移 2 位+C) →(R1)
R2 - =R3 LSR 3, Carry        //((R2) -R3 逻辑右移 2 位- C)- →(R2)
R2 - =R3 LSR 4               //((R2)--R3 逻辑右移 4 位)→(R2)
CMP R3, R4 LSR 2             //(R3)-(R4 逻辑右移 2 位)
R2 = -R3 LSR 1               //(R3 逻辑右移 1 位求补) →(R2)
BP&-=R2 LSR 2                //((BP)^(R2 逻辑右移 2 位))→(BP)
R2 |=SP LSR 2                //((R2) ∨(SP 逻辑右移 2))→(R2)
R3 ^ =PC LSR 4               //((R3)⊕(PC 逻辑右移 4))→(R3)
TEST R4, PC LSR 3            //(R4)^(PC 逻辑右移 3)
R4=BP LSR 2                  //(BP 逻辑右移 2 位)→(R4)
```

[例 9] 比较 R1 的高 8 位与 R2 的低 8 位。

```
R1=R1 LSR 4
R1=R1 LSR 4                  //R1 右移 8 位
R1&=0x00ff                   //保留 R1 低 8 位
R2&-=0x00ff                  //保留 R2 低 8 位
CMP R1,R2
```

3. 算术右移指令 ASR

算术右移指令 ASR 的服务对象是有符号数，操作过程类似于逻辑右移指令，主要差别在于采用带符号扩展方式，对空出的源寄存器最高位进行符号扩展。当 Rs 最高位为“0”时，扩展位为“0”；当最高位为“1”时，扩展位置“1”。可见，在最高位为“0”且移位次数相同的情况下，执行算术右移指令和执行逻辑右移指令，对某操作数而言其移位结果是相同的。

算术右移指令 ASR 的操作过程如图 3-10 所示。图中 Rs 为源寄存器，SB 为移位缓冲器。假设 $nn=3$，即 3 位算术右移，E2、E1、E0 为最高有效位的符号扩展位。可见算术右移 1 位，相当于 Rs 除以 2。

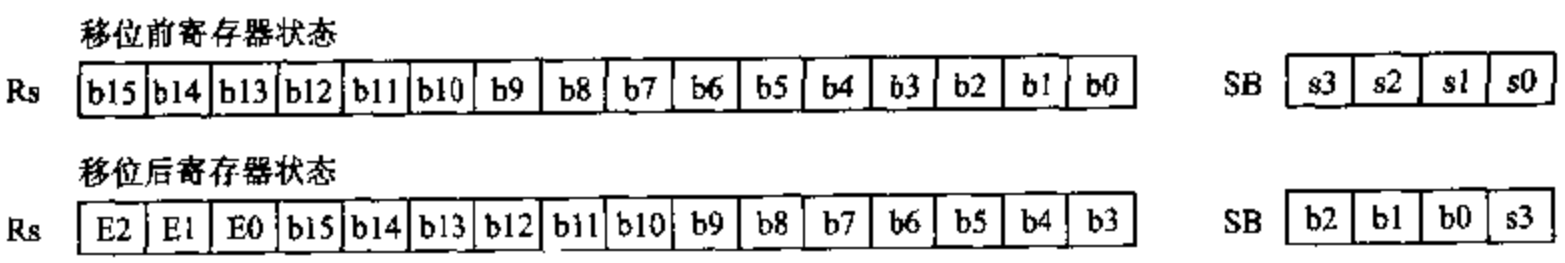


图 3-10　ASR 操作过程示意

表 3-17 给出了所有算术右移指令。

下面给出部分指令：

```
BP+ =R1 ASR 4. Carry         //((BP)+R1 算术右移 4 位+C)- →(BP)
R2 - =R3 ASR 1, Carry        //((R2)-R3 算术右移 1 位-C)- →(R2)
R2 - =-R3 ASR 4              //((R2)-R3 右移 4 位)→(R2)
CMP R3, R4 ASR 2             //(R3)-(R4 算术右移 2 位)
R2-- -R3 ASR 1               //(R3 算术右移 1 位后求补)→(R2)
BP&-=R2 ASR 2                //((BP)^(R2 算术右移 2 位)) →(BP)
R2 |=SP ASR 1                //((R2) ∨(SP 算术右移 1 位))→(R2)
```

R3 ^ =PC ALSR 4　　';((R3)⊕(PC右移4))→(R3)

TEST　R4, PC ASR 3　　';(R4)^(PC右移3)

R4 -BP ASR 2　　';(BP算术右移2位)→(R4)

表3-17　算术右移指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=Rs ASR *nn*,{Carry} Rd-= Rs ASR *nn*,{Carry} CMP Rd, Rs ASR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	√	√
Rd= -Rs ASR *nn* Rd&=Rs ASR *nn* Rd |= Rs ASR *nn* Rd ^ =Rs ASR *nn* TEST Rd, Rs ASR *nn* Rd=Rs ASR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√		
说明：当Rd为PC时，执行周期是表中较长者，且移位后SR中的所有标志位不受影响											

六、循环指令

循环指令共有两种：左循环、右循环。该指令只能采用寄存器寻址，μ'nSP™的CPU中的移位缓冲器SB参与了所有的循环操作，每条指令一次最多能循环4位。该指令除了循环功能外，还具有数据传送、算术运算、逻辑操作功能，即具有双重功能。循环指令对标志位的影响与循环后的操作对标志位的影响一致。

1. 左循环指令ROL

左循环指令ROL的功能是：先对源寄存器Rs和移位寄存器SB进行*nn*(可以设置为1～4)位左移循环，然后再和目的寄存器Rd之间或进行传送数据，或进行算术运算，或进行逻辑操作，最后结果保存在Rd中。左循环ROL的操作过程如图3-11所示。图中设*nn*=3，即3位左循环。

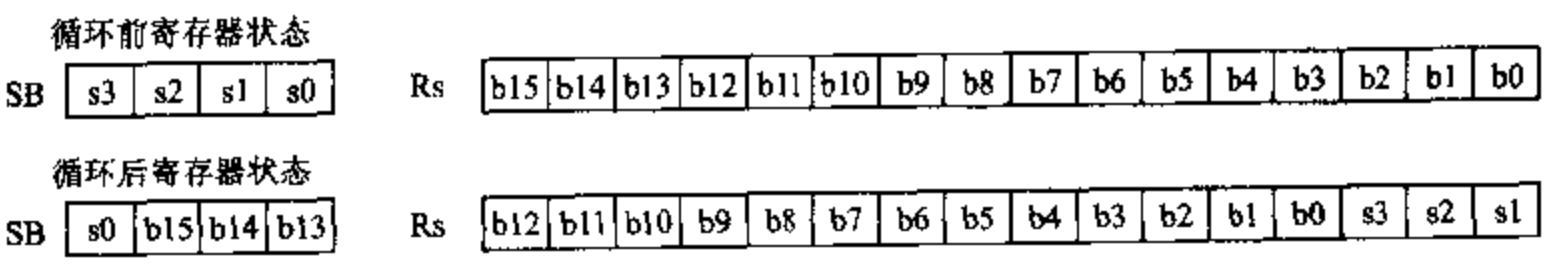


图3-11　ROL操作过程示意

在循环过程中，Rs和SB组成寄存器对循环移位，第一次Rs的最高位移入SB的最低位，而SB的最高位移入Rs的最低位，随后的移位也按同样的过程进行。图中给出了3次移位结束后寄存器的状态。

表3-18给出了所有左循环指令。

表 3-18　左循环指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=Rs ROL *nn*,{Carry} Rd−= Rs ROL *nn*,{Carry} CMP Rd, Rs ROL *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	√	√
Rd= −Rs ROL *nn* Rd&=Rs ROL *nn* Rd |= Rs ROL *nn* Rd ^ =Rs ROL *nn* TEST Rd, Rs ROL *nn* Rd=Rs ROL *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√		−
说明：当 Rd 为 PC 时，执行周期是表中较长者，且循环后 SR 中的所有标志位不受影响											

下面给出部分指令：

```
R1+=R2 ROL 2, Carry      //((R1)+R2 左循环 2 位+C)→(R1)
R1+=R2 ROL 4             //((R1)+R2 左循环 4 位)→(R1)
R2−=R3 ROL 3, Carry      //((R2)−R3 左循环 3 位−C)→(R2)
R2−=R3 ROL 4             //((R2)−R3 左循环 4 位)→(R2)
CMP R3, R4 ROL 2         //(R3)−(R4 左循环 2 位)
R2=−R3 ROL 1             //(R3 左循环 1 位求补)→(R2)
BP&=R2 ROL 2             //((BP)^(R2 左循环 2 位))→(BP)
R2 |=SP ROL 1            //((R2)∨(SP 左循环 1 位))→(R2)
R3 ^=PC ROL 3            //((R3)⊕(PC 左循环 3 位))→(R3)
TEST  R4, PC ROL 2       //(R4)^(PC 左循环 2 位)
R4=BP ROL 2              //(BP 左循环 2 位)→(R4)
```

2. 右循环指令 ROR

右循环指令 ROR 的功能与左循环指令的功能类似，惟一的区别就是向相反的方向循环，即向右循环，其余两者完全相同。右循环指令 ROR 的操作过程如图 3-12 所示。图中设 *nn*=3，即 3 位右循环。

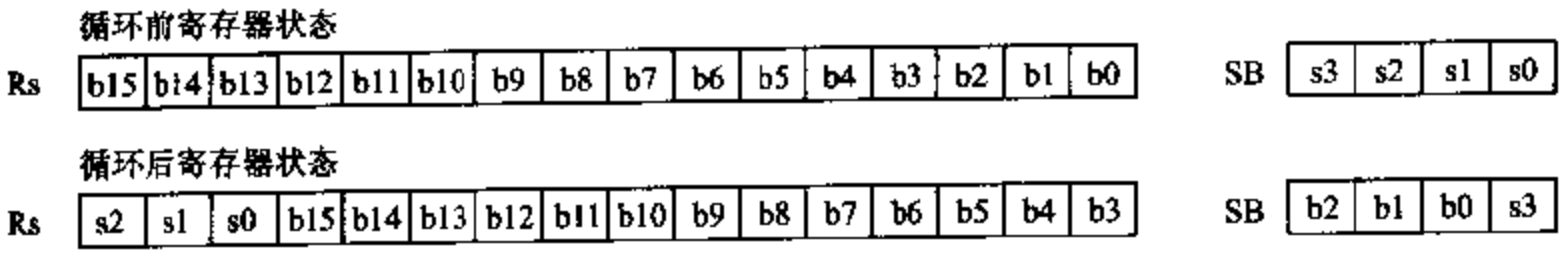


图 3-12　ROR 操作过程示意

表 3-19 给出了所有右循环指令。

表 3-19 右循环指令

指令语法	指令格式						指令周期	标志位状态			
	第一字					第二字		N	Z	S	C
Rd+=Rs ROR *nn*,(Carry) Rd-=Rs ROR *nn*,(Carry) CMP Rd, Rs ROR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	√	√
Rd=-Rs ROR *nn* Rd&=Rs ROR *nn* Rd|=Rs ROR *nn* Rd^=Rs ROR *nn* TEST Rd, Rs ROR *nn* Rd=Rs ROR *nn*	xxxx	Rd	xx	*nn*	Rs	无	3/8	√	√	-	-
说明：当 Rd 为 PC 时，执行周期是表中较长者，且循环后 SR 中的所有标志位不受影响											

下面给出部分指令：

```
R1+=R2 ROR 2, Carry      //((R1)+R2 右循环 2 位+C)→(R1)
R1+=R2 ROR 4             //((R1)+R2 右循环 4 位)→(R1)
R2-=R3 ROR 3, Carry      //((R2)-R3 右循环 3 位-C)→(R2)
R2-=R3 ROR 4             //((R2)-R3 右循环 4 位)→(R2)
CMP  R3, R4 ROR 2        //(R3)-(R4 右循环 2 位)
R2=-R3 ROR 1             //(R3 右循环 1 位求补)→(R2)
BP&=R2 ROR 2             //((BP)∧(R2 右循环 2 位))→(BP)
R2|=SP ROR 1             //((R2)∨(SP 右循环 1 位))→(R2)
R3^=PC ROR 3             //((R3)⊕(PC 右循环 3 位))→(R3)
TEST  R4, PC ROR 2       //(R4)∧(PC 右循环 2 位)
R4=BP ROR 2              //(BP 右循环 2 位)→(R4)
```

在使用移位和循环指令时需注意：

(1) 快速中断请求 FIQ、中断请求 IRQ 以及用户程序有它们自己的移位缓冲区，用户不必为中断保存移位缓冲区。

(2) 移位缓冲区 SB 在移位和循环操作中只起中间缓存的作用，其值在运算前是不确定的，用户不能对其进行任何读操作。用户要进行移位和循环操作时，要先对移位缓冲区 SB 进行初始化，即先执行移位或循环指令，使 SB 中的内容为用户所期望，然后再进行移位和循环操作。

(3) 执行乘法指令 Mul 和内积运算指令 Muls，移位缓冲区的内容是未知的。

3.6 控制转移及设置类指令

控制转移类指令用于控制程序的走向，其作用区间必然是程序存储器空间。μ'nSP™ 指令系统具有丰富的控制转移指令：利用子程序调用指令，可以访问 64 页代码段；利用无条件转

移指令，可以访问页内任一地址单元；利用短转移指令，可在前后 63 个字内转移。特别是丰富的条件转移指令，使得用户可灵活选择，给编程带来了很大的灵活性。软件中断指令给调试程序、编制程序带来许多方便。

控制转移类指令可分为 3 种：① 软件中断指令；② 程序转移指令；③ 子程序调用和返回指令。表 3－20 给出了所有控制程序转移类指令。

另外，μ'nSP™的指令系统还有专门的功能设置指令，用于设置 FIR 算法中的数据自动移动功能、中断的开放与禁止等，属于对 CPU 的控制。

下面分别给以介绍

一、软件中断指令

软件中断(或称为软件陷阱)为单字指令，助记符是 BREAK，采用 16 位直接地址寻址方式，指令周期为 13，不影响标志位，指令格式见表 3－20。其功能是产生软件中断，即利用软件发出中断请求，使 CPU 转到软件服务程序中去。当执行该指令时，CPU 会自动跳转到中断向量[0x00FFF5]处执行软件中断服务程序。软件中断服务程序的地址是固定的，该单元不得挪作它用。

例如：

```
BREAK                    //产生一个软件中断
```

表 3－20　控制程序转移类指令

指令语法	指令格式					指令周期	标志位状态			
	第一字				第二字		N	Z	S	C
BREAK	xxxxxxxxxxxxxxxx				无	13	—	—	—	
Jcond Label JMP Label	COND	xxxxxx	S	IM6	无	3/5			—	—
GOTO Label	xxxxxxxxxxxxxxxx				A16	12		—	—	—
CALL Label	xxxxxxxxxx			A6	无	13	—	—	—	—
RETF	xxxxxxxxxxxxxxxx				无	12	√	√	√	√
RETI	xxxxxxxxxxxxxxxx				无	12	√	√	√	√

二、程序转移指令

程序转移指令不影响标志位，它可分为近转移指令和远转移指令两种。

1. 近转移指令

(1) 条件转移：该类指令执行满足某种特定条件的转移，主要通过检测段寄存器 SR 中的各标志位是否满足要求，若条件满足，则程序跳转到指定的地址，否则顺序执行下一条指令。该条件转移指令采用 PC 相对寻址方式，所以，跳转范围限制在 PC±63 个字(前进或后退 63 个字的范围)，而不影响标志位。指令格式见表 3－20。当条件满足转移时，其指令周期为 5，否则继续执行下一条指令时，其指令周期为 3。

μ'nSP™的指令系统具有丰富的条件转移指令，给编程带来很大的灵活性。表 3－21 给

出了所有条件转移指令Jcond，表中包括指令格式中的操作码、指令助记符、操作数类型、条件说明及相应标志位的状态。

表3-21 近转移指令组

COND操作码	指令助记符	操作数类型	条件描述	标志位状态
0000	JCC	-	清进位位	C=0
0000	JB	无符号数	小 于	C=0
0000	JNAE	无符号数	不大于等于	C=0
0001	JCS		进位位置1	C=1
0001	JNB	无符号数	不小于	C=1
0001	JAE	无符号数	大于等于	C=1
0010	JSC		清符号位	S=0
0010	JGE	有符号数	大于大于	S=0
0010	JNL	有符号数	不小于	S=0
0011	JSS		符号位置1	S=1
0011	JNGE	有符号数	不大于等于	S=1
0011	JL	有符号数	小 于	S=1
0100	JNE	-	不等于	Z=0
0100	JNZ	—	非 零	Z=0
0101	JZ	—	零	Z=1
0101	JE	—	相 等	Z=1
0110	JPL	—	为 正	N=0
0111	JMI		为 负	N=1
1000	JBE	无符号数	小于等于	非(Z=0且C=1)
1000	JNA	无符号数	不大于	非(Z=0且C=1)
1001	JNBE	无符号数	不小于等于	Z=0且C=1
1001	JA	无符号数	大 于	Z=0且C=1
1010	JLE	有符号数	小于等于	非(Z=0且S=0)
1010	JNG	有符号数	不大于	非(Z=0且S=0)
1011	JNLE	有符号数	不小于等于	Z=0且S=0
1011	JG	有符号数	大 于	Z=0且S=0
1100	JVC	有符号数	无溢出	N=S
1101	JVS	有符号数	溢 出	N!=S
1110	JMP	—	无条件转移	—

例如：

```
R1+= 0x24                //((R1)+0x24)→(R1)
```

```
JCC   Label1          //无进位,C=0,转移到标号 Label1 处
CMP   R2, [0x2400]    //比较 R2 和零页[0x2400]单元中的数据大小
JNE   Label3          //不相等,转移到标号 Label3 处
```

(2) 无条件近转移指令:无条件近转移指令控制程序从现行地址直接转移到目标地址,采用 PC 相对寻址,转移范围为向前或后退 63 个字。该指令只有一条,不影响标志位,指令周期为 5,助记符为 JMP,指令格式见表 3-20,操作码一并放在了表 3-21 中。

例如:

```
JMP   Label2          //无条件转移到标号 Label2 地址处
```

2. 无条件远转移指令

无条件远转移指令控制程序从现行地址直接转移到目标地址。该指令采用 16 位直接地址[A16]寻址方式,所以执行该指令可以使程序从当前地址转移到页内 64 KB 程序存储器空间的任何单元,也就是说其转移范围为页内 64K 个字。该指令只有一条,不影响标志位,助记符为 GOTO,指令格式见表 3-20。

例如:

```
CMP   R1, 0x1         //比较 R1 和 0x1
JE    Case1           //若(R1)=0x1,则转移到标号 Case1 地址处
GOTO  Label3          //无条件转移到标号 Label3 地址处
```

三、子程序调用和返回指令

1. 子程序调用指令

该指令用于从指定的代码段调用子程序,采用 22 位直接地址[A22]寻址方式。所以,执行该指令可以在整个程序存储空间即 64 页代码段内调用任何指定的子程序。该指令在执行时,先修改程序指针 PC,待获得下一条指令地址后,自动地把此时 PC 和段寄存器 SR 的内容压入堆栈并保护起来,以便使子程序返回后能顺序执行;然后把目标地址的低 16 位装入 PC 并指向 SR 中的代码段 CS 所指定的代码页,去执行子程序。可以看出,该子程序的地址是由 PC 和 SR 中的 CS 段共同确定的,用户编制的子程序可以放在 64 页代码段中的任何一页。该指令只有一条,不影响标志位,助记符为 CALL,指令格式见表 3-20。

例如:

```
CALL  sub_1           //调用子程序 sub_1
```

2. 子程序返回指令

该指令和 CALL 指令对应,表示子程序已结束,需要返回主程序,也采用 22 位直接地址[A22]寻址方式。执行该指令时将自动分别弹出调用子程序时压入的段寄存器 SR 和程序指针 PC,使程序从调用指令 CALL 的下一条指令处开始继续执行主程序。从子程序返回必须使用该指令,因自动弹出段寄存器 SR,所以影响所有标志位。该指令的助记符为 RETF,指令格式见表 3-20。

例如:

```
sub_1:                //子程序 sub_1 开始
......
RETF                  //从子程序返回
```

3. 中断返回指令

该指令与RETF指令类似，区别在于从子程序返回时必须使用RETF，而从中断子程序返回时必须使用该指令，其操作过程与RETF相同，且在中断返回的同时清除内部相应的FIQ或IRQ的中断请求标志，助记符为RETI，它影响所有标志位，指令格式见表3-20。

例如：

```
IRQ_1:                    ; 中断子程序IRQ_1开始
……
RETI                      ;从中断子程序返回
```

四、功能设置指令

该类指令对标志位都无影响，可分为以下4种。

(1) FIR_MOV：该指令用于设置允许或禁止FIR算法过程中的数据自动移动功能(见Muls指令)。该指令影响着FIR算法的行为，并且具有全局性，在使用中需注意，尤其是在中断服务子程序中使用时要格外小心。该指令有两条(指令格式见表3-22)：

```
FIR_MOV   ON              ;允许FIR算法过程中的数据自动移动
FIR_MOV   OFF             ;禁止FIR算法过程中的数据自动移动
```

表3-22 功能设置类指令

指令语法	指令格式			指令周期	标志位状态			
	第一字		第二字		N	Z	S	C
FIR_MOV ON FIR_MOV OFF	xxxxxxxxxxxxxxxx		无	3				
FIQ ON FIQ OFF	xxxxxxxxxxxxxx	F I	无	3	-	-	—	—
IRQ ON IRQ OFF	xxxxxxxxxxxxxx	F I	无	3	—	—	—	—
INT FIQ INT IRQ INT FIQ, IRQ INT OFF	xxxxxxxxxxxxxx	F I	无	3	—	—		
NOP	xxxxxxxxxxxxxxxx		无	5	—	—	—	—

(2) FIQ：该指令用于设置允许或禁止快速中断请求FIQ。该指令有如下两条，指令格式见表3-22。

```
FIQ   ON                  ;允许FIQ中断
FIQ   OFF                 ;禁止FIQ中断
```

(3) IRQ：该指令用于设置允许或禁止中断请求IRQ。该指令有如下两条，指令格式见表3-22。

```
IRQ   ON                  ;允许IRQ中断
IRQ   OFF                 ;禁止IRQ中断
```

(4) INT:该指令用于设置中断的允许与禁止,指令格式见表 3-22,具体指令如下:

```
INT   FIQ                ;允许 FIQ 中断,禁止 IRQ 中断
INT   IRQ                ;允许 IRQ 中断,禁止 FIQ 中断
INT   FIQ, IRQ           ;允许 FIQ、IRQ 中断
INT   OFF                ;禁止 FIQ、IRQ 中断
FIQ   OFF                ;禁止 FIQ 中断
```

由上面介绍可以看出:指令 FIQ、IRQ、INT 相当于整体与局部的关系,FIQ 指令只对快速中断请求 FIQ 有控制作用,IRQ 指令只对中断请求 IRQ 有控制作用,而 INT 指令对 FIQ 中断和 IRQ 中断同时有控制作用。例如:指令 FIQ　ON,允许 FIQ 中断,对 IRQ 中断无影响;而指令 INT FIQ,在允许 FIQ 中断的同时将 IRQ 中断关闭。

表 3-22 指令格式中的 F、I 分别对应 FIQ、IRQ 中断。当 F=1 时,允许 FIQ 中断;当 F=0时,禁止 FIQ 中断。当 I=1 时,允许 IRQ 中断;当 I=0 时,禁止 IRQ 中断。

(5) NOP:该指令为空操作,是对 CPU 的控制指令。执行本指令时机器不作任何操作,而转向下一条指令去执行,不影响任何寄存器和标志位。该指令常用于精确延时或时间上的等待。

例如,用下列程序,给 R1 赋不同的初值,就可以实现不同的延时:

```
Delay_Loop:                 延时子程序
    NOP                     空操作
    CMP R1, 0xFFFF          ;比较 R1 和 0xFFFF
    JAE Exit_Loop           ;若 R1≥0xFFFF,则转移到标号 Exit_Loop 处,退出延时程序
    R1=R1-1                 ;否则,R1=R1-1
    JMP Delay_Loop          ;跳转到 Delay_Loop,继续延时
```

应用举例:

[例 10]　编写计算 1～100 所有整数和的函数,要求结果存于 R1 中。

```
F_CalculateSum:
  R1=0                      ;数据传送指令
  R2=0
    L_SumLoop:
      R1+=R2                ;加法指令
      R2+=1
      Cmp R2,0x65           ;比较指令,如果小于 100,继续
      Jb L_SumLoop
  Retf                      ;函数返回
```

[例 11]　编写计算 1～200 所有奇数和的函数,要求结果存于 R1 中。

```
F_CalculateOddSum:
  R1=0
  R2=0
  L_SumOddLoop:
    Test R2,0x01            ;测试奇偶
```

```
    Jnz L_AddIt                 //奇数转
    R2+=1
    Cmp R2,200
    Jnb L_LoopFinish            //不小于结束
  L_AddIt:
    R1+=R2
    R2+=1
    Jmp L_SumOddLoop
  L_LoopFinish:
    Retf                        //函数返回
```

[例 12] 利用 IRQ1 中断服务程序完成数据采集和滤波，并将结果从 DAC 输出。

```
_IRQ1:                          //IRQ1 中断子程序开始
    PUSH R1, R5 to [SP]         //将 R5~R1 压入堆栈，并保护现场
    CALL F_IRQ1_Serivce         //调用 IRQ1 服务子程序
    POP R1, R5 from [SP]        //将 R1~R5 弹出，并恢复现场
    RETI                        //中断子程序返回
F_IRQ1_Serivce::                //IRQ1 服务子程序开始
    R1 = Data_Entry             //R1 指向采样数据单元
    R2=Conf_Entry               //R2 指向系数单元
    FIR_MOV ON                  //允许 FIR 运算过程中采样数据自动移动
    MR=[R1]*[R2], n             //执行 n 阶 FIR 运算
    FIR_MOV OFF                 //关闭 FIR 运算数据自动移动功能
    R4=R4 LSL 4, Carry          //移位 11 位获得 16 位输出量
    R3=R3 ROR 4, Carry          //R4 的低 4 位移至 R3 高 4 位
    R4=R4 LSL 4, Carry          //再移 4 位
    R3=R3 ROR 4, Carry          //MR 移位 8 位
    R4=R4 LSL 4, Carry          //再移 4 位
    R3=R3 ROR 3, Carry          //MR 移位 11 位
    [P_DAC1]=R3                 //将 R3 中的数据输出至 DAC1 口
```

习题与思考题

1. μ'nSP™指令系统有几种寻址方式？试举例说明。
2. 装载和存储数据指令有何特点？指令[--R5]=0x3800 是否正确？
3. μ'nSP™的堆栈操作方式与一般 CPU 有何不同？若要将 R1、R3、R5 压入堆栈，应如何编程？若压入的是 R1~R5 或 R1~R3、R5，又应如何编程？
4. 若 R1 中数据是 0x61B3，R2 中数据是 0x5AC7，执行 R1=R2+0x2400 指令后，标志位 N、Z、S 和 C 的状态如何？
5. μ'nSP™指令系统包括哪些逻辑操作指令？
6. 在子程序调用时，哪些寄存器的状态自动保存？

7. 乘法指令是否可用寄存器 R3、R4 寻址？内积运算指令是否可用寄存器 R3、R4 寻址？为什么？执行乘法指令和内积运算指令时，是否都需要将 R3、R4 清零？为什么？
8. 若要将 RAM 单元中连续 32 个数据求平均值，用内积指令应如何编程？
9. μ'nSP™指令系统的移位指令和循环指令有何特点？若要将寄存器 R1 中的最高 4 位和最低 4 位交换位置，应如何编程？
10. 说明 FIQ、IRQ 和 INT 指令有何异同点。若将指令 IRQ ON 和 FIQ OFF 合并为一条指令，应如何完成？

第四章　单片机片内外设部件

4.1　并行 I/O 口

并行接口是单片机 CPU 与外部设备交换信息的桥梁。μ'nSP™有两个 16 位的并行 I/O 口：A 口和 B 口。有 32 根 I/O 线，其中，A 口 16 根，用 IOA0～IOA15 表示；B 口 16 根，用 IOB0～IOB15 表示。它们都是 16 位可编程输入/输出口，且口的每一位都可单独编程，并定义为输入或输出。其中：A 口是通用 I/O 口，且 IOA0～IOA7 具有唤醒功能；B 口除作为通用 I/O 口外，IOB0～IOB10 还具有特殊功能。

一、并行 I/O 口的结构

图 4－1 为 μ'nSP™单片机并行 I/O 口的一位结构框图。它由输出数据寄存器及控制逻辑、三态缓冲器、一对 FET(场效应管)组成的输出驱动电路、数据缓冲寄存器及读寄存器和读引脚控制电路等部分组成。

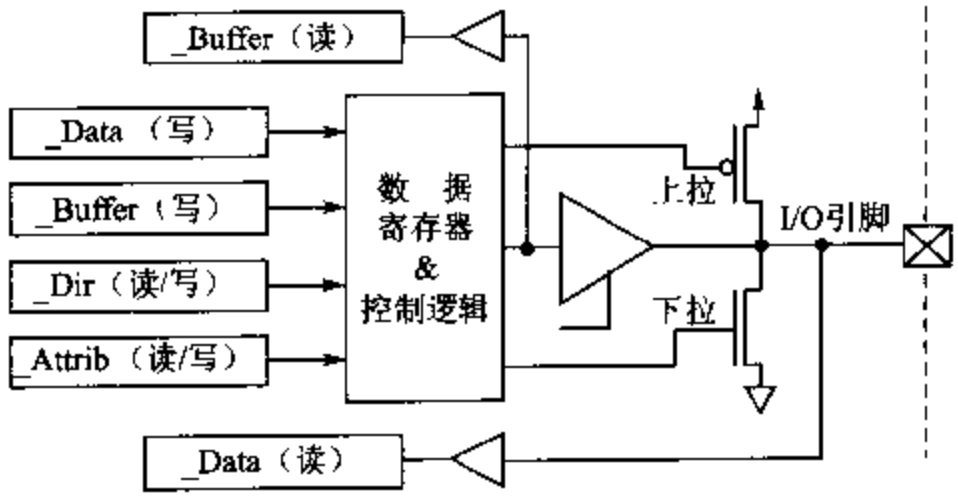


图 4－1　并行 I/O 口的一位组织结构

由一对 FET 组成的输出驱动电路，使得 I/O 的输入、输出方式设置非常灵活：作为输入时可设置为内部带上拉电阻、内部带下拉电阻或悬浮式的输入端口三种方式；而作为输出时可设置为常规的 CMOS 输出端口或 NMOS 开漏输出端口。图中_Data、_Buffer、_Dir 和_Attrib 为 I/O 口的数据及 I/O 控制寄存器。

二、并行 I/O 口数据寄存器及其设置

1. 数据寄存器

μ'nSP™并行口有两个：A 口和 B 口。而每个口都有相应的数据口_Data、数据缓冲寄存器_Buffer、方向寄存器_Dir 和属性寄存器_Attrib。此外，A 口还有一个锁存寄存器 P_IOA_Latch，B 口还有一个反馈寄存器 P_FeedBack。

A 数据口 P_IOA_Data(＄7000H)－ －可读/写。芯片经 I/O 引脚与外界交换信息并由

此 A 数据口完成。P_IOA_Data 为其符号地址名，而物理地址为 $7000H，以后各寄存器表示方法都相同，不再重复。当 A 口设置为输入口时，CPU 对此口有读/写两种操作：读操作是 CPU 读入其引脚输入电平，也称为输入操作；而写操作是 CPU 将命令字数据写到 A 口的数据寄存器中，用于设置 I/O 口的输入方式。当 A 口设置为输出口时，只有写操作，即 CPU 将输出数据写到 A 口数据寄存器中，并从引脚输出。

A 口数据缓冲寄存器 P_IOA_Buffer($7001H)　--可读/写。写操作时，CPU 将数据写到 A 口的数据寄存器中；读操作则是 CPU 从 A 口数据寄存器中读出数据。可以看出，此时的读/写操作仅在芯片的内部进行，没有与芯片外部交换信息。

由图 4-1 和以上介绍可知，向 P_IOA_Data 和 P_IOA_Buffer 写数据时，数据最终将被写入同一个数据寄存器中(图 4-1 中的数据寄存器)。但有两种不同的读操作，即读出数据来自不同的位置。当读 P_IOA_Data 时，是读 A 口芯片外的引脚即 I/O 口状态，而读 P_IOA_Buffer 时，是读片内 A 口数据寄存器中的内容。

A 口方向寄存器 P_IOA_Dir($7002H)——可读/写。该寄存器用于选择 I/O 口是作为输入口还是作为输出口使用。写操作是 CPU 将 A 口的方向向量写到 A 口的方向寄存器中；读操作则是 CPU 从 A 口的方向寄存器中读出 A 口的方向向量。

A 口属性寄存器 P_IOA_Attrib($7003H)——可读/写。A 口作为 I/O 口使用时，它有不同的工作方式，该寄存器用于选择 I/O 口输入或输出时的工作方式。写操作是 CPU 将 A 口的属性向量写到 A 口的属性寄存器中；读操作则是 CPU 从 A 口的属性寄存器中读出 A 口的属性向量。

与上述 A 口的寄存器相对应，B 口也有 P_IOB_Data($7005H)、P_IOB_Buffer($7006H)、P_IOB_Dir($7007H)和 P_IOB_Attrib($7008H)，它们的读/写操作模式和功能与 A 口完全一样。

A 口锁存寄存器 P_IOA_Latch($7004H)——只能读。从其读数可激活 A 口的唤醒功能，并锁存 IOA0～IOA7 上的数据。A 口的 IOA0～IOA7 常作为唤醒源，用于键盘输入。要激活 IOA0～IOA7 的唤醒功能，必须读 P_IOA_Latch 寄存器，以此来锁存 IOA0～IOA7 引脚上采集的数据。随后，系统才可通过指令进入低功耗的睡眠状态，锁存 IOA0～IOA7 采集的引脚电平状态。当有键按下时，IOA0～IOA7 的输入状态将不同于其在进入睡眠前被锁存时的状态，从而引起系统的唤醒。

B 口反馈寄存器 P_FeedBack($7009H)—— 只能写。该寄存器用于 B 口中 IOB2～IOB5 及 IOB8 的特殊功能选择控制，即选择 B 口中这几位是作为普通的并行 I/O 口用，还是作为特殊功能口用。

为了方便，将各寄存器功能总结于表 4-1 中。

2. 并行 I/O 口的设置

μ'nSP™具有位设置功能，即每一个 I/O 口的每一位可以单独定义，并设置有关的包括方向(Direction)寄存器_Dir(P_IOA_Dir、P_IOB_Dir)、属性(Attribution)寄存器_Attrib(P_IOA_Attrib、P_IOB_Attrib)和数据(Data)口_Data(P_IOA_Data、P_IOB_Data)三个控制口。这三个寄存器中每个对应的位组合在一起形成一个控制字，用来定义相应 I/O 口某位的输入/输出状态和方式。

表 4-1 并行 I/O 口寄存器

寄存器	读/写属性	地 址	功能说明
P_IOA_Data	读/写	7000H	A口的数据口。当A口为输入口时,CPU对其执行读时,是读入其引脚输入电平;写操作时,是CPU将命令字数据写到A口的数据寄存器中。当A口为输出口时,写操作是将输出数据写到A口数据寄存器中,并从引脚输出
P_IOA_Buffer	读/写	7001H	A口数据缓冲寄存器。CPU对其执行写操作时,是将数据写到A口的数据寄存器中;读操作则是从A口数据寄存器中读出数据
P_IOA_Dir	读/写	7002H	A口方向寄存器,用于选择I/O口的功能。CPU对其执行写操作时,是将A口的方向向量写到A口的方向寄存器中;读操作则是读出A口的方向向量
P_IOA_Attrib	读/写	7003H	A口属性寄存器,用于选择I/O口输入或输出时的工作方式。CPU对其执行写操作时,是将A口的属性向量写到A口的属性寄存器中;读操作则是读出A口的属性向量
P_IOA_Latch	读	7004H	A口锁存寄存器。CPU从其读数可激活A口的唤醒功能,并锁存IOA0~IOA7上的数据
P_IOB_Data	读/写	7005H	B口的数据口。当B口为输入口时,CPU对其执行读操作时,是读入其引脚输入电平;写操作是CPU将命令字数据写到A口的数据寄存器中。当B口为输出口时,写操作是将输出数据写到B口数据寄存器中,并从引脚输出
P_IOB_Buffer	读/写	7006H	B口数据缓冲寄存器。CPU对其执行写操作时,是将数据写到B口的数据寄存器中;读操作则是从B口数据寄存器中读出数据
P_IOB_Dir	读/写	7007H	B口方向寄存器,用于选择I/O口的功能。CPU对其执行写操作时,是将B口的方向向量写到B口的方向寄存器中;读操作则是读出B口的方向向量
P_IOB_Attrib	读/写	7008H	B口属性寄存器,用于选择I/O口输入或输出时的工作方式。CPU对其执行写操作时,是将B口的属性向量写到B口的属性寄存器中;读操作则是读出B口的属性向量
P_FeedBack	写	7009H	B口反馈寄存器,用于B口特殊功能选择控制

(1) 方向向量_Dir 位:该位用于设置I/O口的对应位是用做输入位还是用做输出位,设置如下:

0——输入;

1——输出。

(2) 属性向量_Attrib 位:该位用于定义I/O口的工作方式,在I/O口位作为输入和输出时的作用不同。

当口位输入状态下,属性向量_Attrib 设置其是悬浮式输入还是非悬浮式输入,设置如下:

0——带上拉或下拉电阻式输入;

1——悬浮式输入。

当口位输出状态下，属性向量_Attrib 设置其输出是反相的还是同相的，设置如下：

0——经反相器反相输出；

1——经缓冲器同相输出。

(3) 数据向量_Data 位：在该位的输入状态下被写入时，与 Attrib 位组合在一起形成输入方式的控制字，其功能如下：

00——带唤醒功能的下拉电阻式输入；

01——带唤醒功能的上拉电阻式输入；

10——带唤醒功能的悬浮式输入；

11——无唤醒功能的悬浮式输入。

_Data 位在该位的输出状态下被写入的是输出数据，数据输出是同相还是经反相输出则取决于_Attrib 位的设置。

带下拉电阻式输入是该位的默认设置(即 3 个控制位的组合是“000”)，而唤醒功能只有 IOA0～IOA7 才具有。

表 4-2 给出了方向向量位_Dir、属性向量位_Attrib 以及数据口位_Data 的状态及并行 I/O口的相应工作方式。

表 4-2　并行 I/O 口某位的设置

_Dir	_Attrib	_Data	口位状态	口位方式描述	唤醒功能
0	0	0	输　入	带下拉电阻的输入口	有
0	0	1	输　入	带上拉电阻的输入口	有
0	1	0	输　入	悬浮的输入口	有
0	1	1	输　入	悬浮的输入口	无
1	0	0	输　出	经数据反相器输出高电平	无
1	0	1	输　出	经数据反相器输出低电平	无
1	1	0	输　出	经数据缓存器输出低电平	无
1	1	1	输　出	经数据缓存器输出高电平	无

I/O 口的设置简单灵活，如要把 A 口的 b0 位设置成下拉电阻式的输入口，则 A 口_Dir、_Attrib和_Data 的三个相应 b0 位都应设成“0”。若要把 A 口的 b1 位用做具有唤醒功能的悬浮式输入口，只需将_Dir、_Attrib 和_Data 向量中相应的 b1 组合设置为“010”即可。I/O 口的结构可以方便地改变其属性功能。如要把悬浮式输入(011)改变为高电平输出(111)，只需要将方向位由“0”改为“1”即可。

例如，要设置 IOA0～IOA3 为带下拉电阻输入，IOA4～IOA7 为带上拉电阻输入，IOA8～IOA11 为带缓冲的低电平输出，IOA12～IOA15 为带缓冲的高电平输出。A 口的初始化程序如下。

```
R1=0xF0F0                    //根据要求设置_Data 的值→R1
[P_IOA_Data]=R1              //将设置值→P_IOA_Data 寄存器
R1=0xFF00                    //根据要求设置_Attrib 的值→R1
[P_IOA_Attrib]=R1            //将设置值→P_IOA_Attrib 寄存器
```

```
R1=0xFF00                    //根据要求设置_Dir的值→R1
[P_IOA_Dir]=R1               //将设置值→P_IOA_Dir寄存器
```

可见，I/O的应用非常灵活，可以根据需要任意设置。如经过上述程序初始化后，此16位的I/O可以当作两个8位的I/O使用，低8位IOA0～IOA7作为8位输入口，高8位IOA8～IOA15作为8位输出口。

三、B口的特殊功能

B口的IOB0～IOB10除了可以用做普通的并行I/O口外，还提供了一些特殊专用功能。表4-3列出了IOB0～IOB10的特殊功能，这些功能必须经过对P_FeedBack寄存器进行初始化功能设置后方可使用。

表4-3 B口特殊功能

B口引脚	特殊功能	功能描述
IOB0	SCK	串行设备接口SIO时钟信号
IOB1	SDA	串行设备接口SIO数据信号
IOB2	EXT1	外部中断源1(负跳沿触发)，IOB2需设置为输入
	FeedBack_Output1	通过与IOB4之间的外接RC反馈电路组成一振荡电路，该振荡信号可作为外部中断源EXT1或从EXT1引脚输出
IOB3	EXT2	外部中断源2(负跳沿触发)，IOB3需设置为输入
	FeedBack_Output2	通过与IOB5之间的外接RC反馈电路组成一振荡电路，该振荡信号可作为外部中断源EXT2或从EXT2引脚输出
IOB4	FeedBack_Input1	反馈输入
IOB5	FeedBack_Input2	反馈输入
IOB6	IRRx	红外通信接口的数据接收信号
IOB7	Rx	通用异步串行口的数据接收信号
IOB8	APWMO	定时器A的PWM输出
	IRTx	红外通信接口的数据发送信号
IOB9	BPWMO	定时器B的PWM输出
IOB10	Tx	通用异步串口的数据发送信号

四、反馈寄存器P_FeedBack及设置

IOB2与IOB4、IOB3与IOB5及IOB8特殊功能的使用由反馈寄存器P_FeedBack进行设置。该寄存器各位的具体功能和格式如下：

b15～b4	b3	b2	b1	b0
…	FBKEN3	FBKEN2	…	IRTxEN

IRTxEN　红外通信使能：

0　禁止红外通信，此为默认设置；

1　允许 IOB8 的红外通信功能。

FBKEN2　IOB2 与 IOB4 之间反馈设置：

0　IOB2 与 IOB4 用做普通并行 I/O 口，此为默认设置；

1　IOB2 与 IOB4 之间形成反馈，作为振荡发生器使用。

FBKEN3　IOB3 与 IOB5 之间反馈设置：

0　IOB3 与 IOB5 用做普通并行 I/O 口，此为默认设置；

1　IOB3 与 IOB5 之间形成反馈，作为振荡发生器使用。

1. 振荡发生器

图 4-2 给出了 IOB2 与 IOB4、IOB3 与 IOB5 组成的 RC 反馈充放电的张弛振荡电路结构。施密特电路的输出信号反相后经 IOB2(或 IOB3)通过 RC 电路输入端充放电，使两种稳定状态不能长期保持，从而形成振荡信号。改变 RC 参数(充放电时间常数)，即可改变振荡周期。可见，只要在 IOB2 与 IOB4 之间或者 IOB3 与 IOB5 之间外加一个简单的 RC 电路，便可在 EXT1 或 EXT2 端得到振荡信号。同时该信号在单片机内作为中断源 EXT1 触发 IRQ3_EXT1 中断，也可以用做定时器的时钟输入。为了确保反馈电路正常工作，必须将 IOB2 或 IOB3 设置成反相输出，将 IOB4 或 IOB5 设置成悬浮输入。

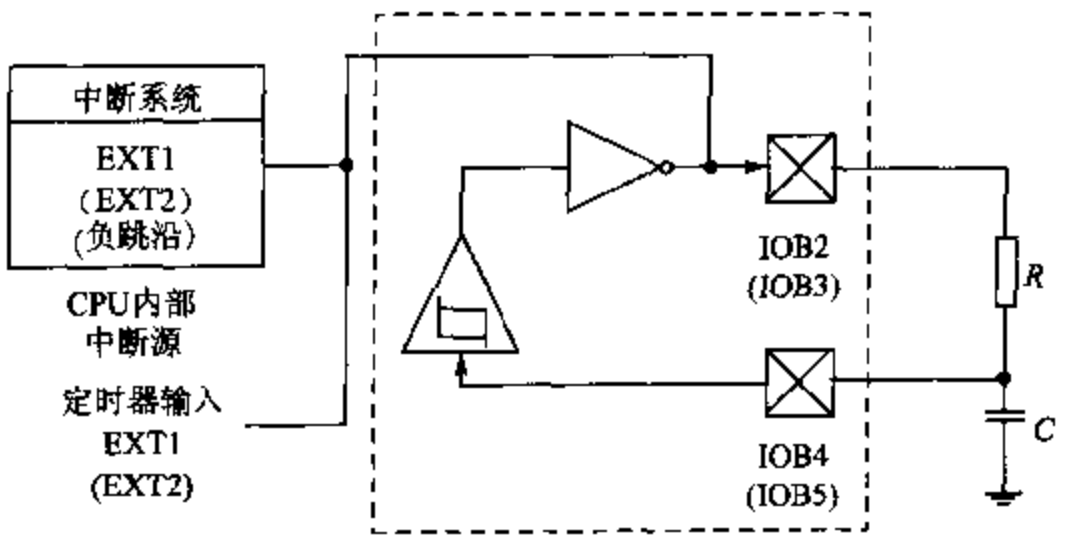


图 4-2　IOB2 与 IOB4、IOB3 与 IOB5 组成的振荡电路结构

例如，在 IOB2 和 IOB4 之间连接一个 RC 电路，通过执行下面的程序，将会在 IOB2 上得到一个方波振荡信号。

```
.R1=0x0004          //IOB4 设置成悬浮输入，IOB2 设置成反相输出，取方向向量值→R1
[P_IOB_Dir]=R1      //方向向量值→P_IOB_Dir 寄存器
R1=0x0010           //取属性向量值→R1
[P_IOB_Attrib]=R1   //属性向量值→P_IOB_Attrib 寄存器
R1=0x0004           //IOB2 与 IOB4 设置成反馈，设置值→R1
[P_FeedBack]=R1     //设置值→P_FeedBack 寄存器
```

2. 红外通信与串行通信控制位

上面已提到，反馈寄存器 P_FeedBack 的 b0 位 IRTxEN 是红外通信数据发送的使能控制位。该位与定时器/计数器 A 的 PWM 输出使能标志位 TAON(见 4.2 节)、通用异步串行通信发送使能信号 TxPinEN(见 4.5 节)共同决定了 IOB8 和 IOB10 是作为一般 I/O 口使用，还

是作为特殊功能口使用。具体如下：

TAON　　0　　IOB8用做普通并行I/O口；

　　1　　IOB8用做特殊功能口APWMO或IRTx。

当IOB8用做特殊功能口时，有两种应用方式：一种是用做定时器A的PWM输出口APWMO；另一种是用做红外通信数据发送口IRTx，由TxPinEN和IRTxEN组合控制。当这两位的组合值为"11"时，IOB8用做IRTx信号；为其它组合值时，则用做APWMO信号。图4-3给出了IOB8的输出组成结构示意框图，图中Tx为通用异步串行通信的数据发送信号。

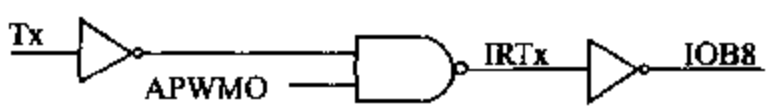


图4-3　IOB8输出组成结构框图

TxPinEN　　0　　IOB10用做普通并行I/O口；

　　1　　IOB10用做普通并行I/O口或特殊功能口Tx。

当TxPinEN设置为1时，IOB10的功能与IRTxEN的设置有关：若IRTxEN=0，即禁止红外通信，则IOB10用做通用异步串行通信的数据发送口Tx；若IRTxEN=1，即允许红外通信，则IOB10用做普通I/O口。

由上可见，IOB8和IOB10的应用方式是通过三个控制位组合控制的。表4-4给出了其控制位的状态和IOB8、IOB10相应的应用方式。

表4-4　IOB8和IOB10的应用方式控制

TAON	TxPinEN	IRTxEN	IOB8	IOB10
0	0	0	普通并行I/O口	普通并行I/O口
0	0	1	普通并行I/O口	普通并行I/O口
0	1	0	普通并行I/O口	Tx
0	1	1	普通并行I/O口	普通并行I/O口
1	0	0	APWMO	普通并行I/O口
1	0	1	APWMO	普通并行I/O口
1	1	0	APWMO	Tx
1	1	1	IRTx	普通并行I/O口

从上表可以看出，B口各位的特殊功能在控制、中断、定时/计数及通信时极为有用。只有在不用上述功能时，某些I/O口的位才能作为一般的输入/输出口使用。

4.2　时基系统与计数器/定时器CTC

在检测、控制及智能仪器等应用领域中，经常要求有一些外部实时时钟，以实现定时检测、定时或延时控制，或要求定时器产生一定宽度的脉冲，以驱动步进电机一类的电气机械。也经常要求有一些外部计数器，以实现对外部事件的计数。例如，转速的测量和频率的测量。另外，随着V/F器件的应用，可以将电压的测量转换为频率的测量，也要求有计数器。μ'nSP™

单片机提供了两个 16 位的定时器/计数器：Timer A 和 Timer B。除了检测频率以外，还常用计数器溢出信号产生定时中断的请求信号，用于需定时测控的场合。Timer A 和 Timer B 的硬件中断事件还可用于锁存 A/D 转换结果和 D/A 音频输出。

另外，由于系统时钟和定时器/计数器的部分输入时基信号以及 IRQ4、IRQ5、IRQ6 定时中断的时基信号都来自于时钟振荡器 32 768 Hz，所以，将系统时钟及时基信号一并在此介绍。

一、系统时钟

系统时钟的信号源为锁相环 PLL 振荡器。锁相环 PLL 用于为系统提供一个实时时钟的基频32 768 Hz，再将基频进行倍频作为系统时钟 f_{osc}。系统时钟 f_{osc} 经 n 次分频后产生 CPU 时钟信号 CPUCLK，系统时钟结构如图 4－4 所示。系统时钟 f_{osc} 和 CPU 时钟 CPUCLK 的设置由系统时钟控制寄存器 P_SystemClock 控制。

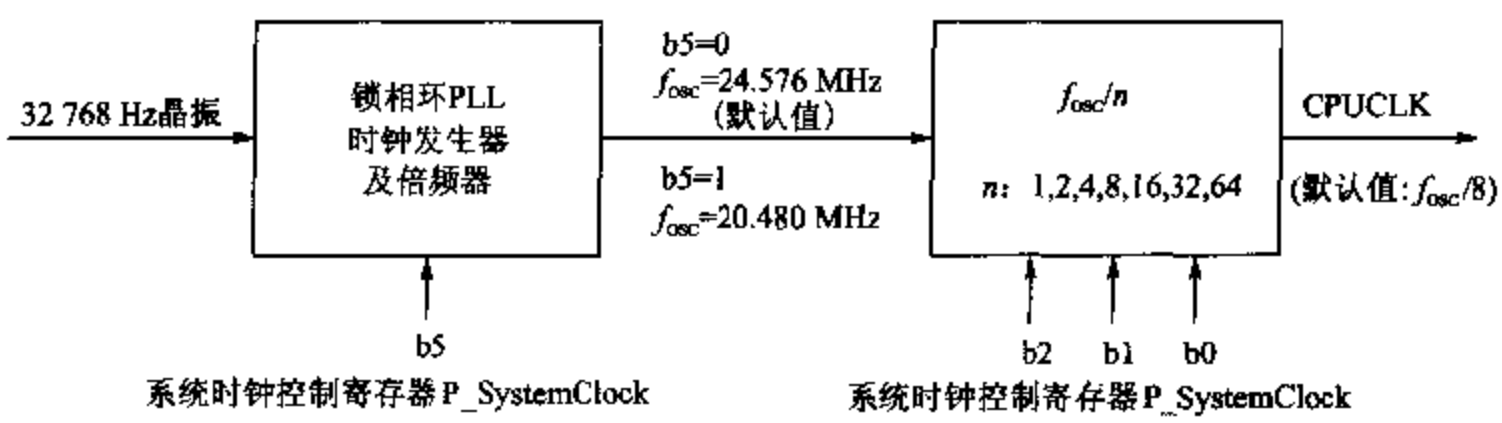


图 4－4　系统时钟结构框图

系统时钟控制寄存器 P_SystemClock（$7013H）——只能写。该寄存器用于选择系统时钟、CPU 时钟信号及工作方式，其各位的具体功能和格式如下：

b15～b6	b5	b4	b3	b2	b1	b0
…	系统时钟选择	32 kHz 睡眠状态	32 kHz 方式选择	CPU 时钟选择		

其中，CPU 时钟信号的选择如表 4－5 所列。

表 4－5　CPU 时钟选择

b2	b1	b0	CPU 时钟频率
0	0	0	f_{osc}
0	0	1	$f_{osc}/2$
0	1	0	$f_{osc}/4$
0	1	1	$f_{osc}/8$（默认设置）
1	0	0	$f_{osc}/16$
1	0	1	$f_{osc}/32$
1	1	0	$f_{osc}/64$
1	1	1	停止（睡眠状态）

b3、b4、b5 位的含义如下：

b3—— 32 kHz 方式选择：

0　　32 768 Hz 时钟处于自动弱振模式，系统默认设置；

1　　32 768 Hz 时钟处于强振模式。

b4——32 kHz 睡眠状态选择：

0　　在备用状态下，32 768 Hz 时钟被关闭；

1　　在备用状态下，32 768 Hz 时钟仍处于工作状态，系统默认设置。

b5—— 系统时钟选择：

0　　系统时钟 $f_{osc}=24.576$ MHz，系统默认设置；

1　　系统时钟 $f_{osc}=20.480$ MHz。

32 768 Hz 振荡器有两种工作模式：强振模式和自动弱振模式。处于强振模式时，32 768 Hz 振荡器始终运行在高能耗的状态下；处于自动弱振模式时，系统在上电复位后的前 7.5 s 内处于强振模式，然后自动切换到弱振模式以降低功耗。CPU 被唤醒后默认的时钟频率为 $f_{osc}/8$，用户可以根据需要调整该值，这样可以避免在系统被唤醒后造成 ROM 读取错误。

当系统时钟控制寄存器 P_SystemClock 的第 0～2 位设置为“111”时，将使 CPU 时钟停止工作，系统切换至低功耗的备用状态。在备用状态下，通过设置该寄存器的 b4 位可以接通或关闭 32 kHz 实时时钟。

二、时基信号及其控制

CPU 的时基信号发生器是最基本的定时脉冲信号，它可以为单片机提供各种不同的频率信号，其组成结构框图如图 4－5 所示。它的输入信号是实时时钟振荡器 32 768 Hz（见图 4－6），32 768 Hz 经 14 级 2 分频器产生各种时间基准信号，也叫实时时钟信号 RTC（Real Time Clock）。它用于产生各种实时时钟信号，输出有两个时基信号即 TMB1、TMB2 和2 Hz、4 Hz、1 024 Hz、2 048 Hz、4 096 Hz 等频率信号。TMB1 提供 8 Hz、16 Hz、32 Hz、64 Hz 的 4 种频率信号之一，TMB2 提供 128 Hz、256 Hz、512 Hz、1 024 Hz 的 4 种频率信号之一。TMB1 和 TMB2 的输出作为中断源 IRQ6，可以为系统提供实时中断源，同时 TMB1 还为定时器/计数器 A 的时钟源 ClkB 提供可选择的频率信号。2 Hz、4 Hz 作为中断源 IRQ5，为系统提供0.5 s、0.25 s 的实时中断信号；1 024 Hz、2 048 Hz、4 096 Hz 作为中断源 IRQ4，为系统提供1 ms、0.5 ms、0.25 ms 的实时中断信号，这些实时中断源都可用于精确的实时控制。

由上可见，μ'nSP™ 为用户提供了 7 种实时中断源，范围从 0.25 ms～0.5 s，这对需要实时跟踪检测、控制的应用场合特别方便，例如，中断源 IRQ5_2 Hz 可应用于系统的睡眠时间控制，即在 0.5 s 之内若无键盘输入的变化，系统便会进入低功耗的睡眠状态。系统的定时信号也可以用于精确的时间计数。

为了设置相应的定时信号，必须对时基信号的控制寄存器进行设置，相关寄存器有两个：

(1) 时基设置寄存器 P_TimeBase_Setup（$700EH）—— 只能写。该寄存器用于选择 TMB1、TMB2 的输出频率，其各位的具体功能和格式如下：

b15～b4	b3	b2	b1	b0
…	TMB2 频率选择位		TMB1 频率选择位	

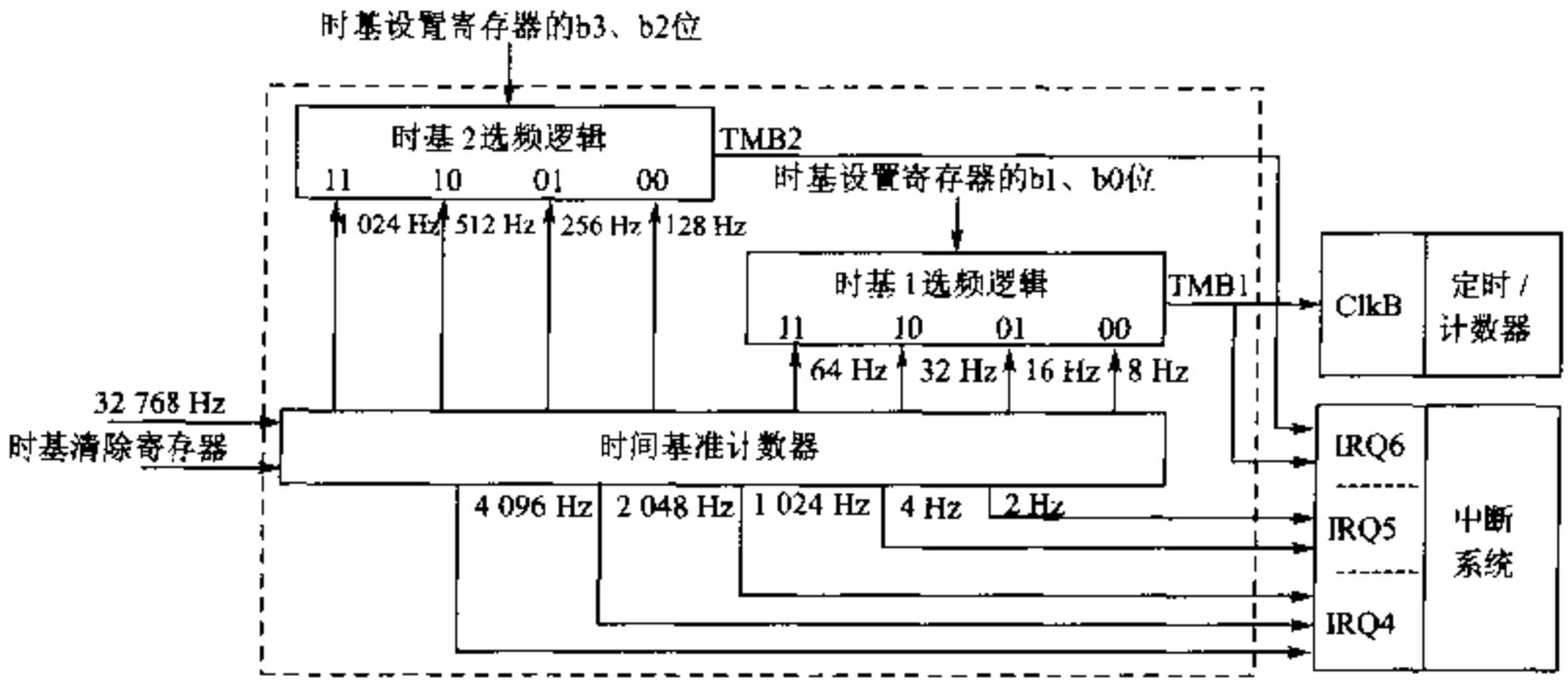


图 4-5　时基信号组成结构框图

表 4-6 给出了 TMB1、TMB2 的频率选择设置，它决定了 TMB1、TMB2 的输出信号频率。

表 4-6　时基信号 TMB1、TMB2 的设置

b3	b2	TMB2	b1	b0	TMB1
0	0	128 Hz	0	0	8 Hz
0	1	256 Hz	0	1	16 Hz
1	0	512 Hz	1	0	32 Hz
1	1	1 024 Hz	1	1	64 Hz
TMB2 默认设置：128 Hz			TMB1 默认设置：8 Hz		

(2) 时基清除寄存器 P_TimeBase_Clear($700FH)——只能写。该寄存器用于对时基信号进行清零。向 P_TimeBase_Clear 寄存器任意写入一个数，将使所有级别的时基计数器复位清零，如图 4-6 所示。此功能可用于对时基发生器进行精确的时间校准。

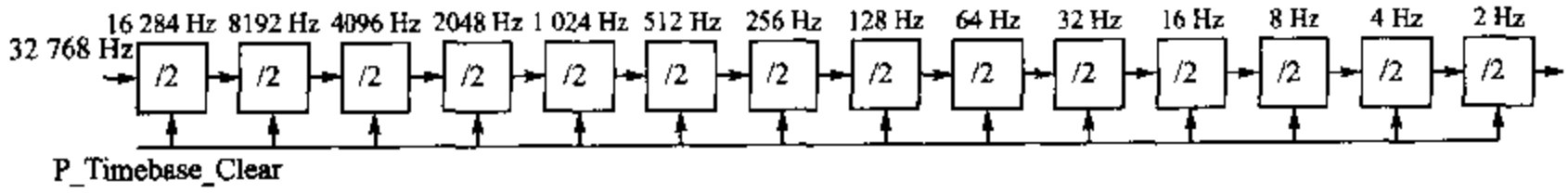


图 4-6　时基信号的清零

三、定时器/计数器的结构

1. 定时器/计数器 A(TimerA)

定时器/计数器 A 由两个时钟源 ClkA 和 ClkB、16 位定时器/计数器、16 位数据寄存器、4 位计数器、4 位半加器以及 16 位的控制寄存器等组成，结构组成框图如图 4-7 所示。由图可见，定时器/计数器 A 的计数脉冲是由两个时钟源 ClkA 和 ClkB 经过一个逻辑门相与而形成的，且 ClkA 的输入时钟频率较高，可取自锁相环晶体振荡器输出 f_{sox} (24.576 MHz)。

ClkA用于一般的应用系统中的时钟源，可以产生定时器/计数器的中断请求信号，控制音频输出通道数/模转换的数据锁存时间及ADC输入通道自动模/数转换的时间。ClkB的输入时钟频率较低，仅取自实时时钟振荡器的32 768 Hz生成的信号，可以用做精确的实时时基信号源，例如，2 Hz定时器的时钟源。图中EXT1和EXT2取自IOB2、IOB4和IOB3、IOB5组成的两个振荡器。ClkA和ClkB经与门组合后为定时器/计数器A提供了各种不同的计数速率。ClkA中的"1"和ClkB中的"1"可看做相互的选择控制信号。如ClkA的输入为"1"，其输出为高电平，则与门输出就是ClkB的输入，16位定时器/计数器的计数脉冲仅取决于ClkB；反之，ClkB输入为"1"时，与门输出就是ClkA的输入，16位定时器/计数器的计数脉冲仅取决于ClkA。ClkA中的"0"可看做定时器/计数器工作与否的控制信号，当ClkA的输入为"0"，其输出为低电平，则与门输出始终保持在低电平"0"，16位定时器/计数器无脉冲输入，定时器/计数器停止工作。

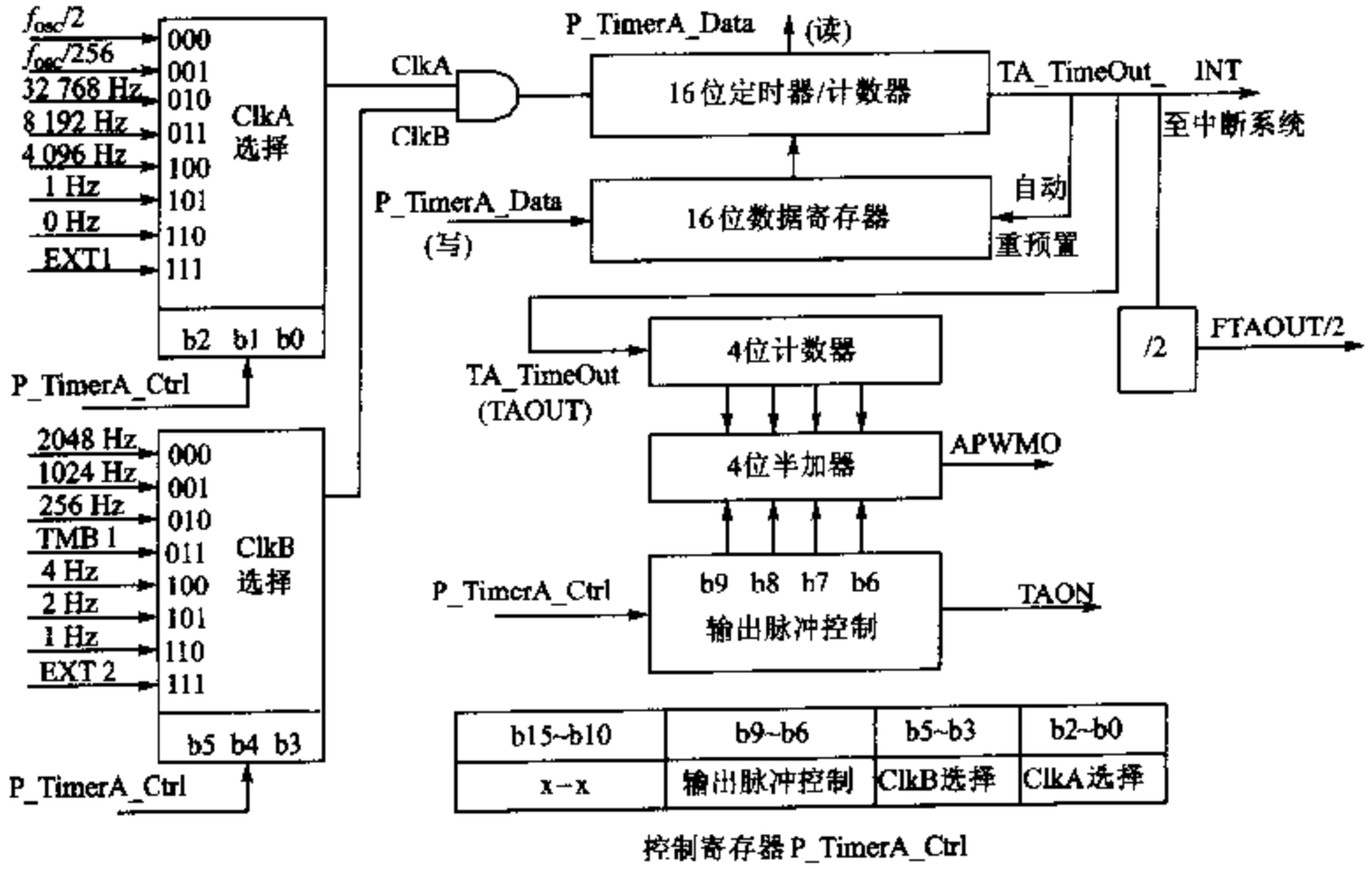
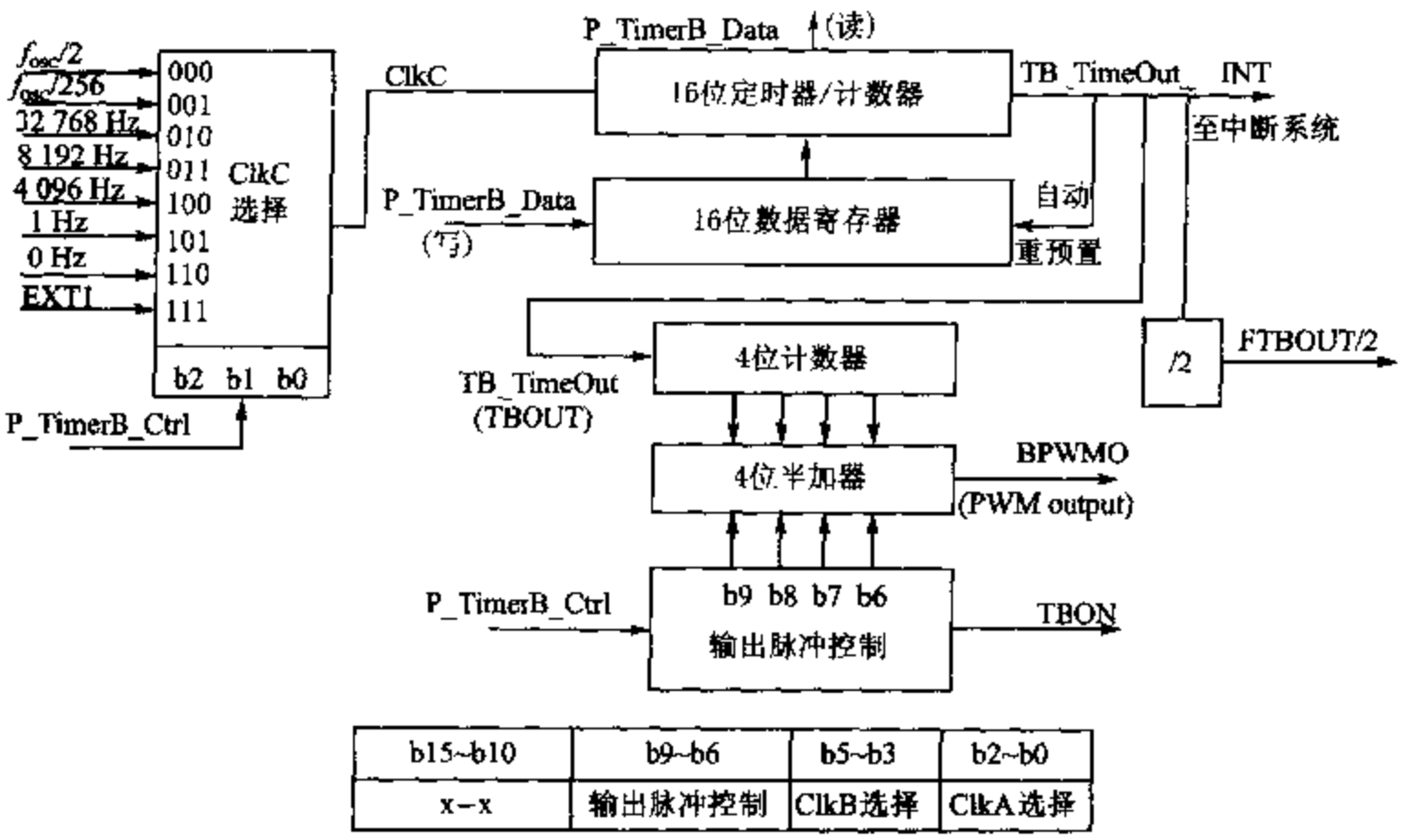


图4-7 TimerA结构组成框图

由图中还可以看出，定时器/计数器的溢出信号TA_TimeOut是4位脉冲宽度调制PWM(Pulse Width Modulation)计数器的输入脉冲信号，该4位计数器和4位半加器配合后，在控制寄存器P_TimeA_Ctrl的控制下，可产生15种不同的PWM信号APWMO输出。

2. 定时器/计数器B(TimerB)

图4-8给出了定时器/计数器B的结构组成框图，将其和图4-7相比可知，定时器/计数器B的结构和定时器/计数器A的结构基本相同，只是前者比后者少了一个时钟源，只有一个时钟源ClkC，其余两者完全相同，且ClkC与ClkA相同。



b15~b10	b9~b6	b5~b3	b2~b0
x~x	输出脉冲控制	ClkB选择	ClkA选择

控制寄存器P_TimerB_Ctrl

图 4-8　TimerB 结构组成框图

四、定时器/计数器控制寄存器及其设置

1. 数据寄存器

定时器/计数器 A 的数据寄存器 P_TimerA_Data($700AH)---可读/写。此数据寄存器实际上有两个寄存器，一个是预置初值寄存器，另一个是当前计数寄存器，它们共用一个地址号。CPU 执行写操作是将计数/定时初值写入到 16 位预置寄存器；CPU 执行读操作时，将从当前计数寄存器中读出 16 位的计数值。

定时器/计数器 B 的数据寄存器 P_TimerB_Data($700CH)——可读/写。功能与 P_TimerA_Data 相同。

2. 控制寄存器

定时器/计数器 A 的控制寄存器 P_TimerA_Ctr($700BH)——只能写。其中，b0～b5 位用于选择 TimerA 的时钟源 ClkA、ClkB 的输入信号，即选择 16 位定时器/计数器的计数速率；b6～b9 用于设置 PWM 脉冲 APWMO 的占空比。b0～b15 各位功能如下：

b15～b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
…	PWM 输出脉宽控制				ClkB 选择位			ClkA 选择位		

CPU 通过写入 P_TimerA_Ctrl 寄存器相应的控制字格式，控制定时器/计数器 A 的输入时钟源频率和脉宽调制输出的占空比。表 4-7 给出了定时器/计数器 A 的时钟源相应输入信号频率的选择。其中，TMB1 为系统实时时钟提供的时基信号 1 的时钟脉冲频率，分别是 64 Hz、32 Hz、16 Hz 和 8 Hz 之一。系统默认设置是无计数脉冲，即系统上电复位后定时/计数器是不工作的。

表 4-7 定时器/计数器 A 的时钟源选择

b5	b4	b3	ClkB 时钟频率	b2	b1	b0	ClkA 时钟频率
0	0	0	2 048 Hz	0	0	0	$f_{OSC}/2$
0	0	1	1 024 Hz	0	0	1	$f_{OSC}/256$
0	1	0	256 Hz	0	1	0	32 768 Hz
0	1	1	TMB1	0	1	1	8 192 Hz
1	0	0	4 Hz	1	0	0	4 096 Hz
1	0	1	2 Hz	1	0	1	1 Hz
1	1	0	1	1	1	0	0 Hz
1	1	1	EXT2	1	1	1	EXT1
默认设置:1(高电平)				默认设置:0(低电平)			

定时器/计数器 B 的控制寄存器 P_TimerB_Ctrl(\$700DH)——只能写。其中,b0～b2 位用于选择 TimerB 的时钟源 ClkC 的输入信号,b6～b9 用于设置 PWM 脉冲 BPWMO 的占空比,其余位没用。ClkC 的输入信号选择同定时器/计数器 A 的 ClkA 选择完全一致,BPWMO 的设置与 APWMO 的设置完全一致,这里不再重复。

五、定时器/计数器及 PWM 工作原理

定时器/计数器实际上是一个加法计数器。当向定时器/计数器的数据寄存器 P_TimerA_Data 或 P_TimerB_Data 中写入一个计数初值 N 后(初值 N 是计数脉冲的补码,如要计 256 个脉冲,则 $N=65\ 536-256$),将自动启动 16 位定时器/计数器在选择的时钟源频率下进行加法计数;当计数器计满即到 0FFFFH 后再加 1 时,产生定时器/计数器溢出中断请求信号 TA_TimeOut_INT 或 TB_TimeOut_INT 信号,并向 CPU 申请中断;与此同时,计数初值 N 会自动重新装入定时器/计数器内,然后重复上述计数的过程。

另外,定时器/计数器的溢出信号 TAOUT(TimerA_TimeOut)或 TBOUT(TimerB_TimeOut)作为 4 位计数器的时钟源,4 位计数器在此时钟源控制下进行加法计数。4 位计数器的计数值经 4 位半加器(即 4 位异或比较器)与输出脉冲控制寄存器的 b9～b6 寄存的 PWM 脉宽调制值相比较。当两者相等时,4 位半加器输出电平极性变反,不等时输出不变,从而达到 PWM 调制的目的。例如,调制值为 5 时,4 位计数器需计够 5 次,PWM 输出极性才改变。而以后 4 位计数器继续计数,当 4 位计数器计满溢出时清零,从 0 开始重新计数,与此同时 PWM 输出极性也改变,这样就产生一个占空比为 5/16 的脉宽调制信号。可见,当输出脉冲控制寄存器的 b9～b6 全为"0"时,PWM 的输出电平恒定,无 PWM 输出。所以,总共可产生 15 种不同占空比的脉宽调制信号。该信号从 IOB8(APWMO)或(IOB9)BPWMO 输出,若经过平滑滤波即可产生相应的直流模拟信号,用来控制马达或其它一些设备的速度。对应不同占空比的 PWM 调制器的波形如图 4-9 所示。

定时器/计数器控制寄存器中的 b9～b6 用于设置 PWM 输出占空比。表 4-8 给出了定时器/计数器 A 脉宽调制输出占空比的设置。其中,TAON 是定时器/计数器 A 的脉宽调制输出 APWMO 的使能标志,默认为"0",表示禁止 PWM 输出。只要 PWM 脉宽设置位 b6～

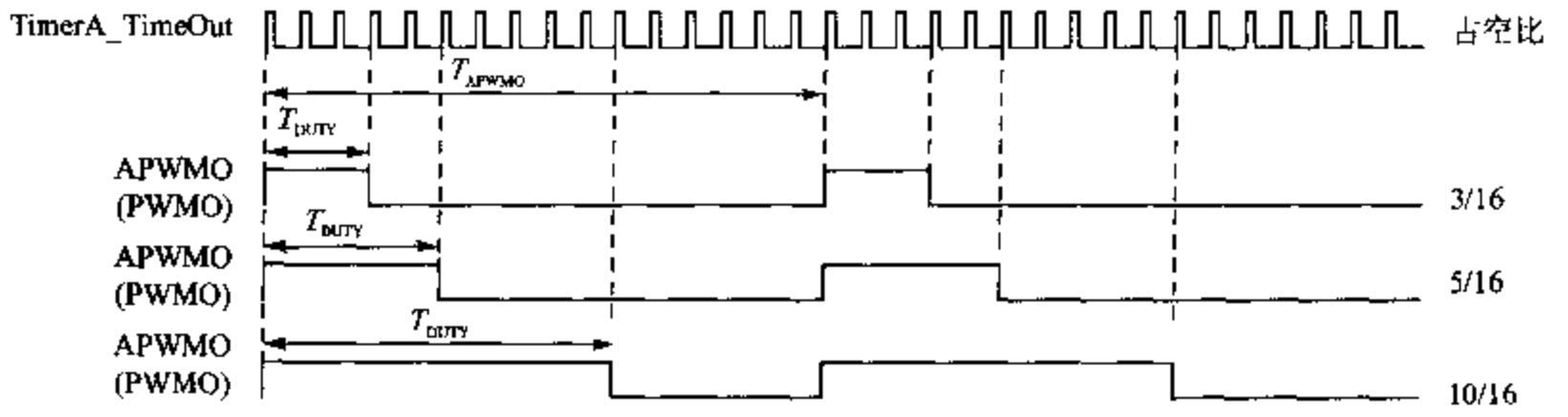


图 4-9　不同占空比的 PWM 输出波形

b9 不全为 0，则 TAON=1，表示允许有 PWM 信号输出，同时输出相应脉宽调制信号 APWMO，共 15 种。当 b6～b9 全为 0 时，TAON=0，同时将 APWMO 关闭。TAOUT 是定时器/计数器 A 的溢出信号，TAOUT 翻转的占空比是 50 %，频率是 $f_{TAOUT}/2$。

表 4-8　定时器/计数器 A 脉宽调制输出占空比的设置

b9	b8	b7	b6	脉宽占空比 APWMO	TAON
0	0	0	0	OFF	0(默认)
0	0	0	1	1/16	1
0	0	1	0	2/16	1
0	0	1	1	3/16	1
0	1	0	0	4/16	1
0	1	0	1	5/16	1
0	1	1	0	6/16	1
0	1	1	1	7/16	1
1	0	0	0	8/16	1
1	0	0	1	9/16	1
1	0	1	0	10/16	1
1	0	1	1	11/16	1
1	1	0	0	12/16	1
1	1	0	1	13/16	1
1	1	1	0	14/16	1
1	1	1	1	TAOUT 翻转	1

六、定时器/计数器应用举例

下面给出了一个输出占空比为 3/16 的定时器/计数器 A 的 PWM 输出例子。在程序中将 IOB8 设置为反相输出，由 P_FeedBack 结合定时器/计数器 A 的 PWM 标志 TAON 设置 IOB8 为 PWM 端口。计数器初值为 65 023，时钟源只有 ClkA，每经 512 个时钟计数器溢出。占空比为 3/16 时，即计数器溢出 16 次后产生一个宽度为 3 个时钟宽度的脉冲。图 4-10 为 TimerA 输出的 3/16 的脉宽调制占空比的时序。

```
R1=0x0100                    //将 IOB8 设置成反相输出状态,设置方向向量
[P_IOB_Dir]=R1               //写入方向向量值(IOB8 输出)
R1=0x0000                    //设置属性向量
[P_IOB_Attrib]=R1            //写入属性向量(IOB8 反相)
R1=0x0001                    //设置反馈寄存器的 b2、b0,并结合 TAON 设置 IOB8 为 APWMO 端口
[P_FeedBack]=R1              //写入反馈寄存器,IRTxEN=1
//设置 TimerA 的 APWMO 信号周期 T_APWMO=(12.288 MHz / 512) / 16 = 1.5 kHz
//设置 APWMO 信号的占空比 T_DUTY= (3/16) * T_APWMO
R1=0xFDFF                    //计数器初值 65 023
[P_TimerA_Data]=R1           //写计数器初值
R1=0x00F0                    // ClkA 为 f_OSC/2,ClkB 为 1,占空比=3/16
[P_TimerA_Ctrl]=R1           //写入控制寄存器
```

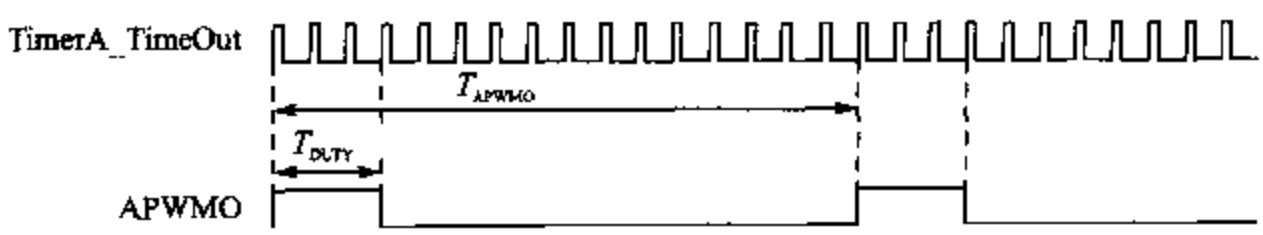


图 4-10 占空比为 3/16 的 PWM 输出时序

4.3 模/数转换器 ADC

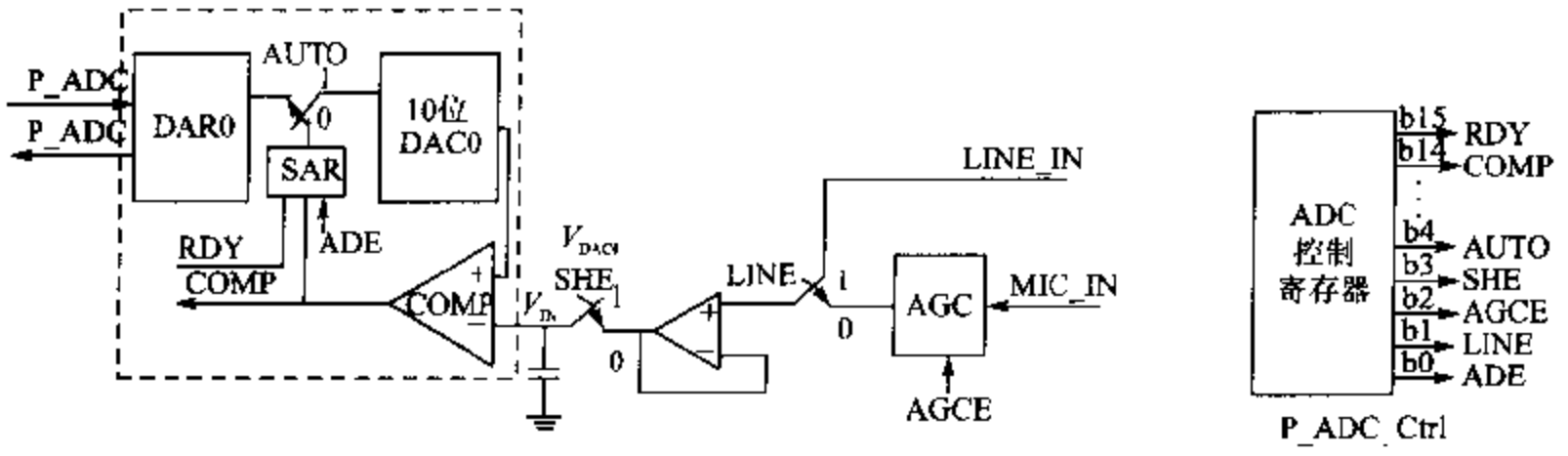
在单片机应用系统中,常常需要将检测到的连续变化的模拟量,如温度、压力、流量、转速、声音、光亮度等转换成数字信号才能输入到单片微机中进行处理。这种将模拟量转换成数字量的过程称为 A/D 转换。

随着微电子技术、单片机技术的发展,许多新一代的单片机已经在片内集成了多路 A/D 和 D/A 转换器及 PWM 输出,这样就可以大大简化片外总线的连接及编程工作。本节主要介绍 μ'nSP™单片机中的 A/D 转换器的结构及工作原理。

一、ADC 的结构及工作原理

μ'nSP™单片机的内部集成了一个 10 位的 A/D 转换器 ADC(Analog to Digital Converter),它采用逐次逼近式原理实现模/数转换。μ'nSP™中的 ADC 结构如图 4-11 所示。它由以下几个部分组成:10 位数/模转换器 DAC0、10 位数据缓存器 DAR0、逐次逼近寄存器 SAR、比较器 COMP 以及 ADC 控制寄存器。其模入信号有两个通道:一个由 LINE_IN 通道输入;另一个由 MIC_IN 通道输入。MIC_IN 一般用于麦克风通道输入,对较弱的信号一般经音频放大器 AGC(Automation Gain Control)自动增益控制放大后再进行 A/D 转换。此二通道可由 ADC 控制寄存器的 LINE 位加以选择。模入通道的信号再经单位放大器阻抗变换后作为 A/D 转换的模入信号。

图 4-11(a)中虚线框内为逐次逼近式实现模/数转换的核心。其实现 A/D 转换的工作过程是依次把设定在逐次逼近寄存器 SAR 中的数字送至 10 位 DAC0 中进行 D/A 转换,其输出的模拟量 V_{DAC0} 与采样后的模入信号 V_{IN} 相比较。比较时采用对半搜索法,即从 SAR 中最高



(a) ADC的结构框图　　(b) ADC的控制寄存器

图 4-11　ADC 原理结构示意图

有效位开始，根据比较的大小逐位确定其数码取“1”还是取“0”。其工作过程如下：

转换开始时，先设定在 SAR 中的最高位为“1”，其余位为“0”。将此假定数据输入到 DAC0 转换成电压 V_{DAC0}。然后 V_{DAC0} 与输入电压 V_{IN} 在具有高增益、二元输出（逻辑 0 或逻辑 1）的比较器中进行电压比较。如果此时 $V_{IN}<V_{DAC0}$，则说明此位置成“1”不合适，应重新设置成“0”；如果 $V_{IN}\geqslant V_{DAC0}$，说明此位设置成“1”是对的。然后对下一位按上述方法进行转换、比较、判断，决定此位取“1”还是取“0”。这种操作反复进行，经过 10 次，即可确定 SAR 中的 10 位数值。此时 SAR 中的数值就是输入量转换成的二进制数。最后将此转换结果存入数据缓存器 DAR0 中等待输出。

现以 3 位模数转换为例说明逐次逼近式转换过程。A/D 转换的分辨率习惯上用输出的二进制位数或者 BCD 码位数表示。如果 3 位模数转换器的分辨率为 3 位，即该转换器的输出数据可以用 2^3 个二进制数进行量化；如果用百分数来表示分辨率，则分辨率为 $1/2^3\times100\ \%=12.5\ \%$。若输入范围为 0～2 V，则分辨率为 0.25 V。假设现有输入模拟电压为 1.30 V，则其逐次逼近转换过程如图 4　12 所示。经过三次转换、比较、判断，就可得到转换后的二进制数为 101（代表 1.25 V，误差为 0.05 V）。μ'nSP™中为 10 位的 ADC，故需经过 10 次逐次比较才能得到最终的转换结果。

二、ADC 控制寄存器及其设置

为了正确进行 A/D 转换，必须掌握如何启动 A/D 转换，如何控制 A/D 转换过程以及如何读取 A/D 转换结果的方法。μ'nSP™单片机提供了两个单元供用户配置数据及控制字，以控制 ADC 的工作过程。一个是 ADC 控制/状态寄存器 P_DAC_Ctrl，另一个是 ADC 数据寄存器 P_DAC。

1. ADC 控制/状态寄存器

ADC 控制/状态寄存器 P_ADC_Ctrl（$7015）— —可读/写。该寄存器有两种含义，既是控制寄存器，又是状态寄存器。写入时为 ADC 控制字，读出时为 ADC 状态字。其功能及格式如下。

b15	b14	b13～b6	b5	b4	b3	b2	b1	b0
RDY(读)	COMP(读)	-	—	AUTO(写)	SHE(写)	AGCE(写)	LINE(写)	ADE(写)

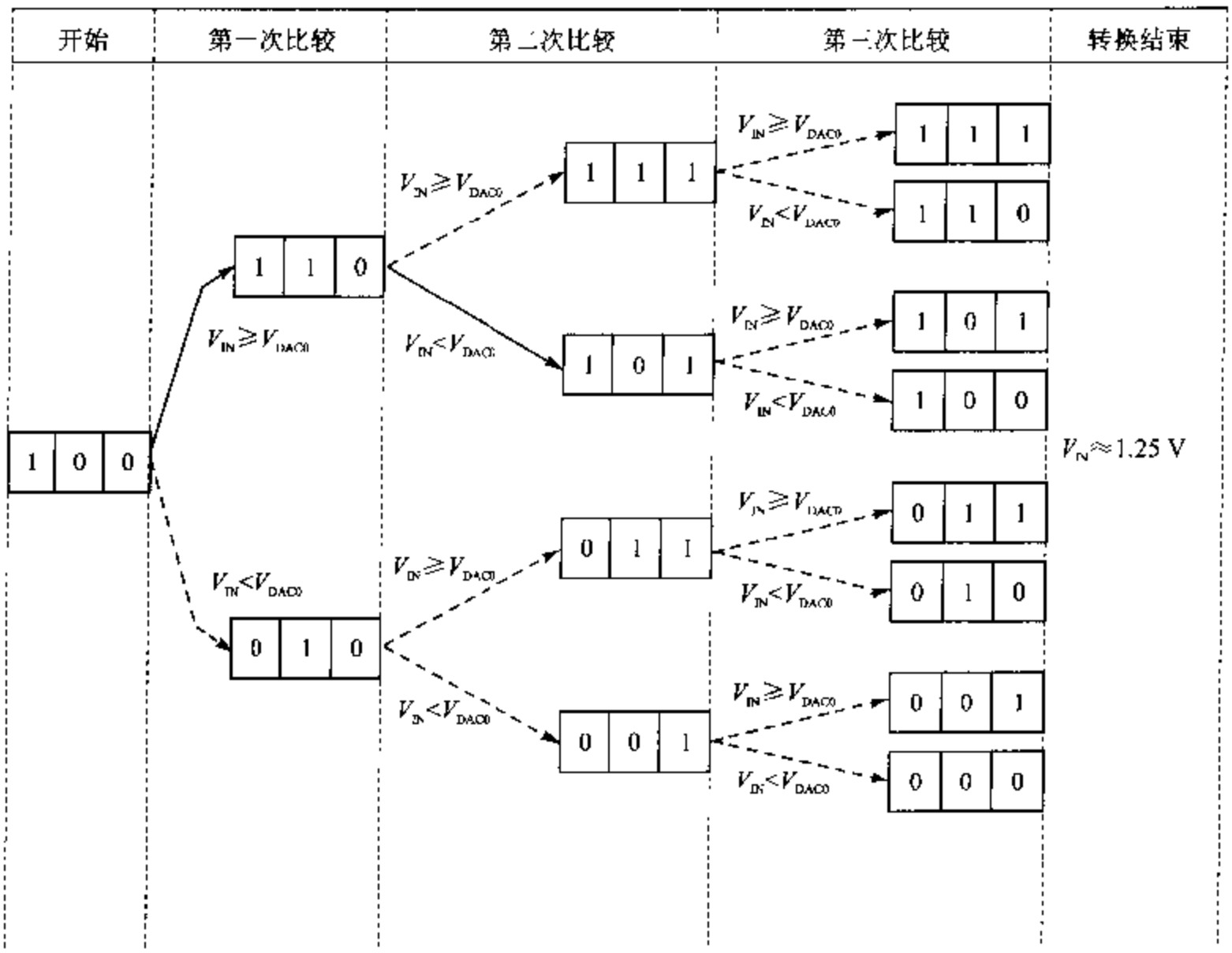


图 4-12 逐次比较过程示意

对控制字的各位说明如下：

ADE 转换控制位：

0 禁止 A/D 转换，此为默认设置；

1 允许 A/D 转换。

LINE —输入通道选择位：

0 模拟信号通过 MIC_IN 通道输入，此为默认设置；

1 模拟信号通过 LINE_IN 通道输入。

AGCE -自动增益控制位：

0 取消自动增益功能，默认设置；

1 设置自动增益功能。

SHE——采样/保持开关控制位：

0 手动方式下，采样/保持开关置于保持位置；

1 手动方式下，采样/保持开关置于采样位置。

AUTO——A/D 工作方式选择：

0 ADC 工作在手动方式下，由用户软件控制 A/D 转换，此为默认设置；

1 ADC 工作在自动方式下，由硬件自动执行 A/D 转换。

状态字的含义说明如下：

RDY —A/D 转换标志位：

0　　A/D 转换正在进行；

1　　A/D 转换已完成，CPU 可以读取 10 位转换结果。

COMP　比较器 COMP 的比较结果位：

0　　DAC0 的输出电压大于模拟输入电压；

1　　DAC0 的输出电压小于模拟输入电压。

设置控制寄存器 P_ADC_Ctrl 时，有以下几点需要注意：(1)SHE 功能仅在手动方式下有效。(2)当模拟信号通过麦克风通道 MIC_IN 输入时，可选择 AGCE 为“1”，即麦克风的增益可在运算放大器的线性区域内自动调整。当模拟信号通过 LINE_IN 通道输入时选择 AGCE 为“0”。

下面给出两个设置控制字的例子：

[例 1] 设置 ADC 工作在自动方式下，模拟量从 LINE_IN 通道输入，设置控制字及启动 ADC 的汇编程序段如下：

```
R1=0x0013                ;根据要求设置控制字
[P_ADC_Ctrl]  R1         ;将控制字送入 ADC 控制寄存器中，同时启动 A/D 转换
```

[例 2] 设置 ADC 工作在自动方式下，模拟量从 MIC_IN 通道输入，带有 AGC 功能，设置控制字的汇编程序段如下：

```
R1=0x0015                ;设置控制字
[P_ADC_Ctrl]=R1          ;将控制字送入 ADC 控制寄存器中，同时启动 A/D 转换
```

2. ADC 数据寄存器

ADC 数据寄存器 P_ADC(＄7014H)　　可读/写。CPU 对数据寄存器 P_ADC 执行读、写操作具有不同的含义：

写操作是 CPU 给 10 位缓存器 DAR0 中写入对半搜索的假定值。可见，只有在选择 A/D 转换为手动方式时才需要写入该单元。

读操作是 CPU 从 P_DAC 中读出 10 位的 ADC 转换结果。无论是自动方式还是手动方式，只有当 A/D 转换结束即状态位 RDY=1 时，CPU 才可以从该寄存器中读转换结果。

注意：ADC 数据寄存器 P_ADC 无论是读还是写，只有高 10 位有效，低 6 位无意义。

三、ADC 工作方式

上面已提到，μ'nSP™单片机的 ADC 有两种工作方式：自动方式和手动方式。可以通过对 ADC 控制寄存器的 AUTO 位设置，选择 ADC 是“自动”还是“手动”的方式。

1. ADC 自动工作方式

当 ADC 控制寄存器中的 AUTO 设置为 1 时，ADC 工作于自动方式，逐次逼近寄存器 SAR 将根据比较结果自动跟踪并给出数字量，逐次比较的 A/D 转换过程由硬件自动完成。当 A/D 转换开始后，控制寄存器中的 A/D 转换标志位 RDY=0，表示 A/D 转换正在进行；当 RDY=1 时，表示 A/D 转换结束，CPU 可以从 DAR0 寄存器中读取转换结果。ADC 自动转换过程有两种启动方式：通过读取 P_ADC 寄存器启动和通过定时器的计数溢出启动。这两种启动方式由数/模转换器的控制寄存器 P_DAC_Ctrl 中的 b3、b4 位选择，具体设置方法见 4.4 节。从 DAR0 寄存器中读取 A/D 转换结果后，将使 RDY 重新变为 0。若选择的是读数

启动A/D转换方式，则读出转换结果使RDY变为0后，将再次自动启动A/D转换。可见，适时读取A/D转换结果，可以控制A/D转换的启动时间，亦即控制了A/D转换的速率。

2. ADC手动工作方式

当ADC控制寄存器中的AUTO设置为1时，ADC工作于手动方式，内部比较器COMP和缓存器DAR0模拟SAR的作用，逐次比较的A/D转换过程由用户通过软件编程的方式控制。首先要使得采样/保持开关控制位SHE=0，即采样/保持开关处于保持状态，然后将对半搜索的假定值写入数据寄存器P_ADC中，再经过D/A转换器DAC0转换成输出电压V_{DAC0}并与输入信号V_{IN}相比较，最后读取ADC控制寄存器，并根据COMP位判断比较结果。若$V_{IN} \geqslant V_{DAC0}$，则数据缓存器DAR0中将比较后的有效位保持"1"，即A/D转换的结果暂存为1000000000B；否则取"0"。接着再比较下一位，这样经过10次比较后，数据缓存器DAR0中的数据即为A/D转换结果。图4-13给出了用手动方式编程实现A/D转换逐次比较的软件流程图。在手动方式下，用户可以选择比较搜索的起始位和停止位，比较次数可按A/D转换精度达到要求即可。

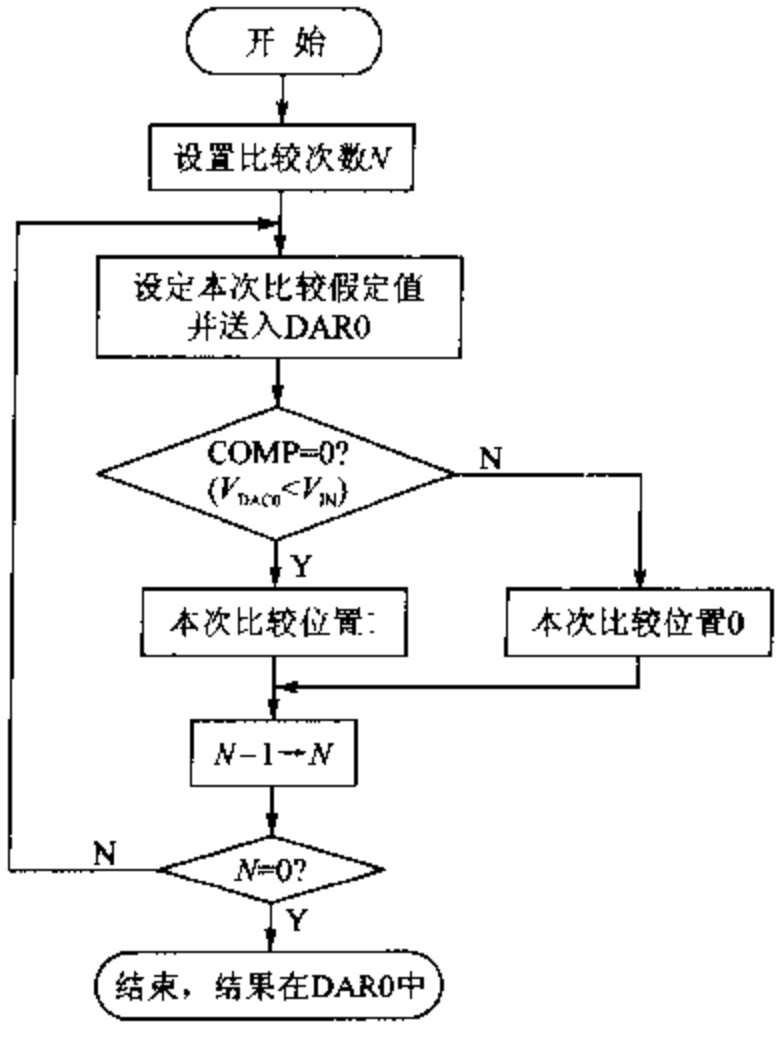


图4-13 软件实现A/D转换流程图

在使用时必须注意，ADC的转换速度有一定的限制，最高转换速度不能超过(f_{osc}/32/12)Hz，若超过此极限，从ADC数据寄存器P_ADC中读出的数据将不正确。

4.4 数/模转换器DAC&PWM

在计算机自动控制系统中需要的控制量一般是模拟量，如电动执行机构、直流电动机、伺服机构、音频信号等。而计算机输出的是数字量，这就要求把计算机输出的数字量变换为被控对象需要的模拟量才能驱动。数/模转换器DAC(Digital to Analogy Converter)就是将输入的数字量转化成电压或电流模拟量的一种器件，在自动控制系统、测控仪表、多媒体技术中有着广泛的应用。μ'nSP™单片机的内部集成了A/D、D/A转换器及脉宽调制PWM输出功能。A/D、D/A转换接口可以方便地用于各种数据采集、处理和控制输出，再与DSP运算的相关指令功能结合在一起，便能较好地实现语音识别功能，使其能方便地用于发声和语音识别领域。脉宽调制PWM则是先将输入的数字量转换成脉宽调制占空比信号，然后再转换成电流模拟量信号输出的一种功能部件，最终将完成数/模转换的功能。

一、音频输出的方式

μ'nSP™单片机中的DAC主要用于将数字信号转换成音频信号输出。μ'nSP™的音频

输出有两种控制方式：一种是语音输出(Speech Mode)方式；另一种为音调输出(Tone Mode)方式。两者的区别在于其输出控制机理不同。语音输出是用于声音(不论是音乐还是语音)数据采样率相同的速率将声音数据通过数/模转换通道还原成声音。音调输出是通过 Timer 溢出所产生的不同频率来决定声音音调的高低，DAC 的模拟量决定声音信号的幅值。图 4-14 给出了两种音频输出方式的硬件结构原理框图。

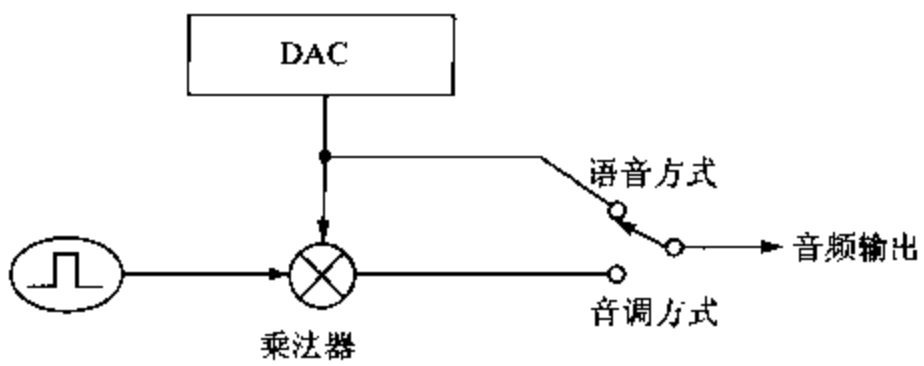
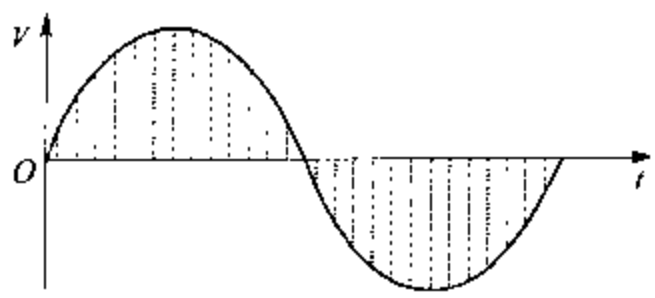
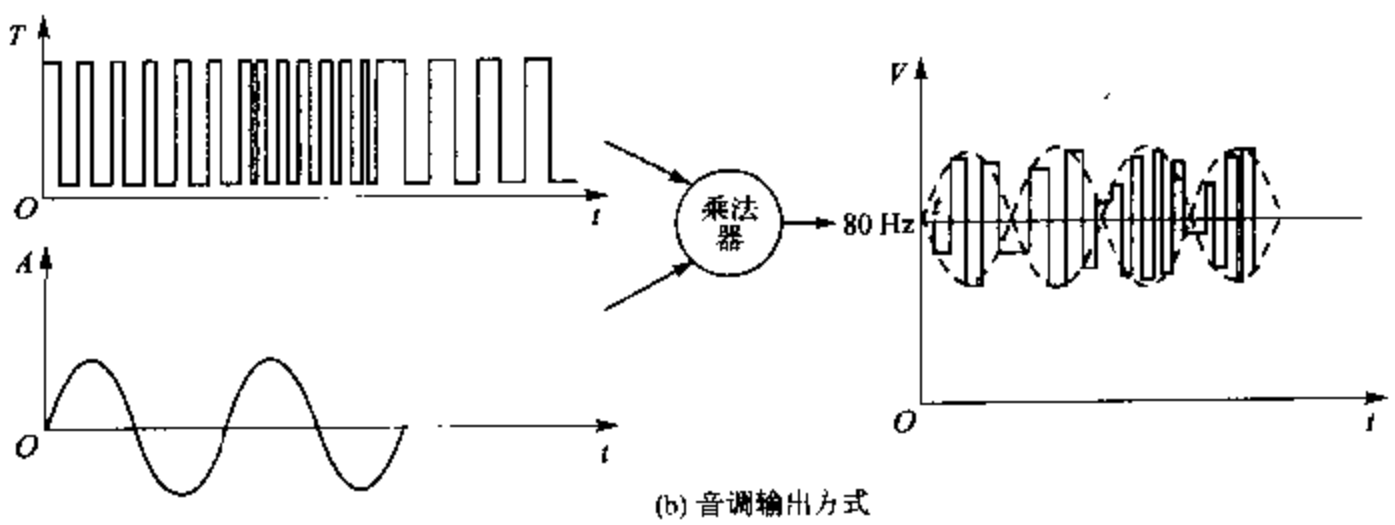


图 4-14　两种音频输出方式硬件结构示意框图

语音输出方式是数/模转换器 DAC 把模拟声波输入的数字量转换成模拟量(即模拟声波信号)实现波形重放。D/A 转换器位数越多，重放波形的音质就越好。一般 16 位的转换器就具有较好的音质。图 4-15(a)为语音输出方式的 D/A 转换波形示意图。其中，横轴表示时间，纵轴表示振幅。语音输出时，CPU 以一固定的频率(采样率)向 DAC 送出一系列的数字量值，此数值被转换成一系列的电流(或电压)模拟量，并经平滑滤波后驱动扬声器得到声音信号。这种方式下，声音数据的采样率决定了声音音质的好坏。采样率越高，恢复的声音波形越接近原来的波形，音质越好。如果输出的声音记录只产生一个波形则是单通道输出；如果产生两个波形则是立体声双声道。立体声听起来比单声道的声音丰满且有一定的空间感，更能反映人们的听觉感受。



(a) 语音输出方式



(b) 音调输出方式

图 4-15　两种音频输出方式

音调输出方式是对 DAC 输出的声音信号采用了双边带的调制方式。双边带 DSB(Doub-

le Side Band)调制方式是一种振幅调制方式,用于抑制载波。若载波信号为 $V_c(t)$,调制波信号为 $V_n(t)$,则调制后的信号为 $V_{DSB}(t)=kV_c(t)\times V_n(t)$,式中,$k$ 为比例常数。由表达式可知,任何输出电压正比于输入电压乘积的器件都可实现双边带的调制。在 μ'nSP™单片机中,载波信号是由定时器 Timer 产生的方波信号,而调制波是由 DAC 输出的模拟量对应的波形。调制电路由模拟乘法器构成。Timer 输出的方波波形在此决定音调的高低,即频率;而 DAC 输出的模拟量幅值决定着音色,即幅值,波形如图 4-15(b)所示。变换的法则如下:假设 Timer 溢出使方波输出用"T"表示,数/模转换器 DAC 输出的模拟量用"A"表示,乘法器输出用"V"表示,则乘法器在一个采样周期内的输出规则是:

当 $T=0$ 时:V = 80H−A;

当 $T=1$ 时:V = 80H+A。

二、DAC/PWM 输出的结构形式

前已提及音频输出方式有两种,它们都需要用到数/模转换器 DAC。在 μ'nSP™单片机中用于实现 D/A 转换通道的硬件结构有两种:一种是直接采用数/模转换器 DAC 实现;另一种是采用单通道脉宽调制 PWM 驱动方式输出。图 4-16 所示为 μ'nSP™中采用的两种实现 D/A 转换的音频输出硬件结构。它由两个 DAC 通道和一个 PWM 发生器组成。图中 DAR1,DAR2 为两个 10 位缓冲器。

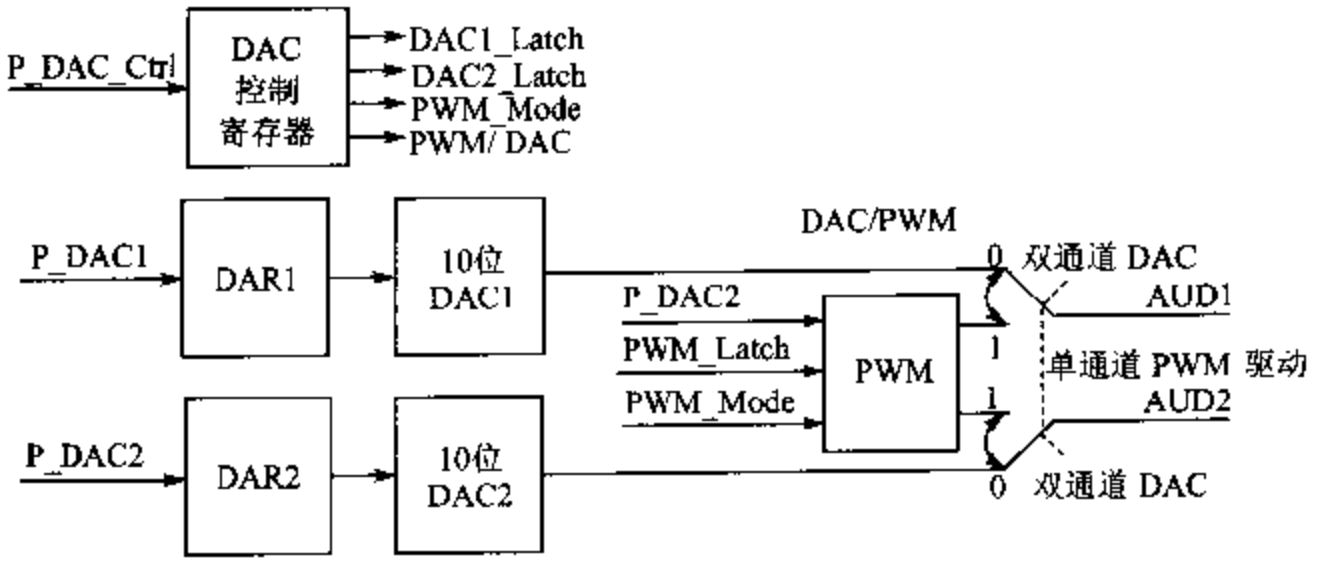


图 4-16 音频输出硬件结构示意图

直接 DAC 方式用于有两个通道的声音数字信息转换成模拟的音频信号输出。实际应用时,应按相同的采样速率,将双通道输入的表示声音的数字量同步且分别地写入到两个数据单元 DAR1 和 DAR2 中,并分别送入 DAC1 和 DAC2 中进行 D/A 转换。转换的电流模拟信号分别从 AUD1 和 AUD2 引脚输出。

采用 PWM 的 D/A 转换方法,只有一个 PWM 通道。PWM 实现 D/A 转换时,先将声音的数字量转换成相应占空比的脉宽信号驱动方式输入的数字量写入 P_DAC2 寄存器(此时 P_DAC1 寄存器不起作用)。AUD1 和 AUD2 引脚则用来驱动 PWM 输出的电流模拟信号。PWM 信号转换的波形见图 4-9。图中 PWM 计数器溢出的输出波形是一个重复的以 16 个脉冲序列为信号周期的波形 PWMO,而占空比周期 T_{DUY} 中所包含的脉冲序列是可变的。占空比可通过向 PWM 计数器写入不同的计数初值来改变。假设 PWM 计数器的计数位数为 n,设定的计数初值为 m,则计数值为 2^n-m,T_{DUY} 为 $(2^n-m)/2^n$。输出的 PWMO 信号经滤波

积分就可得到连续的电流模拟信号。占空比越大，其输出的模拟量也越大，从而实现用 PWM 完成 D/A 转换。

三、D/A 控制寄存器及其设置

为了正确进行 D/A 转换，实现音频输出的功能，μ'nSP™单片机提供了两个单元供用户控制 ADC 的工作过程，这两个单元是 DAC/PWM 控制寄存器 P_DAC_Ctrl 和 DAC 数据寄存器 P_DAC。

1. DAC/PWM 控制寄存器

DAC/PWM 控制寄存器 P_DAC_Ctrl(S702A) —只能写。该寄存器用于设置 D/A 转换过程中的各种功能以及 A/D 转换自动方式的选择。其功能及格式如下。

b15～b9	b8～b7	b6～b5	b4～b3	b2	b1	b0
–	DAC1_Latch	DAC2_Latch	ADC_Latch	PWM_Latch	DAC/PWM	PWM_Mode

对控制字的各位说明如下表述。

PWM_Mode——PWM 输出方式选择位：

0　推挽方式的 PWM 驱动输出，此为默认设置；

1　双脚双端方式的 PWM 驱动输出。

DAC/PWM——音频输出方式选择位：

0　双通道 DAC 方式音频输出，此为默认设置；

1　单通道 PWM 驱动方式音频输出。

PWM_Latch——PWM 输出数据锁存方式选择：

0　将输出数据直接置入 PWM 计数器的预置数方式；

1　通过 PWM 计数器溢出锁存输出数据到 PWM 计数器的自动重置数方式。

ADC_Latch – -ADC 自动转换方式选择位：

00　通过读 P_ADC 寄存器达到 ADC 自动方式的转换，此为默认设置；

01　通过 TimerA 溢出达到 ADC 自动方式的转换；

10　通过 TimerB 溢出达到 ADC 自动方式的转换；

11　通过 TimerA 或 TimerB 的溢出达到 ADC 自动方式的转换。

DAC2_Latch– —DAC2 数据锁存方式选择位：

00　直接将 DAR2 中的数据锁存到 DAC2 中，此为默认设置；

01　通过 TimerA 溢出将 DAR2 中的数据锁存到 DAC2 中；

10　通过 TimerB 溢出将 DAR2 中的数据锁存到 DAC2 中；

11　通过 TimerA 或 TimerB 溢出将 DAR2 中的数据锁存到 DAC2 中。

DAC1_Latch —DAC1 数据锁存方式选择位：

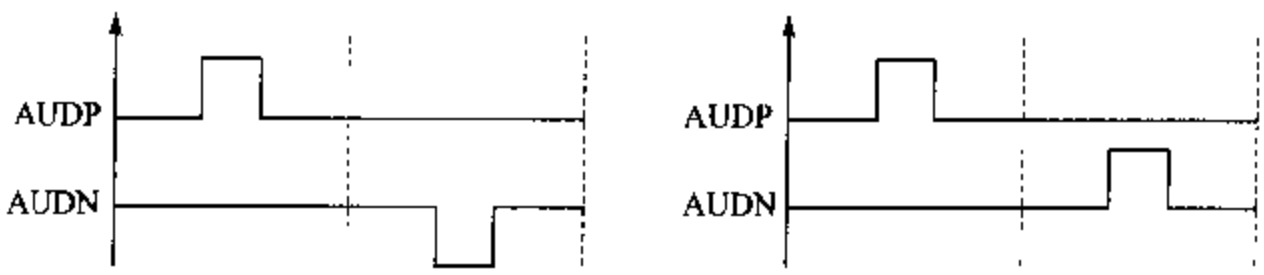
00　直接将 DAR1 中的数据锁存到 DAC1 中，此为默认设置；

01　通过 TimerA 溢出将 DAR1 中的数据锁存到 DAC1 中；

10　通过 TimerB 溢出将 DAR1 中的数据锁存到 DAC1 中；

11　通过 TimerA 或 TimerB 的溢出将 DAR1 中的数据锁存到 DAC1 中。

综上所述，控制寄存器中的DAC/PWM位用于选择音频输出方式，即采用双通道DAC方式或采用单通道PWM驱动方式。若选用单通道PWM驱动方式，则PWM_Mode位进一步选择PWM驱动采用推挽方式输出或是双脚双端方式输出，两种输出方式的波形如图4－17所示。

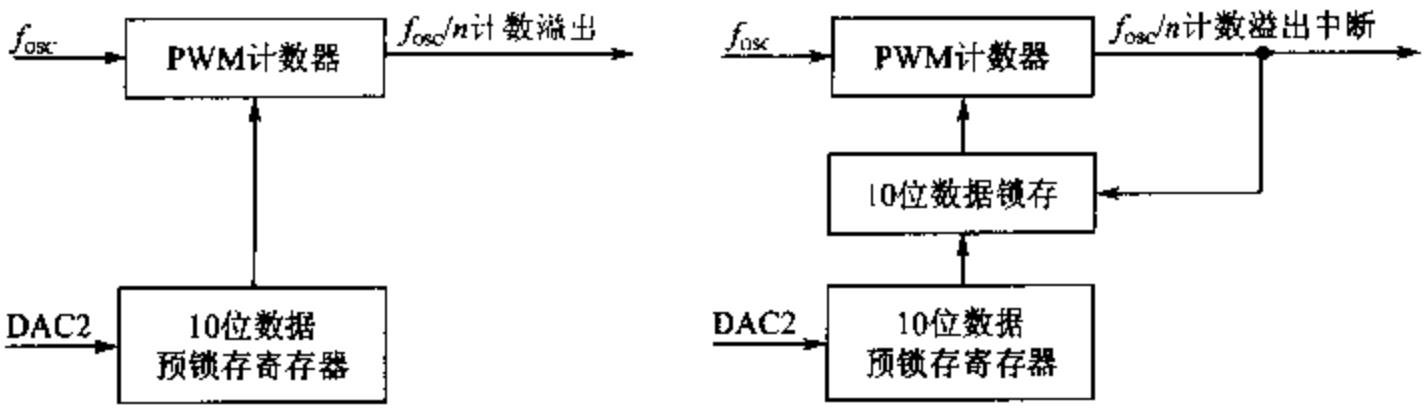


(a) PWM推挽方式　　(b) PWM双脚双端方式

图4－17　PWM驱动输出方式

DAC输出数据锁存方式由DAC1_Latch、DAC2_Latch选择位组成。它有两种锁存方式：直接锁存和通过Timer计数器溢出锁存。Timer计数器锁存方式又可以分为3种：通过TimerA溢出锁存、通过TimerB溢出锁存和通过TimerA或TimerB溢出锁存。

PWM输出数据锁存方式由PWM_Latch选择，也有两种锁存方式：直接置入PWM计数器的预置数方式和通过PWM计数器溢出自动重置数方式。这两种方式如图4－18所示。前者是通过DAC单元将计数初值直接置入PWM计数器内；后者是通过PWM计数器溢出时产生一个自动重置数据信号，将计数初值由锁存器自动重新置入计数器内。在PWM计数溢出产生的中断响应服务程序中，可将下一个计数初值由DAC单元置入预锁存寄存器内。数据的自动重预置和PWM计数溢出中断服务响应是同步发生的，这样可以避免PWM输出波形产生抖动。



(a) 输出数据直接置入PWM计数器　　(b) PWM溢出锁存数据

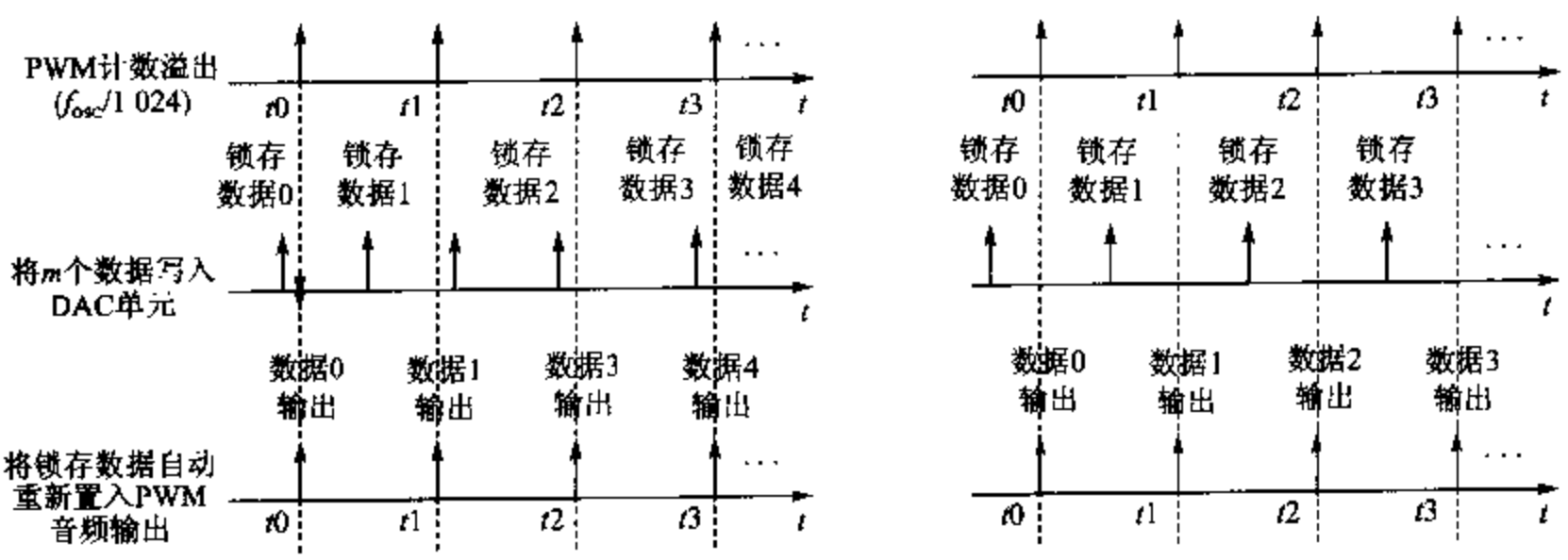
图4－18　PWM数据锁存方式

在编程输出数据锁存时必须注意：① 当采用双通道DAC方式且音频输出由Timer计数器溢出锁存数据；② 采用单通道PWM驱动方式使音频输出时，计数器初值的设置即写入DAC单元的时机非常重要。假设$n=1\,024$，写入数据的时间速率即声音数据的播放率为$f=f_{OSC}/(1\,024*N)$。如果N为非正整数时，则在一次PWM计数溢出信号内会出现两次以上的DAC单元数据写入，从而引起其间数据的丢失，如图4－19(a)所示。若N为正整数，则在一次PWM计数溢出信号内至多只可能出现一次DAC单元数据写入，不会出现数据丢失的现象，如图4－19(b)所示。

2. DAC 数据寄存器

DAC 数据寄存器共有两个，且无论是读还是写，只有高 10 位有效，低 6 位无意义。

DAC 数据寄存器 2 P_DAC2（$7016H）——可读/写。在双通道 DAC 输出方式下，写操作是 CPU 将 10 位无符号数写入带有缓存器 DAR2 的 DAC2 输入口；而在单通道 PWM 驱动输出方式下，写操作是 CPU 将 10 位有符号数置入 DAR2，使 PWM 产生出音频输出。读操作是 CPU 读出 10 位缓存器 DAR2 内的数据。



(a) 数据2出现丢失　　(b) 数据正常输出

图 4-19　写入 DAC 单元的时机

DAC 数据寄存器 1 P_DAC1（$7017H）——可读/写。在双通道 DAC 输出方式下，写操作是 CPU 将 10 位无符号数写入带有缓存器 DAR1 的 DAC1 输入口；而在单通道 PWM 驱动输出方式下不用该寄存器。读操作是 CPU 读出 10 位缓存器 DAR1 内的数据。

四、ADC 和 DAC 应用举例

下面给出从话筒采样声音信号，然后以 8 kHz 的采样速率由 PWM 方式送给扬声器输出的完整的程序。

```
.DEFINE     P_TimerA_data          0x700a;        //硬件资源配置
.DEFINE     P_TimerA_ctrl          0x700b;
.DEFINE     P_INT_ctrl             0x7010;
.DEFINE     P_INT_clear            0x7011;
.DEFINE     P_WATCHDOG_clear       0x7012;
.DEFINE     P_ADC                  0x7014;
.DEFINE     P_ADC_ctrl             0x7015;
.DEFINE     P_DAC2                 0x7016;
.DEFINE     P_DAC1                 0x7017;
.DEFINE     P_DAC_ctrl             0x702a;
.RAM                               //定义预定义 RAM 段
.CODE                              //定义代码段
//------------------ ----- ----- ---------------------------- ----------------
//主程序部分完成以下功能：ADC 及 DAC 的初始化
//Timer A 的初始化，设置中断，清除看门狗计数器
```

```
//------ ------------ . --------- ------------ ------------ ---
.PUBLIC     _main                    //对 MAIN 程序声明
_main:                               //MAIN 程序开始
INT  OFF                             //关闭所有中断
//.....初始化 TimerA .....
r1= 0xFFFF;
r1-=0x05DC                           //采样率 fs=8 kHz
[P_TimerA_data]=r1;
r1=0x0030                            //选择 fosc/2 为 TimerA 的时钟源
[P_TimerA_ctrl]=r1;
//.....初始化 ADC,DAC .....
r1=0x0015;                           //工作在自动模式,从话筒(MIC_IN)输入,带有 AGC 功能
[P_ADC_ctrl]=r1;
   R1=0x0002                         //推挽方式的 PWM 音频输出
[P_DAC_ctrl]=r1;
r1=0x1000                            //允许 TimerA 的 IRQ1 中断
[P_INT_ctrl]=r1;
INT  IRQ                             //开通 IRQ 中断
//.....清除看门狗,循环等待中断发生 .....
ENDLOOP:
   r1=0x0001                         //清除看门狗定时器
   [P_WATCHDOG_clear]=r1;
   jmp  ENDLOOP                      //程序循环
//---------------------- -- - - -- ---- --------------
// Timer A 的溢出中断 IRQ1 的中断子程序
//该子程序完成从 MIC_IN 通道采样和送出数据给 DAC 的功能
//-------------------------------------------------------------------------
.PUBLIC     _IRQ1;
_IRQ1:                               //IRQ1 中断服务程序
push r1,r5 to [sp];
[P_INT_clear]=r1                     //清除 IRQ1 的中断请求
r1=[P_ADC];
[P_DAC2]=r1;
pop     r1,r5     from [sp];
reti;                                //返回主程序
```

4.5 异步串行通信和红外通信

在实际应用中,计算机与其外部设备之间或计算机之间常常需要进行信息交换,这些信息交换就称之为通信。通信的基本方式有并行通信和串行通信两种。

一个信息数据的各位被同时传送的通信方法称为并行通信。并行通信依靠并行 I/O 接口实现。其特点是传输速度快,但当传输距离较远、位数较多时导致通信线路复杂且成本高,

因此，只适合于近距离通信。

一个信息数据的各位被逐位顺序传送的通信方法被称为串行通信。串行通信可通过串行接口来实现。其特点是通信线路简单，只要一对传输线路就可以实现通信，可节省大量的线路成本，特别适合于远距离通信，但传输速率较低。根据信息的传送方向，串行通信可以进一步分为单工、半双工和全双工3种方式。信息只能单方向传送称为单工；信息能双向传送但不能同时双向传送称为半双工；信息能同时双向传送称为全双工。根据时钟控制数据的发送和接收方式的不同，串行通信又有异步通信和同步通信两种方式。μ'nSP™单片机有两个串行接口：一个是通用串行通信接口，主要用于异步串行通信；另一个为串行外围设备接口SIO(Serial interface I/O)，主要用于同步串行通信。

μ'nSP™单片机的通用串行通信接口为标准全双工的通用异步接收器/发送器UART(Universal Asynchronous Receiver/Transmitter)模块。UART的接收信号Rx和发送信号Tx分别是与IOB7和IOB10共用，属于B口的特殊功能。为保证UART的正常通信，IOB7和IOB10应分别被设置成输入和输出口。为了通信的正确发送/接收，必须使Rx和Tx信号有相同的通信规约，其中最基本的是传送的字符格式和波特率应相同。

一、UART接收数据的方式

UART有两种接收数据的方式：一种是通过UART的Rx信号，它来自UART接收器的引脚IOB7；另一种则是通过红外通信模块的接收信号IRRx，它来自引脚IOB6。接收方式的选择取决于红外通信功能的控制寄存器P_IR_Ctrl的功能初始化设置。

二、UART的字符格式

在异步通信时，数据的发送端Tx和接收端Rx必须严格按照一定的格式进行发送和接收，否则，将引起数据帧错误，不能进行通信。μ'nSP™的数据格式如图4-20所示。通信时用起始位"0"表示数据的开始，然后从低位到高位逐位传送数据，再传送奇偶校验位，最后用停止位"1"表示字符结束。一字符帧信息包括1位起始位、8位数据位、1位奇偶校验位和1位停止位。根据采用的奇偶校验方式，在发送时，用户编程检查每个要传送的数据中的"1"的个数，在奇偶校验位添"1"或"0"，使得"1"的总和(包括奇偶校验位)在偶校验时为偶数，在奇校验时为奇数，接收方会自动检测出在串行传送时是否发生奇偶错误。

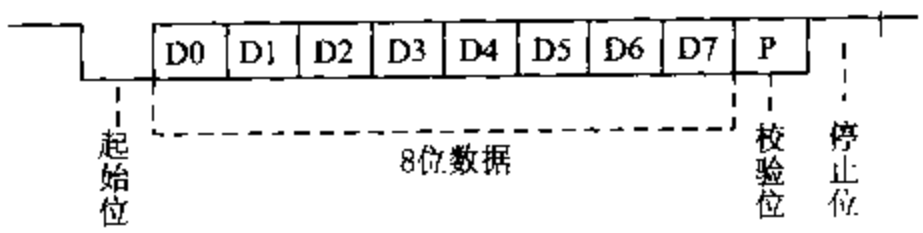


图4-20　UART数据格式

三、UART控制寄存器及其设置

1. UART数据缓冲器

UART数据缓冲器P_UART_Data($7023H)——可读写。μ'nSP™的UART为全双工的通信方式，发送端和接收端各有一个数据缓冲器，分别称为发送数据缓冲器和接收数据缓

冲器。发送数据缓冲器只能写入，不能读出；而接收数据缓冲器只能读出，不能写入。所以，它们只占用一个地址号，统称为UART数据缓冲器P_UART_Data，且只用了其中的低8位。CPU可以把要发送的数据送至发送数据缓冲器，也可以从接收数据缓冲器中读出接收的数据。

2. UART控制寄存器及其设置

UART的控制寄存器有两个：控制命令寄存器1和控制命令/状态寄存器2，用于设置串行通信的工作模式或反映串行通信的工作状态。

(1) 控制命令寄存器1：P_UART_Command1($7021H)——只能写。其功能及格式如图4-21所示。

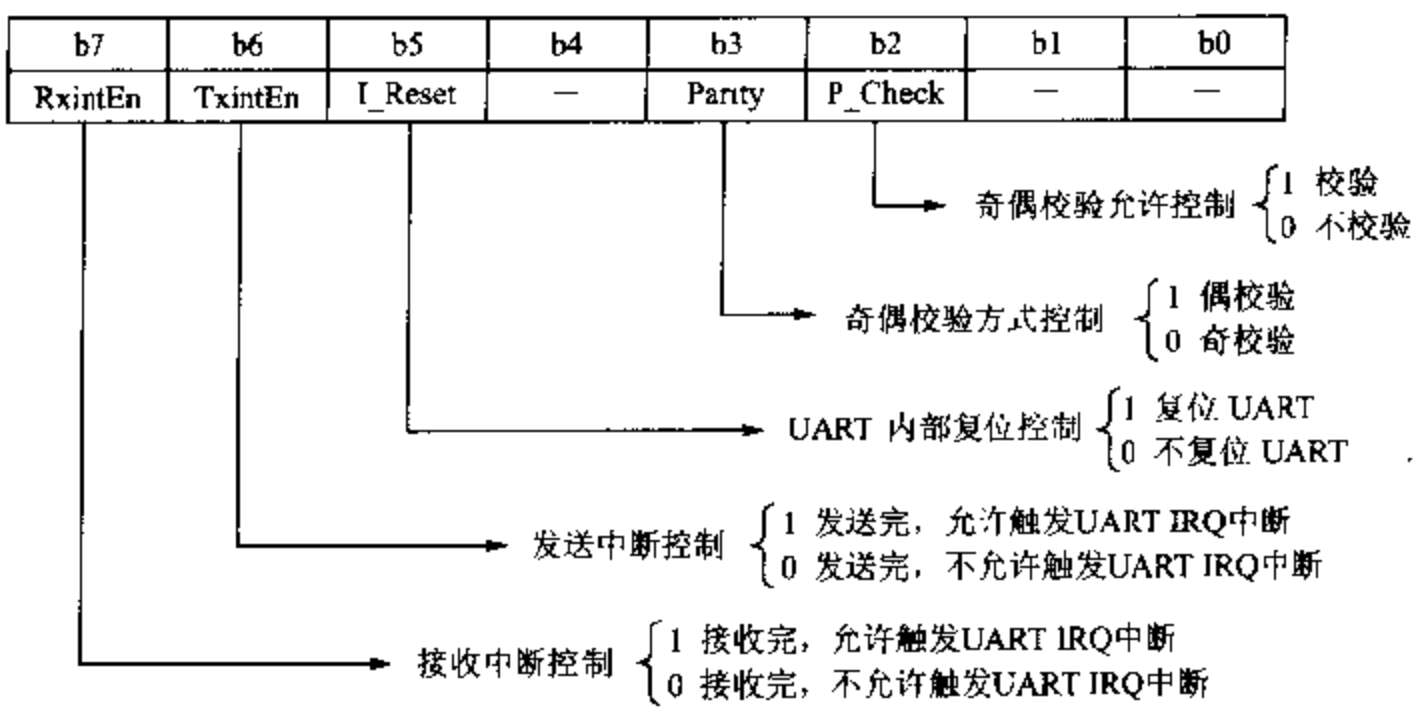


图4-21　控制命令寄存器1的功能及格式

若I_Reset=1，则执行UART模块内部复位，将使UART的所有控制及状态寄存器恢复为默认值；若I_Reset=0，则不执行UART模块内部复位，UART的所有控制及状态寄存器的内容不变。若允许UART模块中断，则UART IRQ发送或接收中断在数据字节发送完成或数据字节接收完成时被触发。

P_UART_Command1的默认设置为00H，即禁止UART中断，不执行UART模块复位，无奇偶校验功能。

(2) 控制命令/状态寄存器2：P_UART_Command2($7022H)——可读/写。控制命令/状态寄存器2具有两种含义，即既是控制寄存器，又是状态寄存器。写入时为UART的控制字，读出时为UART的状态字。其功能及格式如图4-22所示。

P_UART_Command2的默认设置为00H，即禁止Tx引脚发送数据，禁止Rx引脚接收数据，亦即禁止UART通信。

对状态字的各位，说明如下。

奇偶校验错误标志位PE：UART在接收时会检查字符中的每一位(包括奇偶校验位)，计其"1"的总数是否符合要求，若不符合要求，则PE被置1。当通信环境恶劣或有噪音干扰时可能会出现奇偶校验错误。CPU从数据缓冲器P_UART_Data中读出数据后，PE将被清零，为接收下一帧数据作准备。一般情况下，PE=0。

溢出错误标志位OE：溢出错误也称为丢失错误。当CPU尚未从接收数据缓冲器中取走

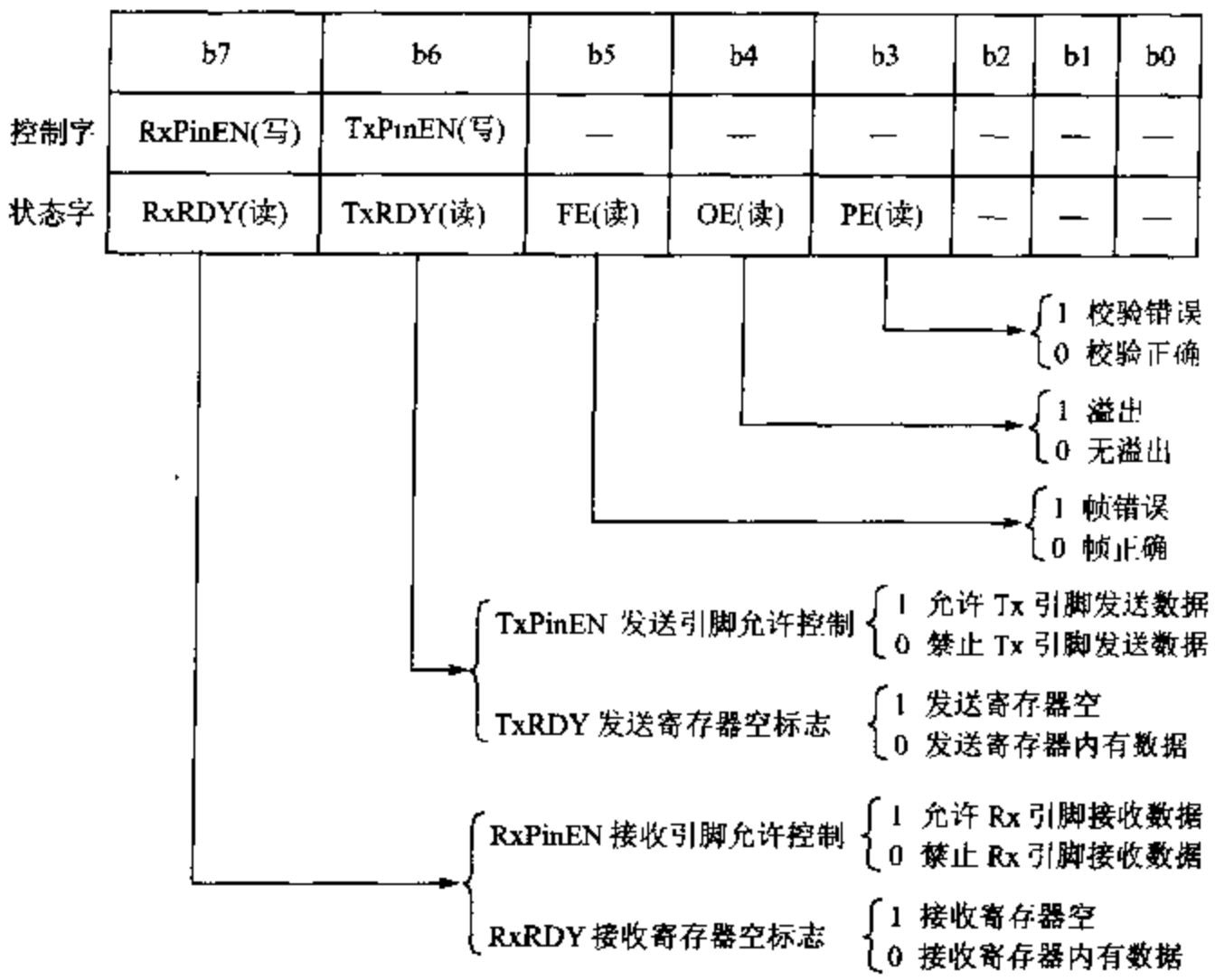


图 4-22　控制命令/状态寄存器 2 的功能及格式

数据，而又有新的数据装入接收数据缓冲器时，新的数据将会覆盖前一个数据，即发生数据丢失，称为溢出错误。这时 OE 将被置 1。CPU 从数据缓冲器 P_UART_Data 中读出数据后，OE 将被清零；没有溢出错误时 OE=0。

帧错误标志位 FE：当通信中 Rx 端与 Tx 端字符帧格式不一致或 Rx 端与 Tx 端波特率不一致时，会发生帧错误，FE 将被置 1。只有从数据缓冲器 P_UART_Data 中读出数据后，FE 才会被清零。一般情况下，FE=0。

发送寄存器空标志位 TxRDY：当一帧数据发送完成后，TxRDY 将被置 1，表示发送寄存器空。若 TxPintEN=1，允许 UART IRQ 发送中断，TxRDY 置 1 将触发 UART IRQ 中断，向 CPU 申请中断，表示 CPU 可以发送下一个数据。当 CPU 响应中断，在中断服务程序中发送数据，即向数据缓冲器 P_UART_Data 中写入数据后，TxRDY 将被清零，为发送下一帧数据作准备。

接收数据完成标志位 RxRDY：当一帧数据接收完成后，RxRDY 将被置 1。若 RxPintEN=1，允许 UART IRQ 接收中断。RxRDY 置 1 将触发 UART IRQ 中断，向 CPU 申请中断，表示 CPU 可从串口数据缓冲器中取走数据。当 CPU 响应中断，在中断服务程序中 CPU 从数据缓冲器 P_UART_Data 中读出数据后，RxRDY 将被清零，为接收下一帧数据作准备。

由于 UART IRQ 的中断优先级别较低，故它的中断触发信号会在其之前发生较高级别的中断响应时一直保持。因此，也常采用查询 RxRDY 或 TxRDY 信号的状态进行数据的接收和发送。

3. UART 的波特率及设置

波特率是指每秒传送的二进制位数，它是衡量数据传送速率的指标，并要求发送端和接收

端都应以相同的波特率传送。μ'nSP™的UART专门有16位的波特率设置寄存器，这些寄存器有如下几种。

波特率低字节寄存器P_UART_BaudScalarLow(\$7024H)——可读/写。它用于存放波特率设置的低字节，默认值为00H。

波特率高字节寄存器P_UART_BaudScalarHigh(\$7025H)——可读/写。它存放波特率设置的高字节，默认值为01H。

上述两个寄存器虽然是16位的，但只取其中低8位即一个字节组合成一个字。

波特率常数也可以用十进制表示。若波特率的十进制数为n，则波特率计算式如下：

$$\text{波特率} = (f_{OSC}/2)/n$$

式中，f_{OSC}=24.567 MHz或f_{OSC}=20.180 MHz，其取决于系统时钟中P_SystemClock寄存器的b5位。

为了方便，表4-9给出了常用波特率以及当f_{OSC}=24.567 MHz时与常用波特率相对应的波特率设置寄存器P_UART_BaudScalarHigh和P_UART_BaudScalarLow设置值。

表4-9 UART的常用波特率及其设置

波特率/B·s⁻¹	高字节(\$7025H)	低字节(\$7025H)	n(十进制)	实际波特率/B·s⁻¹
1 500(最小)	1FH	FFH	8 192	1 500
2 400	14H	00H	5 120	2 400
4 800	0AH	00H	2 560	4 800
9 600	05H	00H	1 280	9 600
19 200	02H	80H	640	19 200
38 400	01H	40H	320	38 400
48 000	01H(默认)	00H(默认)	256	48 000
51 200	00H	F0H	240	51 200
57 600	00H	D5H	213	57 690
102 400	00H	78H	120	102 400
115 200(最大)	00H	6BH	107	114 841

四、红外(IR)通信控制寄存器及设置

前文已提到，UART有两种接收数据的方式，除通过Rx引脚接收数据外，还可以通过红外通信模块的接收端IRRx(IOB6)接收数据。红外通信接口是μ'nSP™的一大特色，用户可以通过红外通信接口与用户设备进行数据的发送和接收，扩大了串行通信的应用范围。

红外通信属于无线传输方式，它是使用非相干红外光线光的收发机进行的。收发双方须互相置于可视线内并对准，可以直接或经浅色表面的反射传递信息。红外不能贯穿墙壁，但可避免微波系统中出现的干扰问题。红外也没有频率分配问题。在μ'nSP™单片机中，红外通信是通过I/O口之IOB6和IOB8的特殊功能实现的，其中IOB6用来接收外部设备的数据，IOB8用来向外设发送数据。为了保证红外通信正常执行，IOB6必须设置为输入，用做红外接收端IRRx；而IOB8必须设置为输出，用做红外发送端IRTx。

红外通信的接收功能由红外通信控制寄存器控制，必须对其写入相应的控制字，确定相应的功能及传送通道时才能使用。图 4－23 给出了红外通信接收通道结构框图。图中，通道中的开关位置由红外通信控制寄存器选择。

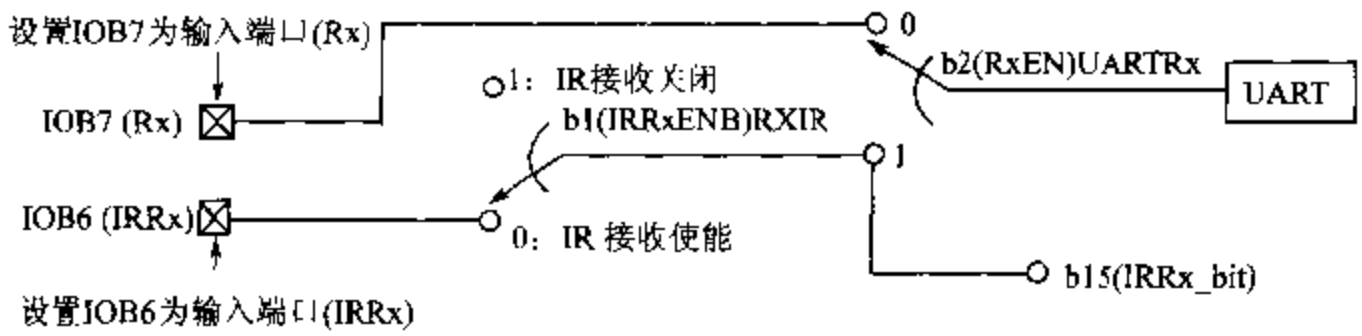


图 4－23　UART 接收通道结构框图

红外通信控制寄存器 P_IR_Ctrl(＄7018H)－ －可读/写。该寄存器用于设置红外通信功能并接收红外通信信号，即该寄存器具有双重意义，既是控制字，又是数据缓冲器。写入时为红外通信的控制字，读出时为红外通信接收的数据。其功能及格式如下。

b15	b14～b3	b2	b1	b0
IRRx_bit(读)	－	RxEN(写)	IRRxENB(写)	－

IRRxENB：红外接收使能控制位，只能写。当 IRRxENB＝1 时，禁止红外接收；当 IRRx-ENB＝0 时，允许红外接收，此为默认设置。

RxEN：UARTRx 接收信号方式控制位，只能写。当 RxEN＝1 时，通用异步串行接口 UART 模块中的 UARTRx 通过 IOB6(IRRx)接收数据；当 RxEN－0 时，UARTRx 通过 IOB7(Rx)接收数据，即为通用的异步通信功能，此为默认设置。

IRRx_bit：红外通信数据接收位，只能读。红外信号可以由通用异步串行接口 UART 模块中的 UARTRx 端口接收，亦可通过直接读 IRRx_bit 的值接收。这种灵活的接收方式使得用户可以自己定义红外通信的数据格式，也可以采用 UART 的数据格式。

红外通信的发送功能与反馈寄存器 P_FeedBack 有关，需将 P_FeedBack 寄存器中的 b0 即红外通信使能控制位 IRTxEN 的值设为“1”，允许 IOB8 的红外通信功能(默认为 0，禁止红外通信，IOB8 为 TimerA 的 PWM 输出 APWMO)。红外通信发送组成原理结构框图如图 4－24所示。

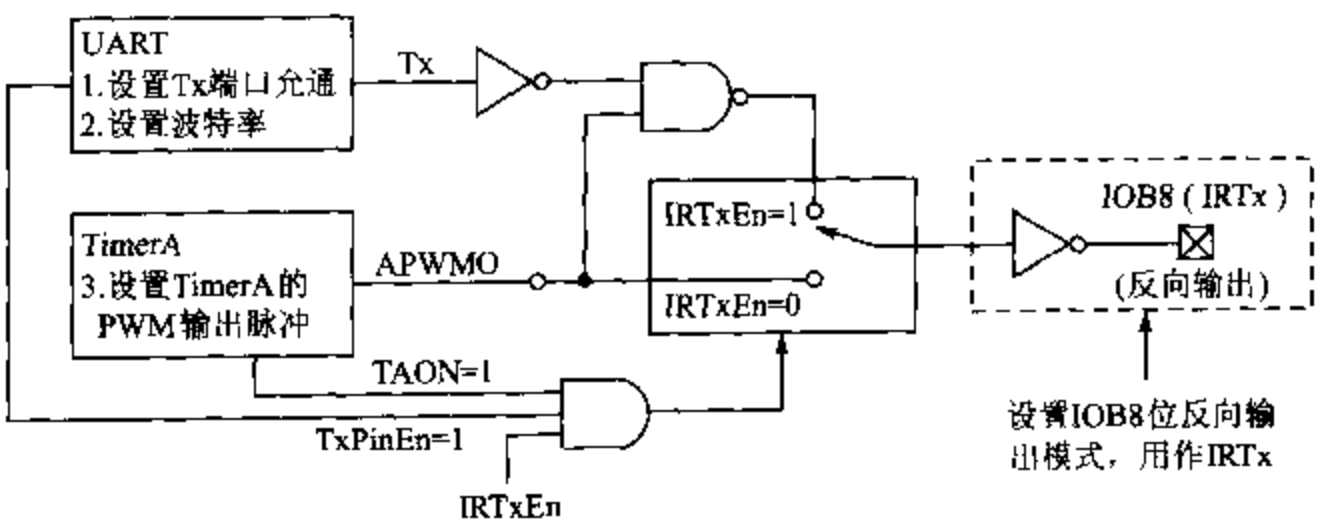


图 4－24　UART 发送组成原理结构框图

图中，IRTx信号是由UART发送端Tx发送的信号和定时器TimerA的PWM输出APWMO信号经过逻辑与门形成的，其输出波形如图4-25所示。为了产生红外发送信号IRTx，必须正确设置定时器/计数器TimerA和UART的传输工作模式。红外输出信号IRTx从IOB8输出。注意：IOB8=IRTx=Tx&APWMO。

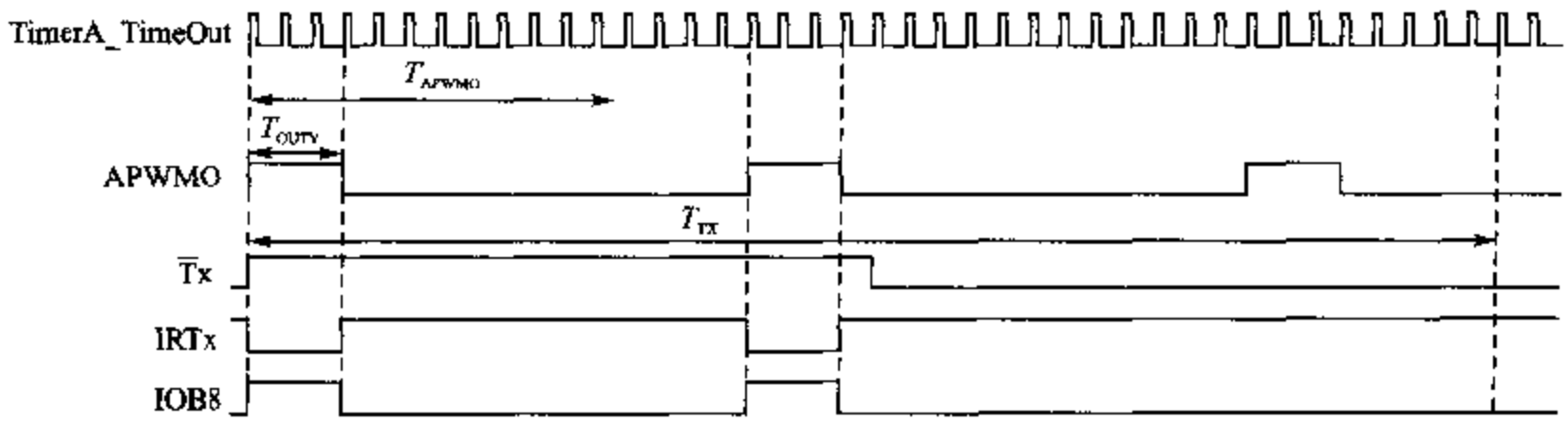


图4-25 IRTx输出波形

五、UART应用程序举例

在分布式控制或计算机网络中，下位机和上位机之间经常要传送数据和命令，这也是异步串行通信最常用的模式之一。下面给出了用通用串行异步通信接口UART接收PC机的RS—232接口上数据的子程序，共接收2 048个数据字。因UART每次只接收8位数据，而存储器是16位，所以，在接收两次数据之后再将其存入16位的SRAM中。

```
    R1=0x0480                    //将IOB10设为输出，IOB7设为输入，以满足UART的要
                                 //求，0x0480为属性向量
    [P_IOB_Attrib]=R1            //写入属性向量
    R1=Cx0400                    //设置方向向量
    [P_IOB_Dir]=R1               //写入方向向量
    R1=0x006B                    //设置波特率=12.288 MHz/107=114.84 kHz（可选择与
                                 //此频率最接近的115.2 kHz），0x006B为波特率低8位

    [P_UART_BaudScalarLow]=R1    //写入波特率低字节寄存器
    R1=0x0000                    //设置波特率高字节
    [P_UART_BaudScalarHigh]=R1   //写入波特率高字节寄存器
    R1=0xC0                      //设置UART工作模式，不进行校验，不复位UART模块，
                                 //允许发送和接收中断

    [P_UART_Command1]=R1         //写入控制寄存器1
    [P_UART_Command2]=R1         //允许Tx和Rx的发送与接收，写入控制寄存器2
L_begin_loop:                    //接收主程序开始
    R4=0                         //设置接收数据队列的SRAM起始地址
L_loop:
    R2=0                         //R2赋值0，为清除移位缓存器作准备
    R2=R2 LSL 4                  //清除移位缓存器，为接收数据移位作准备
    CALL F_UART_RECV             //调用UART接收子程序
    R1=R1 LSL 4                  //接收数据为字节型，左移8位，移至R1的高8位
    R1=R1 LSL 4
```

```
    R3= R1 &  0xFF00                    //屏蔽无用的低 8 位,数据保存在 R3
    R2  0                               //继续清除移位缓存器,为接收下一字节作准备
    R2= R2 LSL 4                        //清除移位缓存器
    CALL F_UART_RECV                    //调用 UART1 接收子程序
    R1= R1 &  0x00FF                    //屏蔽低 8 位
    R1    = R3                          //将接收的高、低 8 位字节组合为数据字,且保留在 R1 中
    [R4++]= R1                          //将接收的数据字存入 R4 指出的 SRAM 中,R4 自动加 1
    CMP R4,0x800                        //比较 R1 和 0x800,判断是否接收完 2 048 个数据字
    JNE L_loop                          //R1≠0x800,未全部接收完,继续接收下一个数据字
    JMP L_begin_loop                    //所有数据接收完,重新开始
//UART 子程序
F_UART_RECV:                            //UART 子程序标号
    PUSH R2, R3 to [SP]                 //保护寄存器 R2、R3
    L_RxRDY:                            //延时等待
    R2= 0x0080                          //R2 赋值以检查 RxRDY 是否为 1
    TEST R2, [P_UART_Command2_          //将 R2 和 UART1 的状态寄存器相与
    JZ  L_RxRDY                         //与结果为 0,表明接收未完成,转回 L_RxRDY 等待
    R1=  P_UART1_Data]                  //与结果不为 0,表明接收已完成,将接收的数据→R1
    POP R2, R3  from [SP]               //恢复现场
    RETF                                //子程序返回,R1 中为接收的数据
```

4.6　串行外设接口 SIO

串行外围设备接口 SIO(Serial interface I/O)主要用于同步串行通信。因此,通过 SIO 允许 CPU 与各种外围设备以串行方式进行通信。外围设备包括简单的 TTL 移位寄存器(用做并行输入或输出口)、复杂的 LCD 显示驱动器、A/D 转换器及 RAM 等。该接口可直接与多个厂家生产的多种标准串行外围接口器件连接,如可以与 PHILIPS 的串行接口 I^2C 及其它 I^2C 器件直接连接。

μ'nSP™的串行设备接口 SIO 采用同步串行数据传输方式与外部串行设备进行同步通信,此时数据线和时钟线是分开的。数据传输的时钟端口 SCK 和数据接收/发送端口 SDA 分别对应于并行 I/O 口 B 口的 IOB0 和 IOB1 的特殊功能。

使用时需要注意的是,SIO 是以字节为单位进行同步串行数据传输的,SIO 设置寄存器的有效读写操作均为字节型。

一、SIO 控制寄存器及其设置

SIO 控制寄存器 P_SIO_Ctrl(\$701EH)　--可读/写。它主要用于设置 SIO 通信的工作模式,其各位的具体功能和格式如下。

b7	b6	b5	b4	b3	b2	b1	b0
SIO_Config	R/W	R/W_EN	Speed_select		-	Addr_select	

Addr_select:SIO设备地址模式选择控制位:

b1	b0	
0	0	SIO设备地址为16位(A15～A0),默认设置;
0	1	SIO设备无地址;
1	0	SIO设备地址为8位(A7～A0);
1	1	SIO设备地址为24位(A23～A0)。

Speed_select:SIO传输速率选择控制位:

b4	b3	
0	0	SIO传输速率为CPUCLK/16,默认设置;
0	1	SIO传输速率为CPUCLK/4;
1	0	SIO传输速率为CPUCLK/8;
1	1	SIO传输速率为CPUCLK/32。

R/W_EN:SIO读/写控制选择控制位:

0	SIO传输使用读/写控制位,默认设置;
1	SIO传输不使用读/写控制位。

R/W:SIO读/写选择控制位:

0	SIO进行读操作,默认设置;
1	SIO进行写操作。

SIO_Config:SIO功能设置选择控制位:

0	IOB0、IOB1用做普通并行I/O口,默认设置;
1	设置IOB0为SIO功能的时钟端SCK,IOB1为数据端SDA。

二、SIO的其它寄存器及设置

设置了SIO的控制寄存器后,还必须对SIO的其它相关寄存器进行设置,才能使用SIO功能。

SIO数据寄存器P_SIO_Data(\$701AH)— 可读/写。这个口为读/写SIO数据,其格式如下。

b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	D2	D1	D0

在SIO写方式下,通过写P_SIO_Data,可由串口传送数据到SIO设备。在SIO读方式下,通过读P_SIO_Data,可接收由SIO设备发送来的数据。

SIO低8位地址寄存器P_SIO_Addr_Low(\$701BH)——可读/写。该寄存器是读/写的SIO设备的低8位地址,其格式如下。

b7	b6	b5	b4	b3	b2	b1	b0
A7	A6	A5	A4	A3	A2	A1	A0

SIO中间8位地址寄存器P_SIO_Addr_Mid(\$701CH)— 可读/写。该寄存器是要读/写的SIO设备的中间8位地址,其格式如下。

b7	b6	b5	b4	b3	b2	b1	b0
A15	A14	A13	A12	A11	A10	A9	A8

SIO 高 8 位地址寄存器 P_SIO_Addr_High($701DH)——可读/写。该寄存器是读/写的 SIO 设备的高 8 位地址，其格式如下。

b7	b6	b5	b4	b3	b2	b1	b0
A23	A22	A21	A20	A19	A18	A17	A16

SIO 启动控制寄存器 P_SIO_Start($70FAH)——可读/写。该寄存器既是 SIO 的启动控制寄存器，又是 SIO 的状态寄存器。向 SIO 启动控制寄存器写入任何一个数据，将启动 SIO 的数据传输。读 SIO 启动控制寄存器可以得到 SIO 传输的状态。P_SIO_Start 的 b7 位是 SIO 的忙标志位“Busy”。当 Busy=1 时，表明 SIO 正在进行数据传输；当 Busy=0 时，表明 SIO 已完成字节数据的传输，数据寄存器 P_SIO_Data 可接收新的数据。

SIO 停止控制寄存器 P_SIO_Stop($7020H)　　只能写。向 SIO 停止控制寄存器写入任何一个数据，将停止 SIO 的数据传输。通常，在用启动寄存器启动数据传输之前应先执行停止 SIO 操作。但在上电复位后第一次进行 SIO 传输前，不必先执行 SIO 停止操作。

三、SIO 的操作及发送/接收时序

在用 SIO 串行口传输数据时，必须对其控制寄存器 P_SIO_Ctrl 进行正确设置。使能 SIO 功能是，选择串行外围设备的地址线宽度(无地址、8 位、16 位、24 位)，设定数据传输速率，并选择是读操作还是写操作。图 4-26 给出了 SIO 串口操作的发送/接收时序。在自起始地址 P_SIO_Addr_Low、P_SIO_Addr_Mid 和 P_SIO_Addr_High 处，并从数据寄存器 P_SIO_Data 读数据或向数据寄存器写数据之前，应先向启动寄存器 P_SIO_Start 写一任意数以启动 SIO 功能。

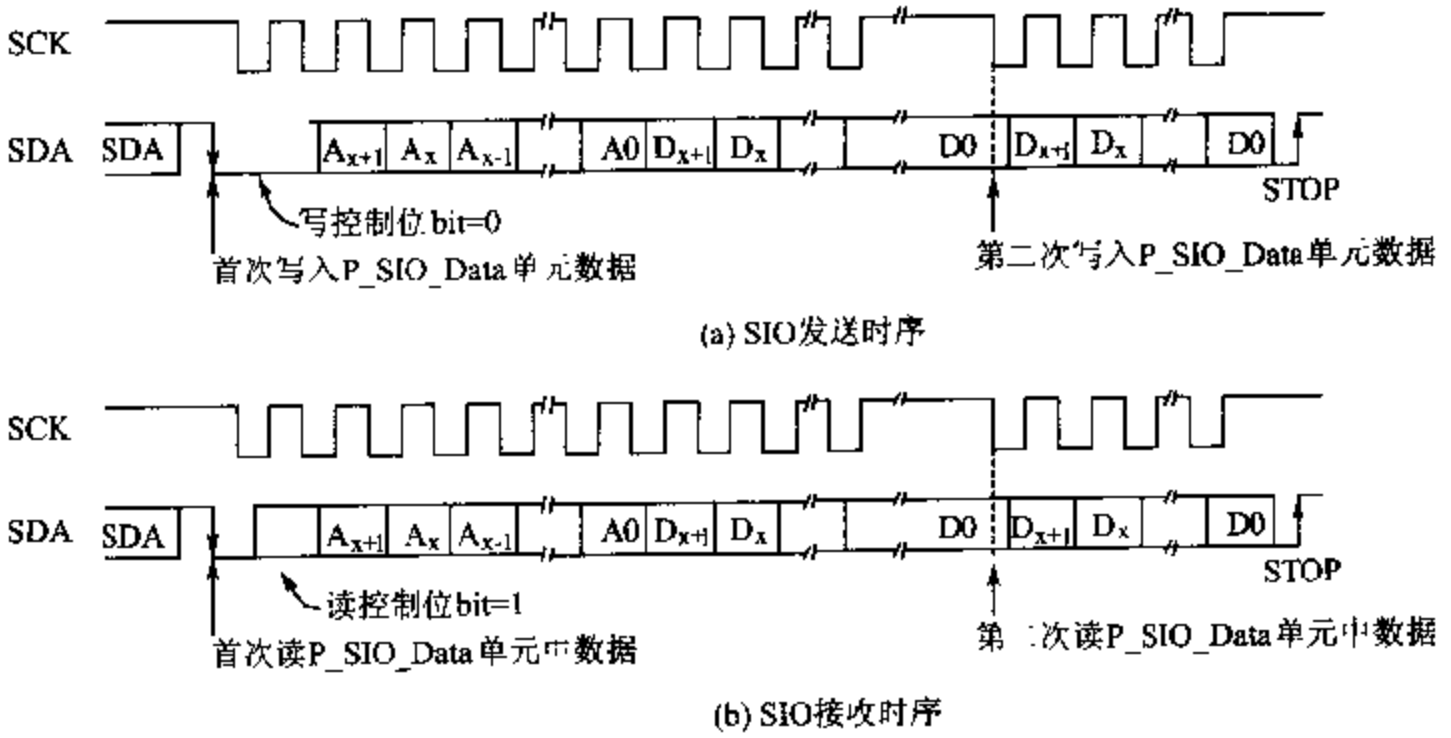


图 4-26　SIO 发送/接收时序

启动SIO功能后，在写模式下(P_SIO_Ctrl寄存器中的读/写控制位 R/W－1)，当第一次将要传输的字节数据写入数据寄存器P_SIO_Data后，SIO将立即由高位到低位逐位发送要写入的起始地址，地址发送完后再发送要写的数据。若要连续写数据，则不用再发送地址，只发送要写入的数据。若要向一个新地址写入数据，则需向SIO停止寄存器P_SIO_Stop写入一数据，停止SIO的发送。将新的地址写入三个地址寄存器中，最后再向P_SIO_Start写入任一数据，重新启动SIO功能。

启动SIO功能后，在读模式下(P_SIO_Ctrl寄存器中的读/写控制位 R/W－0)，当第一次读数据寄存器P_SIO_Data后，SIO将立即由高位到低位逐位发送要读数据的起始地址，地址发送完后再读进要读的数据。若要连续读数据，则不用再发送地址，只需读进数据即可。若要从一个新地址读数据，则需向SIO停止寄存器P_SIO_Stop写入一数据，停止SIO的工作。将新的地址写入三个地址寄存器中，最后再向P_SIO_Start写入任一数据，重新启动SIO功能。

四、SIO应用举例

图4－27给出了一个SIO应用实例的硬件接线。图中SPRS512是凌阳公司自行设计生产的串行静态RAM(SSRAM Serial Static RAM)，内部共有24位地址，在写或读操作下，SSRAM的内部地址指针将自动增或减。

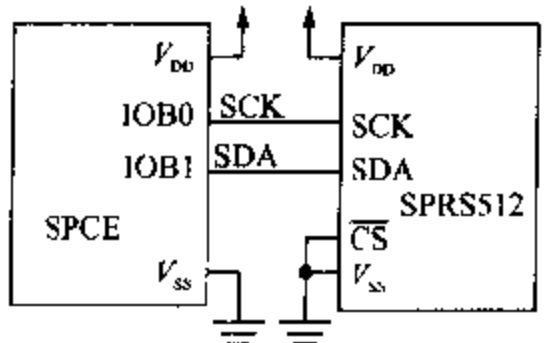


图4－27 SIO应用实例

下面是SPRS512与CPU进行串行数据传输的子程序。传输速率设置为CPUCLK/4，地址线为24位，只进行数据帧的发送操作，传输中SPRS512的地址自动增加。程序如下：

```
R1=0x00CB                    ;使能SIO功能，执行写操作使用读写控制位
                             ;传输速率为CPUCLK/4，24位地址
[P_SIO_Ctrl]=R1              ;写入SIO控制寄存器
R1=0x0000
[P_SIO_Start]=R1             ;写入P_SIO_Start寄存器任一数据，启动串行数据帧的写传输操作
R1=0x0000                    ;写入地址单元
[P_SIO_Addr_High]=R1         ;A23~A16=00000000B
[P_SIO_Addr_Mid]=R1          ;A15~A8=00000000B
[P_SIO_Addr_Low]=R1          ;A7~A0=00000000B
R1=0x0011                    ;要传输的数据赋值给R1
[P_SIO_Data]=R1              ;发送数据
L_busy:                      ;延时
R1=0x0080                    ;R1赋值以检查Busy状态位是否为1
Test R1,[P_SIO_Start]        ;将R1和SIO的状态寄存器相与
JNZ L_Busy                   ;非0，即Busy=1，未发送完，返回等待
[P_SIO_Stop]=R1              ;发送结束，写停止命令，结束数据帧的写传输操作
```

习题与思考题

1. 并行 I/O 口有几种输入方式？有几种输出方式？对应每种方式应如何设置？
2. 哪些 I/O 口有唤醒功能？应如何使用？
3. 简述 B 口有哪些特殊功能？
4. 系统时钟由哪些部分组成？应如何设置？
5. 简述时基信号有哪些作用？
6. TimerA 和 TimerB 有哪些区别？如何启动和停止定时器/计数器的工作？
7. μ'nSP™单片机有几个 A/D 通道？各有何特点？
8. A/D 转换结果是几位？如何读取？
9. A/D 转换的工作方式有几种？有几种启动方法？请说明启动的具体方法。
10. μ'nSP™单片机有哪两种音频输出方式？各有何特点？应如何分别控制？
11. 定时器/计数器的 PWM 输出与音频 PWM 输出是一个通道吗？如不是，请说明它们各自的工作原理和适宜的应用领域。
12. μ'nSP™串行通信的一帧字符有几位？其波特率如何设置？
13. 红外通信有几种接收方式？
14. 异步串行通信和同步串行通信有何不同？SIO 属于哪种串行通信方式？

第五章　中断系统

μ'nSP™的中断系统支持14个中断源，可同时设置11个中断向量。14个中断源来自音频输出、定时器/计数器、时间基准发生器、外部时钟源输入、触键唤醒以及通用异步串行接口等，适用于实时控制要求较高的应用领域。

本章从计算机中断控制的一般概念出发，着重讨论μ'nSP™单片机的中断结构，介绍该单片机各种中断源中断服务程序的设计方法。

5.1　中断的一般概念

中断是计算机进行信息处理的一种特定方式。通常有内部中断和外部中断两类。在μ'nSP™单片机中，内部中断就是软件中断(BREAK)，外部中断则是由独立于CPU内部操作的外部事件通过相应的I/O接口电路引起的。

1. 中断源的中断申请与中断处理过程

按照中断的方式与CPU之间进行信息交换的事件称为中断源。一般来说，来自于外部的中断源何时要求与计算机进行某种信息交换，对于CPU内部操作来说是随机的。在CPU正常执行预定的程序时，中断源按照自身工作的要求，向CPU提出的中断要求称之为中断申请。实际上，中断源总是通过专用的接口电路向CPU提出这种实时信息交换要求的。

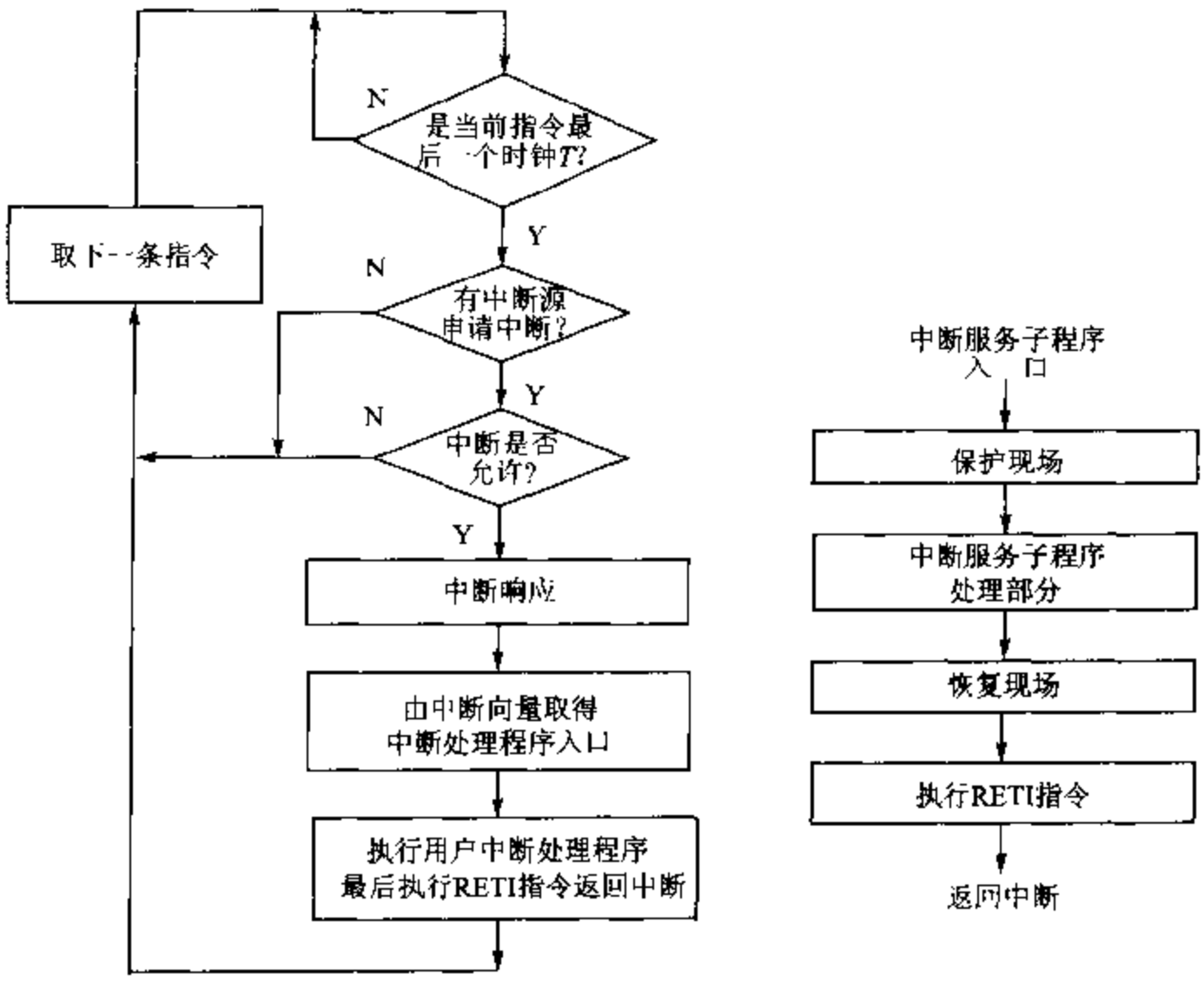
图5-1(a)是CPU中断处理一般过程的示意图。通常，该过程可由中断申请、中断响应、执行中断服务子程序及中断返回等4部分组成。CPU总是在执行到当前指令的最后才检测是否有外部的中断申请。当CPU允许某个中断源的中断申请时，将在结束本指令后进入中断响应总线周期。在中断响应总线周期中，CPU将自动完成以下3种操作：(1)保护断点，关中断；(2)获得当前中断源的中断向量；(3)转向中断服务子程序。

在中断结构的程序中，所谓断点是指中断前CPU状态寄存器的当前内容及在发生程序转移前下一条指令的存储地址。断点的内容将被压入堆栈，以便在中断返回时恢复被打断的程序断点继续执行程序。

在采用向量中断结构的计算机系统中，CPU在响应中断源的中断申请后，必须先获取该中断源的中断向量，然后才能获得进入中断服务子程序的地址信息。所谓中断向量即为指向存储中断服务子程序入口地址的指针。通常，CPU在内存开辟一个特定存储区域，用来集中存放各个中断源的中断服务子程序入口地址，简称为中断向量表。用户在编制应用程序时，先按自己的存储分配编写好相应的中断服务子程序，然后将各个子程序的入口地址填入表中。具有向量中断能力的计算机应用系统，将在对当前提出请求的中断源作出判断，若被允许则随即进入中断响应周期，并在该周期中能自动地取得当前中断源的特征信息，进而从预定的中断向量表中获得中断服务子程序入口地址。

针对特定中断事件的处理程序，称为中断服务子程序。CPU在接受某个中断源的中断申请后，将进入中断响应的处理过程。该过程除了CPU完成一系列内部操作外，还将程序引导

到相应中断服务程序的入口地址，并在最后执行中断返回指令后结束本次中断处理过程。



(a)　　(b)

图 5－1　CPU 执行中断处理流程图

2. 中断优先级与中断嵌套

一个计算机处理系统如果具有多个中断源，则需要针对不同的中断源安排相应的中断优先级。对于紧急事件，应具有较高的优先级，即在多个中断源同时申请中断时，优先级高的可先于低级中断获得响应。

如果一个中断系统在处理当前中断级的过程中，允许被更高级的中断打断，那么就称为该中断系统具有中断嵌套的能力。此时，被打断的低级中断将在较高级中断完成其服务程序并执行中断返回指令后继续进行。通常，在具有优先级中断处理结构的计算机控制系统中，一旦某个中断源的中断申请被允许并获得响应后，与它同级的中断请求将不能再打断它。如果一个正在处理的中断能允许被同级别的其它中断，则也称之为特殊的中断嵌套。

3. 开中断与关中断

在一个采用中断处理的计算机系统中，对于系统紧急复位或严重故障处理程序，往往为其设置为优先级别较高的中断源，像复位这样的中断则是完全不可被禁止的。对于一般的中断请求来说，CPU 能否进入该中断的服务子程序，首先是该中断必须给出有效的请求信号，然后 CPU 没有处在处理的较高级的中断，并且又正处于开中断状态。

为了保证对中断源的中断请求处理不受打扰，CPU 一旦进入中断响应周期并在保护断点后将自动关中断。在用户程序中，可通过专门指令对 CPU 内部的中断状态标志寄存器进行设置。该寄存器设为 1，表示开中断，或称为允许中断；该寄存器设为 0，表示关中断，或称为禁止中断。如果在进入中断服务子程序后，在程序中可用指令设置为开中断，则系统将允许继续

响应同级别的其它中断源的中断请求。

4. 中断服务子程序的一般流程

中断服务子程序的一般流程如图5-1(b)所示。其中：

保护现场：这是指利用堆栈把子程序中将要用到的某些CPU的寄存器内容暂存起来，以便在离开该子程序时予以恢复。值得指出的是，对于在子程序返回时作为参数传递的寄存器则不应列入保护现场的内容。

中断服务子程序处理部分：这是根据请求服务的中断源的具体要求而编写的一系列程序指令，其中往往包含CPU寄存器或内存的一些工作单元与I/O接口的信息交换。如果在本中断服务程序中允许被同级中断源的中断申请打断，则应在处理程序的适当部分编入开中断指令。

恢复现场：在离开该子程序之前，将保存在堆栈中的CPU寄存器的内容弹出，以恢复被中断程序必须保留的寄存器内容。

中断返回：应该通过执行中断返回指令RETI来结束本次中断服务程序，并返回被打断程序的断点。一般的子程序返回指令RET只有恢复断点地址的操作，而RETI指令在恢复断点地址的同时还能恢复中断响应时由CPU自动保存的寄存器内容。

5.2 μ'nSP™单片机的中断系统

μ'nSP™单片机具有多种中断类型及相当丰富的中断处理功能，从而能更充分地发挥其独特的微控制器和数字信号处理器的特点。

1. 中断类型

μ'nSP™单片机的中断系统有三种类型的中断，即异常中断、事件中断和软件中断。其中：

(1) 异常中断：μ'nSP™单片机的异常中断只有复位(RESET)一种。

(2) 事件中断：是外部向CPU申请的中断，也称硬件中断。有两种事件中断方式：一种是一般中断请求即IRQ中断，另一种则是快速中断请求即FIQ中断。这两种中断都可以由程序指令控制开通或关断。

(3) 软件中断：μ'nSP™的指令系统还提供了一种软中断指令(BREAK)，即由程序指令(软件)产生的中断。

2. 中断指令

```
FIQ     OFF       //禁止FIQ中断
FIQ     ON        //允许FIQ中断
IRQ     OFF       //禁止IRQ中断
IRQ     ON        //允许IRQ中断
INT     FIQ       //设置允许FIQ中断的标志
INT     IRQ       //设置允许IRQ中断的标志
INT     FIR,IRQ   //设置允许FIQ和IRQ中断的标志
INT     OFF       //设置禁止FIQ与IRQ中断的标志
RETI              //中断返回
```

3. 中断向量和中断源

中断系统的硬件中断共有 11 个中断向量，即复位、FIQ、IRQ0～IRQ6、UART IRQ 及软中断等。

这 11 个中断向量共可安置 14 个中断源供用户使用。其中，除复位、软中断之外，有三个中断源可安置在 FIQ 或 IRQ0～IRQ2 中，另有 10 个中断源则可安置在 IRQ3～IRQ6 中。以上 13 个中断源分别来自 μ'nSP™的音频输出、定时器/计数器、时间基准发生器、键唤醒和外部时钟源输入。还有一个专门用于通用异步串行口 UART 的中断源，必须安置在 UART IRQ 向量中。

4. μ'nSP™单片机中断处理过程

μ'nSP™单片机的中断处理过程与上述计算机中断处理过程基本相同，即 CPU 在响应所允许中断源的中断请求后，会按照中断优先级进入中断服务子程序，并在中断返回后 CPU 又没有接受新的中断申请，将返回被中断的程序断点继续执行程序。

系统复位 RESET 的优先级别最高。其后，优先级别依次为 FIQ 中断、IRQ 中断、UART IRQ 中断，最后是 BREAK 软中断，如表 5-1 所列。

表 5-1　μ'nSP™中断向量及中断优先级

中断向量表	中断优先级
FFF7H(复位向量)	RESET(最高)
FFF6H	FIQ
FFF8H	IRQ0
FFF9H	IRQ1
FFFAH	IRQ2
FFFBH	IRQ3
FFFCH	IRQ4
FFFDH	IRQ5
FFFEH	IRQ6
FFFFH	UART IRQ
FFF5H	BREAK(最低)

IRQ 中断又分为七级，其优先级别又依次地从 IRQ0 排到 IRQ6，即 IRQ0 的优先级别最高，直至优先级别最低的 IRQ6。

注意，这里所说的 IRQ 中断的优先级别只是在两种以上的 IRQ 中断同时发生时才起作用。该单片机的中断不允许嵌套。即如果先有一个较低级别的 IRQ 中断发生，则后面即便较高级别的 IRQ 中断产生亦是不能打断当前的 IRQ 中断响应的。

5. 中断服务子程序

中断服务子程序由编程者根据自己的需要写出。当中断发生时，这些中断服务子程序就会通过 CPU 响应中断而被执行。若中断服务子程序是用汇编语言编写的，那么程序名应在大写的中断名前加一下画线(_)。μ'nSP™的程序链接器会自动把这些服务子程序的入口地址链接到中断向量表中。例如，符号_IRQ0～_IRQ6、_BREAK、_FIQ 及_RESET 就是用汇编语言编写的一些中断服务子程序和系统复位初始化程序的标号。

例如，若要求通过 TimerA 溢出信号产生 FIQ 中断，在中断服务子程序应做相应的操作。当完成 TimerA 初始化程序后，μ'nSP™将会按照 TimerA 溢出信号产生 FIQ 中断，按 FIQ 向量指出的入口地址进入服务子程序。因为 FIQ 中断可由 FIQ_PWM、FIQ PWM_INT 及 FIQ_TMA 三个中断源产生。因此在进入中断服务程序后应通过查询相应的中断控制单元，来确认是 TimerA 溢出信号产生 FIQ 中断后，然后才可以进行以下的程序处理。

```
//TimerA 初始化
R1=0xFF9F                    // 计数器按加法计数，以补码的形式预置计数值(96)
[P_TimerA_Data]=R1
```

```
    R1=0x0030                    ;CLKB选择1,CLKA选择fosc/2
    [P_TimerA_Ctrl] R1           ;TA_TimeOut/16=(24.576 MHz/2/96)/16=8 kHz
    R1=0x2000                    ; 允许中断源FIQ_TA申请中断
    [P_INT_Ctrl]=R1
    INT FIQ                      ;开FIQ中断
L_loop_fiq:                      ;执行循环,等待中断
    JMP L_loop_fiq
```

当设定了TimerA及其中断源后,μ'nSP™将会按照TimerA的溢出频率(8 kHz)进入FIQ向量指出的服务子程序的入口地址来响应FIQ中断。由于FIQ中断源有FIQ_PWM、FIQ_TMA和FIQ_TB,因此在响应FIQ中断后,通过中断类型的判断仅处理FIQ_TMA中断。

FIQ的中断服务子程序编写如下:

```
_FIQ:
    PUSH R1,R5 to [SP]
    R1 = 0x8000
    TEST   R1,[P_INT_Ctrl]
    JNE    L_FIQ_PWM             ;判别是否为FIQ_PWM
    R1 = 0x2000
    TEST R1,[P_INT_Ctrl]
    JNE    L_FIQ_TimerA          ;判别是否为FIQ_TimerA
L_FIQ_TB:                        ;FIQ_TB服务程序入口
    R1 = 0x0800
    [P_INT_Clear] = R1           ;清除FIQ_TB的中断请求
    POP R1,R5 FROM [SP]
    RETI
L_FIQ_PWM:                       ;FIQ PWM中断服务程序入口
    [P_INT_Clear] = R1           ;清除FIQ_PWM的中断请求
    POP R1,R5 FROM [SP]
    RETI
L_FIQ_TMA:                       ;FIQ_TA服务程序入口
    [P_INT_Clear] = R1           ;清除FIQ_TA的中断请求
    ……                           ;FIQ_TA中断服务程序
    POP R1,R5 FROM [SP]
    RETI                         ;返回到主程序
```

6. 中断控制单元

μ'nSP™有两个配置单元用于设置中断控制标志:P_INT_Ctrl和P_INT_Clear,如表5-2所列。

P_INT_Ctrl单元各位的写入与读出意义是不同的。写入各位"1"或"0",可控制相应的各中断源允许或禁止中断。允许中断的中断源产生中断请求时,CPU会按照中断的优先级别进入由相应的中断向量所指的中断服务子程序入口地址来响应中断。同一个向量所指地址上的中断服务程序要读取P_INT_Ctrl单元中的各位,以判断是哪一个中断源产生了中断请求,以

便进入相应的服务程序入口进行中断处理。

表 5　2　SPCE 中断系统的配置单元

配置单元	读写属性	存储地址	功能说明
P_INT_Ctrl	读/写	7010H	写：控制各中断源允许或禁止中断；读：可判断产生请求的中断源
P_INT_Clear	写	7011H	仅用来清除中断源的中断请求

当中断源的中断请求被 CPU 响应后，该中断请求可通过写入 P_INT_Clear 单元而清除掉。CPU 执行完当前的中断服务程序，会进入下一级的中断向量所指入口地址，继续执行下一级中断服务程序，直至所有的中断服务程序都被执行完毕，便会回到中断发生前的程序指令处，继续执行下面的指令。

P_INT_Clear 单元只能被写入。当某一位被写入“1”时，便清除了该位上相应的中断源的中断请求；写入“0”时不会改变中断源的状态。例如，假设 P_INT_Ctrl 单元的 b5 写入“1”，则表示允许 2 048 Hz 时基信号产生 IRQ4 中断。当此中断发生而被 CPU 响应后进入 IRQ4 中断向量 FFFCH 单元所指的入口地址执行 IRQ4 中断服务程序。由此程序读出 P_INT_Ctrl 单元的 b5 为“1”而转去执行 2 048 Hz 中断服务程序。这时，写入 P_INT_Clear 单元“0020H”(b5 为“1”)，便清除了 2 048 Hz 的中断请求，但对其它中断源状态不会产生影响。

表 5 - 3 是 SPCE 中断控制单元 P_INT_Ctrl 各位与中断源的对应控制关系。

表 5 - 3　SPCE 中断系统的控制

中断控制位[1]	中断向量	中断源	说明
B15[2]	FFF6H	FIQ_f_{OSC}/1 024	f_{OSC}/1 024：系统时钟的 1 024 分频信号
B14		IRQ0_f_{OSC}/1 024	
B13		FIQ_TMA	TA_TimeOut_INT：TimerA 溢出信号
B12	FFF8H	IRQ1_TMA	
B11	FFF9H	FIQ_TB	TB_TimeOut_INT：TimerB 溢出信号
B10	FFFAH	IRQ2_TB	
B9	FFFBH	IRQ3_EXT2	EXT2：外部输入信号[3]
B8		IRQ3_EXT1	EXT1：外部输入信号[4]
B7		IRQ3_KEY	Key_WakeUp：键唤醒信号
B6	FFFCH	IRQ4_4 kHz	4 096 Hz 时基信号
B5		IRQ4_2 kHz	2 048 Hz 时基信号
B4		IRQ4_1 kHz	1 024 Hz 时基信号
B3	FFFDH	IRQ5_4 Hz	4 Hz 时基信号
B2		IRQ5_2 Hz	2 Hz 时基信号
B1	FFFEH	IRQ6_TMB1	TMB1：时基发生器输出的选频信号
B0		IRQ6_TMB2	TMB2：时基发生器输出的选频信号
	FFF7H	RESET	复位输入
	FFFFH	UART IRQ[5]	通用异步串行口数据传输中断
	FFF5H	BREAK	软中断

注：[1] 中断控制位为“1”时，允许相应的中断源中断，反之则禁止其中断；

[2] 控制位 B15、B14、B13、B12、B11 和 B10 两两组合在一起，可分别对系统时钟的分频信号 f_{OSC}/1 024、

TimerA 溢出 TA_TimeOut_INT 或 TimerB 溢出 TB_TimeOut_INT 进行中断允许或禁止以及中断方式的控制，如表5-4所列。

[3] 外部输入信号 EXT2 为负跳沿触发中断。

[1] 外部输入信号 EXT1 为负跳沿触发中断。

[5] UART IRQ 中断的允许或禁止是由通用异步串行口 UART 的 P_UART_Command 单元 B7(RxIntEN)和 B6(TxIntEN)设置的，且可由数据接收标志信号 RxRDY 或数据发送标志信号 TxRDY 触发中断。

表5-4是音频输出及定时器/计数器各位对中断源的对应控制关系。

表5-4 定时器/计数器中断源的控制

中断控制位						控制功能解释
B15	B14	B13	B12	B11	B10	
0	0	X[注]	X	X	X	系统时钟的1 024分频信号 f_{OSC}/1 024禁止中断
0	1	X	X	X	X	f_{OSC}/1 024允许中断且中断方式为IRQ0
1	X	X	X	X	X	f_{OSC}/1 024允许中断且中断方式为FIQ
X	X	0	0	X	X	TimerA溢出信号TA_TimeOut禁止中断
X	X	0	1	X	X	TA_TimeOut允许中断且中断方式为IRQ1
X	X	1	X	X	X	TA_TimeOut允许中断且中断方式为FIQ
X	X	X	X	0	0	TimerB溢出信号TB_TimeOut禁止中断
X	X	X	X	0	1	TB_TimeOut允许中断且中断方式为IRQ2
X	X	X	X	1	X	TB_TimeOut允许中断且中断方式为FIQ

注：X表示中断控制位的值即可为“1”，亦可为“0”。

5.3 中断处理程序举例

作为实际应用的参考，以下拟通过A/D转换、定时中断、串行通信及睡眠/键唤醒等典型程序设计为例，说明在应用μ'nSP™单片机时，实现I/O控制的基本方法。

[例1] 下面是一个关于A/D转换的应用示例。

```
.define    CPU_CLOCK_RATE      24
.define    TIME_DATA_FOR_8 kHz      (0xffff-1500)
        //定义CPU工作速率为24 MHz时，ADC输入信号按8 MHz速率采样
.include    Hardware.inc                    //该资源文件可提供硬件资源应用接口
.CODE
_InitAD_DA:
    INT     OFF                             //关中断
    R1=0x0030                               //选择 fosc/2 为TimerA的时钟源
    [P_TimeA_Ctrl]=R1
    R1= TIME_DATA_FOR_8 kHz                 //采样率 fs=8 MHz
    [P_TimeA_DATA]=R1
    R1-0x0015                               //设置MIC_IN输入通道的自动增益控制AGC功能
    [P_ADC_Ctrl]=R1                         //AIC自动工作方式，输入通道为MIC_IN
```

```
        //设置推挽方式输出的 PWM 数据锁存方式为直接将 PWM 数据送入预置数锁存器
        R1=0x0000
        [P_DAC_Ctrl]=R1
        R1=0x1000
        [P_INT_Ctrl] = R1                    //允许 TimerA 的 IRQ1 中断
        INT     IRQ                          // 开通 IRQ 中断
        retf
.TEXT
.public     _IRQ1
_IRQ1:                                       //IRQ1 中断服务子程序
    push    R1 to [SP]
    R1=[P_ADC]
    [P_ADC1]=R1
    [P_ADC2]=R1
    R1=0x1000                                //清除 IRQ1 中断请求
    [P_INT_Clear]=R1
    pop     R1      from [SP]
    retf
.CODE
.public     _mail                            //监控循环程序
_mail:
    call    _InitAD_DA                       //ADC 初始化
L_DeadLoop:
    R1=0x0001                                //清除定时监视器
    [P_Watchdog_Clear]=R1
    jmp     L_DeadLoop
```

说明：资源文件 Hardware.inc 依据 μ'nSP™提供硬件资源对 I/O 应用接口信息进行了预定义。例如，若已知 Hardware.inc 定义如下：

```
.DEFINE     P_IOA_Dir           0x7002
.DEFINE     P_IOA_Attr          0x7003
.DEFINE     P_IOA_Data          0x7000
.DEFINE     P_IOB_Dir           0x7007
.DEFINE     P_IOB_Attr          0x7008
.DEFINE     P_IOB_Data          0x7005
.DEFINE     P_IOB_Buffer        0x7006
.DEFINE     P_FeedBack          0x7009
.DEFINE     P_WatchDog_Clear    0x7012
```

因此，只要利用包含伪指令 include 将其列入，就不需要在后续程序中再分别对已定义的工作单元赋值。

[例2] 采用中断方式实现开关量的I/O操作示例如下。设IOA0～IOA7作为输入口，连接到实验装置逻辑电平开关S0～S7上，IOA8～IOA15作为输出，连接到实验装置上的发光二极管，如图5-2所示。要求编程实现，将IOA0～IOA7口开关S0～S7状态不断读到内存并送到IOA8～IOA15上的发光二极管显示出来。即，在执行所编程序时，不断改变开关S0～S7状态，可观察发光二极管的变化。实现程序如下。

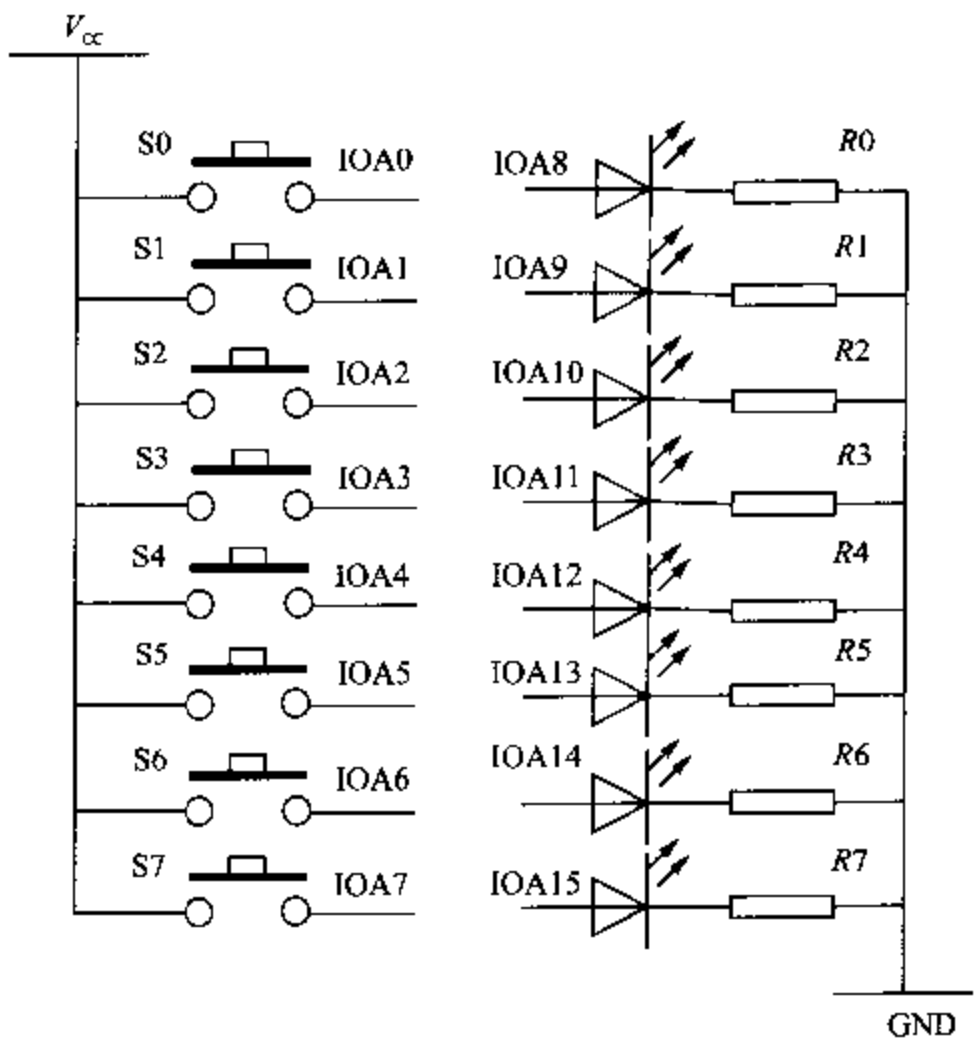


图5-2 采用中断方式实现开关量的I/O操作

```
.include      Hardware.inc                    //该资源文件可提供硬件资源应用接口
.RAM                                          //切换到RAM预定义段
.var P_DisplayData
.CODE                                         //切换到CODE预定义段
.PUBLIC _main
  _main:                                      //开始主程序 main
      irq on                                  //开中断
      int irq
      r1 = 0xff00
//IOA接口初始化，IA0～IA7用于按键输入，IA8～IA15输出到LED驱动显示
        [P_IOA_Dir] = r1
      r1 = 0xff00
        [P_IOA_Attr] = r1
      r1 = 0x0000
        [P_IOA_Data] = r1
//中断系统初始化，采用键唤醒中断的方式处理I/O操作
        r1 = 0x0080
```

```
    [P_INT_Ctrl]=r1
      r1 = [P_IOA_Latch];          // 读 P_IOA_Latch 单元,为键唤醒作准备
        r1=0x0017                  //写 P_SystemClock 单元的 B2~B0 为"111"
        [P_SystemClock] = r1       //使系统进入睡眠状态
L_Wait;                            //循环等待中断
    r1  0x0001                     //清除 WatchDog,以免产生 WatchDog_Reset 信号
    [P_WatchDog_Clear_]=r1
  jmp  L_Wait

//中断服务子程序
.public _IRQ3
 _IRQ3;
    r1=0x0080
    [P_INT_Clear]=r1               //清除中断标志
      r1=[P_IOA_Latch];            // 读 P_IOA_Latch 单元,为下一次键唤醒作准备
      r1 = [P_IOA_Data]            //读 IOA 口至 R1
      r1=r1 and 0x00ff             //屏蔽 R1 的高 8 位
      r1=r1 lsl 4                  //R1 的低 8 位移至高 8 位
      r1=r1 lsl 4
      [P_IOA_Data]=r1              //输出到 LED 驱动显示
      reti
.END
```

[例 3]　μ'nSP™系统的串行通信之功能主要靠 I/O 口的特殊功能的应用和 UART IRQ 的中断实现。它的接收口 Rx 和发送口 Tx 分别与 IOB7 和 IOB10 共用。当 CPU 接收或者发送完一帧的数据后,串行口的状态控制字 P_UART_Command2 中的控制位 RxRDY 和 TxRDY 被系统置为"1",从而触发 UART IRQ 中断。在中断处理程序中可以判断是做接收还是做发送处理,并且在 RxRDY 后,TxRDY 被读出后就消除了 UART IRQ 的中断申请。值得注意的是,在实际应用中,UART IRQ 的中断优先级比较低,故它的中断触发请求信号会在其之前发生的较高级别的中断响应时一直保持,若结合对 RxRDY 或 TxRDY 控制位的查询来保证数据及时地接收或者发送。

以下示例是关于串行口 URAT 接收数据并且触发 UART IRQ 中断的应用,从示例的角度对实际应用的程序做了简化来说明问题,程序如下。

```
//R1,R3 用来存储从 UART 接收的数据
//R2 用来标志数据接收的情况
//R4 用来计数,存储传送的数据字的个数
.include hardware.inc
.code
.public _main
_main;
    INT OFF
    R1=0x0400
```

```
    [P_IOB_Dir]=R1
    R1  0x0480;
    [P_IOB_Attrib]  R1                    //将 IOB10 定义为带数据缓存器的输出口
                                          //将 IOB7 定义为悬浮输入口
    R1  0x0000
    [P_UART_BaudScalarLow]  R1
    R1= 0x0005
    [P_UART_BaudScalarHigh]  R1           //设置波特率为 9 600

    INT IRQ
    R1=0x00c0;
    [P_UART_Command1]=R1
    [P_UART_Command2]=R1                  //状态控制字的设置,具体见串行口的介绍

L_Mainloop:
    R2  0x0000                            //标志位清零
    R4=0x0000                             //计数寄存器清零
  Loop1:
      CMP R4,0x0800                       //是否到了 2 048 个字的数据量
      JNE Loop1
JMP L_Mainloop

_IRQ7:
    R1=[P_UART_Command2]                  //读出状态字,用来清除中断请求
    R1=[P_UART_Data]                      //从缓冲器中读数据
    CMP R2,0x0000                         //检测标志位
    JE IRQ1
    JNE IRQ2
  IRQ1:  R1=R1 LSL 4
         R1=R1 LSL 4
         R3=R1                            //将 R1 的低 8 位变成 R3 的高 8 位
         R2=0x0001                        //改变标志状态
         RETI                             //中断返回
  IRQ2:  R1=R1 and 0x00ff
         R1=R1 or R3                      //形成 16 位的数据格式,用来存储
         [R4 + +]=R1                      //计数标志累加,将数据字存入数据队列
         R2=0x0000                        //改变标志状态
         RETI                             //中断返回
```

[例 4] 睡眠/键唤醒应用程序设计。单片机系统从上电复位开始工作,直到执行睡眠指令,便关闭 PLL 倍频电路,停止 CPU 工作使系统进入睡眠状态。当有唤醒信号时,接通 PLL 倍频电路,启动 CPU 重新开始工作,执行下一条指令。

此唤醒信号是中断源 IRQ3_KEY,它将产生一个 IRQ 中断请求。值得注意的是,若

要唤醒系统使其正常工作，则 IRQ1～IRQ6 都能作为唤醒源。下面用改变 IO 口状态的方法，即通常所说的键盘唤醒的方法来唤醒系统。如下程序说明系统从工作状态到睡眠状态，由键盘引起唤醒的过程。

```
. include hardware. inc
. code
. public _main
_main:
    R1 = 0x0000
    [P_IOA_Dir] = R1          //定义 IO 口 A 为输入口
    [P_IOA_Attrib] = R1
    [P_IOA_Data] = R1         //定义 IO 口 A 为带下拉电阻的输入口

    INT OFF                   //禁止 FIQ 和 IRQ 中断
    R1 = 0x0080
    [P_INT_Ctrl] = R1         //写中断控制位，允许 IRQ3_KEY(即键盘唤醒信号
                              //Key_WakeUp)的中断请求
    INT IRQ                   //允许 IRQ 中断
    R1 = [P_IOA_Latch]        //进行读 P_IOA_Latch 的操作，用来锁存 IOA1～IOA7 的数据
    R1 = 0x0007
    [P_SystemClock] = R1      //P_SystemClock 为系统时钟选频控制字，写入 0x0007 为 CPU 停止工作
  JMP  _main                  //唤醒返回后的任务
        //有键盘唤醒时，进入中断处理程序。在 IRQ3 中有 IRQ3_EXT1，IRQ3_EXT2，
        // IRQ3_KEY 有三个中断源，所以要判断出是哪种中断源产生的中断请求
_IRQ3:
      R1 = 0x0100
      TEST R1,[P_INT_Ctrl]        //判断是否是 EXT1 产生的中断请求
      JNZ  L_IRQ3_EXT1            //结果不是零，则执行 L_IRQ3_1
      R1 = 0x0200
      TEST R1,[P_INT_Ctrl]        //判断是否是 EXT2 产生的中断请求
      JNZ  L_IRQ3_EXT2            //结果不是零，则执行 L_IRQ3_2

      L_IRQ3_KyeChange_Up:
        R1 = 0x0080               //清除 IRQ3_Key 的中断请求
        [P_INT_Ctrl] = R1
            ……                    //唤醒后的中断处理程序
        RETI
      L_IRQ3_EXT2:
        [P_INT_Ctrl] = R1
         RETI                     //清除标志位，不操作返回，系统仍为睡眠状态
      L_IRQ3_EXT1:
        [P_INT_Ctrl] = R1
         RETI                     //清除标志位，不操作返回，系统仍为睡眠状态
```

思考题与习题

1. 试述 μ'nSP™单片机中断系统的基本结构与应用特点。为什么说该单片机适用于实时控制要求较高的应用领域?
2. 试述 μ'nSP™单片机的中断系统处理内部中断就是软件中断的一般过程。
3. 何谓向量中断和中断向量? μ'nSP™单片机的中断系统是如何为 FIQ、IRQ 分配中断向量的?
4. μ'nSP™单片机可由外部事件引起硬件中断 FIQ 和 IRQ,试述这两种硬件中断的应用特点。
5. 试编写一个定时中断处理程序,要求系统按 20 kHz 的频率响应中断,并在中断服务子程序中进行模拟量转换和数据采集。
6. 试应用 μ'nSP™单片机的模拟量输出实现音频波形发生器功能,要求所产生的正弦波频率在 300～3 kHz 可调。
7. 试为一个排列成 3×3 矩阵的键盘(0～9)设计处理程序,其功能为用查询与中断的方法扫描各键,要求有消除键抖动,若有键按下则返回键值。
8. FIQ 中断对应 PWM、TMA、TMB 中断源,FIQ_TMA,FIQ_TMB 中断源分别是通过定时器 A、定时器 B 产生的。当计满溢出时产生中断请求信号 TA_TIMEOUT_INT 或 TA_TIMEOUT_INT,CPU 响应后进入中断执行相应的子程序。试设计一个应用 FIQ 中断的实验系统,并通过写 P_INT_Ctrl 来设置中断允许,在中断程序里通过读取P_INT_Ctrl单元,判断是哪个中断源,并进入相应的子程序控制发光二极管点亮。

第六章　μ'nSP™16 位系列单片机

前面各章介绍了 μ'nSP™单片机的 16 位 CPU 的核心及其各部分的组成与功能，本章将重点对 μ'nSP™16 位系列单片机中的 SPCE 系列产品做扼要介绍，特别对不同点作补充说明，与前面内容相同的地方不再重复。

6.1　μ'nSP™16 位系列单片机技术性能

在第一章中已提及 μ'nSP™16 位系列单片机包括 SPCE 系列、SPT660 系列及 SPMC903 系列等，它们可分别应用于不同的领域，下面将这些系列产品的技术性能进行归纳，作一扼要地介绍。

一、SPCE 系列技术性能

SPCE 系列主要应用于发音和语音识别领域，其主要技术性能指标如下：

- 工作电压范围：2.6～5.5 V。
- 工作频率范围：0.375～49.152 MHz。
- 内置 2 K 字的静态 RAM。
- 内置 ROM 或闪存 ROM(Flash ROM)容量：零页中 32 K 字的快速 ROM 和非零页中 256 K 字的常速 ROM。
- 32 位可编程并行 I/O 口。
- 2 个 16 位可编程定时器/计数器，且都具有 PWM 输出。
- 单通道或 8 通道的 10 位 A/D 转换器，并在其中一个通道内置自动增益控制(AGC)的麦克风。
- 双通道 10 位 D/A 转换器或 1 路 PWM 输出，每个 D/A 通道的输出负载能力为 3 mA。
- 通用异步全双工串行通信接口 UART，具有 RS　232 标准的发送/接收接口时序。
- 红外通信功能，可发送或接收红外信号。
- 串行设备接口，可与串行外围设备进行串行数据传输。
- 14 个中断源，两级优先级。中断可来自系统时钟、定时器/计数器、时间基准发生器、外部中断、键唤醒、通用异步串行通信及软中断。
- 内置 32 768 Hz 实时时钟。
- 锁相环晶体振荡器或 RC 振荡器，为系统提供时钟信号。
- 掉电方式下的系统运行可将功耗降至 5 V 电源电压下的 2 μA。
- 运行/睡眠方式下的看门狗功能。
- 低电压检测/低电压复位功能。

二、SPT660系列技术性能

SPT系列主要应用于通信领域中，可对带LCD显示的来电辨识及语音拨号等功能进行控制，其主要技术性能指标如下。

- 工作电压范围：

  CPU：2.6～5.5 V；

  双音多频(DTFM)接收器/频移键控(SFK)译码器：2.2～3.6 V或2.5～5.5 V。

- 系统时钟：

  外接32 768 Hz晶体振荡器用于电话拨号及解调；

  外接电阻以形成RC振荡器，它用于CPU时钟，频率为1.0 MHz或10 MHz的1、2、4、8分频。

- 内置1 K字的静态RAM。
- 内置12 K字的ROM。
- 26位或29位可编程并行I/O口。
- 定时器/计数器：1个16位的定时器/计数器和1个8位的定时器。
- 模拟信号输入前端：

  运算放大器对双股电话线信号进行前端放大；

  内置4级或5级自动增益控制(AGC)；

  8位A/D转换，其信号采样率为8 kHz。

- 单或双DAC通道用于乐曲/语音回放功能。
- 10个中断/唤醒源(INT/WP)：

  IOC0～IOC1：边沿触发；

  环线/带反转的单线信号检测功能；

  IOA0～IOA7键唤醒：边沿触发；

  TimerA/TimerB溢出；

  系统时钟：由32 768 Hz分频得到的32 kHz、2 kHz、128 Hz和8 Hz。

- 内置LCD驱动：

  最多达224个显示(8COM×28SEG)；

  1/4偏流，1/8负载；

  内置LCD电压调整器。

- 拨号器：内置双音多频(DTFM)发生器。
- 双音多频(DTFM)发生器和频移键控FSK解调器：

  提供了DTFM发生器及FSK解调器的目标代码；

  与BELL202和ITUV2.3FSK规定兼容；

  具有FSK/DTFM译码器自动选择功能。

- 工作方式：运行方式以及用于节省功耗的备用方式和暂停方式。
- 确保系统可靠性的电源管理：

  低电压检测，具有8级电池电压低限的设定；

  上电复位功能；

看门狗复位功能；

SRAM 具有独立的电源，以存储来电记录。

三、SPMC903 技术性能

SPMC903 是一种带多路 A/D 转换和闪存 ROM 的微控制器，非常适合于产品的开发研制以及教学实验，其主要技术性能指标如下。

- 工作电压范围：2.6～3.6 V。
- 最高工作频率：24 MHz。
- 内置 2 K 字的静态 RAM。
- 内置 32 K 字的闪存 ROM。
- 64 位可编程并行 I/O 口。
- 3 个 16 位可编程的定时器/计数器，可自动重预置计数初值。
- 8 通道的 10 位 A/D 转换器。
- 2 通道 10 位 D/A 转换音频输出。
- 3 通道 10 位 PWM 驱动输出。
- 通用异步全双工串行通信接口 UART 或 I^2C 总线串行接口。
- 中断可来自系统时钟、定时器/计数器、时间基准发生器、外部中断、键唤醒及通用异步串行通信。
- 可编程选择晶体振荡器或 RC 振荡器。
- 可编程选择的看门狗功能。
- 可编程设置低电压检测/低电压复位功能。
- 可编程设置程序代码保护功能。
- 具有 SPMC903 仿真板及并行通信接口。

6.2　SPCE 系列单片机

一、概　述

SPCE 系列是 μ'nSP™ 单片机中最早研制的一种，其发展期最长，也是 μ'nSP™ 单片机中最具典型应用的一个系列，本章将主要介绍 SPCE 系列单片机。SPCE 系列单片机是在 μ'nSP™ 内核基础上研制的，其结构如图 6－1 所示。从图中可以看出，SPCE 系列单片机是以 μ'nSP™ 为核心，再结合片内外设部件所构成。其片内外设部件与第四章介绍内容基本相同。

二、SPCE 产品技术概况

SPCE 系列主要产品有 SPCE500A、SPCE060A 和 SPCE061A。表 6－1 给出了它们的技术性能及功能概况。它们都是在 SPCE 框架的基础上减少、增加或改进了一些功能而形成的。后面两节将对 SPCE500A 和 SPCE061A 与前述不同的特点作重点介绍。

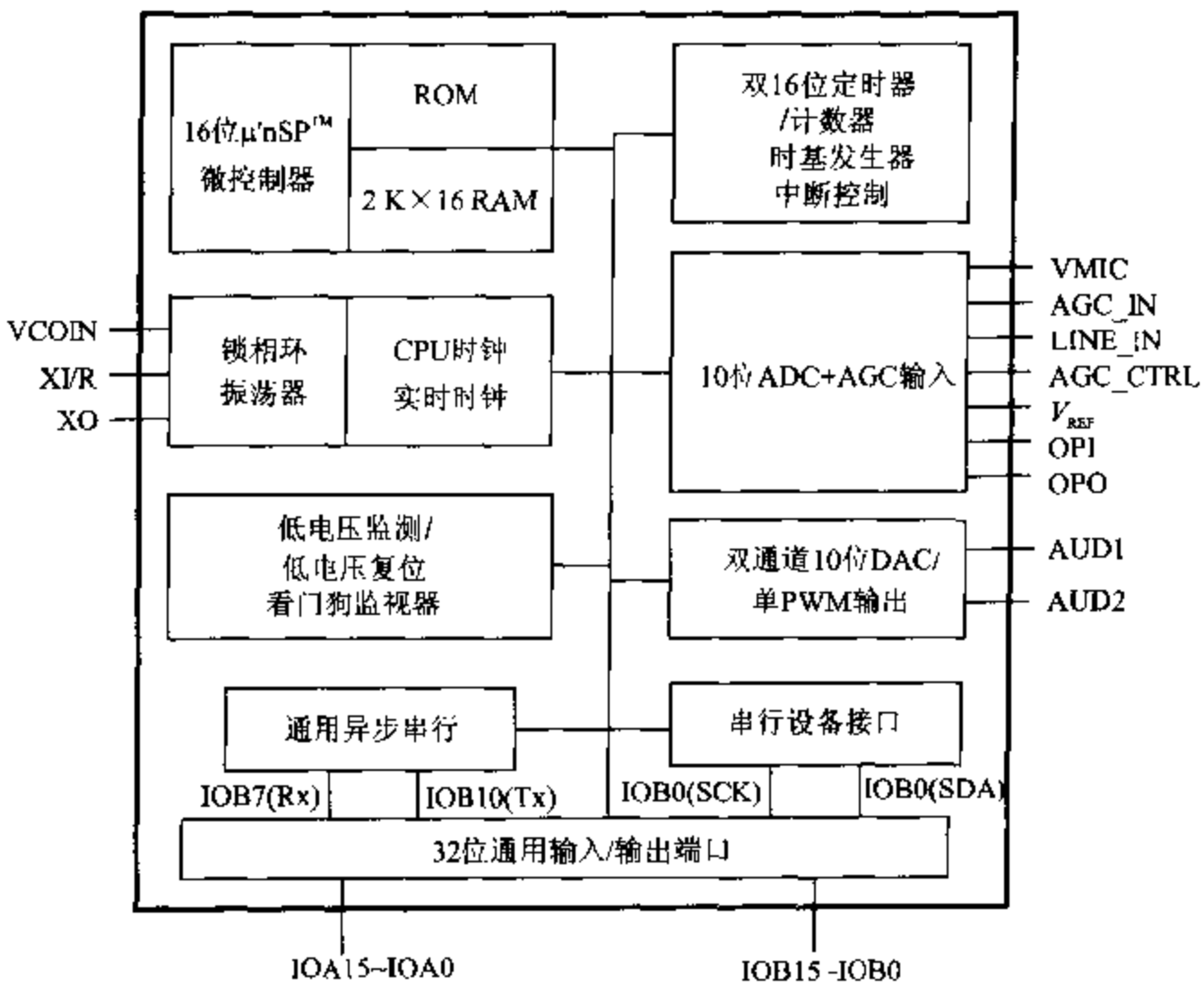


图 6-1 SPCE 系列结构框图

表 6-1 SPCE 系列产品概况

SPCE 系列	SPCE500A	SPCE060A	SPCE061A
工作电压/V	2.6~5.5	2.6~3.6	2.6~3.6
系统工作频率/MHz	24.576	49.152	49.152
SRAM 容量	2 K 字	2 K 字	2 K 字
ROM 容量	零页中 32 K 字的 快速 ROM 和非零页中 256 K 字常速 ROM	32 K 字闪存	32 K 字闪存
并行 I/O 口	16 位 A 口、16 位 B 口	16 位 A 口、16 位 B 口	16 位 A 口、16 位 B 口
ADC	单通道 10 位并内置自动 增益控制功能	8 通道 10 位且 1 个通道 具有自动增益控制功能	8 通道 10 位且 1 个通道 具有自动增益控制功能
音频输出方式	DAC×2	DAC×2	DAC×2
音频输出通道	双扬声器	双扬声器	双扬声器
中断源	定时/计数器 A/B 时基发生器 外部时钟源 键唤醒	定时/计数器 A/B 时基发生器 外部时钟源 键唤醒	定时/计数器 A/B 时基发生器 外部时钟源 键唤醒
唤醒源	IOA0~IOA7 键输入 其它中断源	IOA0~IOA7 键输入 其它中断源	IOA0~IOA7 键输入 其它中断源

续表 6-1

SPCE 系列	SPCE500A	SPCE060A	SPCE061A
定时器/计数器	双 16 位定时器·计数器	双 16 位定时器/计数器	双 16 位定时器/计数器
通用异步串口	有	有	有
红外通信接口	有	无	无
串行 SRAM 接口	无	有	有
晶振/RC 振荡器	可任选	可任选	可任选
低电压复位	可选择设置或取消	可选择设置或取消	可选择设置或取消
低电压检测	可选择设置或取消	可选择设置或取消	可选择设置或取消
看门狗	可选择设置或取消	可选择设置或取消	可选择设置或取消

6.3 SPCE500A 概述

SPCE500A 是凌阳公司第一个以 μ'nSP™为核心的 16 位微控制器。较高的处理速度使 μ'nSP™能够非常容易地、快速地处理复杂的数字信号。因此，以 μ'nSP™为基础的第一个微控制器 SPCE500A 的最初设计目标是用于数字声音和语音识别领域，并在这些领域显示出其高效、廉价的优势。

SPCE500A 的工作电压范围为 2.6～5.5 V，工作范围为 0.375～24.576 MHz，其内置存储器容量包括 32 K 字的快速零页 ROM、256 K 字的非零页常速 ROM 及 2 K 字 SRAM。其它特点包括：32 位可编程的多功能 I/O 端口；两个 16 位定时器/计数器；32 768 Hz 实时时钟；低电压复位/监测功能；内置自动增益控制功能的麦克风输入方式；10 位 DAC 方式的音频输出等。这使得 SPCE500A 能够很方便地应用于数字声音和语音识别领域。

一、SPCE500A 性能特点及应用领域

1. 性能特点

- 16 位 μ'nSP™微处理器；
- 工作电压范围：V_{DD}为 2.6～5.5 V；
- CPU 时钟：0.375～24.576 MHz；
- 内置 2 K 字的静态 RAM；
- 内置 32 K 字的快速零页 ROM 和 256 K 字的非零页常速 ROM；
- 32 位可编程并行 I/O 口；
- 2 个 16 位可编程的定时器/计数器(可自动重预置计数初始值)；
- 10 位 A/D 转换通道(内置麦克风放大器和自动增益控制 AGC 功能)；
- 双通道 10 位 D/A 转换方式的音频输出；
- 通用异步全双工串行通信接口 UART，具有 RS—232 标准的发送/接收时序；
- 14 个中断源，两级优先级，中断可来自系统时钟、定时器/计数器、时间基准发生器、外部中断、键唤醒、通用异步串行通信及软中断；

● 软件基础上的数字音频处理功能，当以 2.4 KB/s 的速率、SACM_S240 方式进行数据处理时最多可有 1 700 s 的语音数据容纳空间；

● 内置 32 768 Hz 实时时钟；

● 锁相环晶体振荡器或 RC 振荡器，为系统提供时钟信号；

● 运行/睡眠方式下的看门狗功能；

● 低电压检测/低电压复位功能；

● 掉电方式下的系统运行可将功耗降至 5 V 电源电压下的 2 μA。

2. 应用领域

● 语音识别类产品；

● 智能语音交互式玩具；

● 高级寓教于乐类玩具；

● 儿童智力开发学习类产品；

● 儿童电子故事书类产品；

● 通用语音合成器类产品；

● 需较长语音持续时间类产品。

二、SPCE500A 结构

SPCE500A 是在 SPCE 结构基础上研制的，其结构基本上与 SPCE 相同，如图 6－2 所示。SPCE500A 没有串行外围设备接口，简化了系统结构。关于其引脚见 2.6 节。

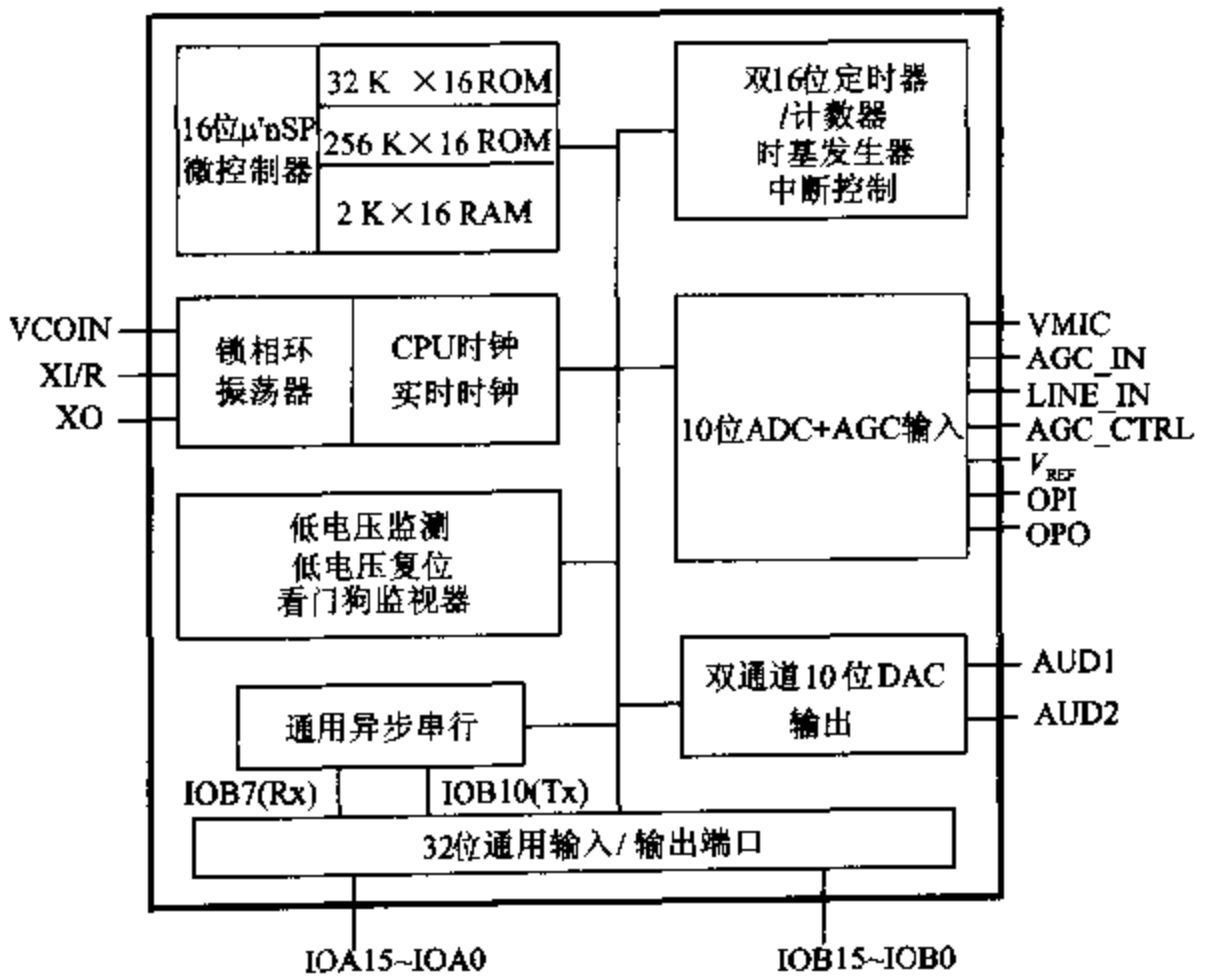


图 6－2 SPCE500A 结构框图

三、SPCE500A 存储器配置

1. 静态 RAM

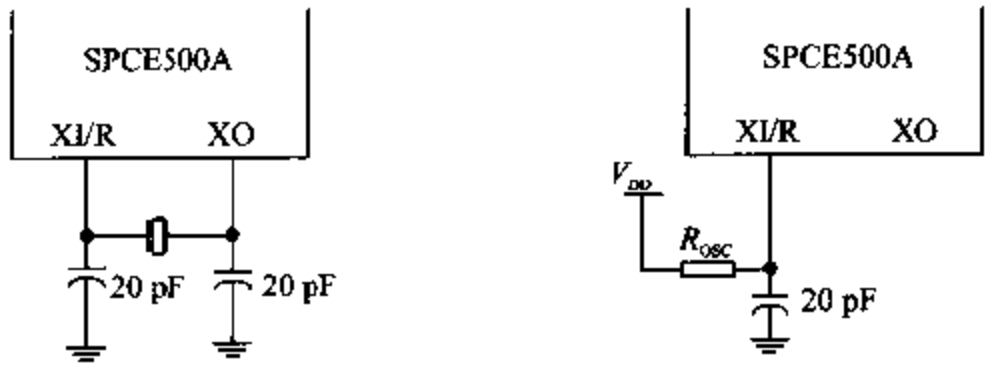
SPCE500A 内置有 2K 字的 SRAM(包括堆栈区),其地址范围从 $000000 到 $0007FF。

2. ROM

内置 ROM 分为两大块:一块是零页 32K 字的快速 ROM,其地址范围从 $008000 到 $00FFFF;另一块是非零页的 256 K 字的常速 ROM,其地址范围从 $010000 到 $04FFFF。当 CPU 时钟为 24.576 MHz 时,零页访问周期只有 3 个时钟周期,非零页访问周期为 6 个时钟周期。在其它 CPU 时钟下,所有 ROM 的访问周期都是 3 个时钟周期。

四、SPCE500A 系统时钟

SPCE500A 系统时钟电路与 4.2 节介绍的系统时钟完全一致,这里只给出 SPCE500A 与晶体振荡器或阻容(Resistor - Capacitor,RC)振荡器的连接电路原理图,如图 6 - 3 所示。用户可以在掩模时选择其中一种方式。



(a) 与晶体或陶瓷振荡器连接　　(b) 与RC振荡器连接

图 6 - 3　SPCE500A 与振荡器的连接

系统的睡眠与唤醒:芯片在上电复位开始工作直到接收到睡眠信号后,才关闭系统时钟(PLL 振荡器),进入睡眠状态。用户可以通过对 P_SystemClock 寄存器写入 CPUCLK 停止控制字(CPU 睡眠信号),使 CPU 时钟停止工作,使系统从运行状态切换至低功耗的备用状态。系统进入睡眠状态后,程序计数器(PC)会停在程序的下一条指令计数上,当有任一唤醒事件发生后开始由此继续执行程序。

系统接收到唤醒信号后接通系统时钟(PLL 振荡器),同时 CPU 会响应唤醒事件的处理和初始化。唤醒源包括:IRQ3_KEY 键唤醒源(IOA7～IOA0)和中断源(IRQ0～IRQ6),而 UART IRQ 中断不能作为唤醒源使用。唤醒操作完成后,程序将会从进入睡眠后指令计数的断点处开始继续执行。

五、SPCE500A 系统维护

1. 看门狗监视器

SPCE500A 看门狗监视器的清除周期为 0.75 s,即每隔 0.75 s 通过向 P_Watchdog_Clear ($7012H)寄存器写入"0x0001H",这样可以清除看门狗计数器。如果写操作失败,计数器溢出将产生一个复位信号。如果在备用状态下 32 768 Hz 振荡器仍工作,则看门狗计数器功能仍有效。用户可以在掩模时选择设置或取消看门狗监视器功能。

2. 低电压监测/低电压复位(LVD/LVR)功能

SPCE500A 的低电压监测/低电压复位(LVD/LVR)功能与第二章所述一样,但4级电压监测低限大小有所不同,分别为:2.6 V、3.0 V、3.6 V 和 4.0 V。低电压监测功能,可通过对P_LVD_Ctrl(\$7019H)寄存器编程进行控制。

如果系统设置了低电压复位(LVR)功能,当电源电压低于 2.4 V 时,将会在 4 个时钟周期之后产生一个复位信号,使系统复位。用户可以在掩模时选择设置或取消低电压监测/低电压复位功能。

六、SPCE500A 中断

SPCE500A 具有两种中断方式:快速中断请求 FIQ 中断和中断请求 IRQ 中断,而 FIQ 的优先级高于 IRQ。中断控制器可处理 3 种 FIQ 中断和 11 种 IRQ 中断以及一个由指令 BREAK 控制的软中断。第五章中有关中断的介绍完全适用于 SPCE500A,这里不再重复。

七、SPCE500A 片内外设部件

SPCE500A 的片内外设部件主要包括:并行 I/O 口、定时器/计数器、A/D 转换器、D/A 转换器、通用异步串行通信接口以及红外通信。它们的组成和原理与第四章介绍的单片机片内外设部件基本相同,其中定时器/计数器、A/D 转换器完全相同,只有下列两点不同:

(1) SPCE500A 不具备串行外围设备接口 SIO,所以 B 口的特殊功能没有表 4-3 中所列的 IOB0 和 IOB1 的串行外围设备接口功能。

(2) SPCE500A 音频输出方式为双 DAC 方式,没有 PWM 单通道的音频输出方式,DAC1、DAC2 转换输出的模拟量电流信号分别通过 AUD1 和 AUD2 引脚输出。

其余内容完全与第四章相同,不再重复。

八、SPCE500A 语音编码类型

SPCE500A 支持的语音编码类型如表 6-2 所列。

表 6-2 SPCE500A 语音编码类型

编码方式	编码名称	功能解释
脉冲编码调制方式	PCM	
对数脉冲编码调制方式 0	LOG PCM	
凌阳音频编码方式	SACM_A3200	
	SACM_A2000	以 24 K 位/s,20 K 位/s,16 K 位/s 的速率编码/解码
	SACM_S720	以 7.2 K 位/s 的速率编码/解码
	SACM_S480	以 4.8 K 位/s 的速率编码/解码
	SACM_S240	以 2.4 K 位/s 的速率编码/解码
	SACM_MS01	用于音调合成器的 FM 合成器和波表合成器
	SACM_A2000_DVR	数字语音记录器(Digital Voice Rccorder)

6.4 SPCE061A 概述

SPCE061A 是继 μ'nSP™ 系列产品 SPCE500A 等之后，凌阳科技推出的又一个以 μ'nSP™为核心的 16 位微控制器。与 SPCE500A 不同的是，在存储器资源方面考虑到用户较少的资源需求以及便于程序调试等功能，SPCE061A 中只内嵌 32 K 字的闪存 FLASH ROM。较高的处理速度使 μ'nSP™ 能够非常容易地、快速地处理复杂的数字信号。因此，与 SPCE500A 相同，SPCE061A 微控制器也特别适合在数字语音识别领域应用。

SPCE061A 在 2.6～3.6 V 工作电压范围内的工作频率范围为 0.32～49.152 MHz，较高的工作速度使其应用领域更加拓宽。SPCE061A 中包括 2 K 字的 SRAM 和 32 K 字的闪存 ROM，仅占一页存储空间；32 位可编程的多功能 I/O 端口；两个 16 位定时器/计数器；32 768 Hz 实时时钟；低电压复位/低电压监测功能，8 通道 10 位模/数转换输入并具有内置自动增益控制功能的麦克风输入方式；双通道 10 位 DAC 方式的音频输出功能等。SPCE061A 尤其适合于数字声音和语音识别领域。

一、SPCE061A 性能特点及应用领域

(1) 性能特点：

- 16 位 μ'nSP™ 微处理器；
- 工作电压范围：V_{DD} 为 2.6～3.6 V(CPU)，V_{DDH} 为 V_{DD}～5.5 V(I/O)；
- CPU 时钟：0.32～49.152 MHz；
- 内置 2 K 字的静态 RAM；
- 内置 32 K 字闪存 ROM；
- 32 位可编程并行 I/O 口；
- 2 个 16 位可编程定时器/计数器(可自动重预置计数初始值)；
- 7 通道 10 位 A/D 转换器和单通道声音 A/D 转换器(内置麦克风放大器和自动增益控制 AGC 功能)；
- 双通道 10 位 D/A 转换器；
- 通用异步全双工串行通信接口 UART，具有 RS—232 标准的发送/接收时序；
- 串行设备接口 SIO，可与串行外围设备进行串行数据传输；
- 14 个中断源，两级优先级，中断可来自系统时钟、定时器/计数器、时间基准发生器、外部中断、键唤醒、通用异步串行通信及软中断；
- 内置 32 768 Hz 实时时钟；
- 锁相环晶体振荡器或 RC 振荡器，为系统提供时钟信号；
- 运行/睡眠方式下的看门狗功能；
- 低电压检测/低电压复位功能；
- 可编程音频处理，使用凌阳音频编码 SACM_S240 方式(2.4 K 位/s)，能容纳 210 s 的语音数据；
- 内置在线仿真板(In - Circuit Emulator，ICE)接口；
- 掉电方式下的系统运行可将功耗降至在 3.6 V 电源电压下的 2 μA。

(2) 应用领域：

- 语音识别类产品；
- 智能语音交互式玩具；
- 高级寓教于乐类玩具；
- 儿童电子故事书类产品；
- 通用语音合成器类产品；
- 需较长语音持续时间类产品。

二、SPCE061A 结构

SPCE061A 是在 SPCE 结构基础上研制的，其结构基本上与 SPCE 相同，如图 6-4 所示。SPCE061A 内置 32 K 字的 FLASH ROM，具有 ICE 接口，方便了用户产品的研制与开发；增加了 7 路 A/D 转换器，简化了多路 A/D 转换的外围电路。

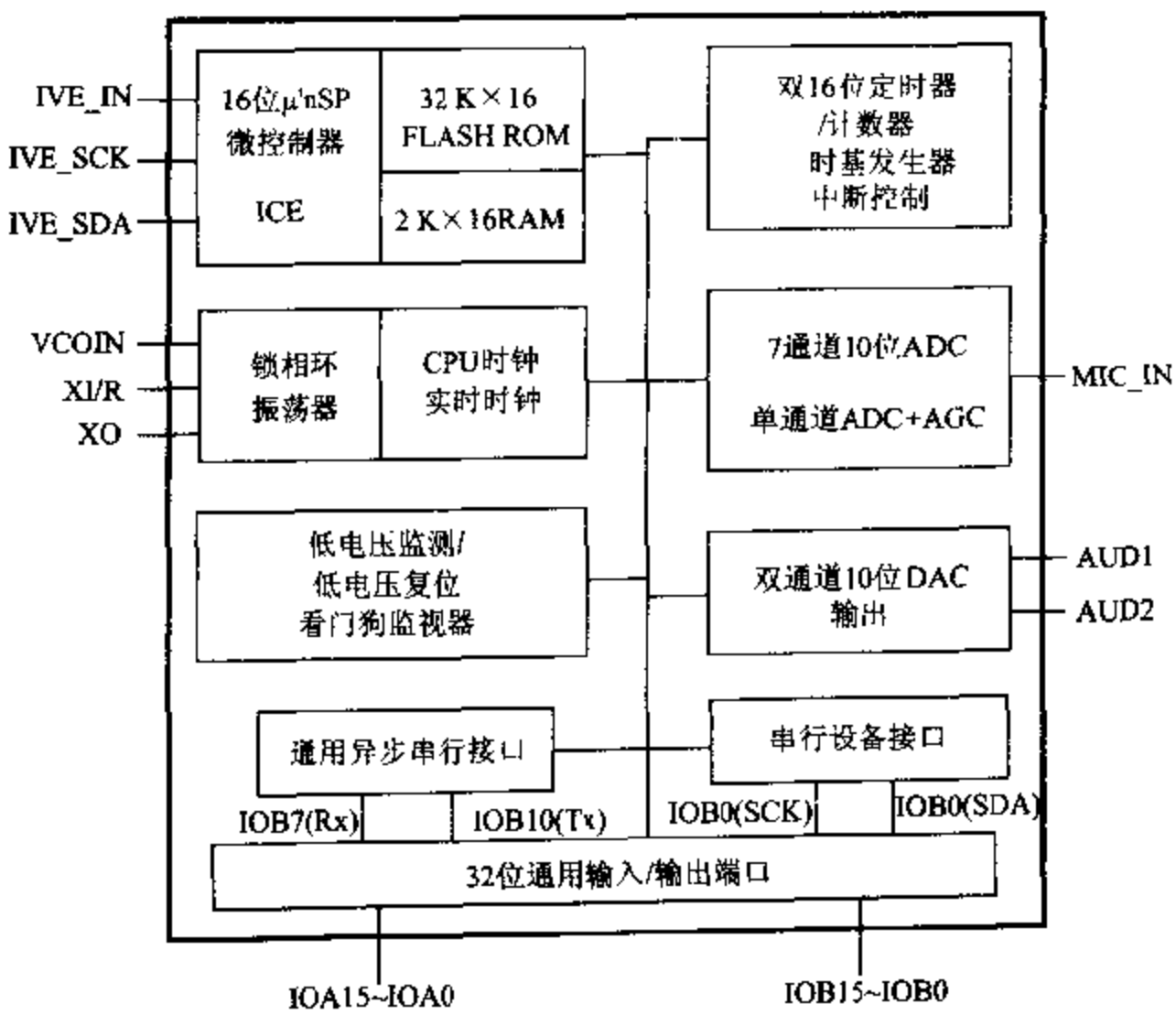


图 6-4 SPCE061A 结构框图

三、SPCE061A 存储器配置

SPCE061A 的片内存储器地址映射如图 6-5 所示，其存储器有 2 K 字的 SRAM 和 32 K 字的闪存 FLASH ROM。

1. 静态 RAM

SPCE061A 内置有 2K 字的 SRAM(包括堆栈区)，其地址范围为 0x0000～0x07FF。前 64 个字即 0x0000～0x003F 地址范围内可采用 6 位地址且以直接地址寻址方法，访问周期为 2 个 CPU 时钟周期，其余在 0x0040～0x07FF 地址范围内的存储器访问周期则为 3 个 CPU 时

钟周期。

2. ROM

内置 32 K 字的闪存 ROM，被划分为 128 个页区域(每个页的存储容量为 256 个字)，其地址范围为 0x8000～0xFFFF。其中，0xFFF5～0xFFFF 是中断向量，它们在 CPU 空闲状态下均可通过编程被设置为只读或读/写工作方式。全部 32 K 字闪存均可在 ICE 工作方式下被编程写入或被擦除。由于 SPCE061A 只有 32 K 字闪存 ROM，仅占一页存储空间，所以代码段选择字段(CS)和数据段选择字段(DS)在 SPCE061A 中不用。

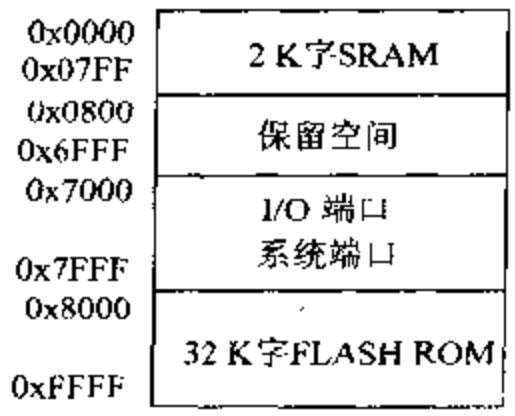


图 6-5　SPCE061A 片内存储器地址映射

闪存内有 64 个字容量的信息块，可用来存储由用户定义的一些重要信息，且必须通过 ICE 来存取。

(1) 闪存读操作：闪速存储器芯片上电以后，芯片就处于读存储单元状态，读存储单元的操作与 SRAM 相同。

(2) 闪存擦除操作：在对 FLASH 存储器芯片编程操作前，必须对 FLASH 存储器芯片进行擦除操作。由于 FLASH 存储器采用模块分区的阵列结构，使得各个存储模块(页)可以被独立地擦除。当给出的地址是在模块地址范围之内且向命令用户接口写入模块擦除命令时，相应的模块就被擦除。要保证擦除操作的正确完成，必须考虑两个因素：① 该 FLASH 存储器芯片的内部模块分区结构；② 每个模块分区的擦除时间。

(3) 闪存编程操作：FLASH 存储器芯片的编程操作是自动字节编程，既可以顺序写入，也可指定地址写入。编程操作时注意芯片的编程时间参数。FLASH 程序空间为 0x8000～0xFFFF，FLASH 命令用户接口地址为 0x7555。第一页范围是 0x8000～0x80FF，最后一页是 0xFF00～0xFFFF。图 6-6 给出了闪存编程操作流程图。

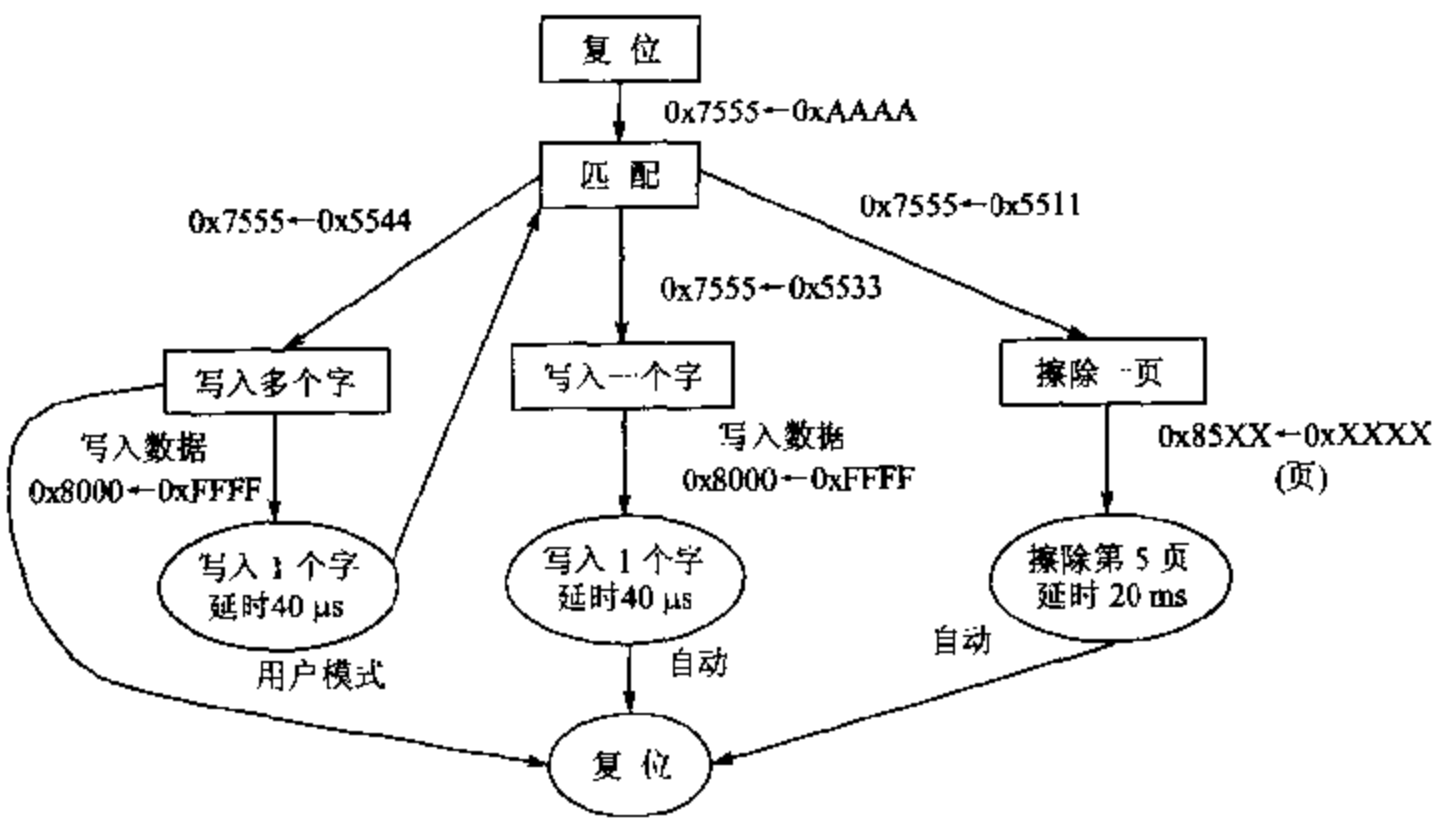


图 6-6　闪存编程操作流程图

擦除一页流程：先给命令用户接口地址 0x7555 中送 0xAAAA，再给命令用户接口地址 0x7555 中送 0x5511，然后给要擦除页地址送任意数，约 20 ms 即可完成擦除操作，随后就可以执行其它操作。例如图 6-6 中擦除第 5 页 0x8500～0x85FF 的流程如下：① 0x7555←0xAAAA；② 0x7555←0x5511；③ 0x85XX←XXXX，X 为任意值。

写入一个字流程：先给命令用户接口地址 0x7555 中送 0xAAAA，再给命令用户接口地址 0x7555 中送 0x5533，然后给要写入字地址送数据，约 40 μs 即可完成写入操作，随后就可以执行其它操作。例如向 0x8000 单元写入 0xFFFF 的流程是：① 0x7555←0xAAAA；② 0x7555←0x5533；③ 0x8000←0xFFFF。

写入多个字流程：先给命令用户接口地址 0x7555 中送 0xAAAA，再给命令用户接口地址 0x7555 中送 0x5544，然后给要写入字首地址送数据，约 40 μs 即可完成 1 个字的写入操作，再给命令用户接口地址 0x7555 中送 0x5544，给要写入字地址送数据，等待 40 μs 即可，循环操作就可完成多字的写入。

上面所提到的延时等待是由硬件完成且不需要软件延时。下面给出了完成上述擦除一页、写一字、写多字的子程序。

```
//=============================================
//程序名称 FLASH.asm
//描述:擦除、写1字、写多字子程序
//=============================================
.include hardware.inc
.define C_FLASH_SIZE              0x8000        //32 K 字
.define C_FLASH_BLOCK_SIZE        0x100         //256
.define C_FLASH_MATCH             0xAAAA
.define C_FLASH_PAGE_ERASE        0x5511        //擦除一页控制字
.define C_FLASH_1WORD_PGM         0x5533        //写一个字的控制字
.define C_FLASH_SEQUENT_PGM       0x5544        //写多个字的控制字
.code
//=============================================
//函数名称:F_FlashWrite1Word()
//描述:写一个字到 FLASH 中
//入口参数:1. 被写数据的存储地址;2. 被写数据
//无出口
//=============================================
.public _F_FlashWrite1Word
.define P_Flash_Ctrl          0x7555
_F_FlashWrite1Word:           .proc
    push bp to [sp]
    bp=sp+1

    r1=C_FLASH_MATCH                          //0xAAAA
    [P_Flash_Ctrl]=r1
    r1=C_FLASH_1WORD_PGM                      //0x5533
```

```
    [P_Flash_Ctrl]=r

    r1=[bp+3]                              //取存储数据地址
    r2=[bp+4]                              //取数据
    [r1]=r2
    //delay 10μs                           //不需要软件考虑而可硬件完成

    pop bp from [sp]
    retf
    .endp
//==================================================================
//函数名称:F_FlashWrite()
//描述:顺序写多个字
//入口参数:1. 被写数据的起始地址;2. 被写数据;3. 写数据的数量
//无出口
//==================================================================
.public _F_FlashWrite
_F_FlashWrite:          .proc
    push bp to [sp]
    bp=sp+1

    r1=[bp+3]                              //取被写数据的首地址
    r2=[bp+4]                              //取被写数据
    r3=[bp+5]                              //写 N 个字

    r4=C_FLASH_MATCH                       //0xAAAA
    [P_Flash_Ctrl]=r4
L_FlashWriteLoop:
    r4=C_FLASH_SEQUENT_PGM                 //0x5544
    [P_Flash_Ctrl]=r4

    r4=[r2++]
    [r1++]=r4
    //delay 40μs                           //不需要软件考虑硬件完成
    r3-=1
    jnz L_FlashWriteLoop
    [P_Flash_Ctrl]=r3                      //写结束

    pop bp from [sp]
    retf
    .endp
//==================================================================
//函数名称:F_FlashErase()
```

```
//描述:擦除256个字节
//入口参数:擦除页的起始地址
;=============================================================
.public _F_FlashErase
_F_FlashErase;              .proc
    push bp to [sp_
    bp=sp+1

    r1 =C_FLASH_MATCH                          //0xAAAA
    [P_Flash_Ctrl]=r1
    r1=C_FLASH_PAGE_ERASE                      //0x5511
    [P_Flash_Ctrl]=r1

    r1=[bp+3]                                  //取擦除页内的地址
    [r1]=r1
    //delay 20ms                               //不需要软件考虑硬件完成

    pop bp from [sp]
    retf
    .endp
    .end
//=============================================================
```

四、SPCE061A 的系统时钟

1. 锁相环 PLL 振荡器

锁相环 PLL 用于为系统提供一个实时时钟的基频 32 768 Hz，再将基频进行倍频，调整至 49.152 MHz、40.96 MHz、32.768 MHz、24.576 MHz 或 20.480 MHz，并作为系统时钟 f_{osc}。系统默认的 PLL 自激振荡频率为 24.576 MHz。系统时钟的结构如图 6-7 所示。

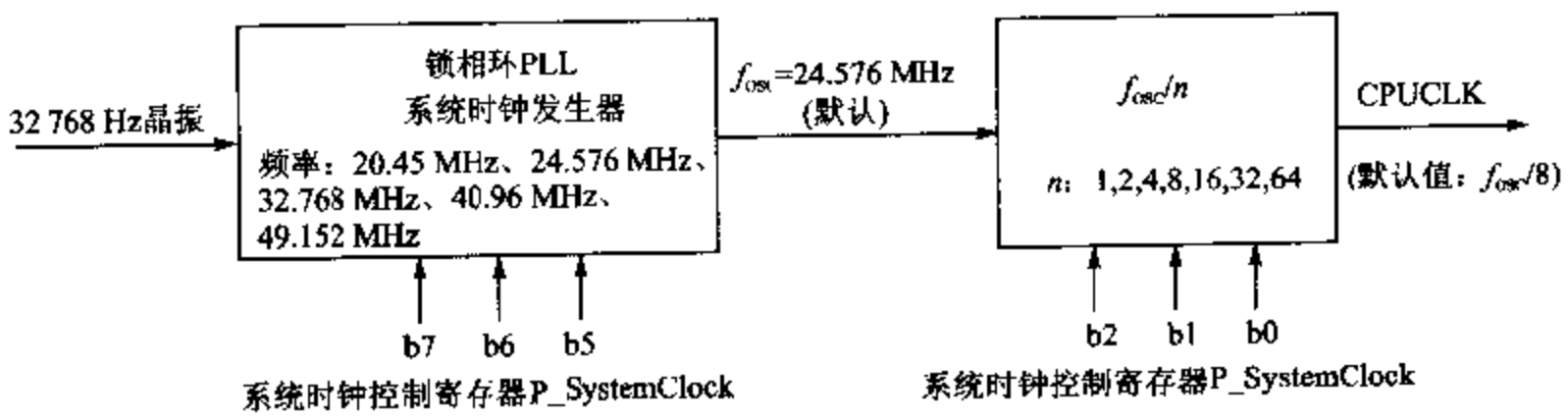


图 6-7 SPCE061A 系统时钟结构框图

2. 系统时钟控制寄存器及其设置

系统时钟的信号源为 PLL 振荡器。系统时钟频率 f_{osc} 和 CPU 时钟频率 CPUCLK 由系统时钟控制寄存器 P_SystemClock(\$7013H)控制。P_SystemClock 寄存器只能写，各位的具体功能和格式如下。

b15~b8	b7~b5	b4	b3	b2	b1	b0
…	系统时钟选择	32 kHz 睡眠状态	32 kHz 方式选择	CPU 时钟选择		

其中，系统时钟频率(f_{osc})和 CPU 时钟频率(CPUCLK)的选择如表 6-3 所列。

表 6-3 SPCE061A 系统频率和 CPU 频率的选择

b7	b6	b5	系统频率/MHz	b2	b1	b0	CPU 时钟频率
0	0	0	24.576(默认设置)	0	0	0	f_{osc}
0	0	1	20.48	0	0	1	$f_{osc}/2$
0	1	0	32.768	0	1	0	$f_{osc}/4$
0	1	1	10.96	0	1	1	f_{osc} 8(默认设置)
1	0	0	49.152	1	0	0	$f_{osc}/16$
1	0	1	49.152	1	0	1	$f_{osc}/32$
1	1	0	49.152	1	1	0	$f_{osc}/64$
1	1	1	49.152	1	1	1	停止(睡眠状态)

b3、b4 位的含义如下：

b3—— 32 kHz 方式选择：

0 32 768 Hz 时钟处于自动弱振模式，系统默认设置；

1 32 768 Hz 时钟处于强振模式。

b4——32 kHz 睡眠状态：

0 在备用状态下，32 768 Hz 时钟被关闭；

1 在备用状态下，32 768 Hz 时钟仍处于工作状态，系统默认设置。

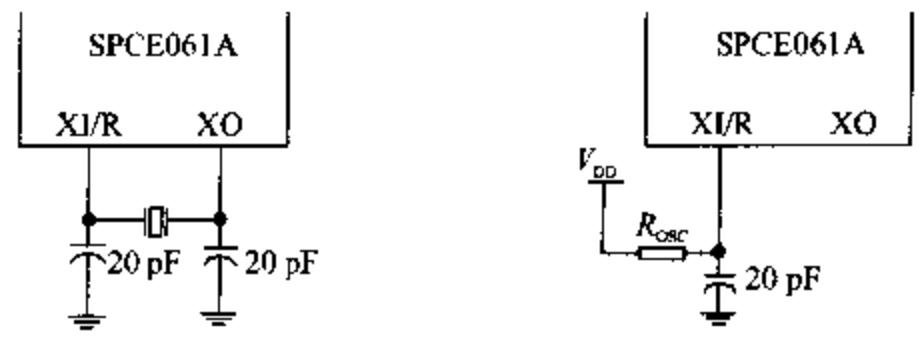
可见，32 768 Hz 振荡器有两种工作模式：强振模式和自动弱振模式。处于强振模式时，32 768 Hz 振荡器始终运行在高耗能的状态下；处于自动弱振模式时，系统在上电复位后的前 7.5 s 内处于强振模式，然后自动切换到弱振模式以降低功耗。CPU 被唤醒后默认的时钟频率为 $f_{osc}/8$，用户可以根据需要调整该值，这样可以避免在系统被唤醒后造成 ROM 读取的错误。

当系统时钟控制寄存器 P_SystemClock 的第 0~2 位设置为“111”时，将使 CPU 时钟停止工作，系统切换至低功耗的备用状态。在备用状态下，通过设置该寄存器的 b4 位可以接通或关闭 32 kHz 实时时钟。注意，只有在备用状态下，b4 设置才有效。

图 6-8 给出了 SPCE061A 与晶体振荡器或阻容 RC 振荡器的连接电路原理图，用户可以在掩模时选择其中一种方式。

3. 系统的睡眠与唤醒

芯片在上电复位开始工作直到接收到睡眠信号后，才关闭系统时钟(PLL 振荡器)，进入睡眠状态。用户可以通过对 P_System Clock 寄存器写入 CPUCLK 停止控制字(CPU 睡眠信号)使 CPU 时钟停止工作。这样，系统将从运行状态切换至低功耗的备用状态。系统进入睡眠状态后，程序计数器(PC)会停在程序的下一条指令计数器上，当有任一唤醒事件发生后开始由此继续执行程序。



(a) 与晶体或陶瓷振荡器连接　　(b) 与RC振荡器连接

图 6-8　SPCE061A 与振荡器的连接

系统接收到唤醒信号后接通系统时钟(PLL 振荡器),同时 CPU 会响应唤醒事件的处理并进行初始化。唤醒源包括:IRQ3_KEY 触键唤醒源(IOA7～IOA0)和中断源(IRQ1～IRQ6及 UART IRQ)。唤醒操作完成后,程序将会从进入睡眠状态后,指令计数的断点处开始继续执行。

五、SPCE061A 系统维护

1. 看门狗监视器

SPCE061A 看门狗监视器的清除周期为 0.75 s,即每隔 0.75 s 通过向 P_Watchdog_Clear ($7012H)寄存器写入"0x0001H",以此清除看门狗计数器。如果写操作失败,计数器溢出将产生一个复位信号。如果在备用状态下,32 768 Hz 振荡器仍工作,则看门狗计数器功能仍有效。用户可以在掩模时选择设置或取消看门狗监视器功能。

2. 低电压监测/低电压复位(LVD/LVR)功能

SPCE061A 的低电压监测/低电压复位(LVD/LVR)功能与第二章所述一样,具有 4 级电压监测低限:2.4 V、2.8 V、3.2 V 和 3.6 V。低电压监测功能,可通过对 P_LVD_Ctrl ($7019H)寄存器编程进行控制。

如果系统设置了低电压复位(LVR)功能,当电源电压低于 2.2 V 时,将会在 4 个时钟周期之后产生一个复位信号,使系统复位。用户可以在掩模时选择设置或取消低电压监测/低电压复位功能。

六、SPCE061A 中断

SPCE061A 同样具有两种中断方式:快速中断请求 FIQ 和中断请求 IRQ。中断控制器可处理 3 种 FIQ 中断和 11 种 IRQ 中断以及一个由指令 BREAK 控制的软中断。表 6-4 给出了各个中断源、中断向量及保留字,其保留字是中断服务程序的入口点。

中断控制寄存器有 3 个:P_INT_Ctrl $7010H)、P_INT_Clear($7011H)和 P_INT_New ($702DH)。前两个控制寄存器的含义及设置参见第五章。P_INT_New 寄存器用于激活和屏蔽中断,每一位对应中断源如下图所示。

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3	IRQ4	IRQ4	IRQ4	IRQ5	IRQ5	IRQ6	IRQ6

b15	b14	B13	b12	b11	b10	b9	b8
FIQ	IRQ0	FIQ	IRQ1	FIQ	IRQ2	IRQ3	IRQ3

表 6－4 SPCE061A 中断源

中断源	中断优先级	中断向量	保留字
f_{OSC}/1024 溢出信号	FIQ/IRQ0	FFF8H/FFF6H	_FIQ/_IRQ0
TimerA 溢出信号	FIQ/IRQ1	FFF9H/FFF5H	_FIQ/_IRQ1
TimerB 溢出信号	FIQ/IRQ2	FFFAH/FFF6H	_FIQ/_IRQ2
外部时钟源输入信号 EXT2	IRQ3	FFFBH	_IRQ3
外部时钟源输入信号 EXT1			
触键唤醒信号			
4 096 Hz 时基信号	IRQ4	FFFCH	_IRQ4
2 048 Hz 时基信号			
1 024 Hz 时基信号			
4 Hz 时基信号	IRQ5	FFFDH	_IRQ5
2 Hz 时基信号			
频选信号 TMB1	IRQ6	FFFEH	_IRQ6
频选信号 TMB2			
UART 传输中断	IRQ7	FFFFH	_IRQ7
BREAK	软中断 FFF5H		

当 P_INT_New 寄存器中某位设置为“1”，则允许中断/唤醒功能；若设置为“0”，则屏蔽中断/唤醒功能，但不清除 P_INT_Ctrl 寄存器相应的中断标志位。

七、SPCE061A 语音编码类型

SPCE061A 支持的语音编码类型如表 6－5 所列。

表 6－5 SPCE061A 语音编码类型

编码方式	编码名称	功能解释
脉冲编码调制方式	PCM	
对数脉冲编码调制方式 0	LOG PCM	--
凌阳音频编码方式	SACM_A3200	--
	SACM_A2000	以 21 K 位/s、20 K 位/s、16 K 位/s 的速率编码/解码
	SACM_S720	以 7.2 K 位/s 的速率编码/解码
	SACM_S480	以 4.8 K 位/s 的速率编码/解码
	SACM_S240	以 2.4 K 位/s 的速率编码/解码
	SACM_MS01	用于音调合成器的 FM 合成器和波表合成器

6.5 SPCE061A 片内外设部件

SPCE061A 的片内外设部件主要包括：并行 I/O 口、定时器/计数器、A/D 转换器、D/A 转换器、通用异步串行通信接口以及串行外围设备接口。它们的组成和原理与第四章介绍的单片机片内外设部件基本相同，其中定时器/计数器和串行外围设备接口完全相同。本节主要针对 SPCE061A 的不同点作简要介绍，其余相同部分可参见第四章内容。

一、并行 I/O 口

SPCE061A 具有两个 16 位可编程并行 I/O：A 口和 B 口。A 口既具有可编程功能的普通 I/O 口，又可与 ADC 的多路输入 LINE_IN 共用。其中 IOA0～IOA7 还具有唤醒功能。B 口除了具有普通 I/O 口的功能外，在特定的引脚上还可以完成一些特殊的功能。除 4.1 节介绍的内容外，SPCE061A 的并行 I/O 口有下列两点不同：

(1) IOA0～IOA6 与 A/D 转换器的输入 LINE_IN1～LLINE_IN7 共用，详见 A/D 转换部分的 P_ADC_MUX_Ctrl 控制寄存器。

(2) SPCE061A 不具备红外通信功能，所以，B 口的特殊功能没有表 4－3 中所列的 IOB6 的红外通信数据接收、IOB8 的红外通信数据发送功能。IOB8 的应用方式由 TAON、TxPinEN 控制，如表 6－6 所列。

表 6－6 IOB8 应用方式设置

TAON	TxPinEN	IOB8
0	0	普通 I/O 端口
0	1	普通 I/O 端口
1	0	APWM0 端口
1	1	UART Tx 端口

关于并行 I/O 口的其它功能参见 4.1 节。

二、模/数转换器 ADC

SPCE061A 有 8 个 10 位模/数转换器，其中 7 个通道用于将模拟量信号（例如电压信号）转换为数字量信号，并直接通过并行 I/O A 口的 IOA0～IOA6 输入。另外有一个通道只作为语音输入通道，通过内置有自动增益控制放大器的麦克风通道 MIC_IN 输入。SPCE061A 的 A/D 转换器与 4.3 节介绍的 A/D 转换器工作原理相同，都采用逐次逼近式转换，但控制有所不同，下面重点阐述其不同点。

1. ADC 控制寄存器

ADC 控制寄存器包括下述 4 个寄存器：

(1) ADC 控制/状态寄存器：ADC 控制/状态寄存器 P_ADC_Ctrl（$7015H）——可读/写。该寄存器有两种含义，既是控制寄存器，又是状态寄存器。写入时为 ADC 转换控制字，读出时为 ADC 状态字。其功能及格式如图 6－9 所示。

设置此控制字时需注意与 4.3 节不同：SPCE061A 的 ADC 只工作于自动方式，没有手动方式；状态位 b15 只用于 MIC_IN 麦克风输入通道；写入时需特别注意没用到的位的特殊要求：b5＝1、b4＝1、b3＝1、b1＝0。

(2) ADC MIC 通道数据寄存器：ADC MIC 通道数据寄存器 P_ADC（$7014H）——可读/写。该寄存器储存 MIC_IN 输入的 A/D 转换的结果。CPU 对该寄存器执行读操作就是

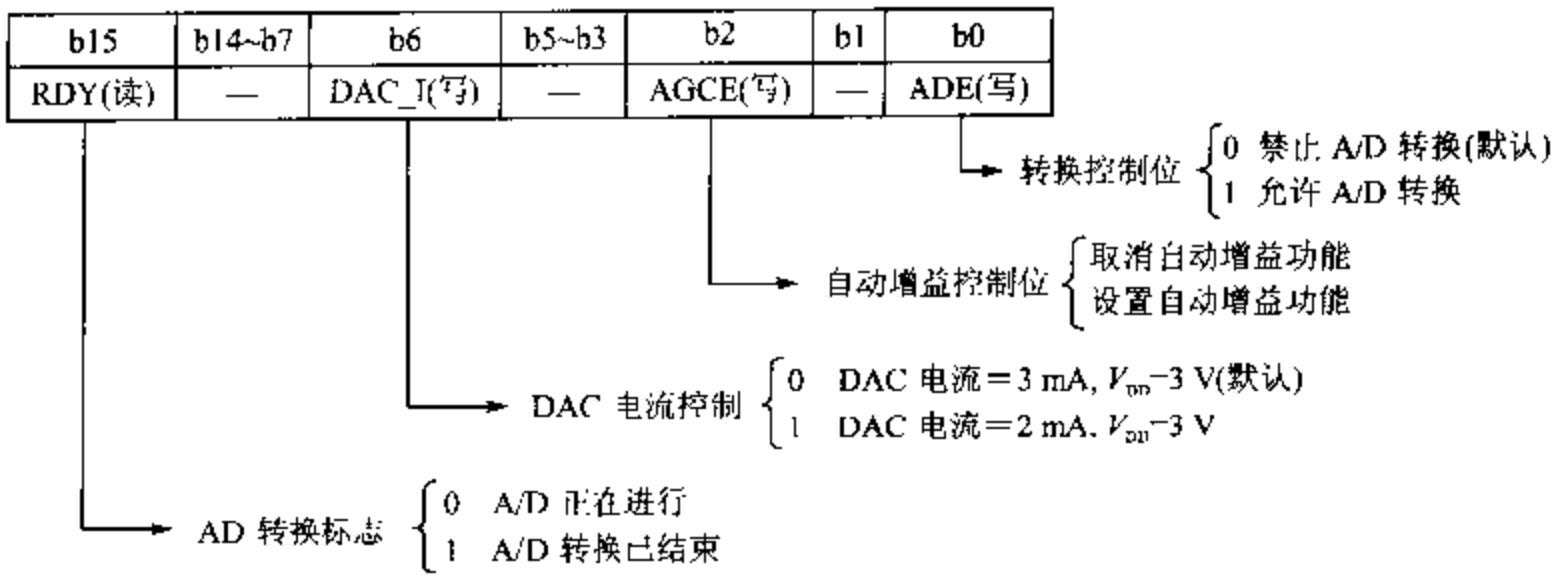


图 6-9　ADC 控制/状态寄存器格式及功能

读 MIC_IN 通道 A/D 转换输出的 10 位数字量。读出的结果只有高 10 位有效，低 6 位无意义。注意：如果 P_DAC_Ctrl 寄存器的第 3、4 位被设为“00”，那么在转换过程中读出本数据寄存器将触发 A/D 转换重新开始。

(3) ADC 多通道控制/状态寄存器：ADC 多通道控制/状态寄存器 P_ADC_MUX_Ctrl ($702BH)——可读/写。该寄存器也有两种含义，即既是多通道控制寄存器，又是多通道状态寄存器。写入时为多通道 ADC 控制字，读出时为多通道 ADC 状态字。其功能及格式如图 6-10 所示。

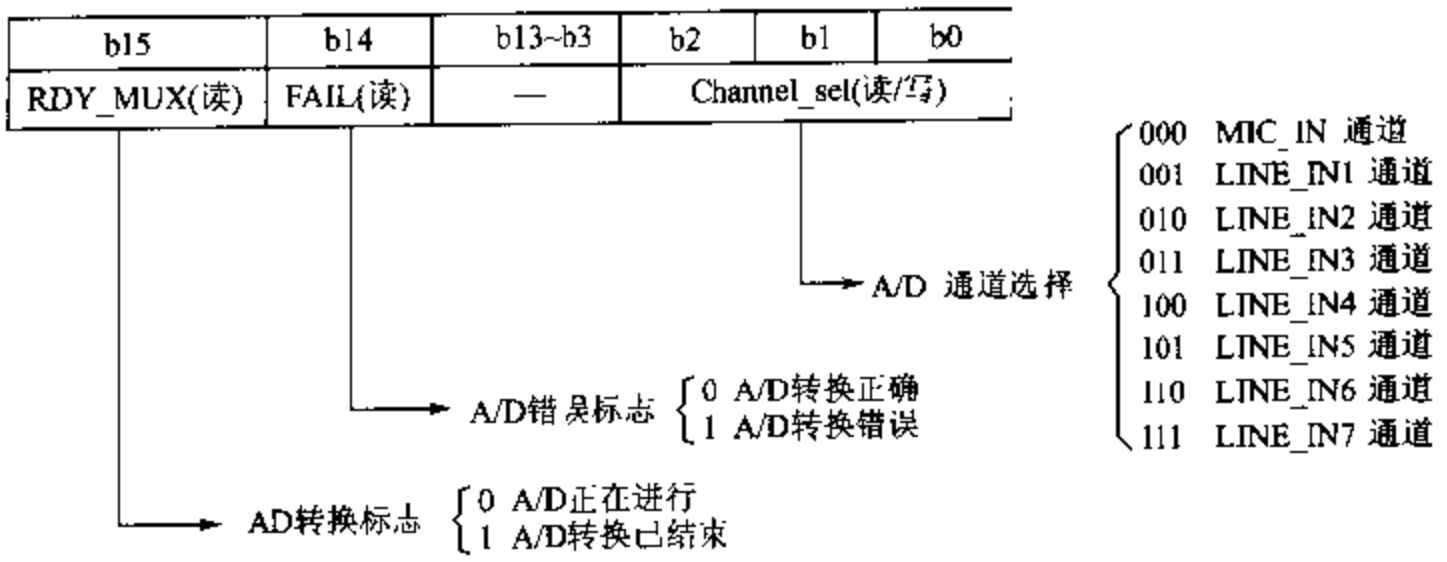


图 6-10　ADC 多通道控制/状态寄存器格式及功能

标志位 FAIL 一般情况下总为“0”。但由于 MIC_IN 的优先级高于 LINE_IN，所以在 LIN_IN 模/数转换过程中又有 MIC_IN 时，则 A/D 切换到 MIC_IN 输入，原 LINE_IN 的数据将会出现问题，此时 FAIL 被置为“1”。MIC_IN 模/数转换完成后，该位被清零。

标志位 RDY_MUX 只用于反映 LINE_IN1～LINE_IN7 的转换结束与否，MIC_IN 的转换结束与否由寄存器 P_ADC_Ctrl 的标志位 RDY 反映。

ADC 的多路 LINE_IN 输入与 IOA0～IOA6 共用，其对应关系如下：

IOA6	IOA5	IOA4	IOA3	IOA2	IOA1	IOA0
LINE_IN7	LINE_IN6	LINE_IN5	LINE_IN4	LINE_IN3	LINE_IN2	LINE_IN1

(4) ADC 多通道数据寄存器：ADC 多通道数据寄存器 P_ADC_MUX_Data ($702CH)——只能读。该寄存器储存 LINE_IN 输入的 A/D 转换的结果。CPU 对该寄存

器执行读操作就是读LINE_IN1~LINE_IN7对应通道A/D转换输出的10位数字量。读出的结果只有高10位有效，低6位无意义。

2. ADC结构及原理

图6-11给出了SPCE061A的A/D转换器结构原理框图。与图4-11的区别主要在于SPCE061A只有自动工作方式，因而没有与手动相关的控制逻辑；MIC_IN和LINE_IN输入的切换由寄存器P_ADC_MUX_Ctrl的低3位完成。

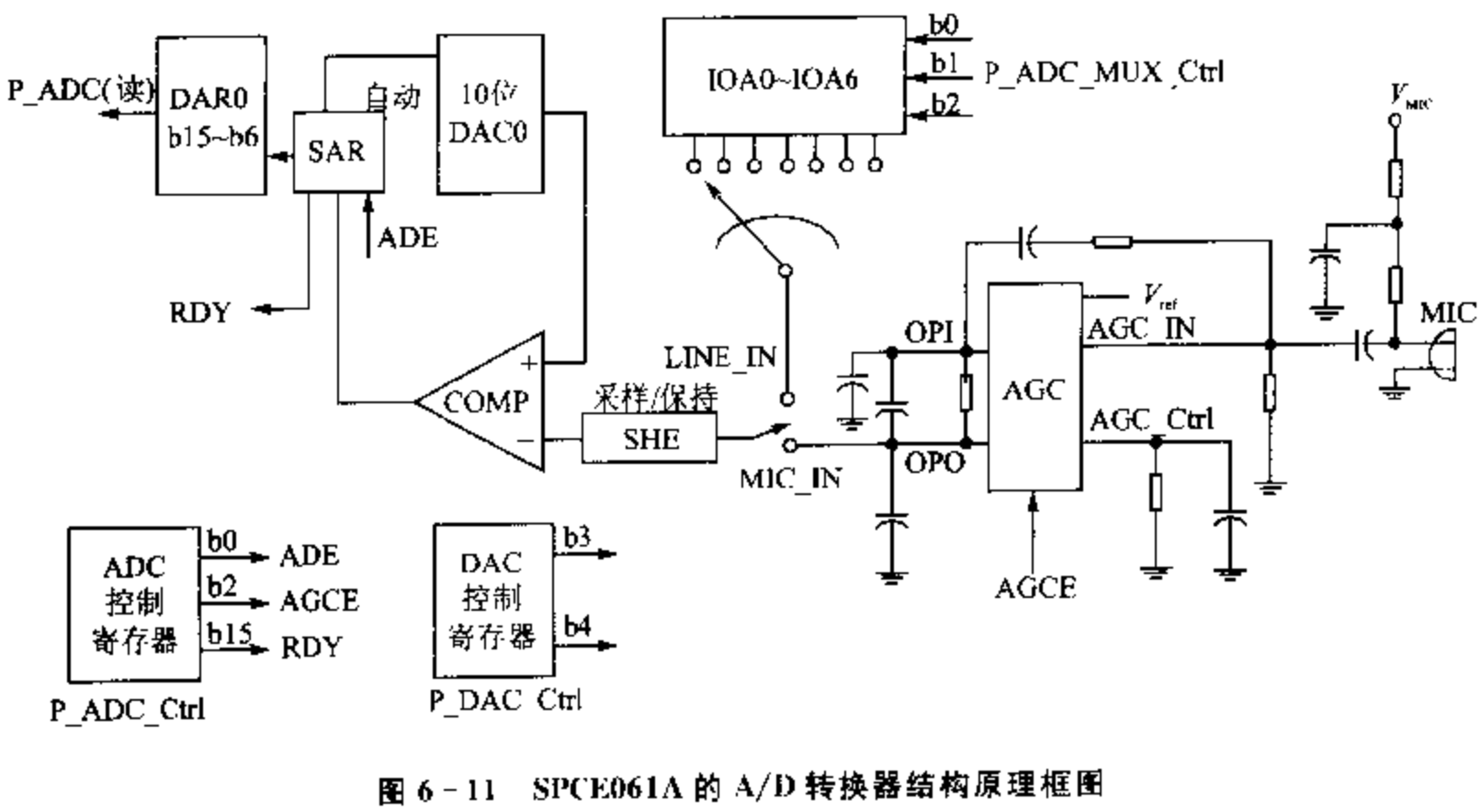


图6-11 SPCE061A的A/D转换器结构原理框图

当A/D转换开始后，控制寄存器中的A/D转换标志位RDY=0，逐次逼近寄存器SAR将根据比较结果自动跟踪给出数字量，逐位比较的A/D转换过程由硬件自动完成，A/D转换的结果保存在SAR内。当10位A/D转换完成时，RDY会被置"1"。此时，用户通过读取寄存器P_ADC或寄存器P_ADC_MUX_Data即获得10位A/D转换的结果。从这两个寄存器读取数据后，又会使RDY自动清零并重新开始进行A/D转换。若未读取P_ADC或P_ADC_MUX_Data寄存器中的数据，RDY仍保持为"1"，则不会启动下一次的A/D转换。可见，适时读取A/D转换结果，可以控制A/D转换的启动时间，亦即控制了A/D转换的速率。

外部信号由LINE_IN1~LINE_IN7即IOA0~IOA6或通道MIC_IN输入。从LINE_IN1~LINE_IN7输入的模拟信号直接送入缓冲器P_ADC_MUX_Data，而从MIC_IN输入的模拟信号则要经过缓冲器和放大器。若允许自动增益功能AGC，则由MIC_IN通道输入的模拟信号将被自动地放大并控制在一定范围内，然后经采样/保持模块送至比较器进行A/D转换，最后将结果送入P_ADC。

ADC自动转换过程有两种启动方式：通过读取数据寄存器(P_ADC或P_ADC_MUX_Data)启动和通过定时器的计数溢出启动。这两种启动方式由数/模转换器的控制寄存器P_DAC_Ctrl中的b3、b4位选择，具体设置方法见4.3节。

在使用时必须注意：ADC的转换速度有一定限制，最高转换速度不能超过$f_{osc}/32/12$，超过此极限时读出的A/D转换结果将不正确。为了方便，表6-7给出了对应不同时钟频率时的A/D转换速率极限。

表 6-7　不同时钟频率时的 A/D 转换速率极限

f_{OSC}/MHz	20.48	24.576	32.768	40.96	49.152
A/D 转换速率/Hz·s^{-1}	610	768	1 021	1 280	1 536

3. ADC 的电气特性

(1) ADC 直流电气特性：ADC 的直流电气特性如表 6-8 所列。

表 6-8　ADC 直流电气特性

名　称	符　号	最小值	典型值	最大值
ADC 分辨率/位	RESO	—	10	—
ADC 有效位数/位	ENOB	8	—	—
ADC 信噪比/dB	SNR	50	—	—
ADC 积分非线性/LSB	INL	—	—	±4
ADC 差分非线性/LSB	DNL	—	—	±2
ADC 转换率/Hz	f_{CONV}	—	—	96 000
电源电流(V_{DD}=3 V)/mA	I_{ADC}	—	3.1	—
功耗(V_{DD}=3 V)/mW	P_{ADC}	—	10.2	—

注：① LSB 表示最小有效单位，在 V_{RT}=3 V 的情况下，1LSB 为 2.93 mV。

② 表中 ADC 转换率由最大采样率求得，即 ADC 转换速率 /16=1 536 kHz/16=96 kHz。

(2) ADC 交流电气特性：ADC 的交流电气特性如表 6-9 所列。

表 6-9　ADC 交流电气特性

名　称	符　号	最小值
ADC 建立时间/s	T_{ini}	0.33
ADC 采样时间/μs	T_{sh}	2
ADC 通道选择时间/μs	T_{sel}	1
V_{RT} 开关时间/μs	T_{sw}	1
ADC 转换时间/CLK	T_{conv}	CLK

(3) ADC 的功能引脚：ADC 的功能引脚表 6-10 所列。

表 6-10　ADC 的功能引脚

ADC 引脚	I/O	功 能 描 述
V_{RT}	O	2 V 输出参考电压
MICP	I	MIC_IN 的同相输入
MICN	I	MIC_IN 的反相输入
AGC	I/O	自动增益控制端
MICOUT	O	麦克风的第一运放输出

续表 6-10

ADC引脚	I/O	功能描述
$V_{IN[7:1]}$	I	模拟信号源输入1~7即IOA0~IOA6
OPI	I	麦克风的第二运放输入
V_{CM}	O	ADC参考电压
V_{REFAD}	I	外部A/D最高参考电压
V_{MIC}	I	麦克风的电源

三、数/模转换器 DAC

SPCE061A音频输出方式为双DAC方式，与4.4节介绍的D/A转换器工作原理相同，但没有PWM单通道的音频输出方式。下面重点阐述其不同点。

1. DAC控制及数据寄存器

DAC控制寄存器P_DAC_Ctrl(\$702AH)与4.4节所述含义基本相同。b8、b7是DAC1数据锁存方式选择位；b6、b5是DAC2数据锁存方式选择位；b4、b3是ADC转换方式控制位；b2~b0在此则保留不用，而b1必须设置为"1"，用于双DAC音频输出。详细见4.4节。

DAC数据寄存器共有两个，即P_DAC2(\$7016H)和P_DAC1(\$7017H)，其作用与4.4节所述相同，但无单通道PWM驱动输出方式下的操作，这里不再重复。

2. DAC结构

SPCE061A音频输出方式与4.4节所述有所不同，在结构上也有所区别，图6-12给出了SPCE061A的D/A转换器结构原理框图。可以看出，在D/A转换时直接将数字量分别写入P_DAC1和P_DAC2中。数据锁存有4种方式：直接锁存、通过TimerA溢出锁存、通过TimerB溢出锁存、通过TimerA或TimerB溢出锁存。DAC1、DAC2转换输出的模拟量电流信号分别通过AUD1和AUD2引脚输出。

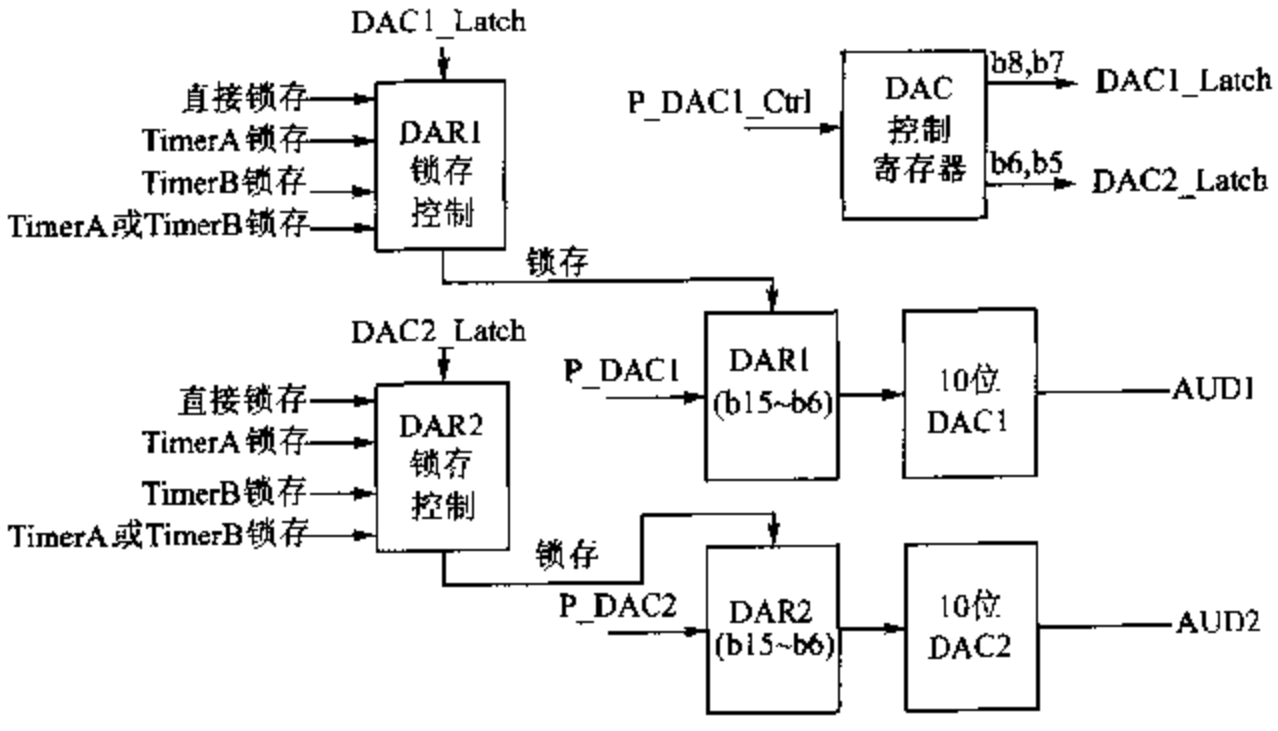


图6-12 SPCE061A的D/A转换器结构原理框图

四、通用串行通信接口 UART

SPCE061A 的通用串行通信接口 UART 与 4.5 节介绍的 UART 原理基本相同，区别在于无红外通信功能，UART 只有一种接收数据的方式：通过 UART 的 Rx 信号，它来自 UART 接收器的引脚 IOB7，在结构上也有所不同。图 6－13 给出了 SPCE061A UART 接收数据的结构框图。

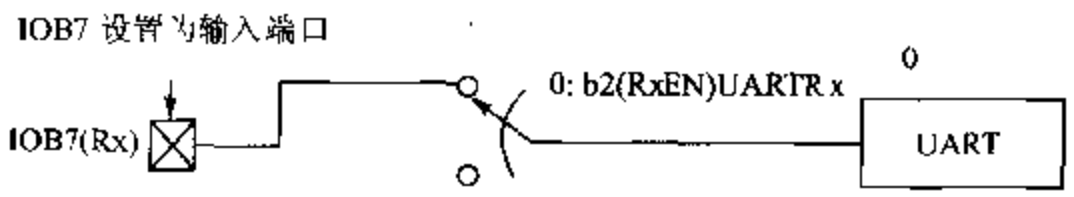


图 6－13　SPCE061A 的异步串行接收端口结构框图

在使用 UART 时须注意，尽管 SPCE061A 没有红外通信功能，但必须对红外通信控制寄存器 P_IR_Ctrl(＄7018H)进行设置，即将该寄存器的 b2 位 RxEN 设置为"0"，使 UARTRx 通过 IOB7(Rx)接收数据，这样 UART 功能才能正常使用。其它详细内容见 4.5 节介绍。

五、保密设置

SPCE061A 提供有保密功能，可以对用户开发的程序进行保密，使用户的知识产权不受侵犯。对内部闪存进行保密设置时，只需将 P_{FUSE} 接 5 V、P_{VIN} 接 GND 并维持 5 s 以上，即可将内部保险丝熔化，此后就无法再实现 download、debug 等功能。

六、端口寄存器总述

为了方便和对 SPCE061A 的硬件控制寄存器有一个全面认识，表 6－11 给出了 SPCE061A 所有寄存器的读/写属性和地址。

表 6－11　SPCE061A 寄存器

寄存器	读/写属性	地　址	功能说明
P_IOA_Data	读/写	＄7000H	写数据到数据寄存器中，读进 I/O A 口管脚上的电平状态
P_IOA_Buffer	读/写	＄7001H	写数据到数据寄存器中，读数据寄存器中的数据
P_IOA_Dir	读/写	＄7002H	I/O A 口的方向向量(输入输出选择)
P_IOA_Attrib	读/写	＄7003H	I/O A 口的属性向量(工作方式选择)
P_IOA_Latch	读	＄7004H	进入睡眠前锁存 I/O A 口的数据，为睡眠后触键引起唤醒准备
P_IOB_Data	读/写	＄7005H	写数据到数据寄存器中，读进 I/O B 口引脚上的电平状态
P_IOB_Buffer	读/写	＄7006H	写数据到数据寄存器里，读数据寄存器中的数据
P_IOB_Dir	读/写	＄7007H	I/O B 口的方向向量(输入输出选择)
P_IOB_Attrib	读/写	＄7008H	I/O B 口的属性向量(工作方式选择)

续表 6·11

寄存器	读/写属性	地 址	功能说明
P_FeedBack	写	$7009H	B11反馈寄存器(B11特殊功能控制)
P_TimerA_Data	读 写	$700AH	TimerA 数据寄存器,写计数初值,读出当前计数值
P_TimerA_Ctrl	写	$700BH	TimerA 控制寄存器
P_TimerB_Data	读/写	$700CH	TimerB 数据寄存器,写计数初值,读出当前计数值
P_TimerB_Ctrl	写	$700DH	TimerB 控制寄存器
P_Timebase_Setup	写	$700EH	时基设置寄存器
P_Timebase_Clear	写	$ 700FH	时基复位寄存器
P_INT_Ctrl	读/写	$7010H	中断控制寄存器
P_INT_Clear	写	$7011H	中断请求清除寄存器
P_SystemClock	写	$7013H	系统时钟控制寄存器(包括系统进入睡眠状态的时钟频率选择)
P_ADC	读/写	$7014H	ADC 的 MIC 通道数据寄存器
P_ADC_Ctrl	读/写	$7015H	ADC 控制/状态寄存器
P_DAC2	读/写	$7016H	DAC2 数据寄存器
P_DAC1	读/写	$7017H	DAC1 数据寄存器
P_IR_Ctrl	读/写	$7018H	红外通信控制寄存器
P_LVD_Ctrl	读/写	$7019H	低电压监测控制寄存器
P_SIO_Data	读/写	$701AH	串行设备接口 SIO 数据寄存器
P_SIO_Addr_Low	读/写	$701BH	SIO 低字节地址寄存器
P_SIO_Addr_Mid	读 写	$701CH	SIO 中字节地址寄存器
P_SIO_Addr_High	读 写	$701DH	SIO 高字节地址寄存器
P_SIO_Ctrl	读/写	$701EH	SIO 控制寄存器
P_SIO_Start	读/写	$701FH	SIO 启动寄存器
P_SIO_Stop	写	$7020H	SIO 停止寄存器
P_UART_Command1	写	$7021H	UART 控制寄存器 1(通信设置)
P_UART_Command2	读/写	$7022H	UART 控制寄存器 2(接收/发送的开通/关断以及通信状态)
P_UART_Data	读 写	$7023H	UART 数据缓冲器
P_UART_BaudScalarLow	读/写	$7024H	UART 波特率设置低字节
P_UART_BaudScalarHigh	读/写	$7025H	UART 波特率设置高字节
P_DAC_Ctrl	读/写	$702AH	音频输出方式控制寄存器
P_ADC_MUX_Ctrl	读/写	$702BH	ADC 多通道(LINE)控制/状态寄存器
P_ADC_MUX_Data	读	$702CH	ADC 多通道(LINE)数据寄存器
P_INT_Ctrl_New	读/写	$702DH	激活和屏蔽中断控制寄存器

第七章　μ'nSP™单片机程序设计

μ'nSP™单片机的汇编指令针对C语言进行了优化，在该单片机应用系统的开发中，利用汇编语言与C语言可设计出紧凑和高效的用户程序来。通常，在实时要求高的应用场合，一些使用频率较高的低层子程序往往是用汇编语言开发的，用C语言编写较高层运算和管理主程序，然而以过程或函数的形式调用由汇编语言编写的子程序。在应用汇编语言进行程序设计时，由于必须直接面对单片机的CPU、存储器及I/O接口硬件资源，因此要熟练掌握单片机的指令系统与寻址方式，有较大的难度。μ'nSP™单片机在基于Windows的集成开发环境IDE(Integrated Development Environment)支持下，为学习和掌握μ'nSP™单片机汇编语言和C语言程序设计方法提供了良好的条件。

本章在概述汇编语言一般概念的基础上，着重介绍应用μ'nSP™单片机汇编语言编写应用程序的基本方法，并通过一系列应用示例说明μ'nSP™单片机汇编语言与程序设计中数据类型、全局/局部标号、伪指令、子程序的定义与调用、以及宏汇编与条件汇编的编程方法与特点。对于C语言与汇编语言在编程中相互调用的详细方法在本章节中也将予以具体介绍。

7.1　概　述

一、汇编语言语句格式及其源程序的汇编与链接

1. 汇编语言语句格式

汇编语言是计算机采用低级语言编程时，借助于助记符的方式来开发应用程序的一种常用方法。通常，μ'nSP™单片机汇编语言语句按行以下3字段格式编排：

[标号:]　　指令　　[;或 // <注释字符>]

其中：方括号所括的内容表示是可选的。

(1) 标号以冒号结束，作为指令所在地址的符号表示，只有在必要时才需写入。标号按使用的有效范围又可分为全局标号和局部标号两种。

(2) 指令可分为硬指令(即μ'nSP™单片机指令系统中已经介绍的机器指令)和伪指令(也称软指令)两种。从μ'nSP™指令系统介绍中可以看出，硬指令是可执行语句，其语句的指令形式与高级语言(特别是C语言)很相似，通过汇编和链接后将生成相应的机器代码。伪指令是仅在汇编过程中执行的语句，汇编器将按所用的伪指令要求完成诸如存储器分配、变量赋值、符号声明等操作。

(3) 汇编语句的注释仅作为程序编写与阅读时的参考，汇编器将不对其作任何处理。注释行必须用分号(;)或双斜线(//)起始，它可与指令在同一行，亦可在指令的前一行或后一行。好的程序注释如同一个好的文档，能有效地改善程序的可读性和可维护性。

汇编语言因CPU指令系统的不同而各不相同，它必须依托称之为相应的“汇编”和“链接”的系统软件，使其转换为CPU能直接运行的可执行程序。

2. 源程序的汇编与链接

μ'nSP™单片机集成开发环境IDE为应用汇编语言开发应用程序提供了功能齐全的编辑器、汇编器及链接器。其功能主要是：

(1) 编辑器：编辑器可在IDE的Edit窗口中直接输入，也可借助于其它文本编辑程序编辑好后复制到该项目的Edit窗口中。值得指出的是，汇编语言源程序除注释之外，程序指令必须用西文方式输入所要求的字符。

(2) 汇编器：μ'nSP™汇编语言的源程序用汇编器Xasm16进行汇编。只要PC机有足够的内存空间，Xasm16便可对任意容量的汇编源文件进行汇编。当然，它是根据需要来申请和扩充内存容量的。而源代码的输入、目标代码及列表文件的输出均出入于硬盘，因而只受硬盘容量的限制。

(3) 链接器：μ'nSP™链接器Xlink16的作用是把一个个单独的汇编目标程序模块、段或文件链接成一个整体的程序。链接的过程实际上是通过有机地融入所有的外部引用符号，对段地址重新定位，并对显示程序运行地址和最终指令代码的列表文件进行修改，最后产生出编程调试所需要的一系列文件。

二、在IDE开发环境下创建汇编语言源程序

在开始讨论汇编设计之前，先通过一个简单的汇编程序设计来建立一个整体的概念。以计算从1～100的累加和为例，要进行一个简单的汇编语言源程序，打开IDE其界面如图7－1所示。

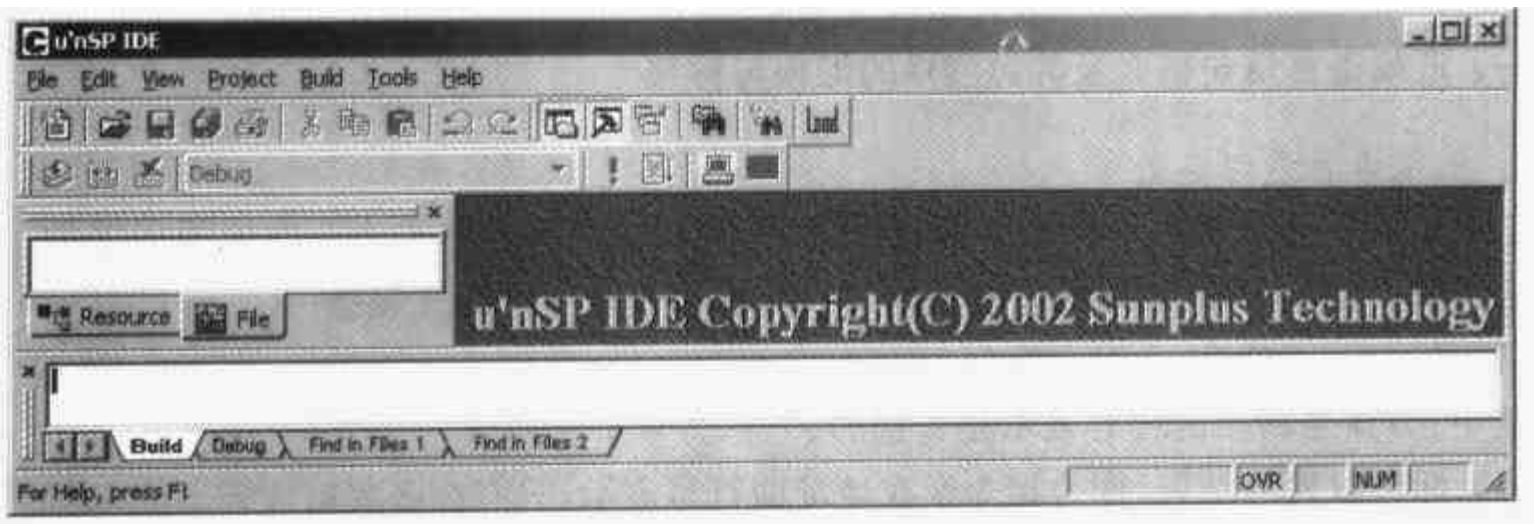


图7－1 打开IDE集成开发环境

然后单击菜单Fill>>New，会弹出如图7－2所示的新建项目的对话框。

在File栏目下输入项目名称Exam1，并在Location栏目下输入要存储的路径。比如："D:\sunplus"。在Body栏目下选择SPCE061芯片，单击"OK"按钮确认。此时，将建立新项目的界面，如图7－3所示。

在这个新建的项目中，可以看到IDE已经自动建立了一个项目的辅助文件。其中，包含了三种文件夹：源文件夹Source File、头文件夹Head File和外部描述文件夹External dependendes。

目前，可以不用理会这些由IDE自动生成的文件(将在第八章再详细给以介绍)，直接再次单击菜单Fill>>New，那么，就会弹出如图7－4所示的对话框。

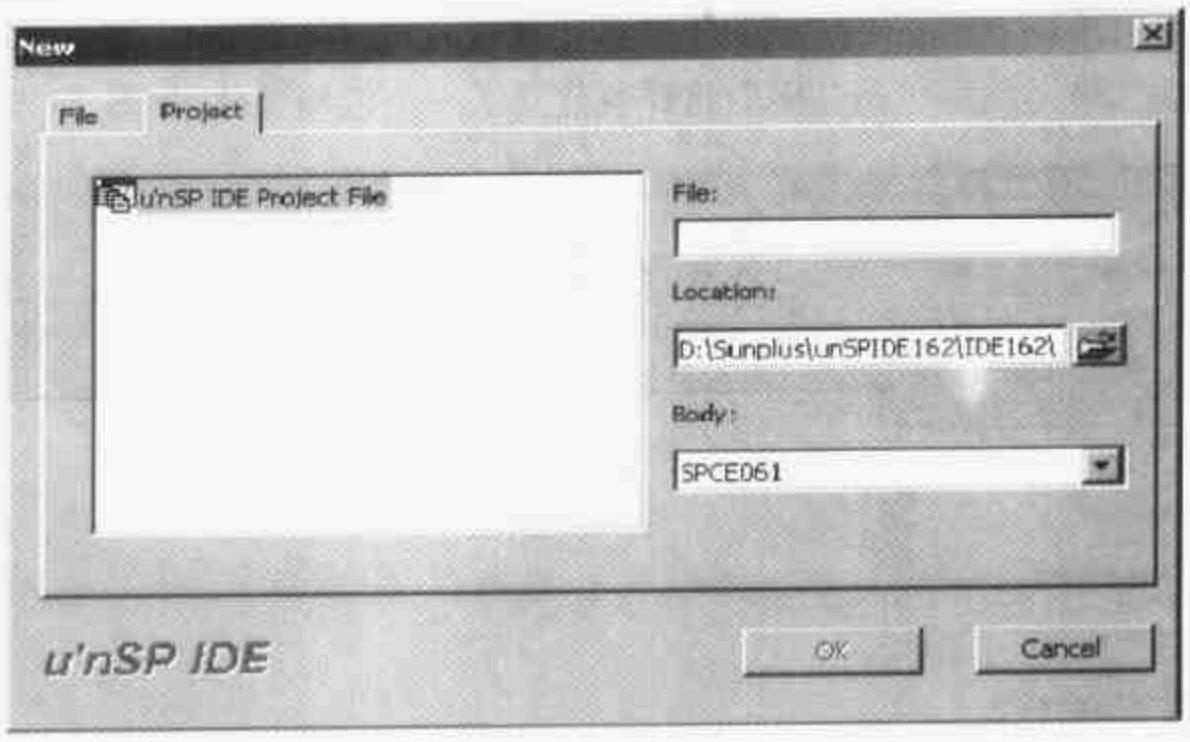


图 7-2 新建项目对话框

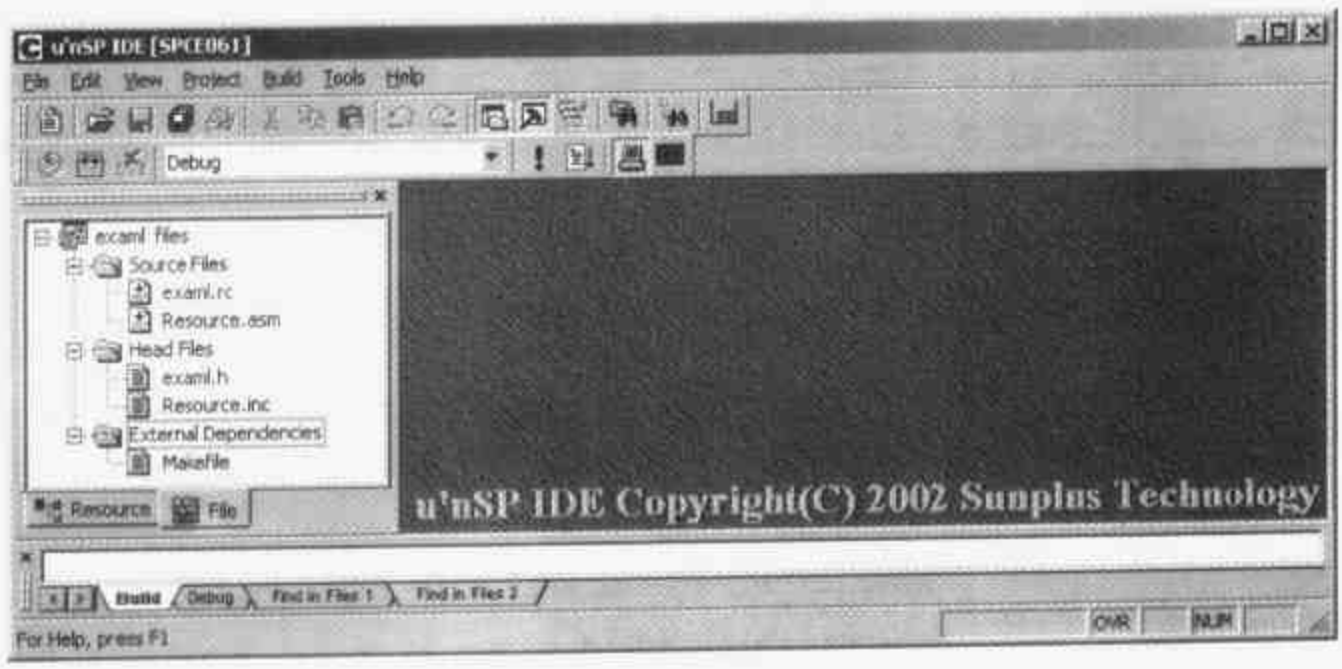


图 7-3 建立了一个新的项目

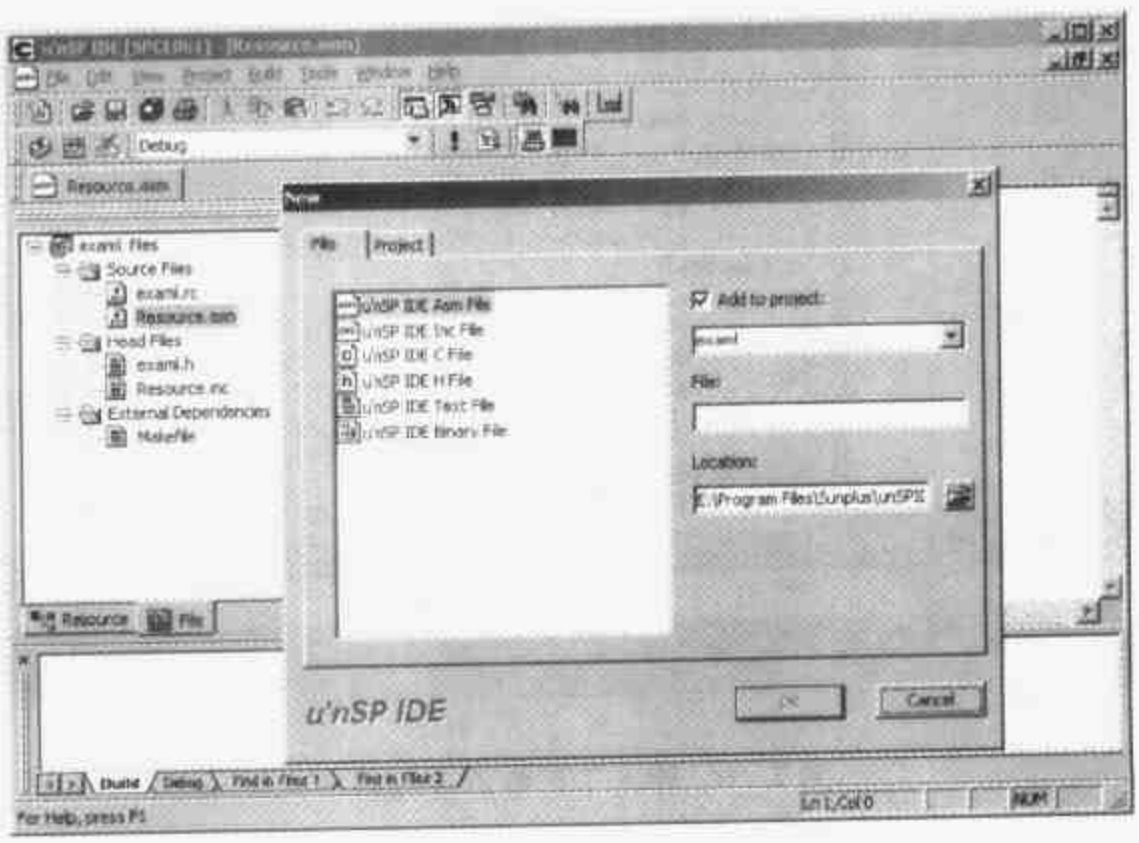


图 7-4 建立新的汇编文件的对话框

在此对话框内可以直接在File栏目下输入汇编文件的名称Examl(在左侧则选择的文件类型缺省是汇编文件),然后单击"OK",这样就建立了一个汇编文件,如图7-5所示。

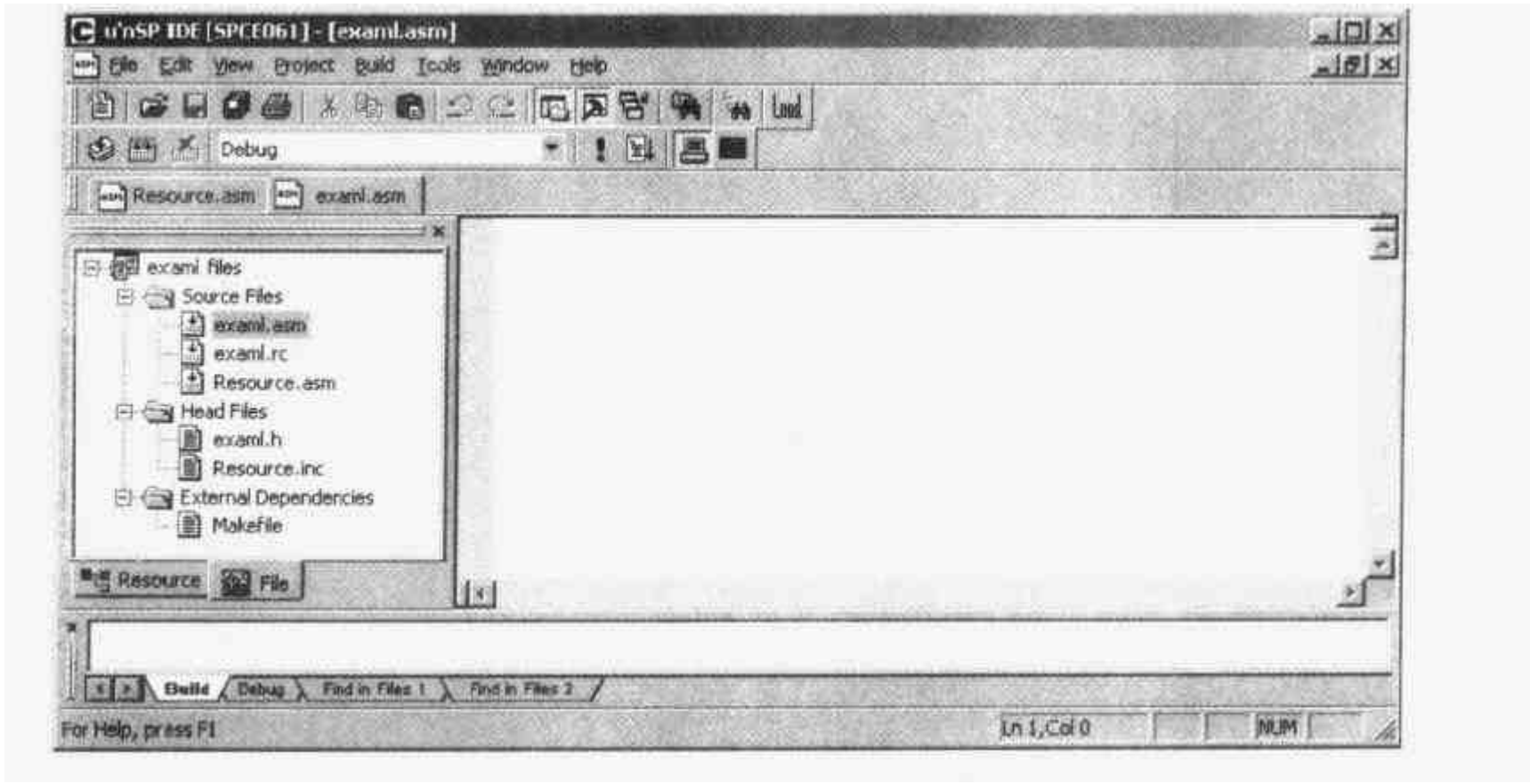


图7-5 建立一个新的汇编文件

在此汇编文件中的编辑框里可以输入汇编程序,如图7-6所示。

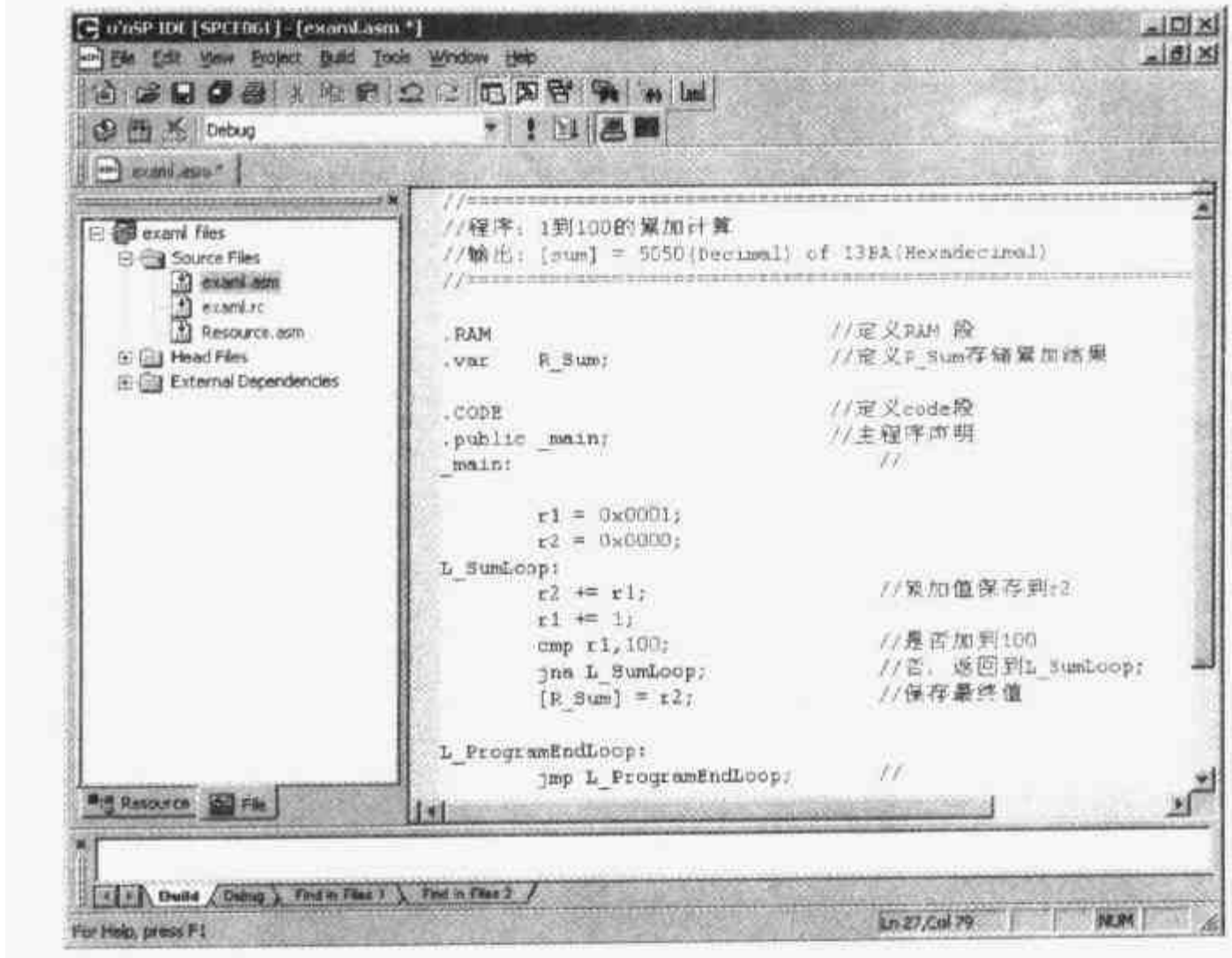


图7-6 输入了汇编语言源程序

在此例中可以看到,汇编语言源程序必须有一个主函数的标号"_main",而且必须通过伪

指令".public"声明此"_main"为全局型标号。程序代码没有定义实际的物理地址，而是用伪指令".code"声明定位在任何一个程序存储区内。另外，用伪指令".ram"数据存储区内声明了一个变量"R_Sum"。我们无需关心"R_Sum"的实际物理地址，IDE 将负责安排这些地址的管理。

语句"r1＝0x0000"实现了对寄存器 r1 赋值 0x0000 的过程。尽管看上去该语句似乎是一条 C 的语句，但它的确是一条汇编代码。

语句"r2＋＝r1"，实现了 r2＝r2＋r1 的过程。

语句"cmp r1,100"是进行了 r1 与 100 相减的运算过程，其结果只影响标志位，而不影响 r1 和其它寄存器的内容。

语句"jna L_Sumloop"是判断标志位，如果结果不大于 0，则跳转到"L_Sumloop"。

当单击菜单"Build＞＞Build"后可以对此汇编进行编辑，并对此文件进行检错生成 obj 文件。然后，单击菜单"Build＞＞Compile"，进行全局编译定位从而生成在 SPCE061 芯片上运行的代码。

最后，通过单击按钮  选择软仿真，单击按钮  进行下载到软件模拟的 SPCE061 芯片上，且通过单击按钮  打开变量观察窗口，并输入"S_Sum"来观察此变量的值。

若把光标放到语句"jmp L_ProgramEndLoop"的地方，按功能键 F9，于是就建立了一个断点。最后，单击运行按钮  ，则程序运行到此断点处将暂停，这样就可以在变量窗口来观察。程序运行的结果如图 7－7 所示。

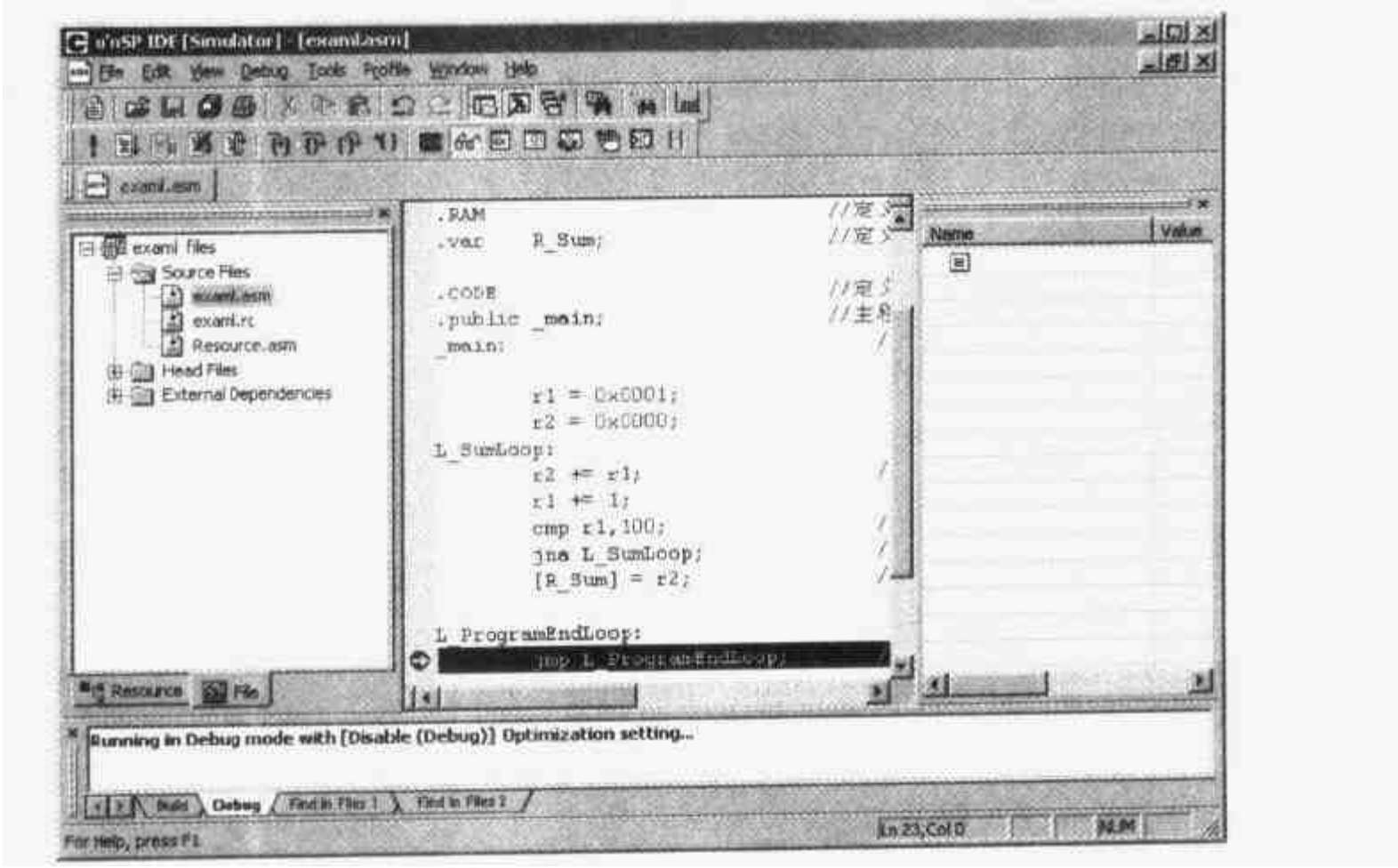


图 7－7　程序下载及运行界面

至此，就完成了一个最简单的汇编代码的开发过程。从中，可以初步体会到应用 μ'nSP 单片机的 IDE 开发环境，进行用户程序汇编编程的基本操作过程。有关 IDE 开发应用更详细

内容，请参见本书的第八章。

三、μ'nSP™汇编语言源程序的结构

μ'nSP™汇编语言源程序由一系列CPU指令和汇编器伪指令遵循一定语法规则的数据和代码段组成的。因此，其源程序最基本的部分是程序段的定义和使用。

在μ'nSP™汇编语言中，"段"其实就是汇编器Xasm16的地址标签。Xasm16把程序段分为预定义段和用户定义段两种。

1. 预定义段

在Xasm16中共有9个预定义段：

(1) 由伪指令.CODE和.TEXT设置的程序指令段。

(2) 由伪指令.DATA、.ORAM、.OSRAM、.RAM、.IRAM、.SRAM及.ISRAM规定的数据段。

这些预定义段分别规定了段内存储数据类型(指令/数据，以及数据中有/无初始值)，指明了存储介质类型(ROM/RAM)，存储范围是在零页(或零页中前64个字)或当前页/整个64页，以及数据在存储区中是同类型合并/重叠排放等。

2. 用户定义段

为了使程序在链接时具有更大的灵活性，用户可以用伪指令.SECTION来定义某些段(不超过4 096个段)。该段可嵌套使用，其属性(attribute)可以是上述9个预定义段中的任意一个。

四、数制、数据类型

(1) 数制：μ'nSP™的汇编程序(汇编器)将十进制作为默认数制。十六进制数可用符号"0x"或"$"作为前缀，或用符号"H"作为后缀。表7-1列出各种数制的后缀规定。

表7-1 μ'nSP™的数制及其后缀规定

数　制	后　缀
二进制	B
八进制	O或Q
十进制	D或不写
十六进制	H
ASCII字符串	用双引号或单引号括起"5"或'5'

(2) 数据类型：μ'nSP™汇编指令中所用的数据类型列在表7-2中。

表7-2 μ'nSP™指令中的数据类型

数据类型	字长度(位数)	无符号数值域	有符号数值域
字型(DW)	16	0～65 535	−32 768～+32 767
双字型(DD)	32	0～4 294 967 295	2 147 483 648～+2 147 483 647

续表 7－2

数据类型	字长度(位数)	无符号数值域	有符号数值域
单精度浮点型(FLOAT)X	32	无	以 IEEE 格式表示的 32 位浮点数[1]
双精度浮点型(DOUBLE)X	64	无	以 IEEE 格式表示的 64 位浮点数[2]

注：

[1] IEEE32 位浮点数，只保证 6 位有效精度。其单精度 32 位浮点数 X 可表示为：

$X = (-1)^S * 2^{e-127} * (1.f)$

32 位数据由高到低表示的含义与上述符号对应如下：

1 位　　数符 S，S=0，正数；S=1，负数；

8 位　　指数 e，取值范围为 0<e<255；

23 位　　尾数的小数部分为 f，其整数部分规格化为 1。

例如，实数　178.125　单精度浮点数　43322000H

100.0　42C80000H

125　42FA0000H

－178.125　C3322000H

[2] IEEE64 位浮点数，只保证 12 位有效精度。其双精度 64 位浮点数 X 可表示为：

$X = (-1)^S * 2^{e-1024} * (1.f)$

64 位数据由高到低的含义与上述符号对应如下：

1 位　　数符 S。S=0，正数；S=1，负数；

11 位　　指数 e，取值范围为 0<e<255；

52 位　　尾数的小数部分为 f，其整数部分规格化为 1。

例如，实数　178.125　浮点数　4066 4400 0000 0000 H

100.0　4059 0000 0000 0000 H

－178.125　C066 4400 0000 0000 H

五、表达式

在 μ'nSP™的汇编指令中采用了与 C 语言类似的运算符及其运算规则，用于连接常数或变量，或者用一些修饰符对常量数值进行修饰，以更便于程序员灵活编程及 μ'nSP™汇编器的辨认或操作。表 7－3 将这些运算符或修饰符列出，供程序员编程时选用。

表 7－3　μ'nSP™汇编指令中的连接运算符/修饰符

运算符/修饰符	操作内容描述
!，&&，|	逻辑非、逻辑与、逻辑或
～或 NOT，& 或 AND，| 或 OR	按位非、按位与、按位或
+，－	(一元操作符) 任意指定一个正、负操操作数或表达式
*，/，+，－	无符号数乘法、除法、加法、减法
++，－－	增量(+1)、减量(－1)
+=，－=，^=	复合赋值(算术加、减、逻辑异或)
>>，<<	把移位操作符前的数值或表达式向右、左移位；右移时最高位用 0 填充，左移时最低位用 0 填充
==，!=，>，<，>=，<=	等于、不等于、大于、小于、大于等于、小于等于

定义可以安全地重复使用在整个程序各处而不致产生混乱错误。

如下面的程序例子中，仅LABEL1，LABEL2和LABEL3都是全局标号，而LABEL1以下的标号?a及?b与LABEL2中的标号?a及?b，虽然标号的名称相同，但由于它们都是局部标号，将只有在本局部程序段才有效，故不会产生程序的混乱。

```
LABEL1:          或者      LABEL1:
?a:  NOP                   a?:  NOP
?b:  JMP  ?a               b?:  JMP  a?
     JMP  ?b                    JMP  b?

LABEL2:          或者      LABEL2:
?a:  NOP                   a?:  NOP
?b:  JMP  ?a               b?:  JMP  a?
     JMP  ?b                    JMP  b?

LABEL3:          或者      LABEL3:
?a:  NOP                   a?:  NOP
?b:  JMP  ?a               b?:  JMP  a?
     JMP  ?b                    JMP  b?
```

局部标号应当注意以下几点：

(1) 局部标号也像全局标号那样最多可有32个字母或数字字符，且必须以字母字符或问号(?)开头。μ'nSP™的汇编器通常规定用问号(?)作为局部标号的前缀或后缀。除此之外，局部标号最好也遵循全局标号的使用规则。

(2) 在不同的局部区域里所定义的局部标号都有不同的含义，且标号?a是不同于标号a?的。

(3) 切忌将诸如"+"这类运算符用在局部标号中。

(4) 伪指令VAR,SECTION或ENDS是不可以用在局部标号结尾处的(见下面μ'nSP™汇编器的伪指令内容)。

7.2 μ'nSP™汇编器的伪指令

伪指令用来控制μ'nSP™汇编器的操作，仅在源程序汇编过程中执行。μ'nSP™的汇编器规定伪指令不必区分字母的大小写，亦即书写伪指令时既可全用大写，也可全用小写，甚至可以大小写混用。但所有定义的标号包括宏名、结构名、结构变量名、段名及程序名则一律区分其字母的大小写。

伪指令是学习使用汇编语言程序设计的重要一环，作为高性能的16位单片机的μ'nSP™汇编器配置使用了功能强大的一系列伪指令，有利于应用程序的开发设计。当然，要熟练掌握和应用这些数量可观的伪指令，也决非易事。本节着重介绍基本和常用的部分伪指令，读者可通过上机操作并参阅《SUNPLUS μ'nSP™ PROGRAMMING GUIDE》，逐步深入学习和掌握μ'nSP™汇编器的其它伪指令。有关宏汇编与条件汇编中使用的伪指令，将集中放在下一节