S3C84H5/F84H5

8-BIT COMOS

Revision 1.00

April 2010

用户手册

SAMSUNG ELECTRONICS

SAMSUNG

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

S3C84H5/F84H5 8-BIT COMOS
用户手册, Revision 1.00

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City, Gyeonggi-Do, Korea 446-711

TEL : (82)-(31)-209-3865
FAX : (82)-(31)-209-6494

Home Page: http://www.samsungsemi.com

Printed in the Republic of Korea

Revision History

修订版本号	修订日期	修订内容	作者
1.00	2010年4月	- 第一版	HM, CHENG

Table of Contents

List of Figures

List of Tables

List of Examples

1 产品概述

1.1 S3C8-系列 MCU

三星的S3C8系列8位单片机向用户提供了高效快速的CPU，丰富的外围接口，以及各种大小的可编程ROM。

其中重要的CPU特性包括：

- 高效的寄存器结构
- 可选的CPU时钟源
- 可由中断唤醒的 IDLE 和 STOP 低功耗模式
- 内置有看门狗功能的Basic Timer

精心设计的中断结构支持8个中断优先级，每个中断优先级支持一个或多个中断源及中断向量。任意指定的中断优先级都可以实现快速中断（最少只需4个CPU时钟）。

1.2 S3C84H5/F84H5 MCU

S3C84H5/F84H5 单片 CMOS MCU 由先进的 CMOS 工艺制造，并基于三星最新的 CPU 架构。

S3C84H5是一款嵌入16-K字节mask ROM的微控制器。

S3F84H5是一款嵌入16-K字节可多次编程Full Flash ROM的微控制器。

通过成熟的模块化设计方法，和基于强大的SAM8 RC 内核，在 S3C84H5/F84H5 中成功地集成了以下外围模块:

- 4个可编程 I/O 口(32 SOP/SDIP: 22 个管脚, 30 SDIP: 20 个管脚, 28 SOP: 18 个管脚)
- 4个提供外部中断的管脚。
- 1个8位Basic Timer用来保证上电后系统时钟的稳定和看门狗功能(系统复位)
- 2个8位Timer/Counter和2个16位Timer/Counter，可选择工作模式
- 1个UART和1个SIO接口
- 1个10位PWM输出模块
- 9路模数转换通道，10位转换结果
- 可实现实时时钟的Watch Timer

S3C84H5/F84H5 是一款多功能的适用于家电和ADC等应用的MCU。目前可提供的封装类型有 32 SOP/SDIP, 30 SDIP, 28 SOP.

1.3 特性

1.3.1 CPU

- SAM8RC CPU内核

1.3.2 存储器

- 272 字节的内部通用寄存器
- 16K 字节可多次编程内部程序存储器

1.3.3 时钟源

- 主时钟振荡器(晶体，陶瓷)
- CPU时钟分频器 (1/1, 1/2, 1/8, 1/16)

1.3.4 指令集

- 78条指令
- 用于低功耗模式的 IDLE 和 STOP 指令

1.3.5 指令执行时间

- 最小400 ns （10 MHz晶振时）

1.3.6 中断

- 16个中断源，16个中断向量
- 8个中断优先级，16个中断向量

1.3.7 I/O PORTS

- 总共22个可位编程的管脚(32 SOP/SDIP)
  总共20个可位编程的管脚(30 SDIP)
  总共18个可位编程的管脚(28SOP)

1.3.8 TIMER/COUNTERS

- 一个用作时钟源稳定控制和看门狗的可编程 8 位 Basic Timer (BT)
- 一个具有 Interval（定时）、Capture（捕获）和 PWM 三种工作模式的 8 位 Timer/Counter
- 一个可作为载波频率发生器（或PWM）的8位Timer/Counter (Timer B)
- 两个具有 Interval（定时）、Capture（捕获）和 PWM 三种工作模式的16位Timer/Counter (Timer 10, 11)

1.3.9 WATCH TIMER

- 实时和定时检测
- BUZ四种频率输出

1.3.10 A/D转换器

- 10 位转换精度
- 八个模拟输入通道
- f_{ADC} 频率10MHz 时，20us转换速度

1.3.11 异步 UART

- 1路异步UART
- 波特率可编程
- 支持8位和9位UART串行数据发送/接收

1.3.12 PWM 模块

- 1路10位可编程的 PWM 输出

1.3.13 SIO

- 一个同步SIO模块
- 发送接收速率可调

1.3.14 内置RESET电路(LVR)

- 低电压检测系统复位
- V_{LVR} = 2.8V (通过 smart option选择)

1.3.15 震荡频率

- 支持1MHz到 10MHz 外部晶振

1.3.16 工作温度范围

- -25°C ~ + 85°C

1.3.17 工作电压范围

- LVR 使能 : LVR ~ 5.5 V (8MHz)
- LVR 禁止 : 2.5 V ~ 5.5 V(8MHz)
- LVR禁止/使能 : 4.5 V ~ 5.5 V(10MHz)

1.3.18 现有封装形式

- 32-管脚 SOP/SDIP, 30-管脚 SDIP
- 28-管脚 SOP

1.4 内部模块框图

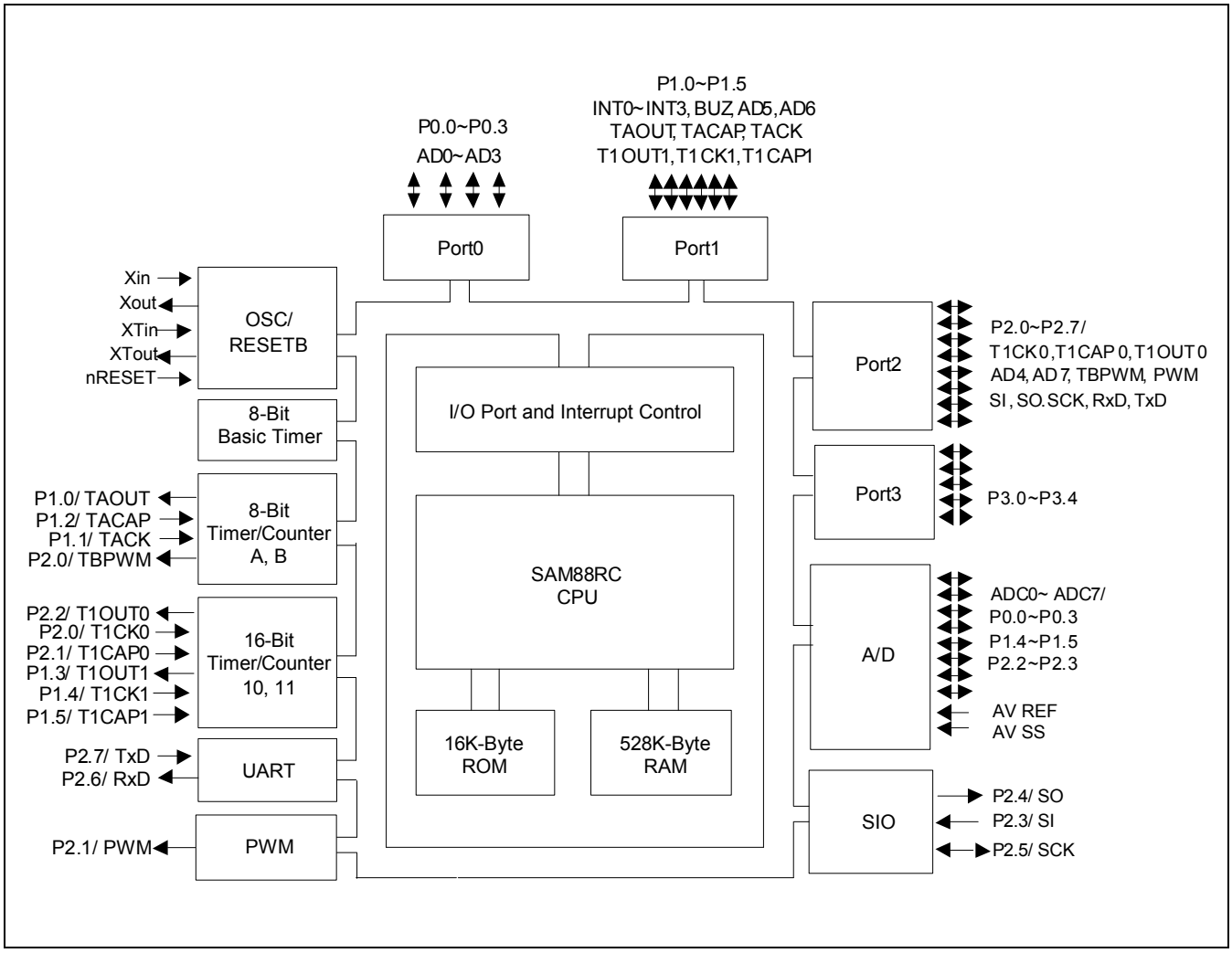


图 1-1 S3C84H5/F84H5 系统框图

1.5 管脚分布图

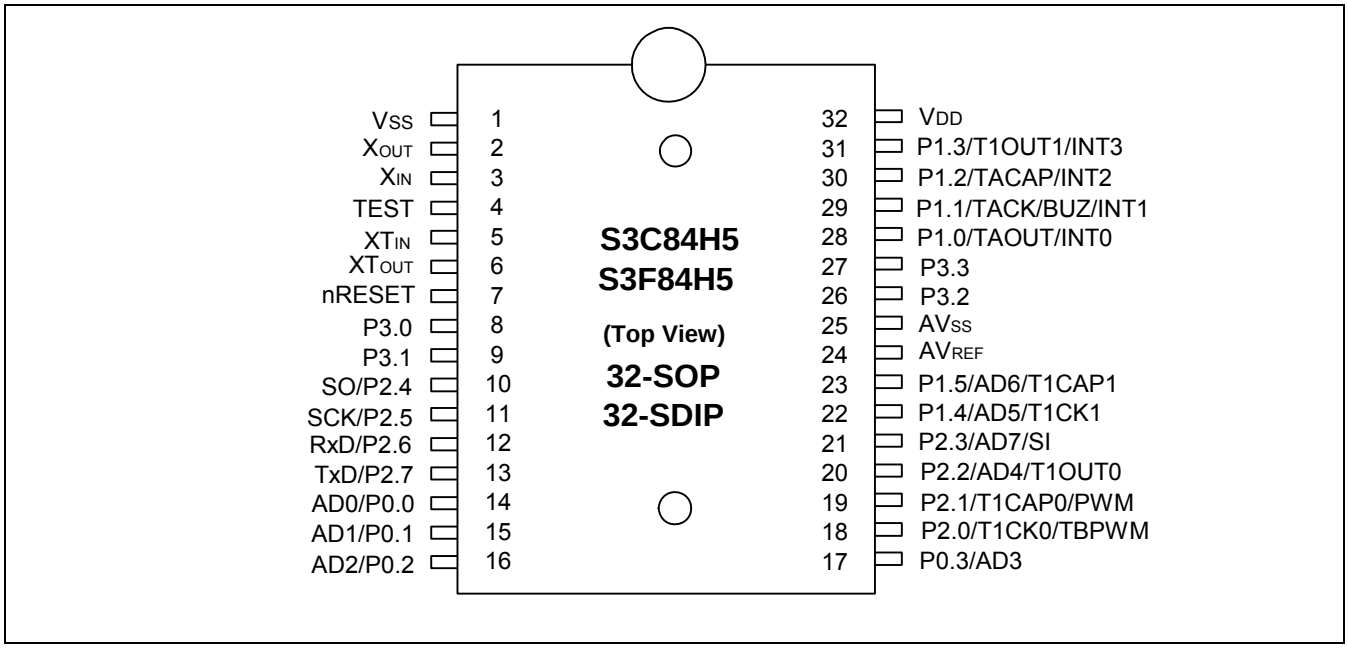


图 1-2 S3C84H5/F84H5 管脚分布图 (32-pin SOP/SDIP)

1.6 管脚分布图

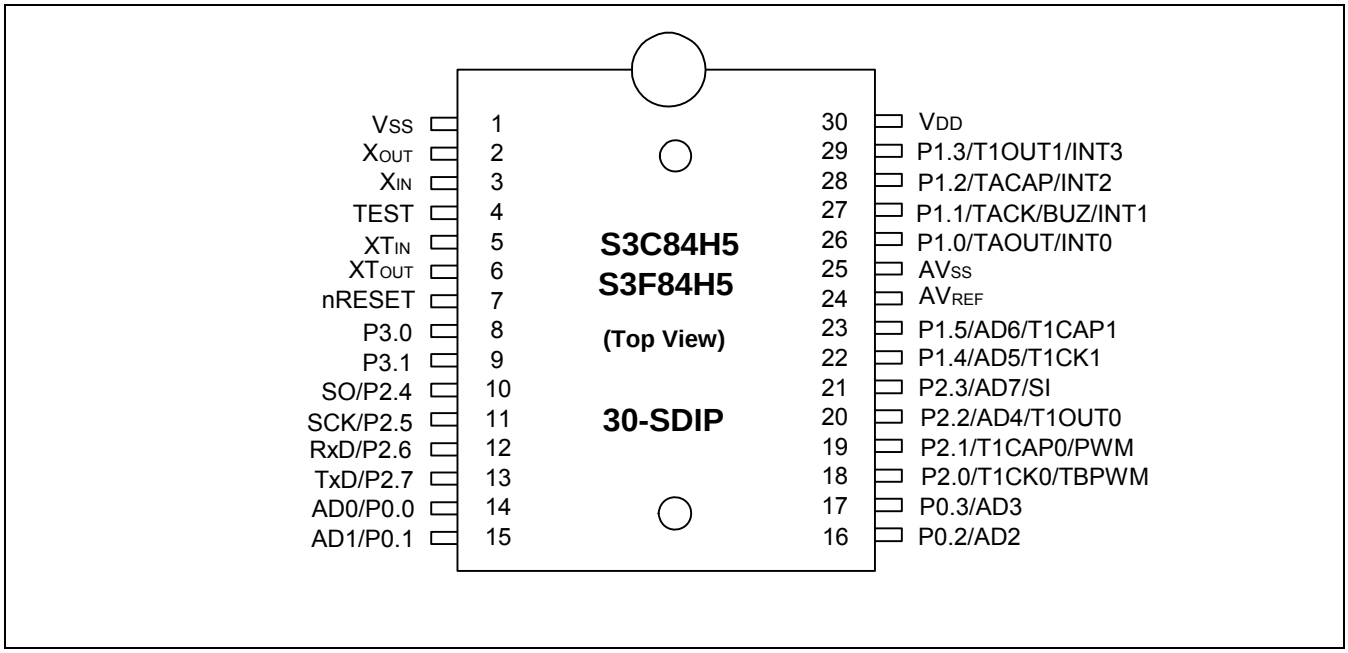


图 1-3 S3C84H5/F84H5 管脚分布图 (30-pin SDIP)

1.7 管脚分布图

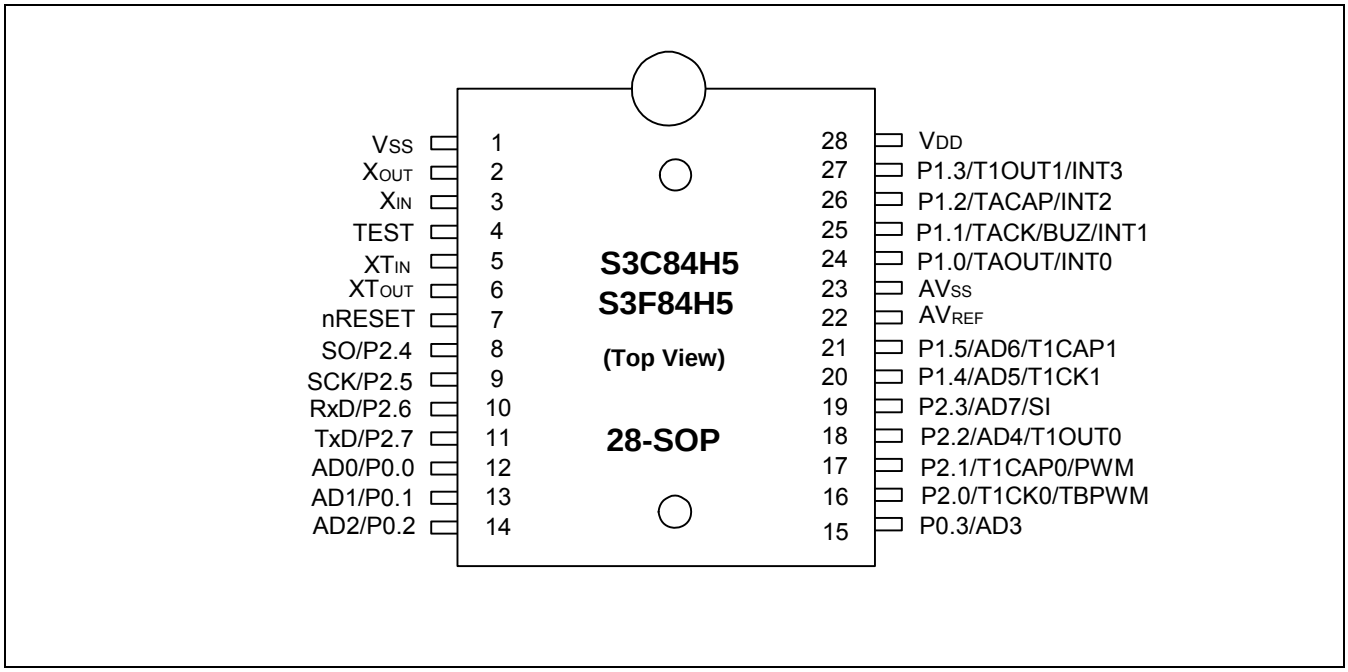


图 1-4 S3C84H5/F84H5 管脚分布图 (28-pin SOP)

1.8 管脚特性描述

表 1-1 S3C84H5/F84H5 管脚特性 (32SOP/32SDIP/28SOP)

管脚名称	输入/输出	管脚特性描述	管脚类型	管脚号	共用管脚
P0.0-P0.3	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作AD0~AD3	E	14-17 (12-15)	ADC0, ADC1, ADC2, ADC3
P1.0-P1.5	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作 INT0~INT3, TAOUT, TACK, TACAP, T1CAP1, T1CK1, T1OUT1, AD5, AD6.	D-5 E	22-23 28-31 (20-21 24-27)	INT0~INT3 TAOUT, TACK TACAP, T1CK1 T1CAP1, AD5 T1OUT1, AD6, BUZ
P2.0-P2.7	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作 ADC4, ADC7, SI, TBPWM, PWM, T1CAP0, T1CK0, SO, SCK, RxD, TxD	E D-5	10-13, 18-21 (16-19 8-11)	AD7 SO, SCK RxD, TxD, T1CAP0, T1CAP1, T1CK0, PWM, TBPWM
P3.0-P3.3	I/O	可位编程，该I/O口为输入或推挽式输出或者开漏输出. 上拉电阻可用软件设定.	G	8-9, 26-27	
INT0-INT3	I	外部中断输入管脚 也可用作普通I/O P1	D-5	28-31 (24-27)	P1.0-P1.3
ADC0-ADC7	I	A/D 转换输入口 也可用作普通I/O P 0, P1 和 P 2.	E	14-17, 20-23, (12-5, 18-21)	P0.0–P0.3 P2.2–P2.3 P1.4-P1.5
AVREF, AVSS	-	A/D 转化器参考电压和地	-	24,25 (22,23)	-
RxD	I	串行数据接收输入，发送输出(模式 0)	E	12(10)	P2.6
TxD	O	串行数据发送输出和时钟输出(模式 0)	E	13(11)	P2.7
TACK	I	Timer A外部时钟输入管脚	D-5	29(25)	P1.1
TACAP	I	Timer A捕获输入管脚	D-5	30(26)	P1.2
TAOUT	O	Timer A PWM 输出管脚	D-5	28(24)	P1.0
TBOUT	O	Timer B 载波频率输出管脚	D-5	18(16)	P2.0
T1CK0	I	Timer 1(0) 外部时钟输入管脚	D-5	18(16)	P2.0
T1CAP0	I	Timer 1(0) 捕获输入管脚	D-5	19(17)	P2.1
T1OUT0	O	Timer 1(0) 16位 PWM 输出或定时输出	D-5	20(18)	P2.2
T1CK1	I	Timer 1(1) 外部时钟输入管脚	E	22(20)	P1.4
T1CAP1	I	Timer 1(1) 捕获输入管脚	E	23(21)	P1.5

管脚名称	输入/输出	管脚特性描述	管脚类型	管脚号	共用管脚
T1OUT1	O	Timer 1(1) 16位 PWM 输出或定时输出	E	31(27)	P1.3
nRESET	I	系统复位管脚	B	7	-
TEST	I	内部连接下拉电阻	-	4	-
V_{DD}, V_{SS}	-	电源输入管脚	-	1,32	-
Xin, Xout	I,O	主时钟振荡器连接管脚	-	2,3	-

注释: 括号"( )" 里为 28-管脚 SOP 封装的管脚号.

表 1-2 S3C84H5/F84H5 管脚特性 (30 SDIP)

管脚名称	输入/输出	管脚特性描述	管脚类型	管脚号	共用管脚
P0.0-P0.3	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作AD0~AD3	E	14-17	ADC0, ADC1, ADC2, ADC3
P1.0-P1.5	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作 INT0~INT3, TAOUT, TACK, TACAP, T1CAP1, T1CK1, T1OUT1, AD5, AD6.	D-5 E	22-23 26-29	INT0~INT3 TAOUT, TACK TACAP,T1CK1, T1CAP1, AD5 T1OUT1, AD6, BUZ
P2.0-P2.7	I/O	可位编程，该I/O口为输入或推挽式输出. 上拉电阻可用软件设定. 该口也可用作 ADC4, ADC7, SI, TBPWM, PWM, T1CAP0, T1CK0, SO, SCK, RxD, TxD	E D-5	10-13, 18-21	ADC6~ADC7 SO, SCK RxD, TxD, T1CAP0, T1CAP1, T1CK0, PWM, TBPWM
P3.0-P3.3	I/O	可位编程，该I/O口为输入或推挽式输出或者开漏输出. 上拉电阻可用软件设定.	G	8-9	
INT0-INT3	I	外部中断输入管脚 也可用作普通I/O P1	D-5	26-29	P1.0–P1.3
ADC0-ADC7	I	A/D 转换输入口 也可用作普通I/O P 0, P1 和 P 2.	E	14-17, 20-23	P0.0–P0.3 P2.2–P2.3 P1.4–P1.5
AV_{REF}, AV_{SS}	-	A/D 转化器参考电压和地	-	24, 25	-
RxD	I	串行数据接收输入，发送输出(模式 0)	E	12	P2.6
TxD	O	串行数据发送输出和时钟输出(模式 0)	E	13	P2.7
TACK	I	Timer A外部时钟输入管脚	D-5	27	P1.1
TACAP	I	Timer A捕获输入管脚	D-5	28	P1.2
TAOUT	O	Timer A PWM 输出管脚	D-5	26	P1.0
TBOUT	O	Timer B 载波频率输出管脚	D-5	18	P2.0
T1CK0	I	Timer 1(0) 外部时钟输入管脚	D-5	18	P2.0
T1CAP0	I	Timer 1(0) 捕获输入管脚	D-5	19	P2.1
T1OUT0	O	Timer 1(0) 16位 PWM 输出或定时输出	D-5	20	P2.2
T1CK1	I	Timer 1(1) 外部时钟输入管脚	E	22	P1.4
T1CAP1	I	Timer 1(1) 捕获输入管脚	E	23	P1.5
T1OUT1	O	Timer 1(1) 16位 PWM 输出或定时输出	E	29	P1.3
nRESET	I	系统复位管脚	B	7	-
TEST	I	内部连接下拉电阻	-	4	-
V_{DD}, V_{SS}	-	电源输入管脚	-	1, 32	-
Xin, Xout	I,O	主时钟振荡器连接管脚	-	2, 3	-

1.9 管脚电路

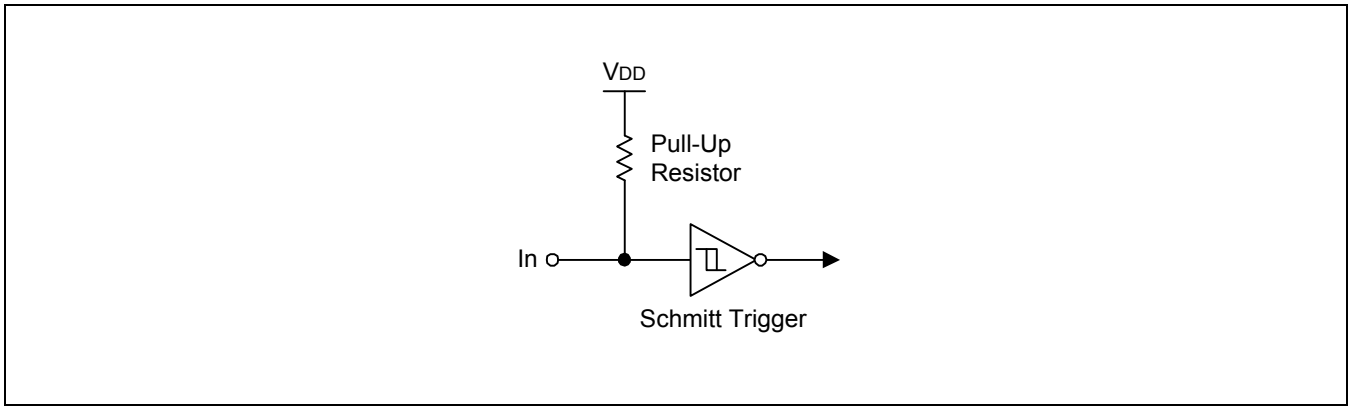


图 1-5 管脚电路类型 B (nRESET)

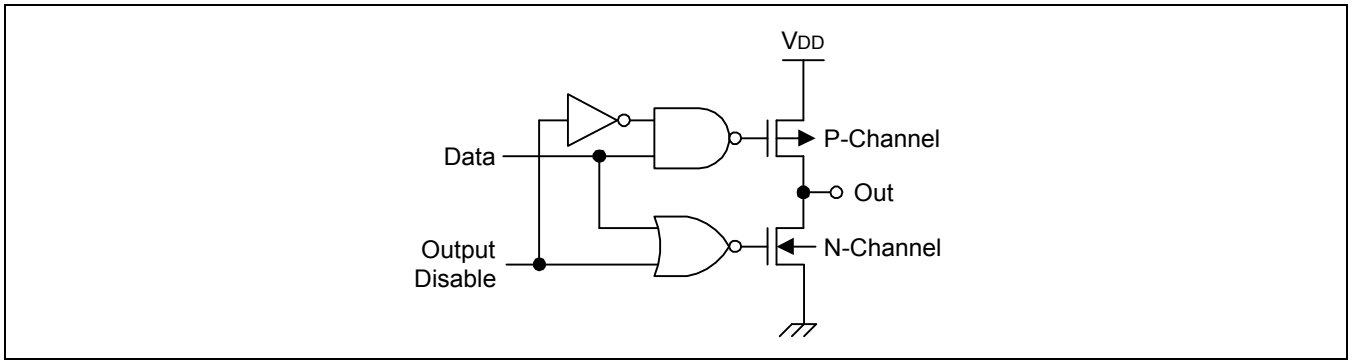


图 1-6 管脚电路类型 C

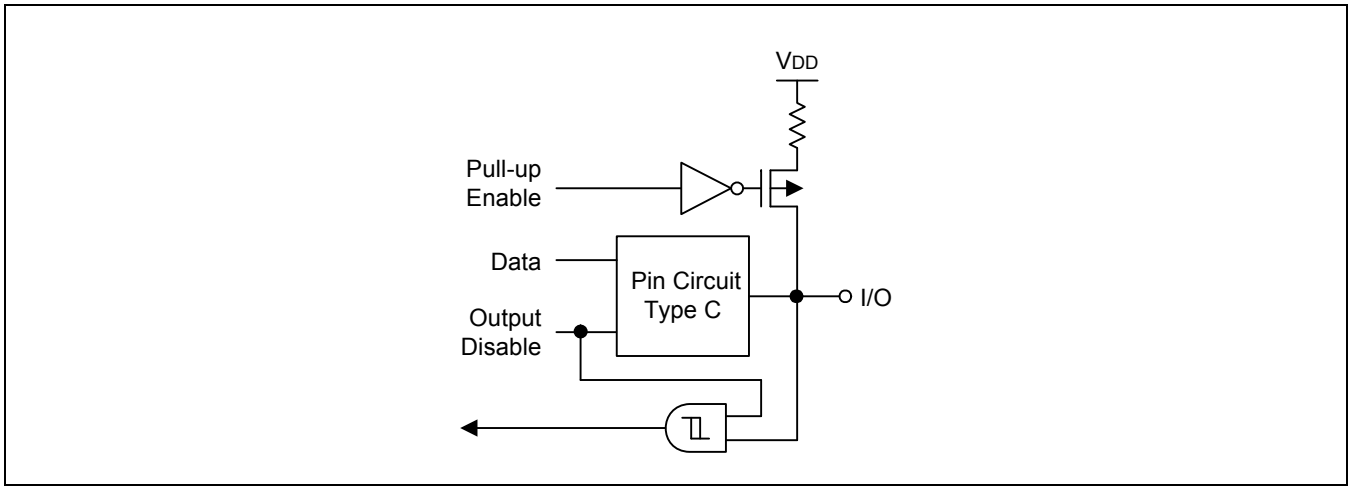


图 1-7 管脚电路类型 D

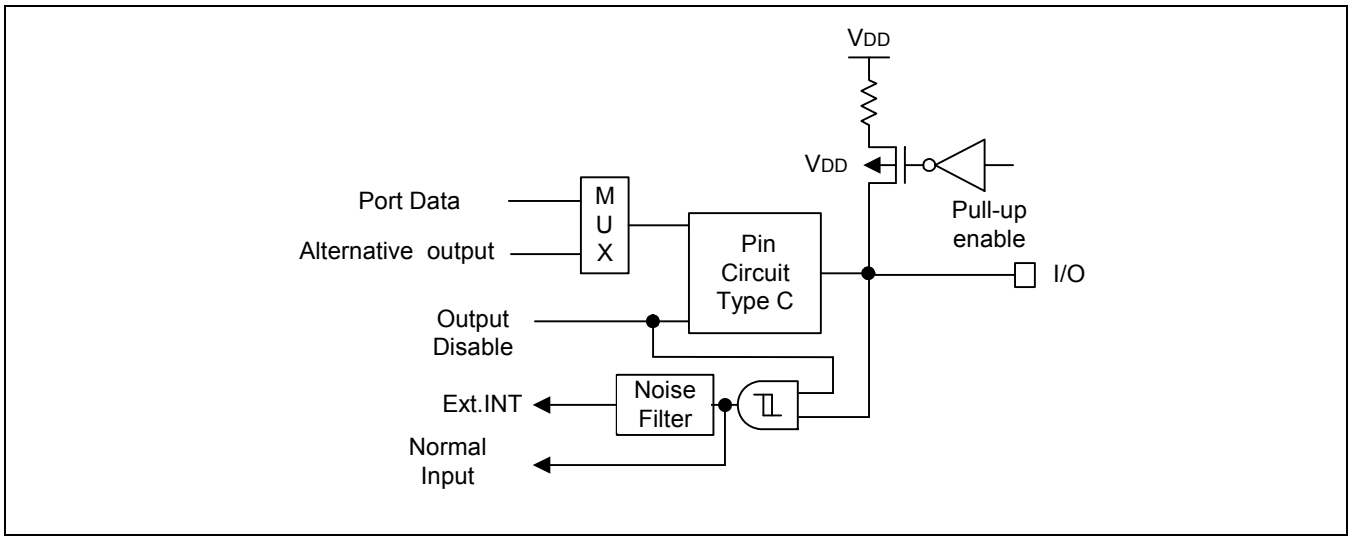


图 1-8 管脚电路类型 D-5 (P1.0~P1.3)

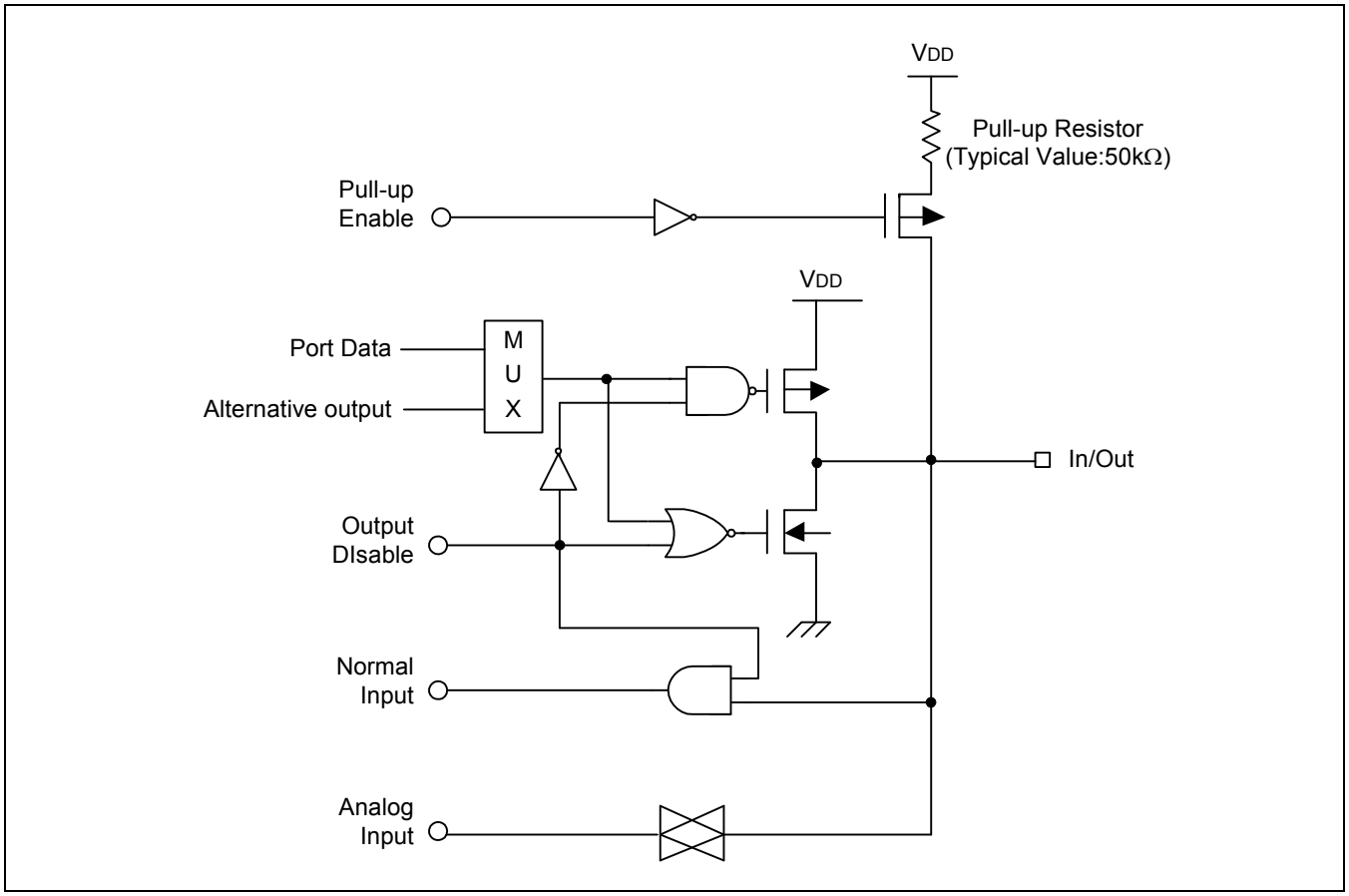


图 1-9 管脚电路类型 E (P2.2~P2.3,P1.4~1.5)

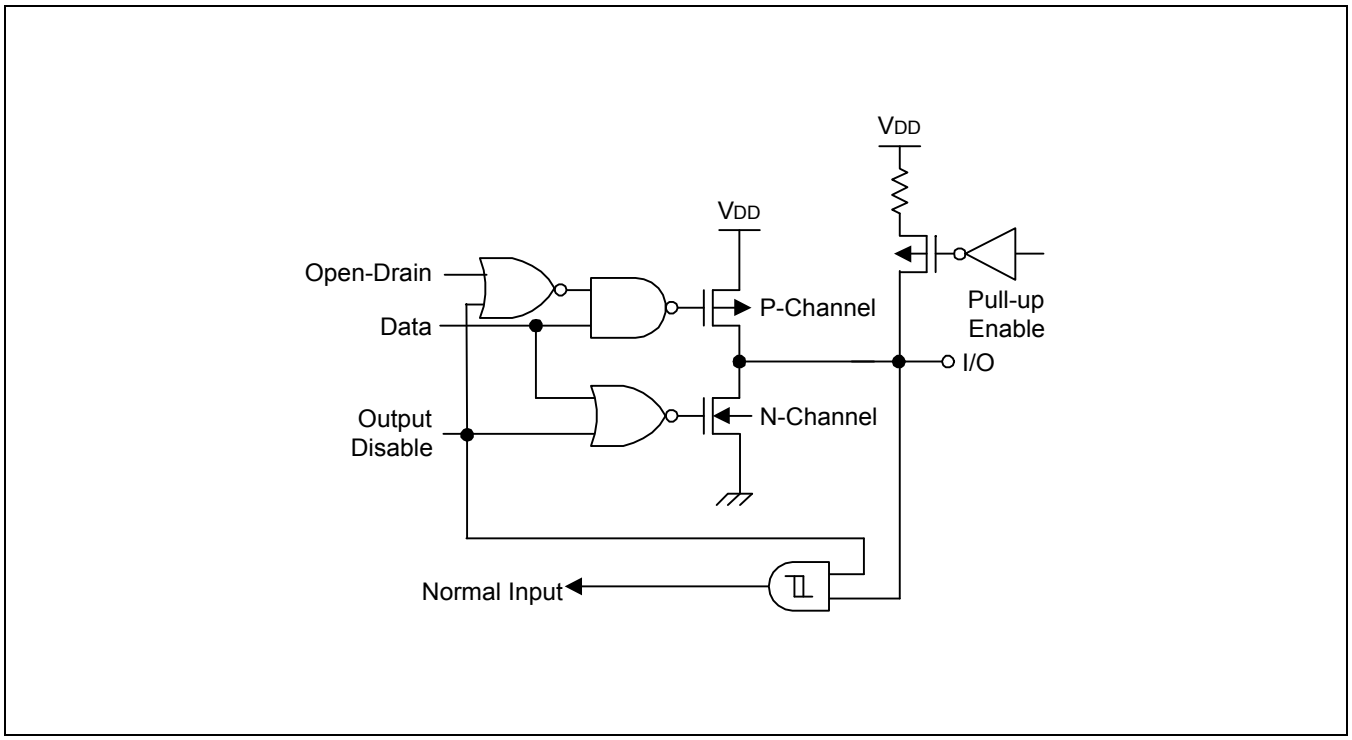


图 1-10 管脚电路类型 G (P3.0-P3.4)

2 地址空间

2.1 概述

S3C84H5/F84H5有两类地址空间:

- 内部程序存储空间(ROM)
- 内部寄存器卷(RAM)

MCU 通过16位的地址总线访问程序存储空间，通过独立的8位地址线和数据线访问内部寄存器卷。

S3C84H5/F84H5 内部集成了的16K 字节 mask ROM / 16K 字节Full Flash ROM 和272字节的 RAM。

2.2 程序存储空间(ROM)

程序存储空间 (ROM) 保存程序代码或表格数据。S3C84H5/F84H5 有16K 字节的内部 mask 可编程 Flash 程序存储空间。程序存储空间为0H-3FFFH (*图 2-1*).

ROM (0H–0FFH)中初始 256 字节预留作中断入口地址。 未使用的地址都可用作普通的程序存储空间。 在使用中断地址区域作为程序代码空间时，注意不可覆盖中断入口地址。

在 ROM 中程序的复位地址是 0100H。

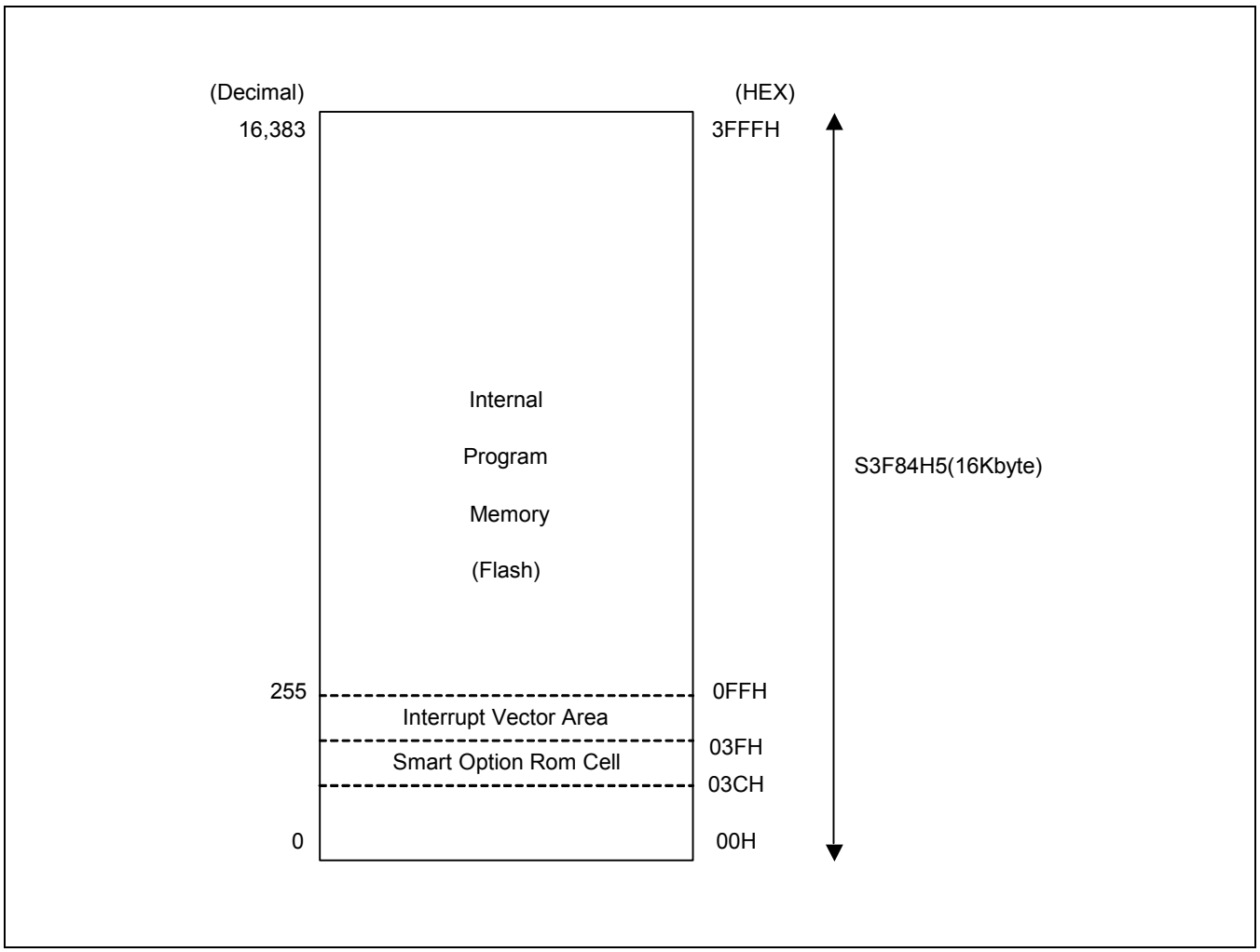
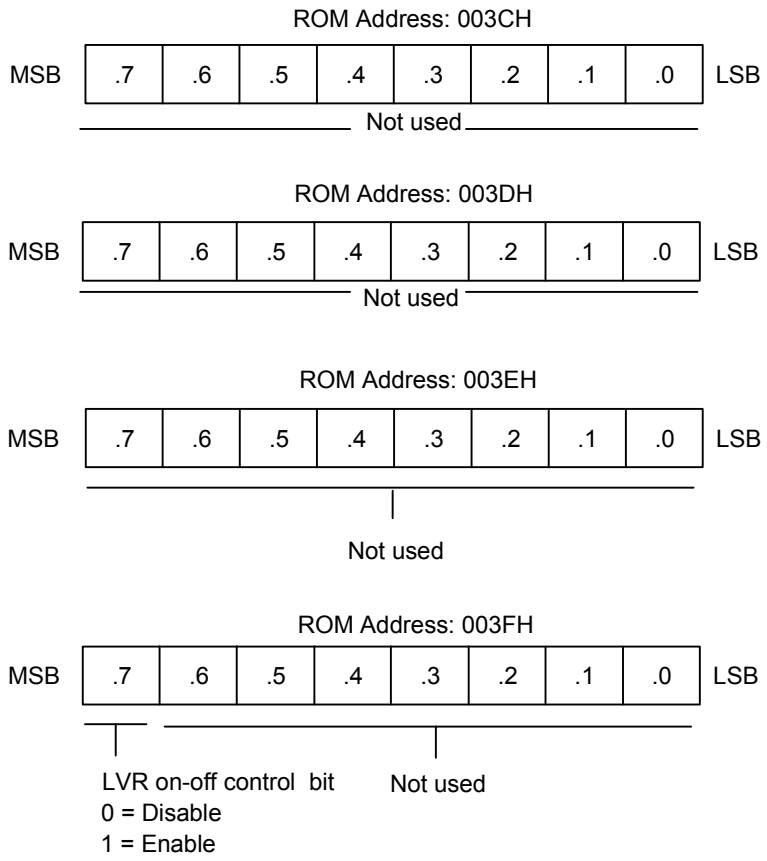


图 2-1 程序存储地址空间

2.2.1 SMART OPTION

Smart Option 是 ROM 中用于初始化芯片的智能选项，决定芯片的启动状态。 用于 Smart Option 的地址是 003CH 到003FH。 默认值为“FFH”。



注释： 03CH,03DH,03EH和 03FH 不使用的位必须写"1"

图 2-2 Smart Option

2.3 寄存器结构

在 S3C84H5/F84H5，寄存器卷的高 64 字节被扩展为两个 64 字节区域，叫做 set 1 和 set 2。Set 1 的高 32 字节区域又进一步扩展为两个 32 字节寄存器块(bank 0 和 bank 1)，低 32 字节是独立的公共存储区域。

S3C84H5/F84H5 共有 334 个可寻址的 8 位寄存器。其中 13字节用于 CPU 和系统控制寄存器，49 字节用于外设控制和数据寄存器，16 字节用于工作寄存器，还有 256个通用寄存器供用户使用。

不管当前选的是哪个寄存器页，都可以对 set 1 的寄存器进行寻址。但只能通过寄存器寻址模式进行。

通过不同寻址模式的限制，bank 选择指令(SB0,SB1)，以及寄存器页面指针(PP)， 得以将寄存器空间扩展为各个独立的可寻址区域(set, bank 和 page)。

表 2-1 总结了内部寄存器卷中特殊功能寄存器类型和所占字节数。

表 2-1 S3C84H5/F84H5 寄存器类型总结

寄存器类型	所占字节数
通用寄存器(包括 16 个工作寄存器，2个172字节的主寄存器和2个64字节set 2区域) CPU 和系统控制寄存器 外设控制和数据寄存器	272 13 49
所有可寻址的字节数	334

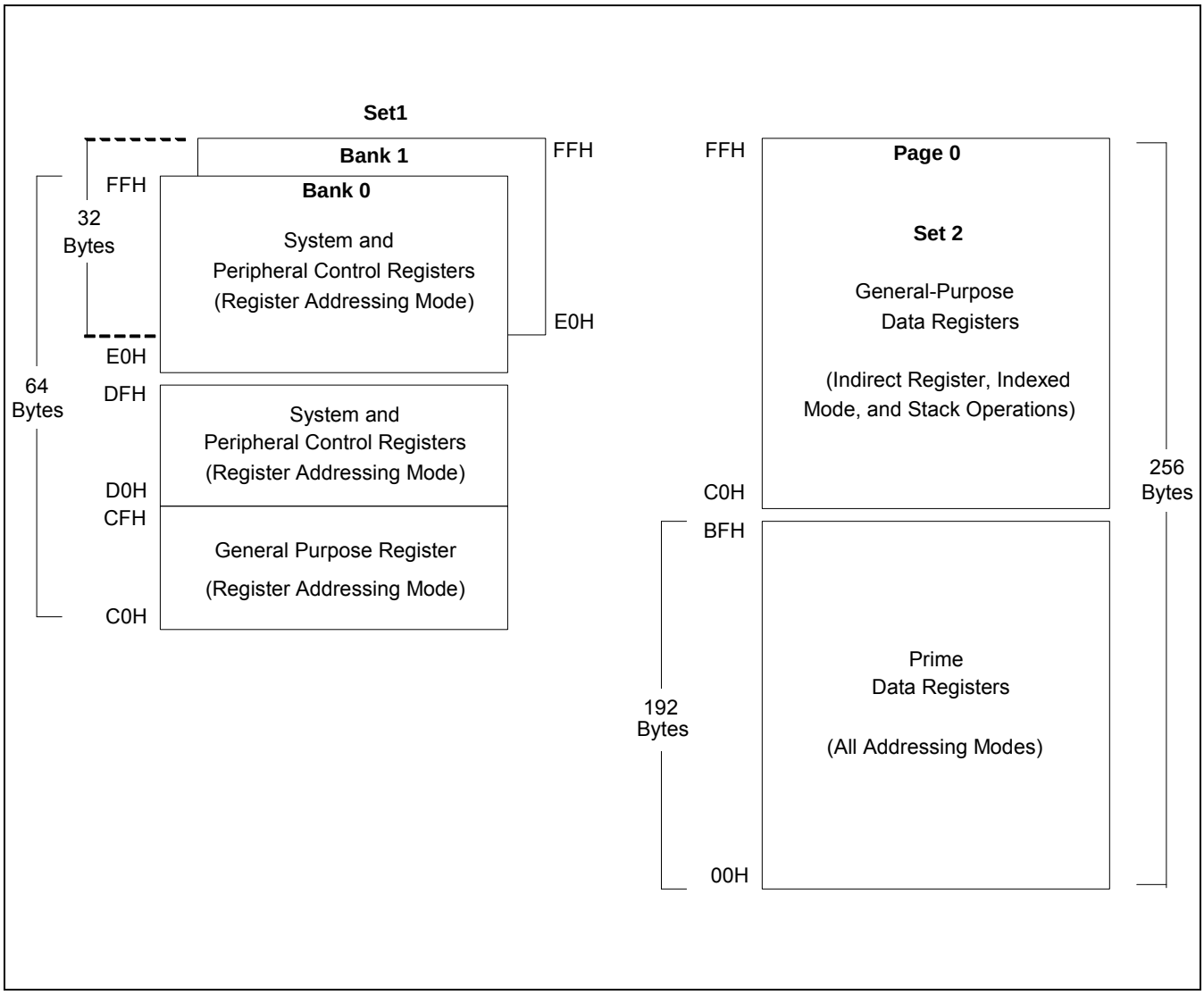


图 2-3 S3C84H5/F84H5 内部寄存器卷的地址空间

2.3.1 寄存器页面指针(PP)

S3C8- 系列 MCU 支持对物理上 256 字节的内部寄存器页进行逻辑扩展（通过 8 位数据总线）， 最多可实现2个可独立寻址的寄存器页。页面寻址由寄存器页面指针（PP, 地址：DFH）来控制。

复位之后，页面指针的源页（低四位）和目标页（高四位）数值缺省为“0000B”，自动选择页面 0 作为源和目标 页来进行寄存器寻址。

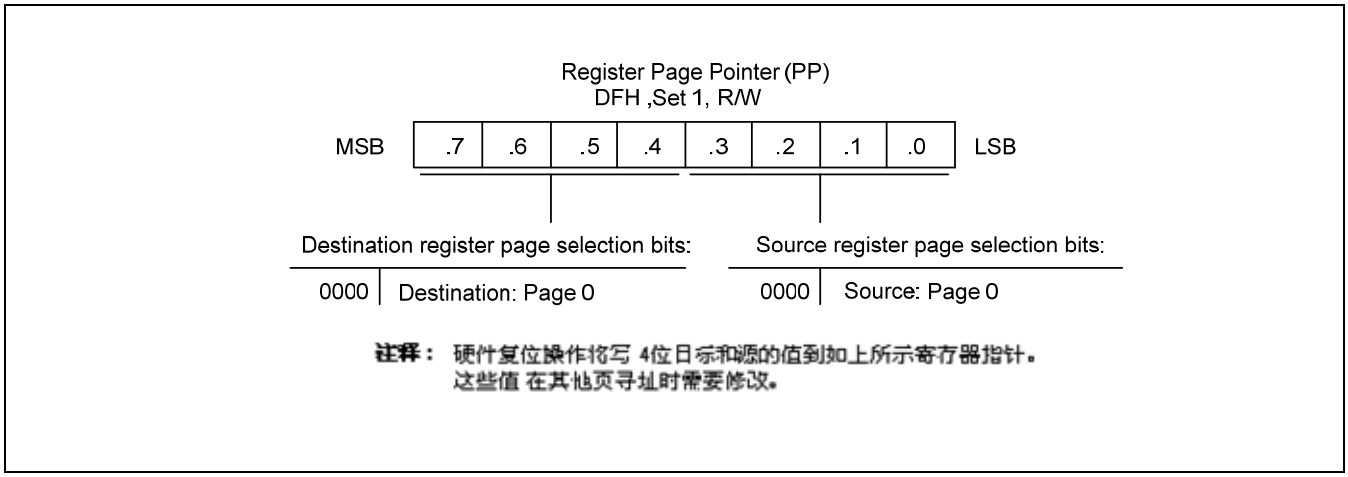


图 2-4 寄存器页面指针 (PP)

编程实例 2-1 在清RAM时使用页面指针

```
        LD      PP,#00H     ; 目标 ← 0, 源 ← 0
        SRP     #0C0H
        LD      R0,#0FFH    ; 开始对页 0 进行 RAM 清零
RAMCL0: CLR     @R0
        DJNZ    R0,RAMCL0
        CLR     @R0         ; R0 = 00H
        LD      PP,#10H     ; 目标 ← 1, 源 ← 0

        LD      R0,#0FFH    ; 开始对页 1 进行 RAM 清零
RAMCL1: CLR     @R0
        DJNZ    R0,RAMCL1
        CLR     @R0         ; R0 = 00H
```

2.3.2 寄存器SET 1

Set 1 指的是寄存器卷中的高 64 字节，地址为 C0H–FFH。

这 64 字节空间(E0H-FFH）的高 32 字节又被扩展为两个 32 字节的寄存器块，bank 0 和 bank 1。通过指令 SB0 或 SB1 来访问 bank 0 或 bank 1。硬件复位后默认选择 bank 0进行寻址。

Set 1(E0H-FFH)的两个高 32 字节区域（bank0 和 bank1）包含64个系统和外设控制寄存器，低 32 字节区域包含16 个系统寄存器（D0H-DFH）和 16 字节的通用工作寄存器（C0H-CFH）。在其它寄存器空间执行的数据操作，也可以通过工作寄存器。

Set 1 中的寄存器可以随时通过寄存器寻址模式进行访问。16 字节的工作寄存器区域，则只能使用工作寄存器寻址模式（更多关于工作寄存器寻址的信息，请查阅第 3 章,“寻址方式”）。

2.3.3 寄存器SET 2

对 set 1 的 64 字节地址空间（C0H-FFH）进行逻辑复制,以增加额外的 64 字节寄存器空间。 这个扩展的空间被称为 set 2。在 S3C84H5/F84H5 中，只有 page 0 可对 set 2 的地址空间 (C0H-FFH) 进行寻址。

通过寻址模式的限制，得以实现 set 1 和 set 2 的逻辑分割。Set 1 只支持寄存器寻址模式，而对 set 2 的寻址只能采用寄存器间接寻址模式或偏址寻址模式。

Set 2 寄存器区常用作堆栈操作。

2.3.4 主寄存器空间

在 S3C84H5/ F84H5 中， 256 字节寄存器页的低192 字节(00H-BFH)是主寄存器空间。 主寄存器空间可通过七种寻址模式中的任何一种进行访问（见第 3 章，“寻址方式”）。

芯片复位后，可立即对 page 0 的主寄存器空间进行寻址。为了在page 0 访问主寄存器空间，必须对寄存器页面指针（PP）的源页和目标页进行正确设置。

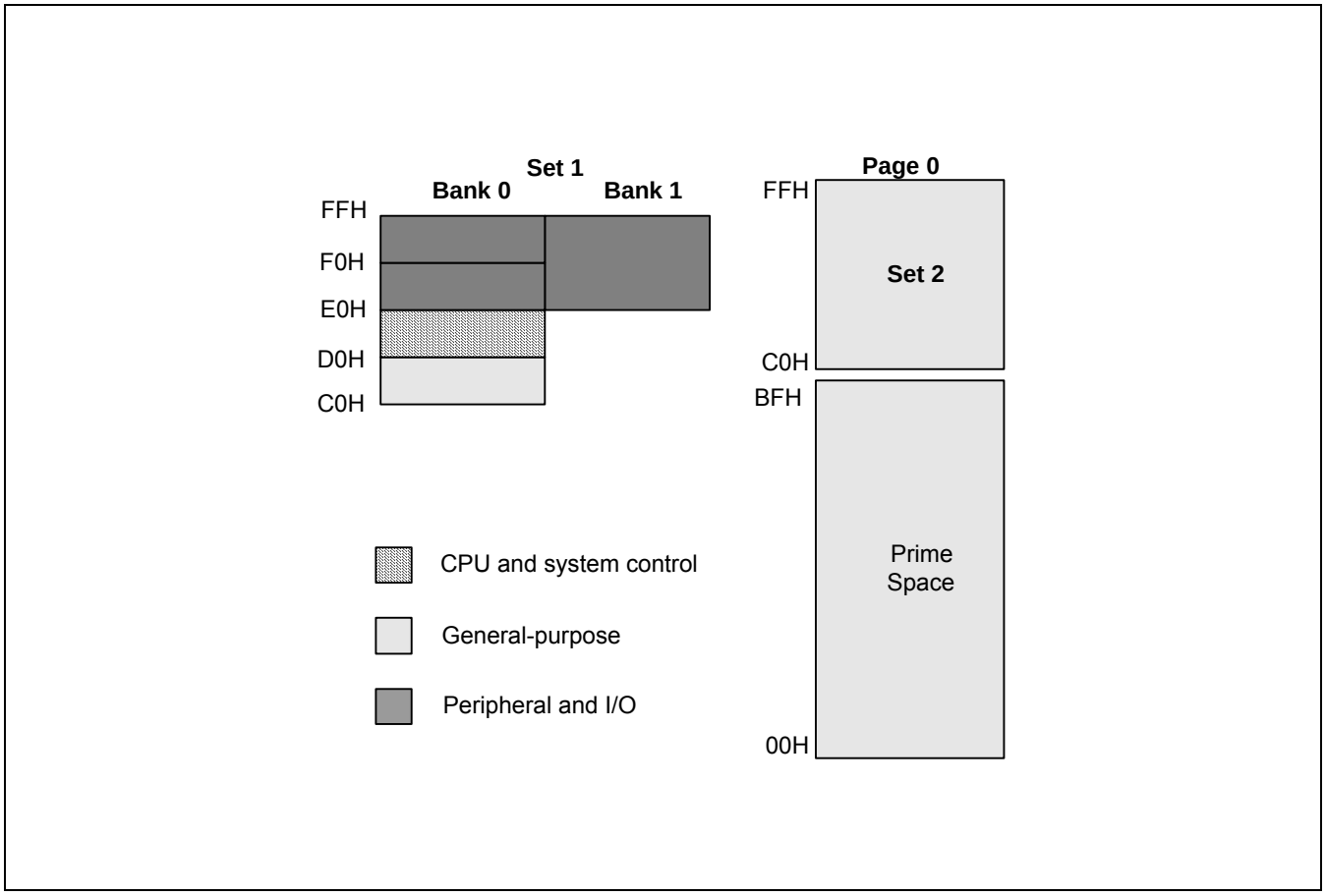


图 2-5 Set 1, Set 2 主寄存器空间

2.3.5 工作寄存器

指令可以使用 4 位或 8 位地址来访问指定的 8 位寄存器或者 16 位寄存器对。当使用 4 位工作寄存器寻址模式时，256 字节的寄存器卷可以被看成是由 32 个 8 字节寄存器组或“片(slice)”组成的。每个片包含 8 个 8 位寄 存器。

通过两个 8 位寄存器指针 RP1 和 RP0，可以随时选择两个工作寄存器片，组成一个 16 字节的工作寄存器块。使用这两个指针，可以将 16 字节的寄存器块移动到除 set 2 以外的任何可寻址空间。

在本手册中，术语“片(slice)”和“块(block)”用来帮助理解工作寄存器空间的大小和相对位置：

- 一个工作寄存器“片”是 8 个字节（8 个 8 位寄存器，R0-R7 或 R8-R15）
- 一个工作寄存器 “块” 是 16 个字节（16 个 8 位寄存器，R0-R15）

所有 8 字节的工作寄存器“片”中，寄存器地址的高 5 位数值完全相同。这使得各个寄存器指针都可以指向寄存器 卷中除 set 2 以外的 32 个寄存器“片”中任何一个。寄存器指针 RP0 和 RP1 中存放的是选定的两个 8 位寄存 器 “片” 的起始地址。

系统复位后，RP0 和 RP1 总是指向 Set 1 中的 16 字节公用空间（C0H-CFH）。

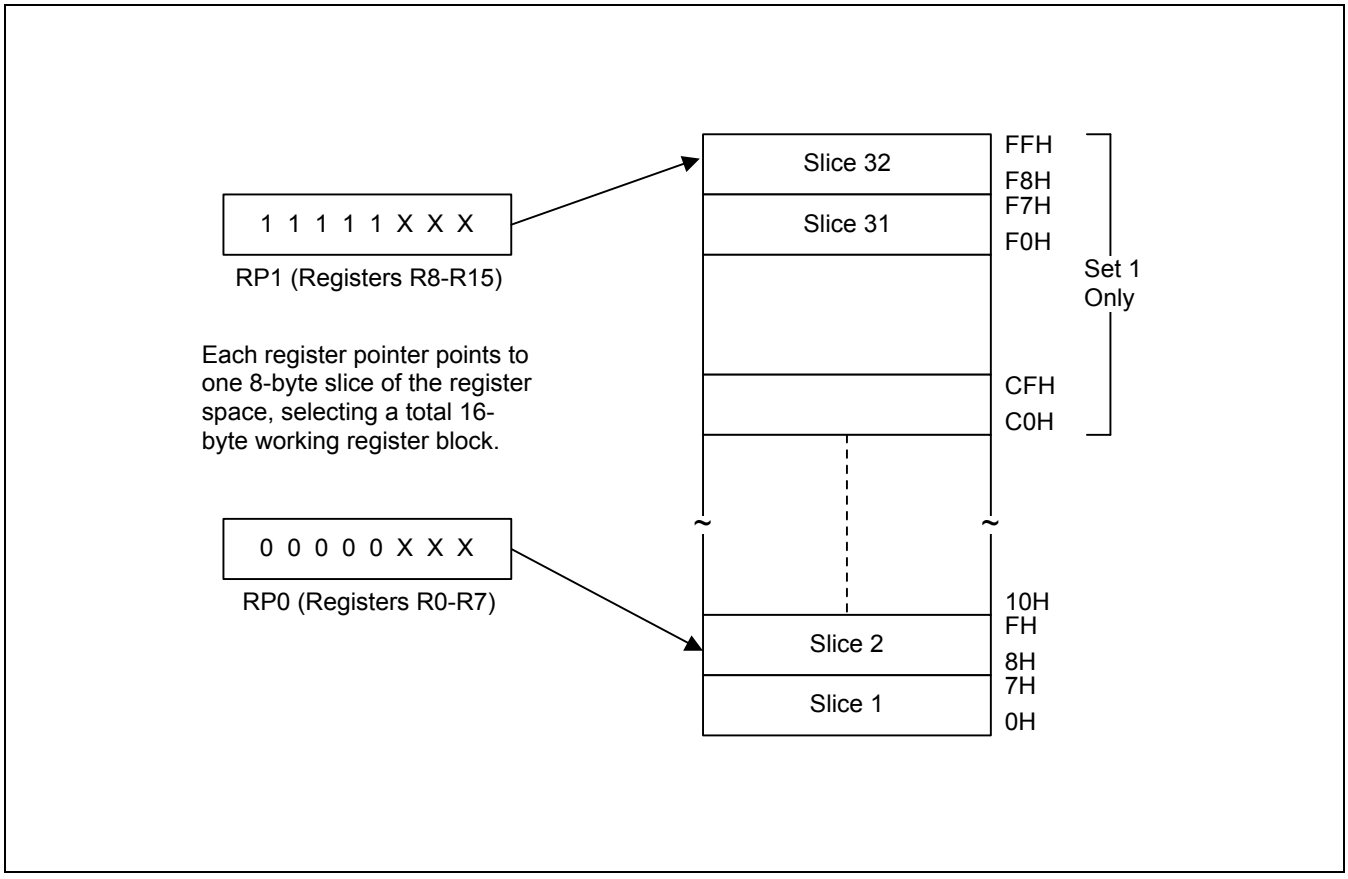


图 2-6 8 字节工作寄存器区 (Slices)

2.3.6 使用寄存器指针

寄存器指针 RP0 和 RP1(地址：D6H 和 D7H，Set 1) 用于选择寄存器卷中两个可移动的 8 字节工作寄存器片. 复位后，RP# 指向工作寄存器的公共空间：RP0 指向地址 C0H-C7H，RP1指向 C8H-CFH。

通过 SRP 或 LD 指令可以改变寄存器指针 RP0/ RP1 的值(*图 2-7*, *图 2-8*)。

用工作寄存器寻址方式，只能访问当前 RP0 和 RP1 指定的两个 8 字节的工作寄存器片。但不能用寄存器指针来选择 set 2 (C0-FFH) 中的工作寄存器，因为这些地址空间只支持间接寄存器寻址模式和偏址寻址模式。

16 字节的工作寄存器块通常由两个相邻的 8 字节工作寄存器片组成。编程时，一般建议将 RP0 指向“低”地址的 片，而 RP1 指向“高”地址的片（*图 2-7*）。某些情况下，可能需要将工作寄存器定义在不同的（不相邻的）寄存器空间中(*图 2-8*)，RP0 指向“高”地址的片，而 RP1 指向“低”地址的片 。

由于寄存器指针可以指向两个工作寄存器片中的任意一个，用户可根据程序需求灵活定义工作寄存器空间。

编程实例 2-2 设置寄存器指针

```
SRP    #70H          ;   RP0 ← 70H, RP1 ← 78H
SRP1   #48H          ;   RP0 ← 不变, RP1 ← 48H,
SRP0   #0A0H         ;   RP0 ← 0A0H, RP1 ← 不变
CLR    RP0           ;   RP0 ← 00H, RP1 ← 不变
LD     RP1, #0F8H    ;   RP0 ← 不变, RP1 ← 0F8H
```

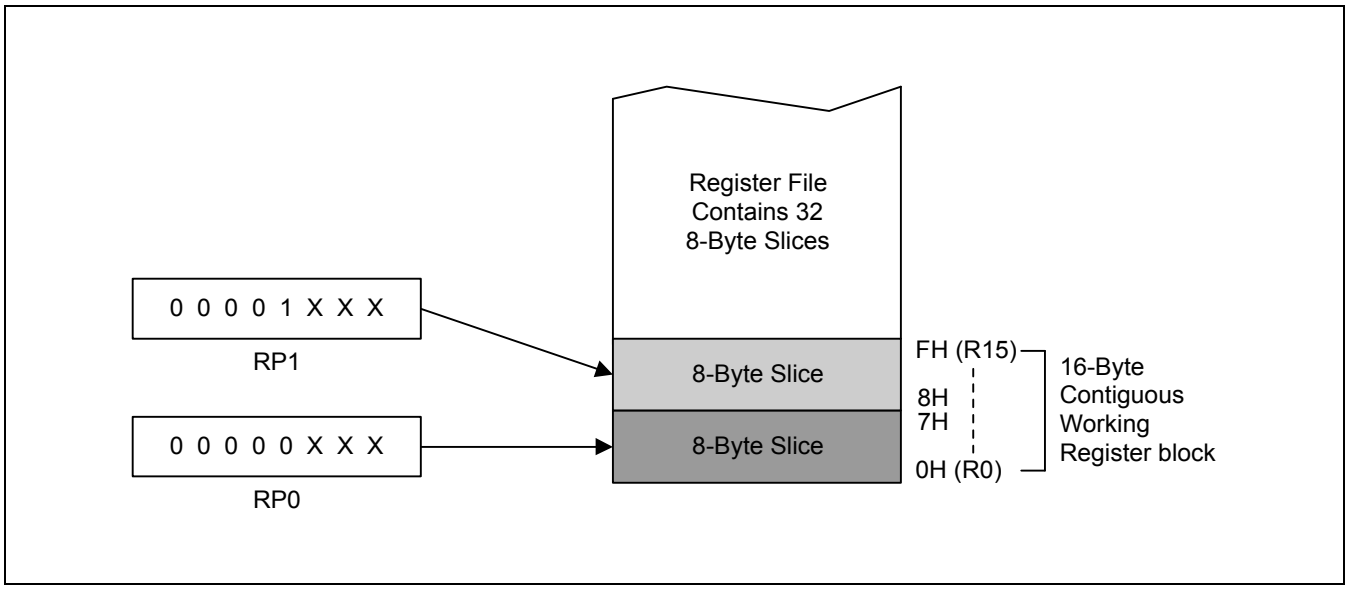


图 2-7 相邻的16 字节工作寄存器块

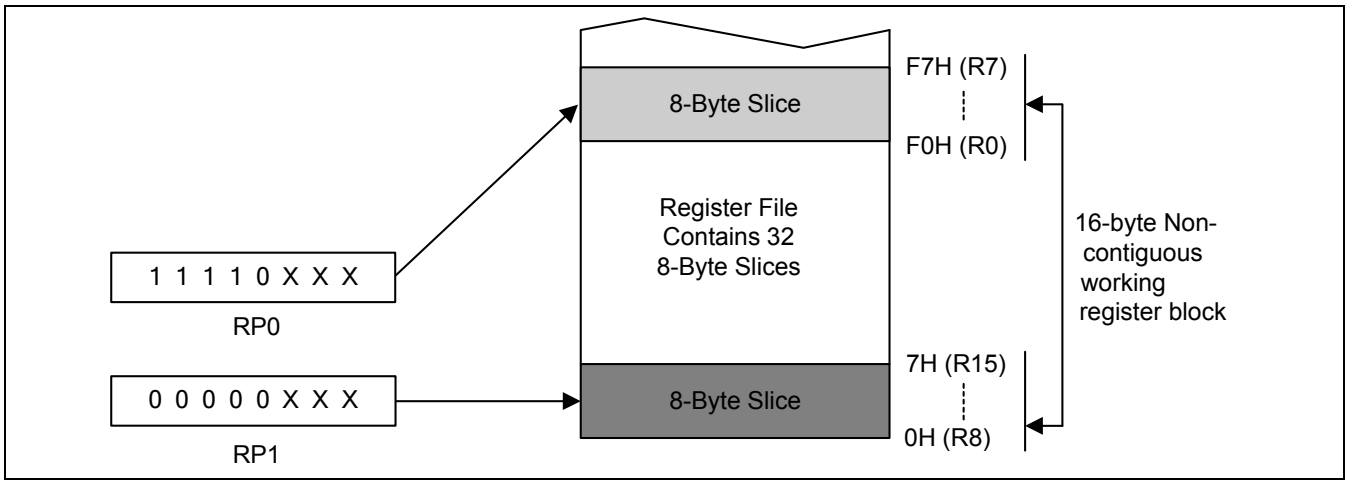


图 2-8 非相邻的16 字节工作寄存器块

编程实例 2-3 使用 RP 对寄存器 80H-85H 的内容求和

用寄存器指针来对寄存器 80H–85H 中的内容求和. 寄存器 80H-85H 中的存放数据分别为 10H, 11H, 12H, 13H, 14H和15H :

```
SRP0   #80H          ;    RP0  ←  80H
ADD    R0, R1        ;    R0   ←  R0 + R1
ADC    R0, R2        ;    R0   ←  R0 + R2 + C
ADC    R0, R3        ;    R0   ←  R0 + R3 + C
ADC    R0, R4        ;    R0   ←  R0 + R4 + C
ADC    R0, R5        ;    R0   ←  R0 + R5 + C
```

6 个寄存器的和 (6FH) 存放在 R0 (80H) 中. 例子中的指令共占用 12 字节的代码长度, 执行时间为 36 个时钟周期. 如果不用寄存器指针, 那就得按照下面的指令顺序来做 :

```
ADD    80H, 81H      ;    80H  ←  (80H) + (81H)
ADC    80H, 82H      ;    80H  ←  (80H) + (82H) + C
ADC    80H, 83H      ;    80H  ←  (80H) + (83H) + C
ADC    80H, 84H      ;    80H  ←  (80H) + (84H) + C
ADC    80H, 85H      ;    80H  ←  (80H) + (85H) + C
```

现在, 6 个寄存器的和依然放在 80H 当中, 但是所有指令总共占用了 15 字节的代码长度而非 12 字节, 且执行时间为 50 个时钟周期而非 36 个.

2.4 寄存器寻址

S3C8- 系列单片机的寄存器结构提供了一种效率高的工作寄存器寻址方式，该方式充分利用了短指令格式以缩短程序执行时间。

在寄存器寻址模式中，操作数存放在某个特定的寄存器或寄存器对中。用寄存器寻址模式可以访问寄存器空间中除 set 2 以外的的任何地址。使用工作寄存器寻址时，通过寄存器指针指定一个 8 字节的工作寄存器空间和该空间内的一个 8 位寄存器。

寄存器寻址时，既可以视其为单独的 8 位寄存器，也可以视为成对的 16 位寄存器空间。在 16 位寄存器对中，第一8位寄存器的地址总是偶数，而另一个寄存器地址则是奇数。16 位数据的高位字节存放在偶地址寄存器中，低位字节存放在邻近的 (+1) 奇地址寄存器中。

工作寄存器寻址模式与寄存器寻址模式的不同之处在于，它通过寄存器指针来指定一个 8 字节工作寄存器空间，以及其中的某个 8 位工作寄存器。

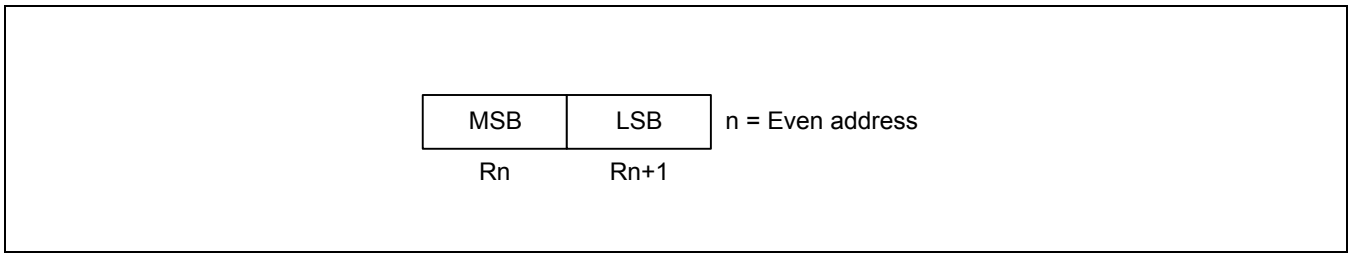


图 2-9 16 位寄存器结构

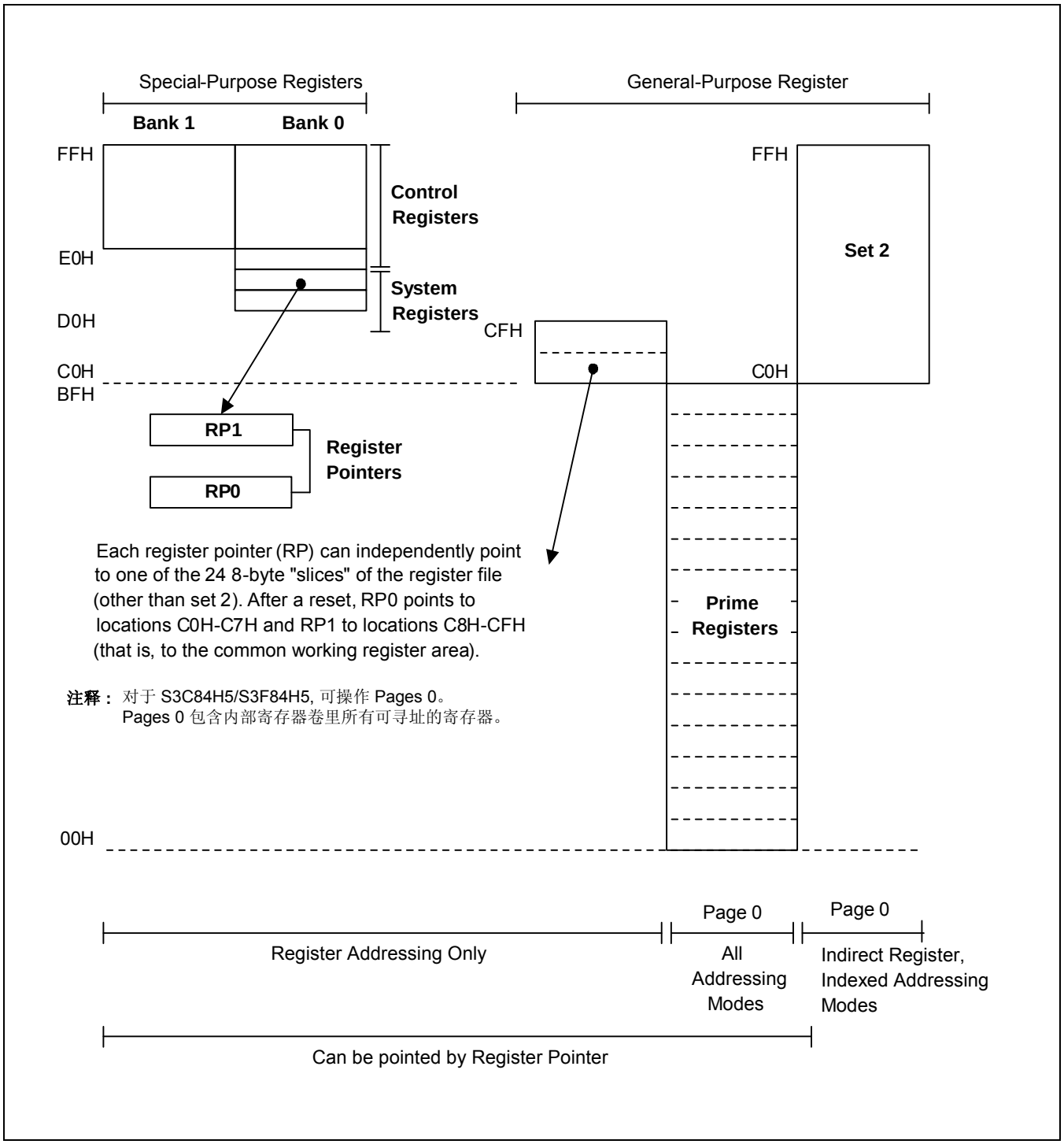


图 2-10 寄存器卷寻址模式

2.4.1 通用工作寄存器区(C0H-CFH)

系统复位后，寄存器指针 RP0 和 RP1 自动指向 set 1中两个 8 字节的寄存器片(C0H-CFH)，组成 16 字节的寄存器块:

RP0 → C0H-C7H
RP1 → C8H-CFH

这 16 字节的地址空间称为通用工作寄存器区。就是说无论当前操作所要访问的是哪个页面，都可以把该空间的寄存器作为工作寄存器使用。典型地，在不同页面间进行数据交换操作时，可使用工作寄存器作为临时存储。

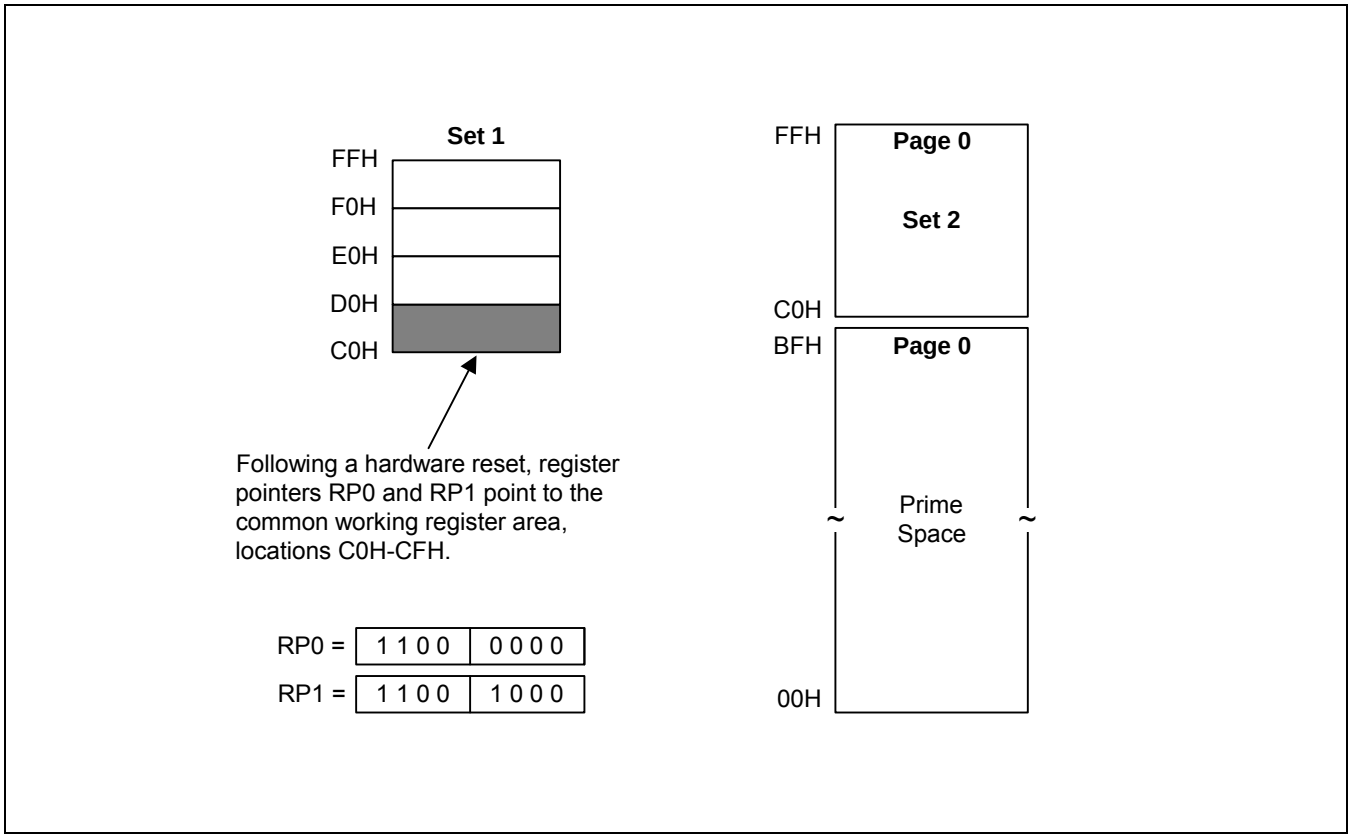


图 2-11 通用工作寄存器区

编程实例 2-4 访问通用工作寄存器区

如下例所示，要访问通用工作寄存器(C0H-CFH)，只能采用工作寄存器寻址模式：

例1:

```
LD      0C2H, 40H      ;       非法寻址模式!
```

改为工作寄存器寻址模式:

```
SRP     #0C0H
LD      R2, 40H        ;       R2 (C2H) ← 地址 40H 中存放的值
```

例2:

```
ADD     0C3H, #45H     ;       非法寻址模式!
```

改为工作寄存器寻址模式:

```
SRP     #0C0H
ADD     R3, #45H       ;       R3 (C3H) ← R3 + 45H
```

2.4.2 4 位工作寄存器寻址方式

每个寄存器指针定义了一个可移动的 8 字节寄存器片，其中的地址信息作为一扇寻址的“窗”，使得指令只须 4 位 地址就可以实现对工作寄存器的有效访问。在工作寄存器寻址时，8 位地址是采用下述方法构成的：

- 4 位地址的最高位选择一个寄存器指针（“0” 选择 RP0, “1” 选择 RP1）
- 寄存器指针的高 5 位选择寄存器卷中的某个 8 字节寄存器片
- 指令中 4 位地址的低 3 位选定 8 字节寄存器片中的一个

图 2-11，操作的结果是，寄存器指针的高 5 位和指令地址的低 3 位一起组成完整的 8 位寄存器地址。 只要寄存器指针中保存的地址不变，指令中 4 位地址的低 3 位总是指向同一个 8 字节寄存器片。

图 2-13 给出了一个典型的 4 位工作寄存器寻址实例。指令“INC R6”的最高位是“0”，这将选择 RP0。RP0 的高 5 位（01110B）与指令中 4 位地址的低 3 位（110B）一起组成了寄存器地址 76H（01110110B）。

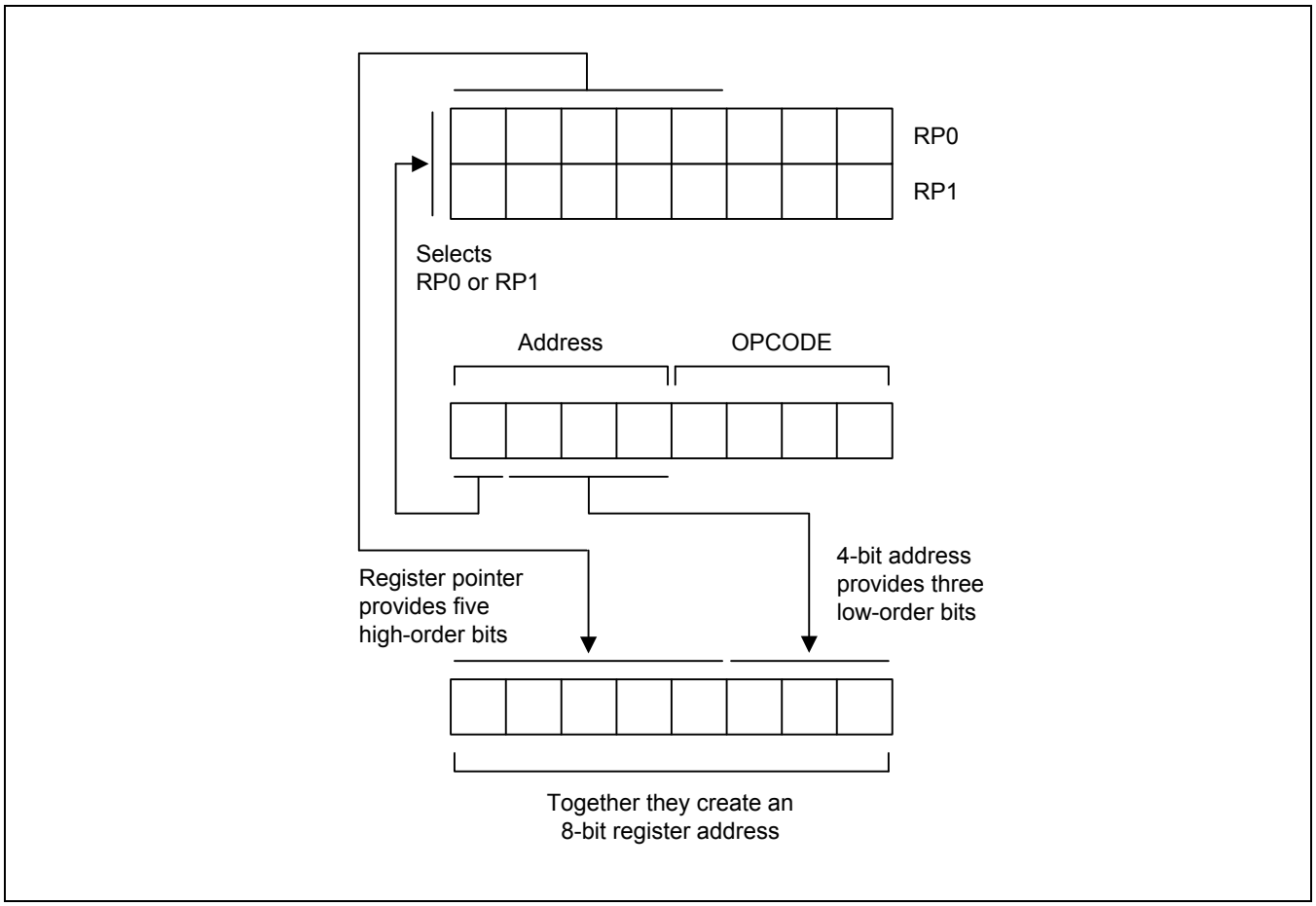


图 2-12 4 位工作寄存器寻址方式

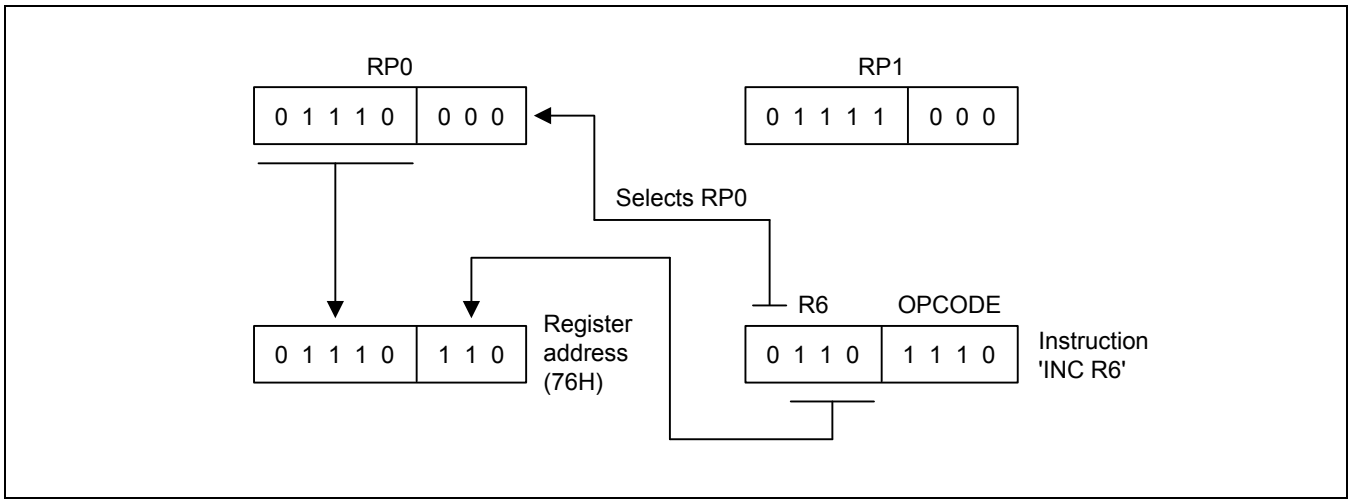


图 2-13 4 位工作寄存器寻址实例

2.4.3 8 位工作寄存器寻址方式

还可以通过 8 位工作寄存器寻址方式来访问工作寄存器区。首先，指令地址的高 4 位必须是“1100B”，这 4 位 数据 (1100B) 表示余下的 4 位与 4 位工作寄存器寻址方式相同。

图 2-13，8 位地址的低阶位形成机制与4位工作寄存器寻址时类似：第 3 位选择 RP0 或 RP1，用来产生最 终地址的高 5 位；而最终地址的低 3 位则由原始指令提供。

图 2-14 给出了一个典型的 8 位工作寄存器寻址实例。指令地址的高 4 位 (1100B) 表明寻址方式为 8 位寄 存器寻址。第 3 位(“1”)选择 RP1，所以 RP1中的高 5 位（10101B）成为寄存器地址的高 5 位，寄存器地址的 低 3 位（011B）则由 8 位指令地址的低 3 位提供。RP1 中的 5 个地址位和指令中的 3 个地址位组合，形成 了完整的寄存器地址，0ABH（10101011B）。

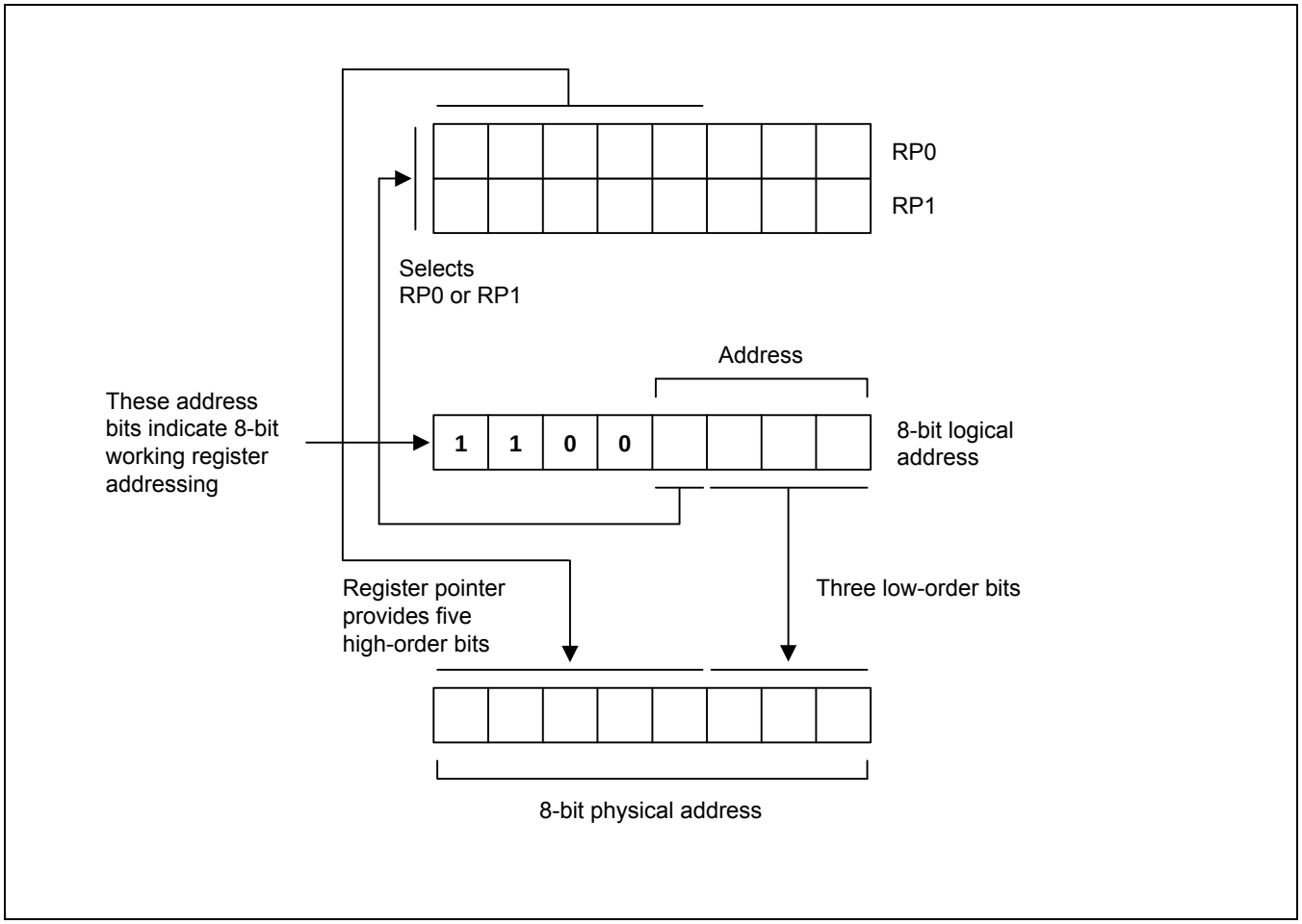


图 2-14 8 位工作寄存器寻址方式

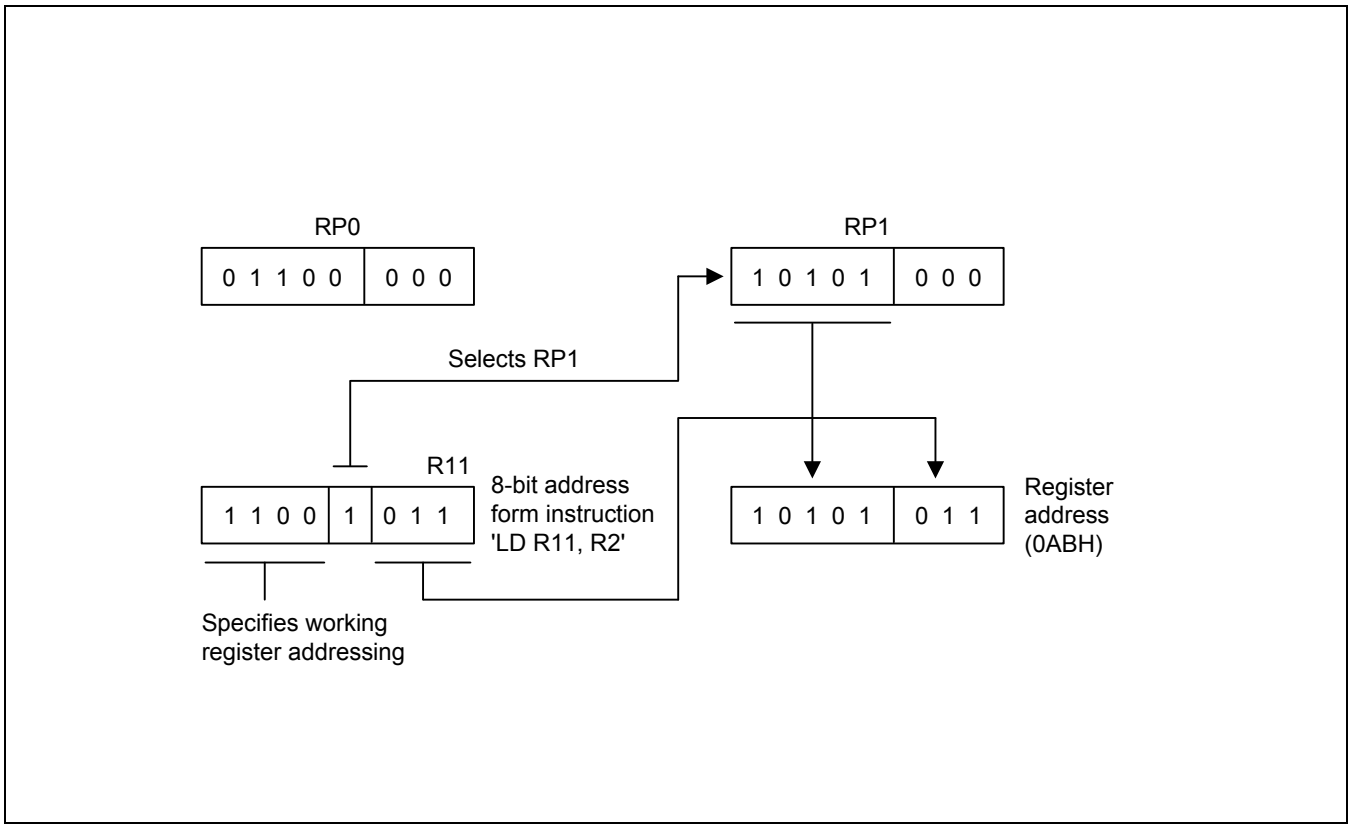


图 2-15 8 位工作寄存器寻址实例

2.5 系统和用户栈

S3C8- 系列单片机通过栈来处理数据存储，子程序调用和返回。PUSH 和 POP 指令用于控制栈操作。S3C84H5/F84H5 支持在内部寄存器卷中进行栈操作。

2.5.1 栈操作

栈用来储存程序调用以及中断的返回地址，也用来储存数据。CALL 指令将 PC 的值压入栈，而 RET 指令将其从栈中弹出。当中断发生时，PC 和 FLAGS 寄存器的值被压入栈，指令 IRET 则将这些值弹出到它们原来的地址。栈指针总是在 PUSH 操作之前先减“1”，在 POP 操作之后加“1”。栈指针(SP)总是指向栈的顶端，*图 2-16* 所示。

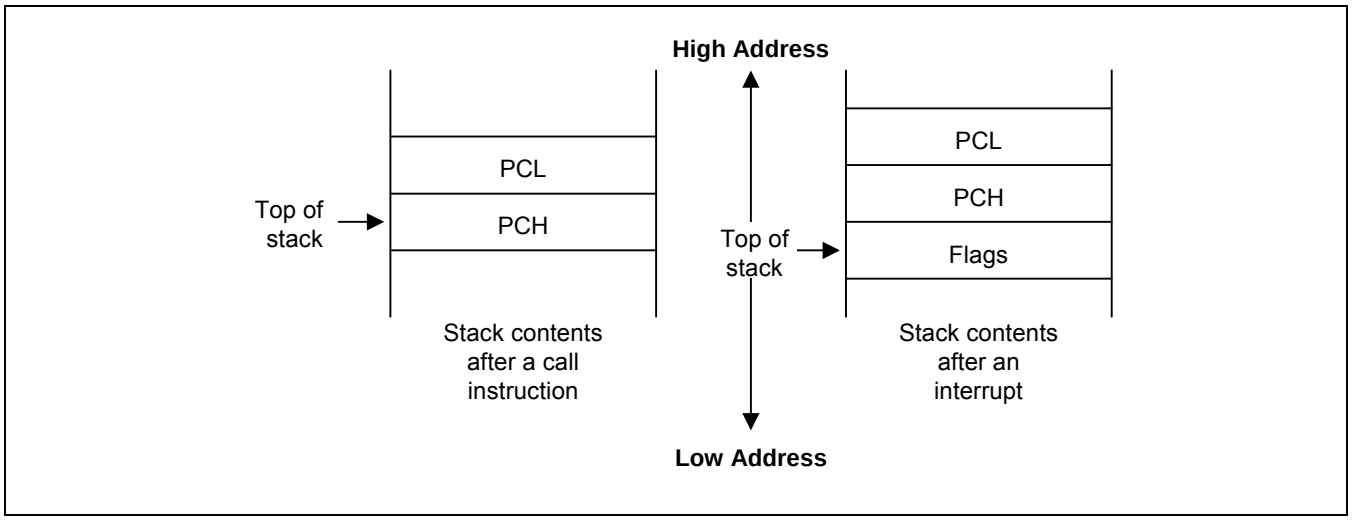


图 2-16 栈操作

2.5.2 用户自定义栈

可以在内部寄存器卷中堆栈自由定义栈数据储存区域。指令 PUSHUI，PUSHUD，POPUI，POPUD 支持用户自定义栈操作。

2.5.3 栈指针(SPL, SPH)

寄存器地址 D8H 和 D9H 存放用于栈操作的 16 位栈指针(SP)。SP 的高字节地址，SP15-SP8，存放在 SPH 寄存器（D8H）中；低字节地址，SP7-SP0，存放在 SPL 寄存器（D9H）中。系统复位后，SP 的值不确定。

由于 S3C84H5/F84H5 只有内部存储空间，SPL 必须被初始化成介于 00H-FFH 之间的 8 位数值。

SPH不使用，可以用作通用寄存器。

当SPL只包含栈指针（也就是说，当它指向寄存器卷中的一个系统堆栈时），SPH可以用作通用数据寄存器。不过，如果发生溢出或下溢导致SPL里的堆栈地址加一或减一，那么SPL的值 会溢出（或下溢）到SPH寄存器， 会覆 盖当前存储在 SPH 里的数据。为了避免SPH的值会被覆盖掉，可以初始化SPL的值为"FFH" 而不是"00H"。

编程实例 2-5 用 PUSH 和 POP 实现标准栈操作

下面的例子演示了在内部寄存器卷中如何通过 PUSH 和 POP 指令进行栈操作：

```
LD      SPL, #0FFH          ;       SPL         ←       FFH
                            ;       (通常 SPL 被初始化为 0FFH )
•
•
•
PUSH    PP                  ;       栈地址 0FEH ←       PP
PUSH    RP0                 ;       栈地址 0FDH ←       RP0
PUSH    RP1                 ;       栈地址 0FCH ←       RP1
PUSH    R3                  ;       栈地址 0FBH ←       R3

•
•
•
POP     R3                  ;       R3          ←       栈地址 0FBH
POP     RP1                 ;       RP1         ←       栈地址 0FCH
POP     RP0                 ;       RP0         ←       栈地址 0FDH
POP     PP                  ;       PP          ←       栈地址 0FEH
```

3 寻址模式

3.1 概述

通过程序指针(PC)读取程序存储空间的指令然后执行。指令隐含着要进行的操作和操作数。寻址模式是用于确定操作数地址的一种方法。SAM8RC 指令中的操作数可以是条件代码、立即数，或者寄存器卷、程序存储区、 数据存储区的地址。

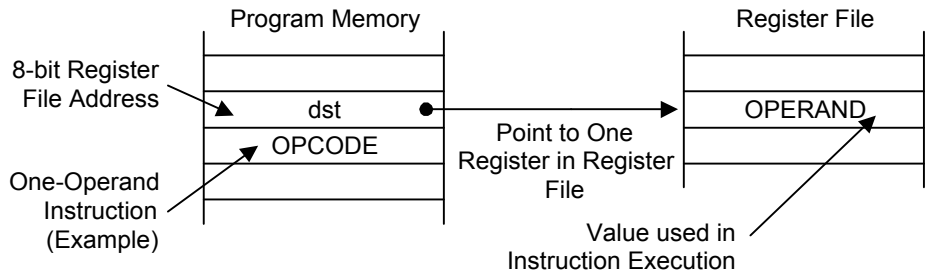
S3F8-系列的指令集支持 7 种寻址模式，但这些寻址模式并非适用于所有指令。7 种寻址模式和它们的符号表示:

- 寄存器寻址模式(R)
- 间接寄存器寻址模式(IR)
- 偏址寻址模式(X)
- 直接寻址模式(DA)
- 间接寻址模式(IA)
- 相对地址寻址模式(RA)
- 立即数寻址模式(IM)

3.1.1 寄存器寻址模式(R)

寄存器寻址模式（R）中，操作数是具体寄存器或寄存器对中的内容，*图 3-1*。

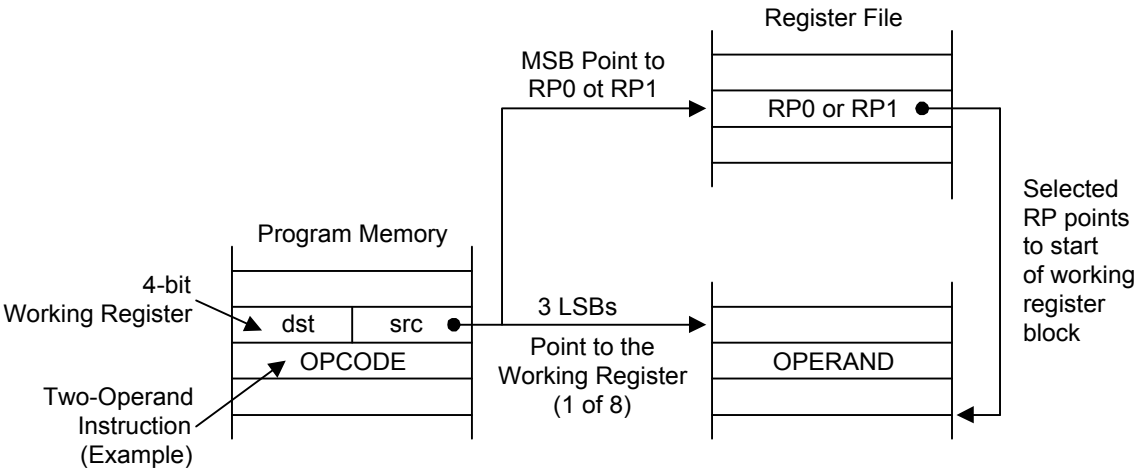
工 工作寄存器寻址模式与寄存器寻址模式不同。因为工作寄存器是通过一个 16 位的寄存器指针，来指定 8 字节的工作寄存器空间，和空间内的某个 8 位寄存器，*图 3-2*。



Sample Instruction:

DEC CNTR ; Where CNTR is the label of an 8-bit register address

图 3-1 寄存器寻址模式



Sample Instruction:

ADD R1, R2 ; Where R1 and R2 are registers in the currently selected working register area.

图 3-2 工作寄存器寻址模式

3.1.2 间接寄存器寻址模式(IR)

在间接寄存器寻址模式中，指定寄存器或寄存器对中存放的是操作数的地址。根据所用的指令，物理地址可能是寄存器卷中的寄存器、程序存储器（ROM），或者外部数据存储器（如图 *图 3-3* 至 *图 3-6*）。

可以用任意的 8 位寄存器来访问其它的寄存器，也可以用任意的 16 位寄存器组来访问其它的存储空间。但要注意，不能通过间接寄存器寻址模式对 Set 1 中地址段 C0H-FFH 进行访问。

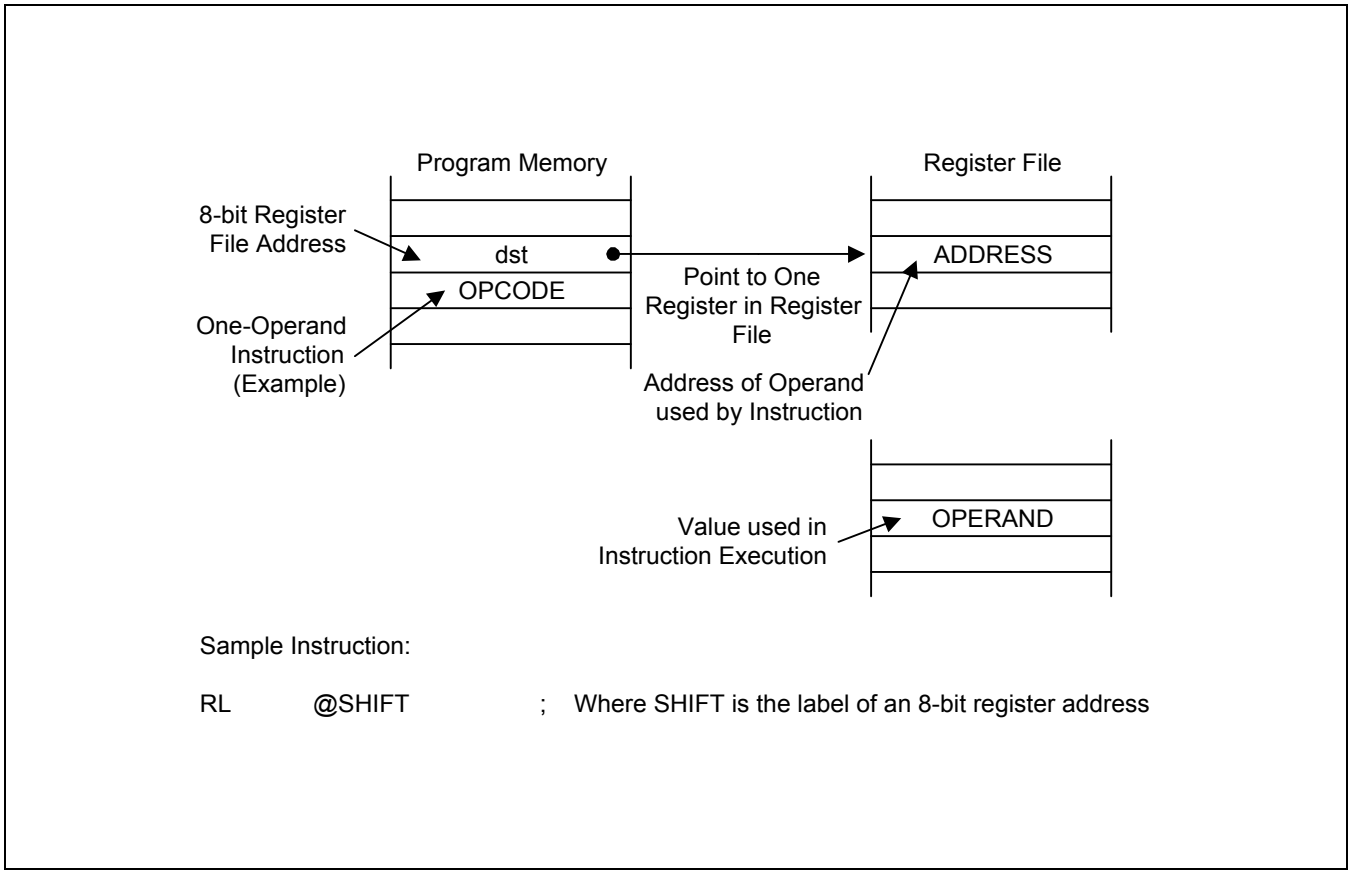


图 3-3 寄存器卷中的间接寄存器寻址模式

3.1.2.1 间接寄存器寻址模式(续)

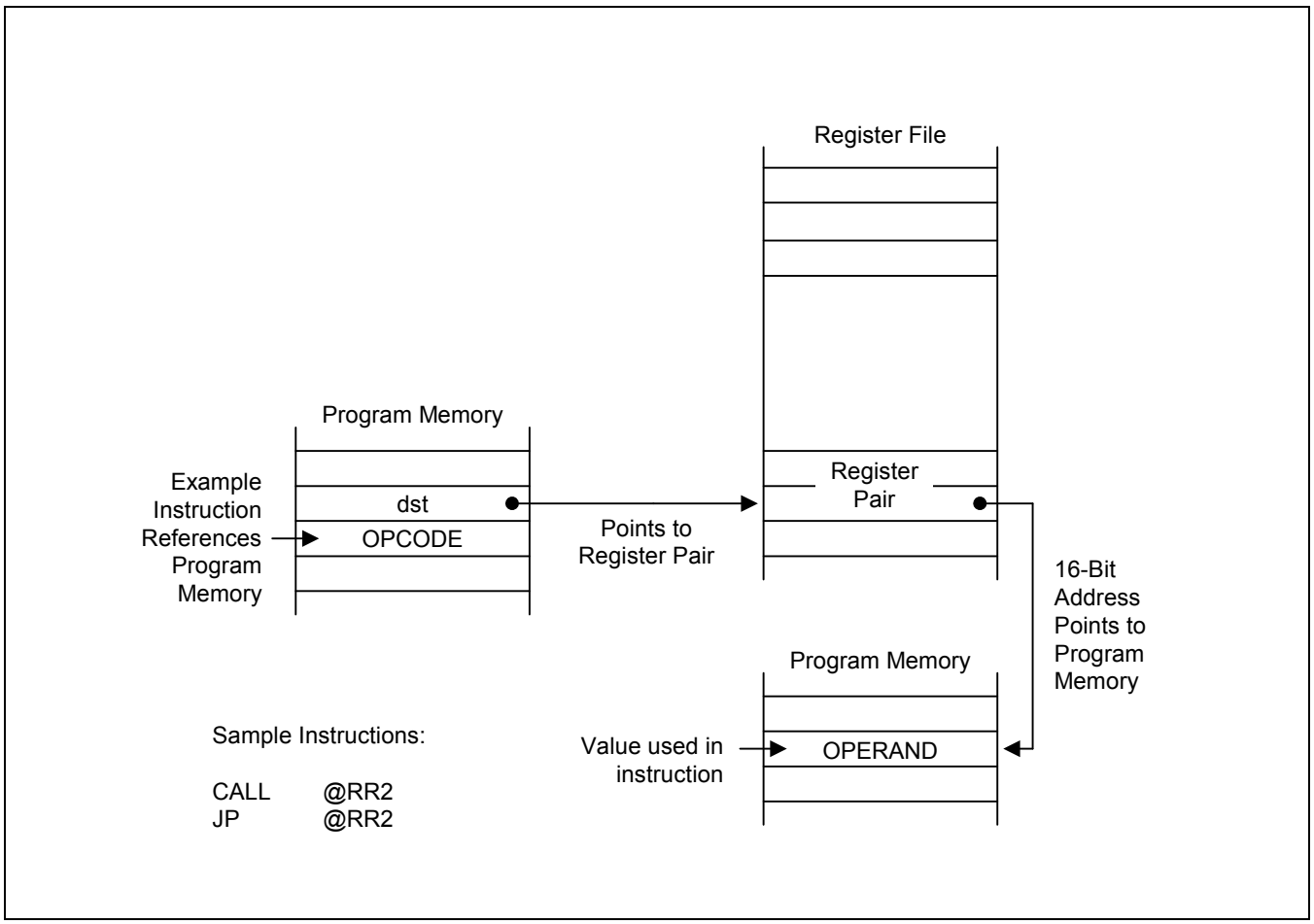


图 3-4 程序存储空间的间接寄存器寻址

3.1.2.2 间接寄存器寻址模式(续)

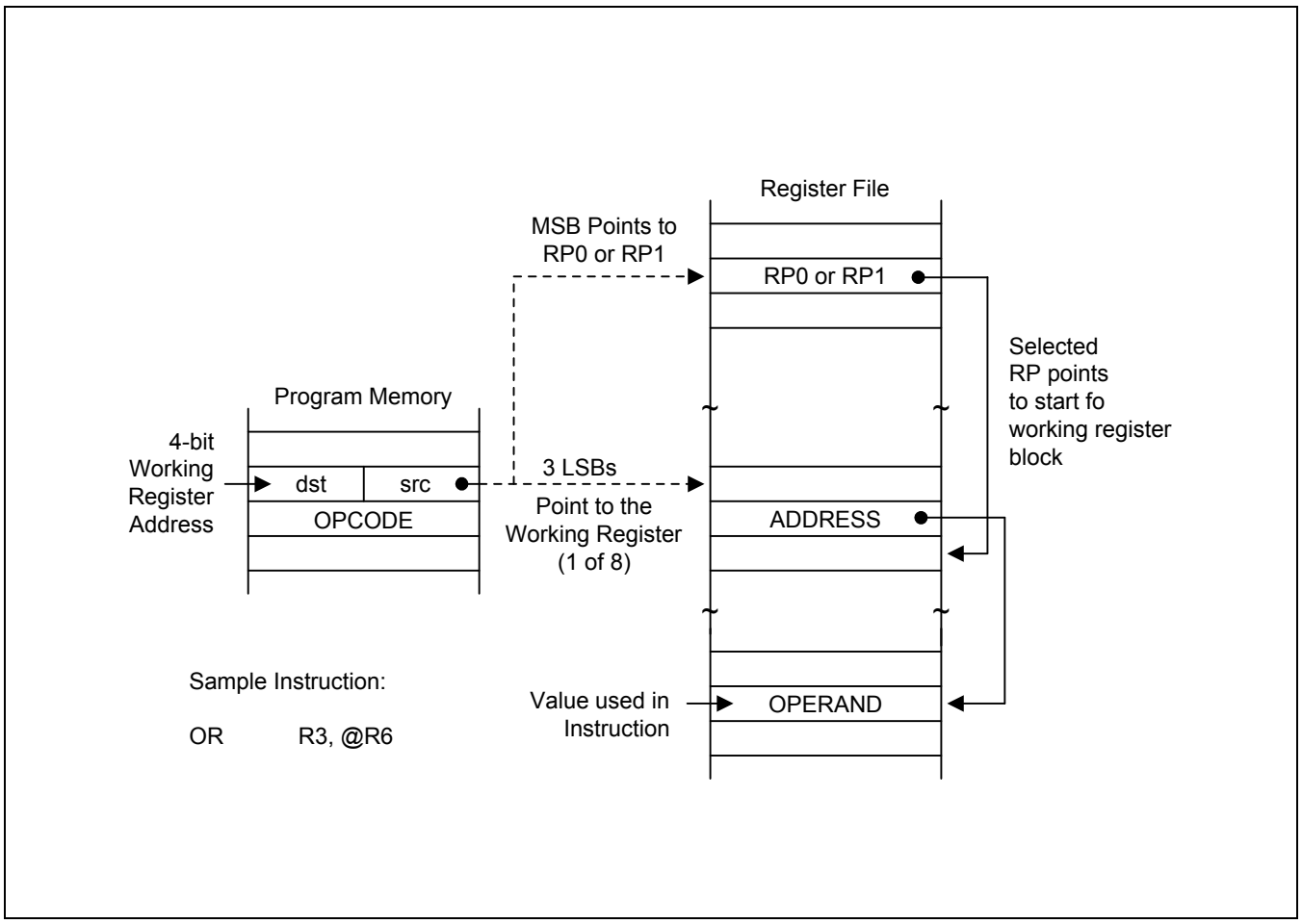


图 3-5 寄存器卷中的间接寄存器寻址

3.1.2.3 间接寄存器寻址模式(续)

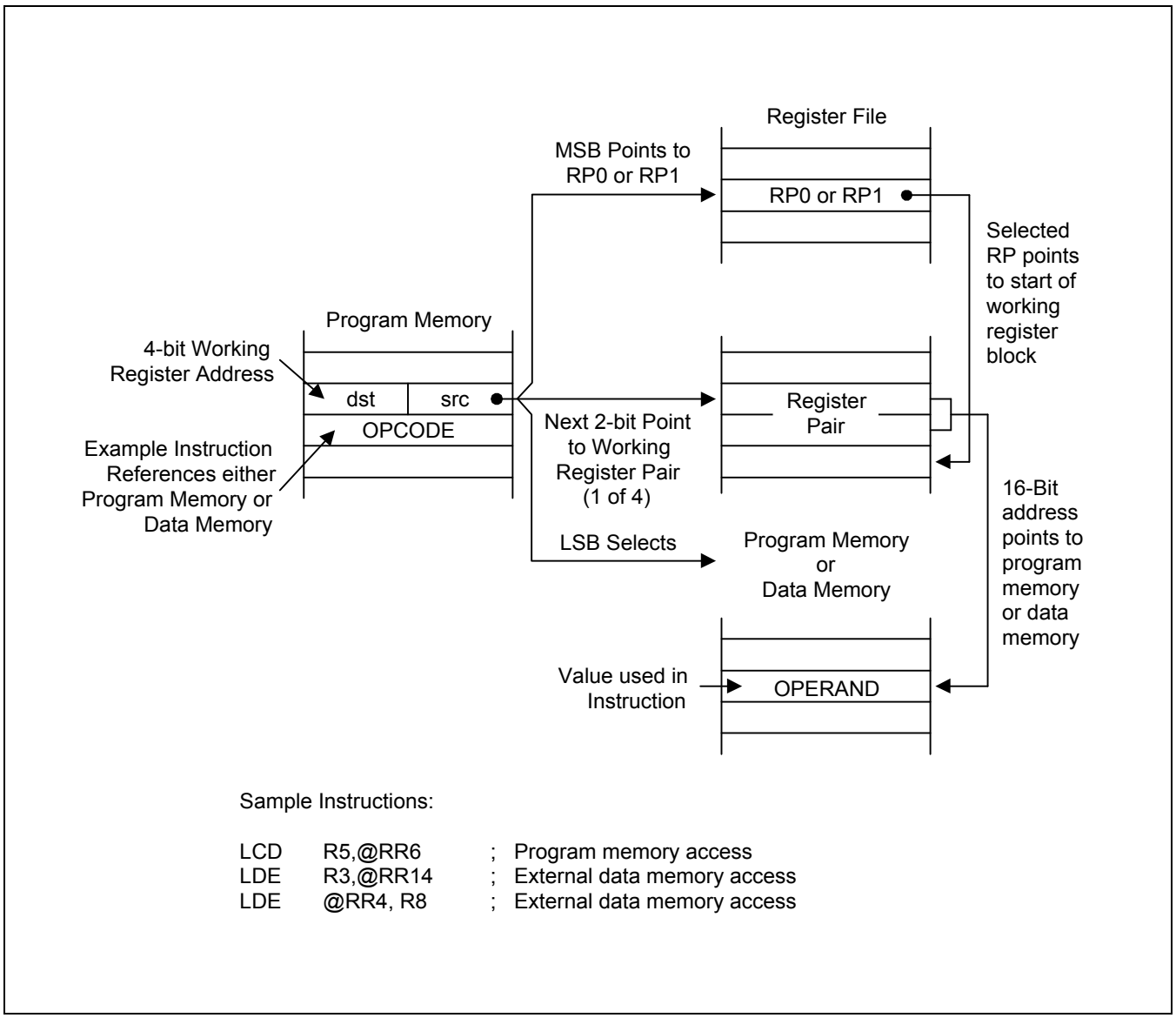


图 3-6 工作寄存器间接访问程序存储器或数据存储器

3.1.3 偏址寻址模式(X)

在指令执行时，偏址寻址模式（X）是在基地址的基础上加上一个偏移地址量，计算出有效的操作数地址（如图 *图 3-7*）。偏址寻址可以用来访问内部寄存器卷或外部数据存储器空间。但要注意，不能通过它对 Set 1 中地 址段 C0H-FFH 进行寻址。

在短指令寻址模式下，8 位的偏移量被认为是介于 -128 到 +127 之间的一个有符号整数，这只用于外部存储器访问（如图 *图 3-8*）。

对寄存器卷寻址时，指令提供的 8 位基地址与工作寄存器中的 8 位偏移地址相加得到操作数地址。对外部存储器访问时，基地址存放在指令指示的 16 位工作寄存器中，指令中给出的 8 位或 16 位偏移地址加到基地址上，得到操作数地址（如图 *图 3-9*）。

支持对寄存器卷进行偏址寻址的指令只有 Load（LD）。LDC 和 LDE 支持内部程序存储器和外部数据存储器 (如果存在）的偏址寻址模式。

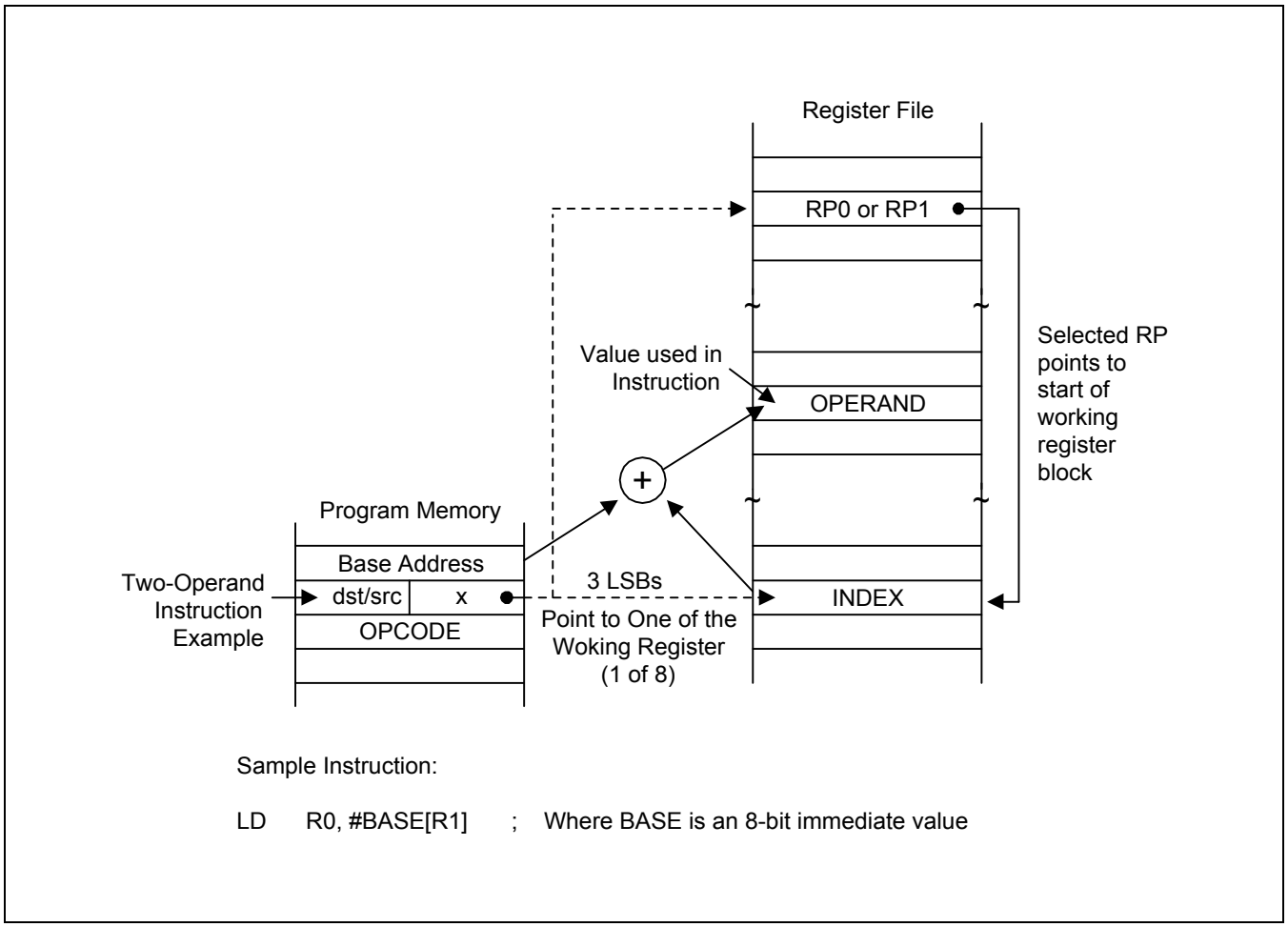


图 3-7 寄存器空间的偏址寻址模

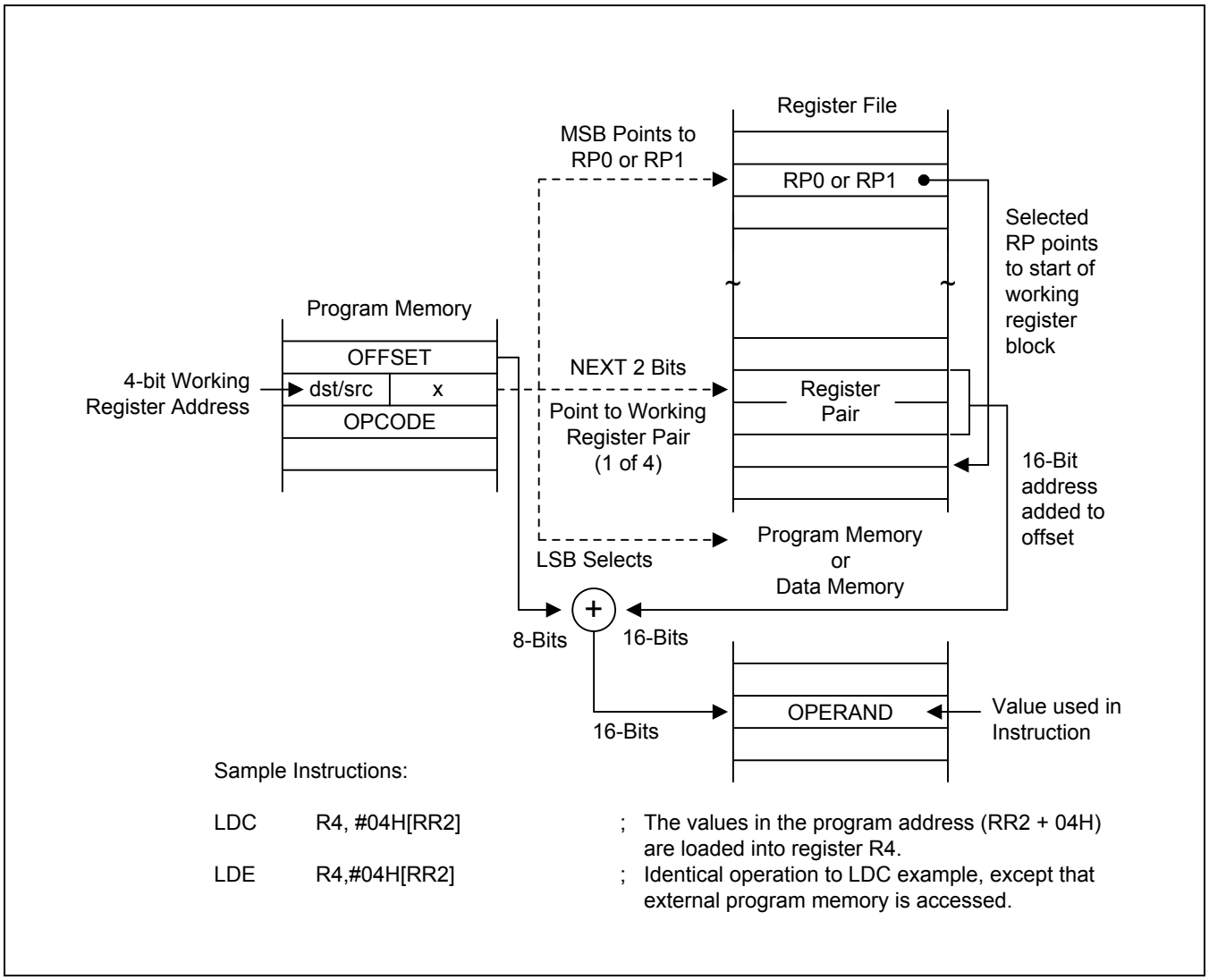


图 3-8 偏址寻址模式中短格式访问程序或数据存储空

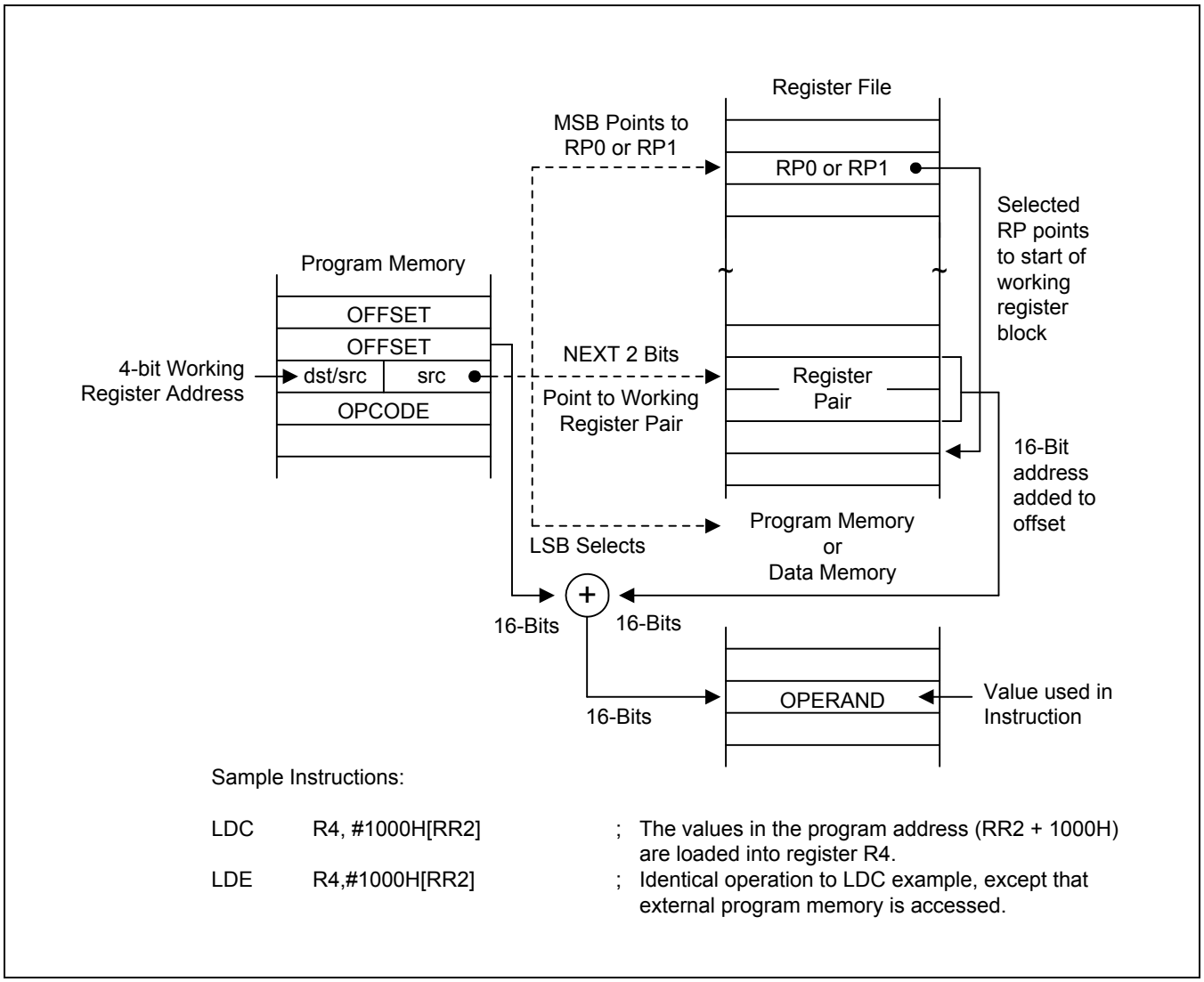


图 3-9 偏址寻址模式中长格式访问程序或数据存储空间

3.1.4 直接寻址模式(DA)

在直接寻址模式中，指令提供操作数的 16 位存储器地址。执行 Jump(JP) 和 Call(CALL) 指令时，就是采用这种寻址模式指定 16 位目标地址并将其装入PC。

LDC 和 LDE 指令即运用直接寻址模式为数据传送操作提供源地址或目标地址，LDC 访问程序存储空间，LDE 访问外部数据存储空间。

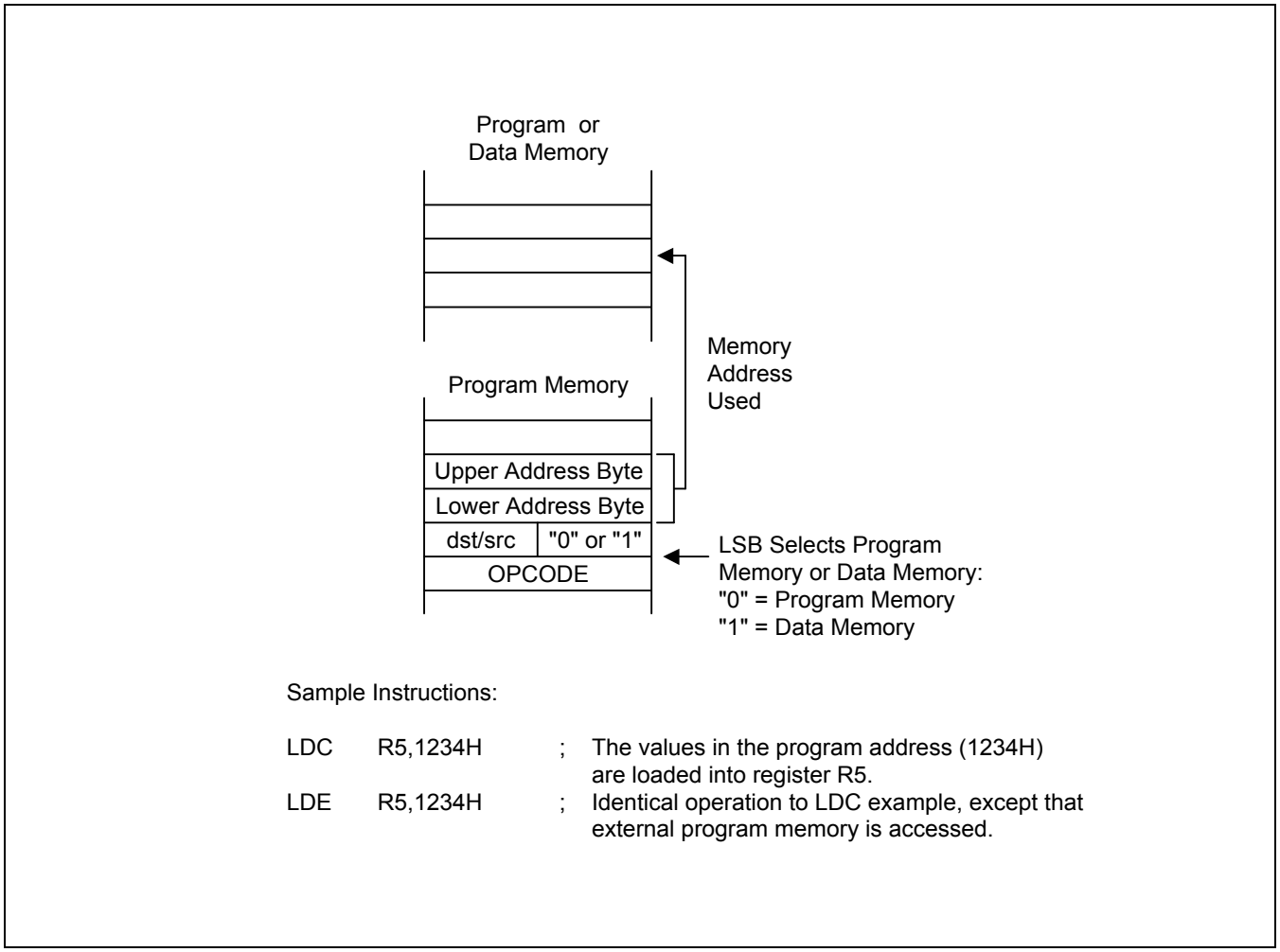
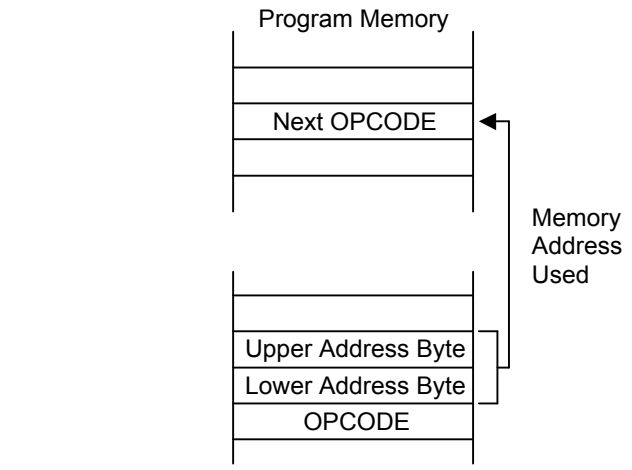


图 3-10 Load 指令的直接寻址

3.1.4.1 直接寻址模式(续)



Sample Instructions:

```
JP      C,JOB1          ;   Where JOB1 is a 16-bit immediate address
CALL    DISPLAY         ;   Where DISPLAY is a 16-bit immediate address
```

图 3-11 Call, Jump 的直接寻址

3.1.5 间接寻址模式(IA)

在间接寻址模式中，须指定程序存储空间中低 256 字节中的某个地址，其中放有要执行的下一条指令地址。 只有 CALL 指令支持间接寻址模式。

由于间接寻址模式规定操作数只能存放在程序存储空间的低 256 字节中，所以指令中只有一个 8 位地址；目标地址的高 8 位全部为 0。

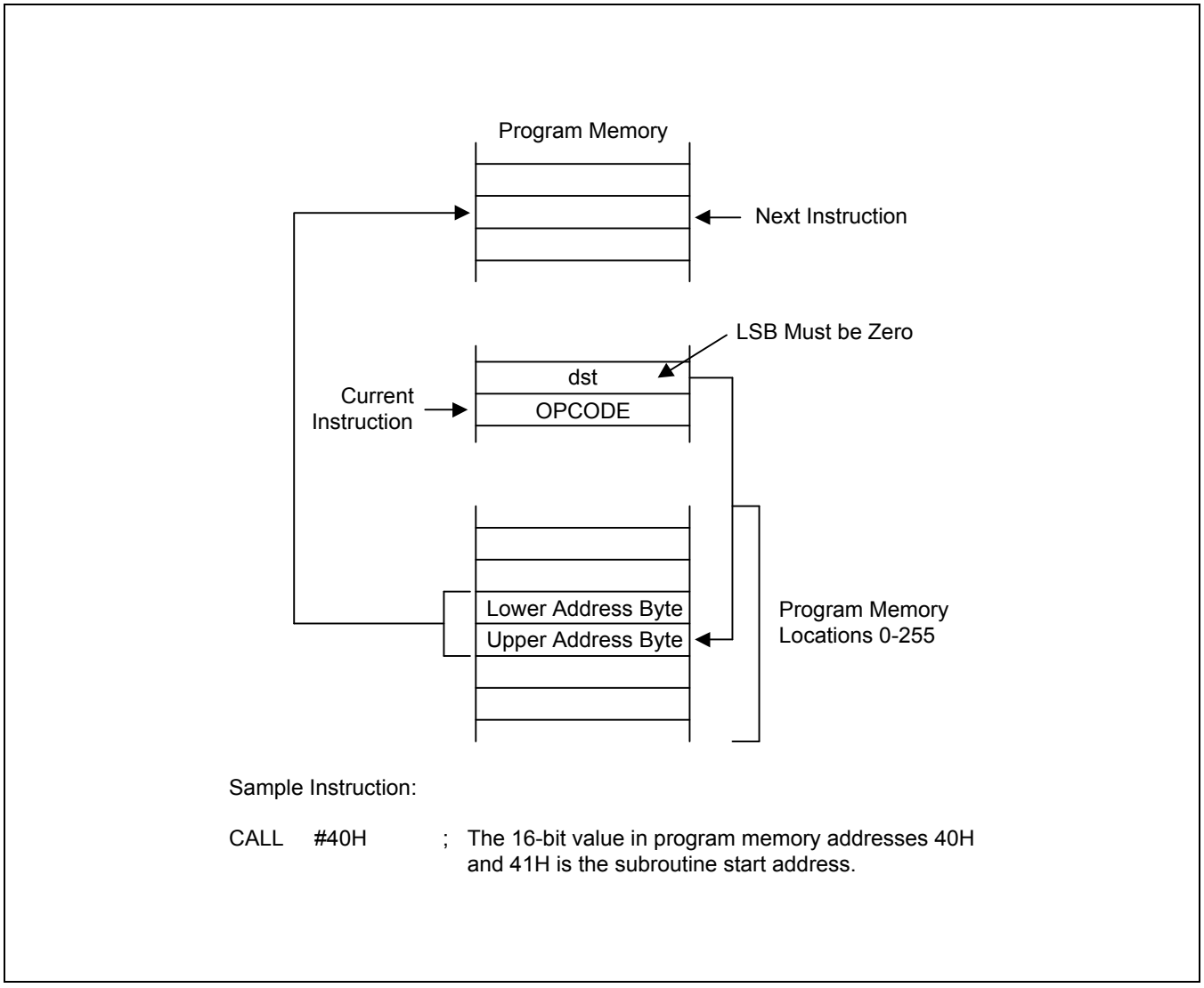


图 3-12 间接寻址模式

3.1.6 相对地址寻址模式(RA)

在相对寻址模式中，指令的调转范围只能在有符号数 -128 到 +127 之间。偏移量加上当前 PC 值，即为下一条要执行指令的地址。在加偏移量之前，PC 中的内容是紧跟着当前指令的后一条指令地址。

程序控制指令用相对寻址模式来实现条件跳转。支持相对寻址模式的指令有： BTJRF, BTJRT, DJNZ, CPIJE, CPIJNE 和 JR 。

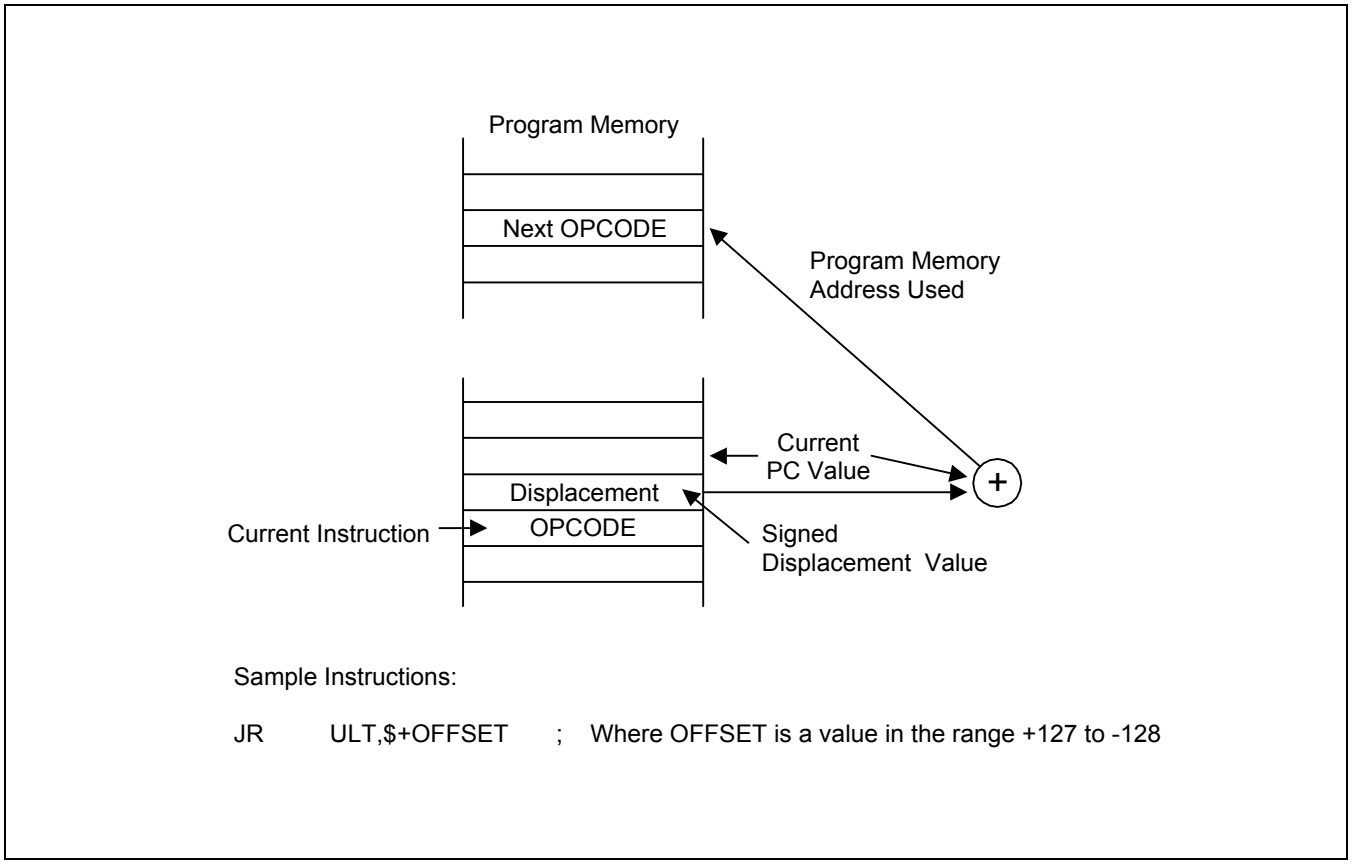


图 3-13 相对寻址

3.1.7 立即数寻址模式(IM)

立即数寻址模式中，操作数本身就包含在指令当中。根据指令的不同，操作数的长度可以是一个字节或一个字。立即数寻址模式常用来将常量赋值给寄存器。

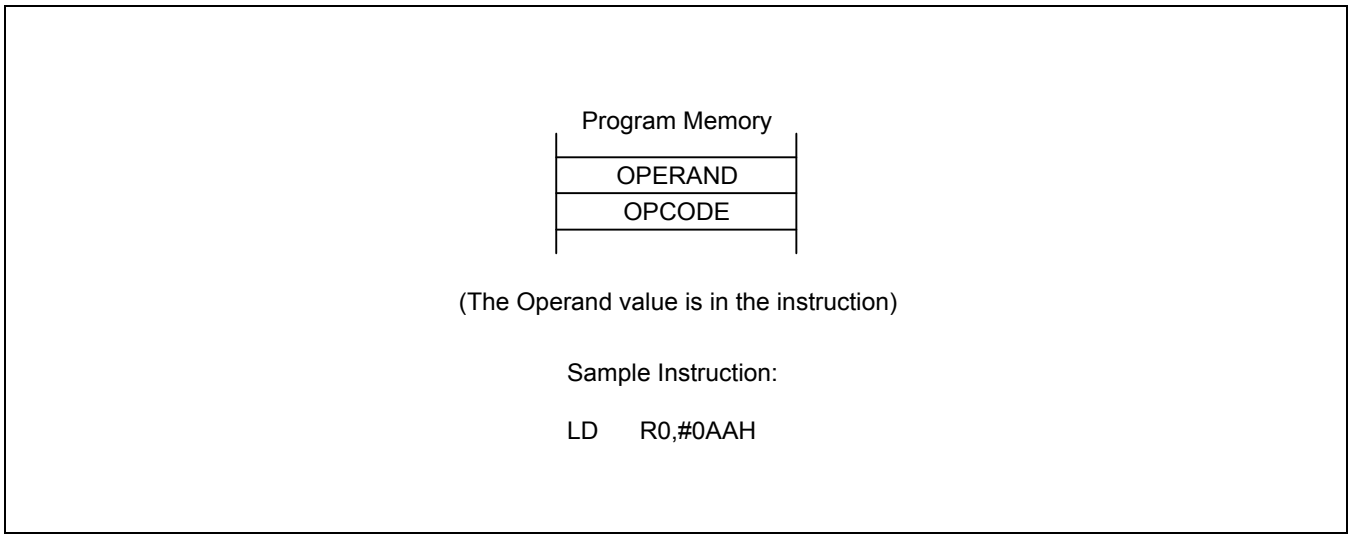


图 3-14 立即数寻址模式

4 控制寄存器

4.1 概述

控制寄存器描述按照寄存器代表符号的字母顺序排列。更多有关控制寄存器的信息在本手册第二部分硬件资源描述中。

表 4-1、*表 4-2* 和 *表 4-3* 列举了 S3C84H5/F84H5 所有映射寄存器的位置和读/写特性。
每个映射寄存器的硬件复位值在第八章“复位和省电模式”中描述。

4.2 REGISTER DESCRIPTION

4.2.1 REGISTER MAP

表 4-1 Set 1 寄存器

寄存器名字	助记标号	十进制地址	十六进制地址	R/W
Timer B 控制寄存器	TBCON	208	D0H	R/W
Timer B 数据寄存器 (高字节)	TBDATAH	209	D1H	R/W
Timer B 数据寄存器 (低字节)	TBDATAL	210	D2H	R/W
Basic timer 控制寄存器	BTCON	211	D3H	R/W
系统时钟控制寄存器	CLKCON	212	D4H	R/W
系统标志寄存器	FLAGS	213	D5H	R/W
寄存器指针 0	RP0	214	D6H	R/W
寄存器指针 1	RP1	215	D7H	R/W
堆栈指针（高字节）	SPH	216	D8H	R/W
堆栈指针（低字节）	SPL	217	D9H	R/W
指令指针（高字节）	IPH	218	DAH	R/W
指令指针（高字节）	IPL	219	DBH	R/W
中断请求寄存器	IRQ	220	DCH	R
中断屏蔽寄存器	IMR	221	DDH	R/W
系统模式寄存器	SYM	222	DEH	R/W
寄存器页指针	PP	223	DFH	R/W

表 4-2 Set 1, Bank 0 寄存器

寄存器名字	助记标号	十进制地址	十六进制地址	R/W
P0 口数据寄存器	P0	224	E0H	R/W
P1 口数据寄存器	P1	225	E1H	R/W
P2 口数据寄存器	P2	226	E2H	R/W
P3 口数据寄存器	P3	227	E3H	R/W
E4H 地址空间未映射				
STOP 控制寄存器	STOPCON	229	E5H	R/W
P0口控制寄存器	P0CON	230	E6H	R/W
E7H 地址空间未映射				
P1口控制寄存器 (高字节)	P1CONH	232	E8H	R/W
P1口控制寄存器 (低字节)	P1CONL	233	E9H	R/W
P1口中断标志位寄存器	P1INTPND	234	EAH	R/W
P1口中断控制寄存器	P1INT	235	EBH	R/W
P2口控制寄存器 (高字节)	P2CONH	236	ECH	R/W
P2口控制寄存器 (低字节)	P2CONL	237	EDH	R/W
EEH 地址空间未映射				
P3口控制寄存器 (低字节)	P3CONL	239	EFH	R/W
F0H,F1H 地址空间未映射				
时钟控制寄存器	OSCCON	242	F2H	R/W
F3H 地址空间未映射				
UART 标志位寄存器	UARTPND	244	F4H	R/W
UART 数据寄存器	UDATA	245	F5H	R/W
UART 控制寄存器	UARTCON	246	F6H	R/W
A/D 转换控制寄存器	ADCON	247	F7H	R/W
A/D 转换数据寄存器（高字节）	ADDATAH	248	F8H	R
A/D 转换数据寄存器（低字节）	ADDATAL	249	F9H	R
P2口上拉使能控制寄存器	P2PUR	250	FAH	R/W
FBH 地址空间未映射				
FCH 地址空间保留				
Basic timer 计数器	BTCNT	253	FDH	R
FEH 地址空间保留				
中断优先级寄存器	IPR	255	FFH	R/W

表 4-3 Set 1, Bank 1 寄存器

寄存器名字	助记标号	十进制地址	十六进制地址	R/W
Timer A, Timer 1 中断标志位寄存器	TINTPND	224	E0H	R/W
Timer A 控制寄存器	TACON	225	E1H	R/W
Timer A 数据寄存器	TADATA	226	E2H	R/W
Timer A 计数器	TACNT	227	E3H	R
Timer 1(0) 数据寄存器 (高字节)	T1DATAH0	228	E4H	R/W
Timer 1(0) 数据寄存器 (低字节)	T1DATAL0	229	E5H	R/W
Timer 1(1) 数据寄存器 (高字节)	T1DATAH1	230	E6H	R/W
Timer 1(1) 数据寄存器 (低字节)	T1DATAL1	231	E7H	R/W
Timer 1(0) 控制寄存器	T1CON0	232	E8H	R/W
Timer 1(1) 控制寄存器	T1CON1	233	E9H	R/W
Timer 1(0) 计数器 (高字节)	T1CNTH0	234	EAH	R
Timer 1(0) 计数器 (低字节)	T1CNTL0	235	EBH	R
Timer 1(1) 计数器 (高字节)	T1CNTH1	236	ECH	R
Timer 1(1) 计数器 (低字节)	T1CNTL1	237	EDH	R
UART 波特率数据寄存器 (高字节)	BRDATAH	238	EEH	R/W
UART 波特率数据寄存器 (低字节)	BRDATAL	239	EFH	R/W
SIO 预分频寄存器	SIOPS	240	F0H	R/W
SIO 数据寄存器	SIODATA	241	F1H	R/W
Serial I/O 控制寄存器	SIOCON	242	F2H	R/W
PWM 数据寄存器 (高)	PWMDATAH	243	F3H	R/W
PWM 数据寄存器 (低)	PWMDATAL	244	F4H	R/W
PWM 控制寄存器	PWMCON	245	F5H	R/W
F6, F7H 地址空间未映射				
Watch timer 控制寄存器	WTCON	248	F8H	R/W
F9H–FFH 地址空间未映射				

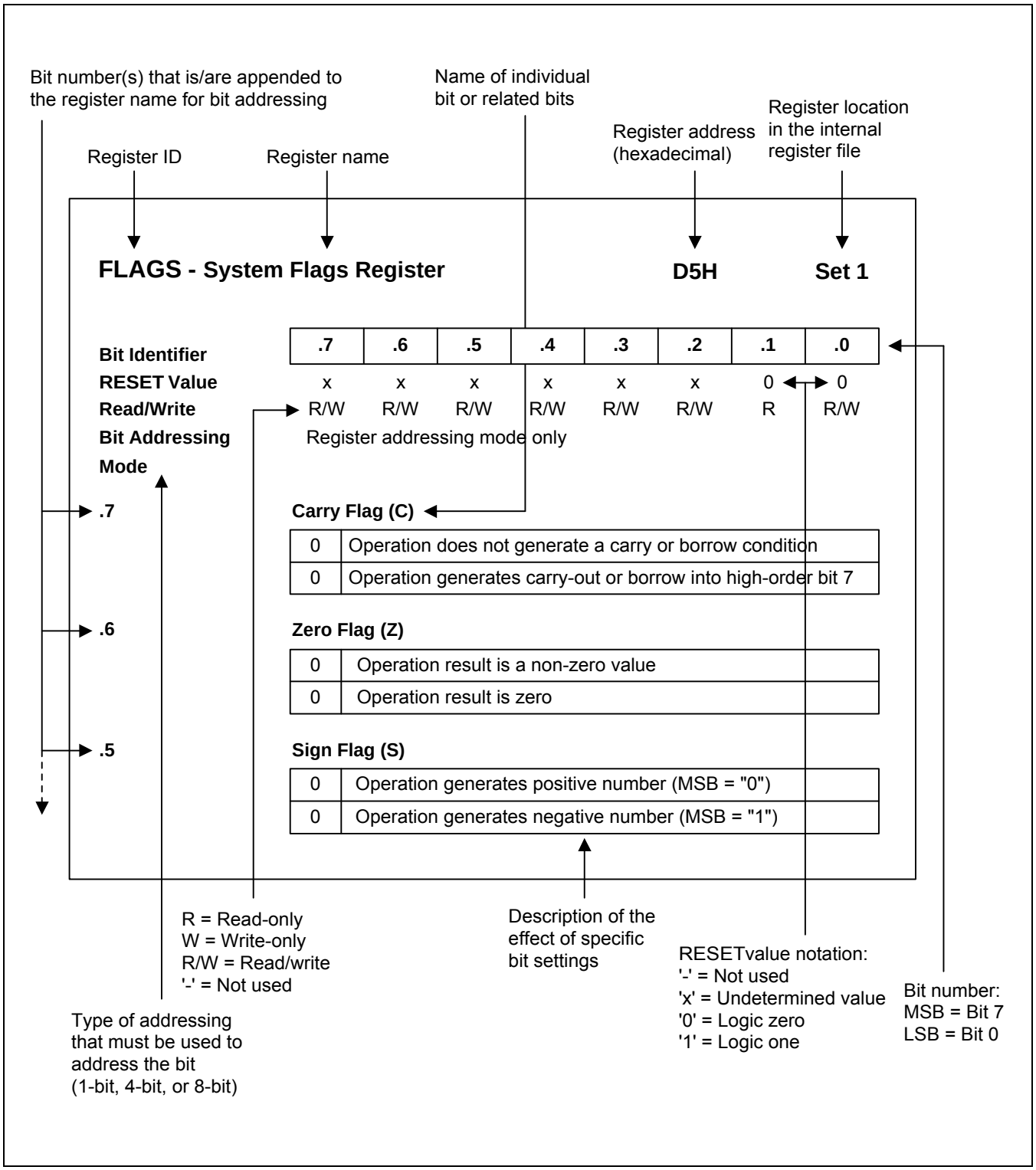

Bit number(s) that is/are appended to the register name for bit addressing
Name of individual bit or related bits
Register location in the internal register file
Register address (hexadecimal)
Register ID
Register name
FLAGS - System Flags Register
D5H
Set 1
Bit Identifier
RESET Value
Read/Write
Bit Addressing Mode
.7 .6 .5 .4 .3 .2 .1 .0
x x x x x x 0 0
R/W R/W R/W R/W R/W R/W R R/W
Register addressing mode only
.7
.6
.5
Carry Flag (C)
0 Operation does not generate a carry or borrow condition
0 Operation generates carry-out or borrow into high-order bit 7
Zero Flag (Z)
0 Operation result is a non-zero value
0 Operation result is zero
Sign Flag (S)
0 Operation generates positive number (MSB = "0")
0 Operation generates negative number (MSB = "1")
R = Read-only
W = Write-only
R/W = Read/write
'-' = Not used
Description of the effect of specific bit settings
RESETvalue notation:
'-' = Not used
'x' = Undetermined value
'0' = Logic zero
'1' = Logic one
Bit number:
MSB = Bit 7
LSB = Bit 0
Type of addressing that must be used to address the bit
(1-bit, 4-bit, or 8-bit)

图 4-1 寄存器描述格式

4.2.1.1 ADCON - A/D 转换控制寄存器 ：F7H Set1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	R/W	R/W	R/W	R	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7

S3C84H5/S3F84H5 不使用(必须始终保持 0)

.6-.4 **A/D 转换输入管脚选择位**

0	0	0	ADC0
0	0	1	ADC1
0	1	0	ADC2
0	1	1	ADC3
1	0	0	ADC4
1	0	1	ADC5
1	1	0	ADC6
1	1	1	ADC7

.3 **转换结束（EOC）状态位**

0	A/D 转换正在进行
1	A/D 转换结束

.2-.1 **时钟源选择位**

0	0	fxx/16 (f_{OSC}=8MHz)
0	1	fxx/8 (f_{OSC}=8MHz)
1	0	fxx/4 (f_{OSC}=8MHz)
1	1	fxx (f_{OSC}=2.5MHz)

.0 **A/D 转换启动位**

0	禁止转换
1	启动转换

4.2.1.2 BTCON - Basic Timer 控制寄存器 ：D3H Set1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.4 **看门狗时钟功能使能位**

1	0	1	0	禁止看门狗时钟功能
其他值				使能看门狗时钟功能

.3-.2 **Basic Timer 输入时钟选择位**

0	0	$f_{OSC}/4096$
0	1	$f_{OSC}/1024$
1	0	$f_{OSC}/128$
1	1	无效设置

.1 **Basic Timer 8 位计数器清 0 控制位**

0	没有作用
1	清除 Basic Timer 的计数值

.0 **Basic Timer 分频器清除位**

0	没有作用
1	清除两个分频器

注释:

1. 当写“1”到 BTCON.1, basic timer 计数器的值清零。之后， BTCON.1 的值也立即自动清零。
2. 当写“1”到 BTCON.0 (或 BTCON.1)时，Basic Timer (或 Basic Timer 分频器) 被清除，之后该位也自动清为零。
3. fxx 位选择的系统时钟 (主时钟或副时钟)。

4.2.1.3 CLKCON - 系统时钟控制寄存器 ：D4H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	-	-	R/W	R/W	-	-	-
寻址模式	仅寄存器寻址							

.7-.5

S3C84H5/S3F84H5 不使用(必须始终保持 0)

.4-.3 **CPU 时钟（系统时钟）选择位** (住釋)

0		0
0		0
1		1
1		1

.2-.0

S3C84H5/S3F84H5不使用(必须始终保持 0)

注释: 复位后，选择最慢时钟(16 分频) 作为系统时钟。可通过向 CLKCON.3 和 CLKCON.4 位写合适的值选择更快的时钟速率。

4.2.1.4 FLAGS - 系统标志寄存器 ：D5H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
寻址方式	仅寄存器寻址模式							

.7 Carry Flag (C)

0	操作没有产生进位或借位
1	操作产生进位或者借位

.6 Zero Flag (Z)

0	操作结果不是“0”
1	操作结果是“0”

.5 Sign Flag (S)

0	操作产生正数（MSB = “0”）
1	操作产生负数（MSB = “1”）

.4 Overflow Flag (V)

0	操作结果在 -128 ~ + 127 之间
1	操作结果不在 -128 ~ + 127 之间，即溢出

.3 Decimal Adjust Flag (D)

0	加操作完成
1	减操作完成

.2 Half-Carry Flag (H)

0	加法操作时第 3 位未产生进位或减法操作时第 3 位未产生借位
1	加法操作时第 3 未产生进位或减法操作时第 3 位未产生借位

.1 Fast Interrupt Status Flag (FIS)

0	中断返回（IRET）正在进行（读此位时）
1	快速中断服务程序正在进行（读此位时）

.0 Bank Address Selection Flag (BA)

0	选择 Bank 0
1	选择 Bank 1

4.2.1.5 IMR - 中断屏蔽寄存器 ：DDH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7 中断级 7（IRQ7）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.6 中断级 6（IRQ6）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.5 中断级 5（IRQ5）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.4 中断级 4（IRQ4）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.3 中断级 3（IRQ3）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.2 中断级 2（IRQ2）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.1 中断级 1（IRQ1）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

.0 中断级 0（IRQ0）使能位

0	禁止（屏蔽）
1	使能（未屏蔽）

注释: 当某个中断级被屏蔽，任何发起的中断请求都不能被 CPU 识别。

4.2.1.6 IPH - 指令指针(高字节) : DAH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.0 **指令指针地址(高字节)**

高字节指令指针指向 16 位指令指针地址的高 8 位（IP15-IP8）。
IP 地址的低字节在IPL 寄存器（地址：DBH）中。

4.2.1.7 IPL - 指令指针(低字节) : DBH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.0 **指令指针地址(低字节)**

低字节指令指针指向 16 位指令指针地址的低 8 位（IP7-IP0）。
IP 地址的高字节在IPH 寄存器（地址：DAH）中。

4.2.1.8 IPR - 中断优先级寄存器 ：FFH Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7, .4, and .1 中断 A、B 和 C 组优先级控制位

0	0	0	组优先级未定义
0	0	1	B > C > A
0	1	0	A > B > C
0	1	1	B > A > C
1	0	0	C > A > B
1	0	1	C > B > A
1	1	0	A > C > B
1	1	1	组优先级未定义

.6 中断 C 子分组优先级控制位

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

.5 中断 C 组优先级控制位

0	IRQ5 > (IRQ6, IRQ7)
1	(IRQ6, IRQ7) > IRQ5

.3 中断 B 子分组优先级控制位

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

.2 中断 B 组优先级控制位

0	IRQ2 > (IRQ3, IRQ4)
1	(IRQ3, IRQ4) > IRQ2

.0 中断 A 组优先级控制位

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

4.2.1.9 IRQ - 中断请求寄存器 ：DCH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R	R	R	R	R	R	R	R
寻址方式	仅寄存器寻址模式							

.7 中断级 7 （IRQ7）请求标志位

0	没有中断
1	标志位置起

.6 中断级 6（IRQ6）请求标志位

0	没有中断
1	标志位置起

.5 中断级 5（IRQ5）请求标志位

0	没有中断
1	标志位置起

.4 中断级 4（IRQ4）请求标志位

0	没有中断
1	标志位置起

.3 中断级 3（IRQ3）请求标志位

0	没有中断
1	标志位置起

.2 中断级 2（IRQ2）请求标志位

0	没有中断
1	标志位置起

.1 中断级 1（IRQ1）请求标志位

0	没有中断
1	标志位置起

.0 中断级 0（IRQ0）请求标志位

0	没有中断
1	标志位置起

4.2.1.10 OSCCON - 时钟控制寄存器 : F2H Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	-	-	R/W	R/W	R/W	-	R/W
寻址方式	仅寄存器寻址模式							

.7-.4

S3C84H5/F84H5 不使用(必须始终保持 0)

.3 **主时钟控制位**

0	主时钟运行
1	主时钟停止

.2 **副时钟控制位**

0	副时钟运行
1	副时钟停止

.1

S3C84H5/F84H5 不使用(必须始终保持 0)

.0 **系统时钟选择位**

0	选择主时钟
1	选择副时钟

4.2.1.11 P0CON - Port 0 控制寄存器(高字节) : E6H Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.6 **P0.3/AD3**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	AD3 输入

.5-.4 **P0.2/AD2**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	AD2 输入

.3-.2 **P0.1/ AD1**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	AD1 输入

.1-.0 **P0.0/ AD0**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	AD0 输入

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 *编程实例 9-1*)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.12 P1CONH - Port 0 控制寄存器(高字节) : E8H Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	-	-	-	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.4

S3C84H5/F84H5不使用(必须始终保持 0)

.3-.2 **P1.5/T1CAP1/AD6**

0	0	输入模式(T1CAP1 输入)
0	1	输入模式，带上拉电阻(T1CAP1 输入)
1	0	推挽式输出
1	1	AD6

.1-.0 **P1.4/T1CK1/AD5**

0	0	输入模式(T1CK1 输入)
0	1	输入模式，带上拉电阻(T1CK1 输入)
1	0	推挽式输出
1	1	AD5

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 ***编程实例 9-1***)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.13 P1CONL - Port 1 控制寄存器(低字节) : E9H Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.6 P1.3/T1OUT1/INT3

0	0	输入模式; 中断输入(INT3)
0	1	输入模式，带上拉电阻; 中断输入 (INT3)
1	0	推挽式输出
1	1	T1OUT1 mode

.5-.4 P1.2/TACAP/INT2

0	0	输入模式; 中断输入(INT2); TACAP
0	1	输入模式，带上拉电阻; 中断输入(INT2) ; TACAP
1	0	推挽式输出
1	1	不使用

.3-.2 P1.1/TACK/BUZ/INT1

0	0	输入模式; 中断输入(INT1); TACK
0	1	输入模式，带上拉电阻; 中断输入(INT1); TACK
1	0	推挽式输出
1	1	BUZ 输出

.1-.0 P1.0/TAOUT/INT0

0	0	输入模式; 中断输入(INT0)
0	1	输入模式，带上拉电阻; 中断输入(INT0)
1	0	推挽式输出
1	1	TAOUT

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 *编程实例 9-1*)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.14 P1INTPND - Port 1 中断标志位寄存器 ：EAH Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	-	-	-	-	0	0	0	0
读/写	-	-	-	-	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.4

S3C84H5/F84H5不使用

.3 **P1.3/INT3 中断标志位**

0	没有中断， 写 0 时清除中断标志
1	中断挂起

.2 **P1.2/INT2中断标志位**

0	没有中断， 写 0 时清除中断标志
1	中断挂起

.1 **P1.1/INT1中断标志位**

0	没有中断， 写 0 时清除中断标志
1	中断挂起

.0 **P1.0/INT0中断标志位**

0	没有中断， 写 0 时清除中断标志
1	中断挂起

4.2.1.15 P1INT - Port 1 中断使能 ：EBH Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.6 **P1.3 中断使能/禁止选择位**

0	X	中断禁止
1	0	中断使能，下降沿触发
1	1	中断使能，上升沿触发

.5-.4 **P1.2 中断使能/禁止选择位**

0	X	中断禁止
1	0	中断使能，下降沿触发
1	1	中断使能，上升沿触发

.3-.3 **P1.1 中断使能/禁止选择位**

0	X	中断禁止
1	0	中断使能，下降沿触发
1	1	中断使能，上升沿触发

1-.0 **P1.0 中断使能/禁止选择位**

0	X	中断禁止
1	0	中断使能，下降沿触发
1	1	中断使能，上升沿触发

4.2.1.16 P2CONH - Port 2 控制寄存器(高字节) : ECH Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.6 **P2.7/ TxD**

0	0	输入模式
0	1	不使用
1	0	推挽式输出
1	1	TxD 输出

.5-.4 **P2.6/ /RxD**

0	0	输入模式; RxD 输入
0	1	不使用
1	0	推挽式输出
1	1	RxD 输出

.3-.2 **P2.5/ SCK**

0	0	输入模式; SCK输入
0	1	不使用
1	0	推挽式输出
1	1	SCK 输出

.1-.0 **P2.4/ SO**

0	0	输入模式
0	1	不使用
1	0	推挽式输出
1	1	SO 输出

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 *编程实例 9-1*)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.17 P2CONL - Port 2 控制寄存器(低字节) : EDH Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址模式	仅寄存器寻址							

.7-.6 P2.3/AD7/SI

0	0	输入模式; SI 输入
0	1	不使用
1	0	推挽式输出
1	1	AD7

.5-.4 P2.2/AD4/T1OUT0

0	0	输入模式;
0	1	T1OUT0
1	0	推挽式输出
1	1	AD4

.3-.2 P2.1/PWM/T1CAP0

0	0	输入模式; T1CAP0 输入
0	1	不使用
1	0	推挽式输出
1	1	PWM 模式

.1-.0 P2.0/TBPWM/T1CK0

0	0	输入模式; T1CK0 输入
0	1	T1CK0 输入
1	0	推挽式输出
1	1	TBPWM

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 ***编程实例 9-1***)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.18 P2PUR - Port 2 上拉电阻控制寄存器 ：FAH Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	1	1	1	1	1	1	1	1
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7 **P2.7 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.6 **P2.6 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.5 **P2.5 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.4 **P1.4上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.3 **P2.3 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.2 **P2.2 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.1 **P2.1 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

.0 **P2.0 上拉电阻使能/禁止**

0	上拉电阻禁止
1	上拉电阻使能

4.2.1.19 P3CONL - Port 3 控制寄存器(低字节) : EFH Set 1, Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.6 **P3.3/ SEG7**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	开漏输出

.5-.4 **P3.2/ SEG6**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	开漏输出

.3-.2 **P3.0/ SEG5**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	开漏输出

.1-.0 **P3.0/ SEG4**

0	0	输入模式
0	1	输入模式，带上拉电阻
1	0	推挽式输出
1	1	开漏输出

注释: 调试模式时，必须包含三句额外命令来初始化端口。
如果漏掉这几句命令，端口操作会不正确。 (参考页 *编程实例 9-1*)
程序写好以后，编译之前，需去掉这几句命令。

```
.ORG    100H
SB1                     ; 调试时的额外命令
LD      0F7H,#5FH       ; 调试时的额外命令
SB0                     ; 调试时的额外命令
```

4.2.1.20 PP - 寄存器页指针 ：DFH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址							

.7-.4 **目的寄存器页选择位**

0	0	0	0	目的寄存器：Page 0
其他值				无关

.3-.0 **源寄存器页选择位**

0	0	0	0	源寄存器：Page 0
其他值				无关

注释: S3C84H5/F84H5 内部寄存器卷被分为 1 页 (Page 0).
Page 0 为通用寄存器卷和数据寄存器。

4.2.1.21 PWMCON - PWM 控制寄存器 ：F5H Set 1, Bank 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	-	0	0	0	0	0
读/写	R/W	R/W	-	R/W	R/W	R/W	R/W	R/W

.7-.6 **PWM 输入时钟选择位**

0	0	$f_{OSC}/64$
0	1	$f_{OSC}/8$
1	0	$f_{OSC}/2$
1	1	$f_{OSC}/1$

.5

S3C84H5/F84H5不使用

.4 **PWMDATA 重新加载时间间隔选择位**

0	10位计数器计满后重新加载
1	8位计数器计满后重新加载

.3 **PWM 计数器清零**

0	没有作用
1	清除 8 位计数器（写此位时）

.2 **PWM 计数器使能位**

0	停止计数器
1	启动计数器 (重新计数)

.1 **PWM 溢出中断使能位(8位溢出)**

0	禁止中断
1	使能中断

.0 **PWM 溢出中断标志位**

0	没有中断 (读此位时)
0	清除中断标志位 (写此位时)
1	中断挂起 (读此位时)

注释: PWMCON.3 不会自动清零。清除中断标志位时，请一定谨慎。(参考 *编程实例 13-1*).

4.2.1.22 RP0 - 寄存器指针 0 : D6H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	1	1	0	0	0	-	-	-
读/写	R/W	R/W	R/W	R/W	R/W	-	-	-
寻址方式	仅寄存器寻址模式							

.7-.3 **寄存器指针 0 地址值**

寄存器指针 0 可以单独指向寄存器卷中的某个 256 字节工作寄存器区域。 通过使用寄存器指针 RP0 和 RP1 同时选择 2 个 8 字节寄存器组作为有效的工作寄存器空间。复位后，RP0 指向寄存器 Set 1 的 C0H 地址，选择从C0H 到 C7H 的 8 位工作寄存器。

.2-.0

S3C84H5/F84H5不使用

4.2.1.23 RP1 - Register Pointer 1 : D7H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	1	1	0	0	1	-	-	-
读/写	R/W	R/W	R/W	R/W	R/W	-	-	-
寻址方式	仅寄存器寻址模式							

.7-.3 **寄存器指针 1 地址值**

寄存器指针 1 可以单独指向寄存器卷中的某个 256 字节工作寄存器区域。通过使用寄存器指针 RP0 和 RP1 同时选择 2 个 8 字节寄存器组作为有效的工作寄存器空间。复位后，RP1 指向寄存器 Set 1 的 C8H 地址，选择从C8H 到 CFH 的 8 位工作寄存器。

.2-.0

S3C84H5/F84H5不使用

4.2.1.24 SIOCON - SIO 控制寄存器 : F2H Set 1, Bank1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7 **SIO 时钟选择位**

0	内部时钟 (系统预分频时钟)
1	外部时钟 (SCK)

.6 **数据方向控制位**

0	MSB优先
1	LSB优先

.5 **SIO 模式选择位**

0	只接受模式
1	发送/接收模式

.4 **时钟边沿选择位**

0	下降沿发送，上升沿接受。
1	下降沿接受，上升沿发送。

.3 **SIO 计数器清零和启动传输位**

0	没作用
1	清零3位计数器，开始传输

.2 **SIO 传输使能位**

0	禁止移位和计数器
1	使能移位和计数器

.1 **SIO 中断使能位**

0	禁止SIO中断
1	使能SIO中断

.0 **SIO 中断标志位**

0	没有中断
1	中断挂起 (写此位时清零标志位)

4.2.1.25 SIOPS - SIO 预分频寄存器 : F0H Set 1, Bank1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.0

波特率 = 输入时钟 (fxx)/[(SIOPS + 1) ×4] 或者SCK 输入的时钟

4.2.1.26 SPH - 堆栈指针(高字节) : D8H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.0 **堆栈指针地址(高字节）**

高字节堆栈指针的值是 16 位堆栈指针地址（SP15–SP8）的高 8 位。低字节堆栈指针的值位于寄存器 SPL（D9H）中。复位后，堆栈指针（SP）的值不确定。

4.2.1.27 SPL - 堆栈指针(低字节) : D9H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	x	x	x	x	x	x	x	x
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.0 **堆栈指针地址(高字节）**

高字节堆栈指针的值是 16 位堆栈指针地址（SP15–SP8）的高 8 位。低字节堆栈指针的值位于寄存器 SPL（D9H）中。复位后，堆栈指针（SP）的值不确定。

4.2.1.28 STPCON - STOP 控制寄存器 : D1H Set 1，Bank0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.0 **STOP 控制位**

1 0 1 0 0 1 0 1	使能使用 STOP 指令
其他值	禁止使用 STOP 指令

注释: 在执行 STOP 指令前，需要将 STPCON 寄存器设置为 “10100101B”，否则，STOP 指令不会被执行。

4.2.1.29 SYM - System Mode Register : DEH Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	x	x	x	0	0
读/写	-	-	-	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.5

不使用，但必须保持0

.4-.2 **快速中断级选择位**

0	0	0	IRQ0
0	0	1	IRQ1
0	1	0	IRG2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

.1 **快速中断使能位**

0	快速中断禁止
1	快速中断使能

.0 **全局中断使能位** (NOTE)

0	快速中断禁止
1	快速中断使能

注释: 复位后，必须通过执行 EI 指令使能全局中断处理（而不是往 SYM.0 写“1”）

4.2.1.30 T0CON - Timer 1(0) 控制寄存器 ：E8H Set 1, Bank 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.5 **Timer 1(0) 输入时钟选择位**

0	0	0	fxx/1024
0	0	1	fxx/256
0	1	0	fxx/64
0	1	1	fxx/8
1	0	0	fxx
1	0	1	外部时钟（下降沿）
1	1	0	外部时钟（上升沿）
1	1	1	计数器停止

.4-.3 **Timer 1(0) 工作模式选择位**

0	0	Interval（定时）模式
0	1	Capture（捕获）模式 （上升沿时捕获，允许溢出发生）
1	0	Capture（捕获）模式 （下降沿时捕获，允许溢出发生）
1	1	PWM 模式

.2 **Timer 1(0) 计数器清零位**

0	没有作用
1	清除 Timer 0 的计数器（自动清除位）

.1 **Timer 1(0) 匹配/捕获中断使能位**

0	禁止中断
1	使能中断

.0 **Timer 1(0) 溢出中断使能位**

0	禁止溢出中断
1	使能溢出中断

4.2.1.31 T1CON - Timer 1(1) 控制寄存器 ：E9H Set 1, Bank 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.5 **Timer 1(1) 输入时钟选择位**

0	0	0	fxx/1024
0	0	1	fxx/256
0	1	0	fxx/64
0	1	1	fxx/8
1	0	0	fxx
1	0	1	外部时钟（下降沿）
1	1	0	外部时钟（上升沿）
1	1	1	计数器停止

.4-.3 **Timer 1(1) 工作模式选择位**

0	0	Interval（定时）模式
0	1	Capture（捕获）模式 （上升沿时捕获，允许溢出发生）
1	0	Capture（捕获）模式 （下降沿时捕获，允许溢出发生）
1	1	PWM 模式

.2 **Timer 1(1) 计数器清零位**

0	没有作用
1	清除 Timer 0 的计数器（自动清除位）

.1 **Timer 1(1) 匹配/捕获中断使能位**

0	禁止中断
1	使能中断

.0 **Timer 1(1) 溢出中断使能位**

0	禁止溢出中断
1	使能溢出中断

4.2.1.32 TACON - Timer A 控制寄存器 ：E1H Set 1, Bank 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.6 **Timer A 输入时钟选择位**

0	0	fxx/1024
0	1	fxx/256
1	0	fxx/64
1	1	外部时钟(TACK)

.5-.4 **Timer A 工作模式选择位**

0	0	Interval(定时) 模式(TAOUT 模式)
0	1	Capture(捕获) 模式(上升沿时捕获，计数器计数，允许溢出发生)
1	0	Capture(捕获) 模式(下降沿时捕获，计数器计数，允许溢出发生)
1	1	PWM 模式(允许溢出中断发生)

.3 **Timer A 计数器清零位**

0	没有作用
1	清除 Timer A 计数器(计数器清零后，此位变为 0)

.2 **Timer A 溢出中断使能位**

0	禁止中断
1	使能中断

.1 **Timer A 匹配/捕获中断使能位**

0	禁止中断
1	使能中断

.0 **Timer A 开始/停止位**

0	停止 Timer A
1	开始 Timer A

4.2.1.33 TBCON - Timer B 控制寄存器: D0H Set 1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.6 Timer B 输入时钟选择位

0	0	fxx/4
0	1	fxx/8
1	0	fxx/64
1	1	fxx/256

.5-.4 Timer B 中断时间选择位

0	0	低字节数据经过的时间
0	1	高字节数据经过的时间
1	0	低字节数据和高字节数据经过的时间
1	1	不使用

.3 Timer B 中断使能位

0	禁止中断
1	使能中断

.2 Timer B 启动/停止位

0	停止 timer B
1	启动 timer B

.1 Timer B 模式选择位

0	One-shot mode
1	Repeating mode

.0 Timer B 输出触发器控制位

0	T-FF 为低
1	T-FF 为高

注释: fxx 为所选择的系统时钟。

4.2.1.34 TINTPND - Timer A, Timer 1 中断标志位寄存器 ：E0H Set 1, Bank1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	-	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7-.6 不使用 (必须保持0)没有中断（写 0 时清除中断标志）

.5 Timer 1(1) 溢出中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

.4 Timer 1(1) 0 匹配/捕获中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

.3 Timer 1(0) 溢出中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

.2 Timer 1(0) 匹配/捕获中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

.1 Timer A 溢出中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

.0 Timer A 匹配/捕获中断标志位

0	没有中断
0	写 0 时清除中断标志
1	中断挂起

4.2.1.35 UARTCON - UART 控制寄存器 : F6H Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7-.6 工作模式和波特率选择位

0	0	模式 0 = 移位寄存器，波特率为 [fxx/(16 × (16bit BRDATA + 1))]
0	1	模式 1 = 8位 UART，波特率为 [fxx/(16 × (BRDATA1 + 1))]
1	X	模式 2 = 9位 UART，波特率为 [fxx/(16 × (BRDATA1 + 1))]

.5 多芯片通讯 (1) 使能位（仅对模式2）

0	禁止
1	使能

.4 串行数据接收使能位

0	禁止
1	使能

.3

如果奇偶校验禁止模式(PEN = 0)，模式2发送第9位数据("0" or "1").
如果奇偶校验使能模式(PEN = 1)，模式2发送数据的奇偶校验位。
0 = 发送数据偶校验
1 = 发送数据奇校验

.2

如果奇偶校验禁止模式 (PEN = 0)，模式2接收到第9位数据("0" or "1").
如果奇偶校验使能模式(PEN = 1)，模式2收到数据的奇偶校验位。
0 = 接收数据偶校验
1 = 接收数据奇校验
接收数据奇偶校验之后，校验结果保存到UARTPND 寄存器的RPE位。

.1 接收中断使能位

0	禁止接收中断
1	使能接收中断

.0 发送中断使能位

0	禁止发送中断
1	使能发送中断

注释:

1. 在模式2，写"1"到MCE 位(UARTCON.5) ，如果接受的第9位数据为"0"，则不会产生接收中断。在模式1，如果MCE = "1" 并且没有收到有效的停止位，接收中断也不会发生。在模式 0, MCE 位(UARTCON.5) 应该为 "0"。
2. 8位和9位UART模式没有数据收发的启动停止位。
3. 奇偶校验位 PEN位于地址为F4H, bank 0的寄存器 UARTPND 里。
4. 只有9位UART模式(模式 2)才有奇偶校验使能和错误校验功能。

4.2.1.36 UARTPND - UART 标志位和奇偶控制位: F4H Set 1, Bank 0

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	-	-	R/W	R/W	-	-	R/W	R/W

.7-.6 不使用 (必须保持0)

.5 **UART 奇偶校验使能位 (PEN)**

0	禁止
1	使能

.4 **UART 接受奇偶校验报错位 (RPE)**

0	没有错误
1	奇偶校验错误

.3-.2 不使用 (必须保持0)

.1 **UART 接收中断标志位**

0	没有中断
0	清除标志位t (当写此位时)
1	中断挂起

.0 **UART 发送中断标志位**

0	没有中断
0	清除标志位t (当写此位时)
1	中断挂起

注释:

1. 要清零数据发送或者接受中断标志位，必须写 "0" 到相应的标志位。
2. 操作UARTPND寄存器时，为了避免编程错误，建议使用 load 指令 (除 LDB外)。
3. 只有9位UART模式(模式 2)才有奇偶校验使能和错误校验功能。
4. 当第八位接受的数据发生移位时，将刷新奇偶校验报错位 (RPE)。

4.2.1.37 WTCON - Watch Timer 控制寄存器 : F8H Set 1, Bank1

位	.7	.6	.5	.4	.3	.2	.1	.0
复位值	0	0	0	0	0	0	0	0
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
寻址方式	仅寄存器寻址模式							

.7 **Watch Timer 时钟选择位**

0	主时钟256 分频(fxx/256)
1	副时钟 (fxt)

.6 **Watch Timer 中断使能位**

0	禁止watch timer 中断
1	使能watch timer 中断

.5-.4 **蜂鸣器频率选择位**

0	0	蜂鸣器输出信号 (BZOUT) ， 0.5 kHz
0	1	蜂鸣器输出信号 (BZOUT) ， 1 kHz
1	0	蜂鸣器输出信号 (BZOUT) ， 2 kHz
1	1	蜂鸣器输出信号 (BZOUT) ， 4 kHz

.3-.2 **Watch Timer 速度选择位**

0	0	0.5 s 时间间隔
0	1	0.25 s 时间间隔
1	0	0.125 s 时间间隔
1	1	1.955 ms 时间间隔

.1 **Watch Timer 使能位**

0	禁止 watch timer; 清除分频电路
1	使能 watch timer

.0 **Watch Timer 中断标志位**

0	没有中断
1	清除标志位 (当写此位时)
1	中断挂起

5 中断结构

5.1 概述

S3C8- 系列的中断结构由三个部分组成：中断级，中断向量和中断源。SAM8RC 的 CPU 最多可以识别 8 个中断级和 128 个中断向量。当多个中断向量同属于一个中断级时，这几个向量的优先级由向量地址的前后决定。从芯片设计的角度来看，单个或多个中断源可以共用同一个中断向量地址。

5.1.1 中断级(LEVELS)

中断级是中断优先级分配和识别的主要操作单元。所有的外围和 I/O 模块都可以发出中断请求。 换句话说，所有外围和 I/O 模块的操作都可以是中断驱动的（interrupt-driven）。一共有 8 个可能的中断级：IRQ0–IRQ7，也被称为中断级 0 ~ 中断级 7。每个中断级直接与中断请求序号(IRQn)关联。但是每款芯片中，中断级的数量可以相同也可以不同。S3C84H5/F84H5的中断结构中就有 8 个中断级。

中断级序号 0~7 仅仅是一个标号，并不直接表示优先级。优先级由中断优先级寄存器 IPR 中的设置决定。 IPR 中，中断组和子分组结构更细化了各个中断级之间的优先级定义。

5.1.2 中断向量(VECTORS)

每一个中断级中可以包括一个或多个中断向量，又或者没有任何中断向量。SAM8RC 的 CPU 结构最多支持 128 个中断向量（但 S3F8- 系列产品中实际用到的向量个数往往远小于 128）。当多个中断向量同属于一个优先级时，这几个向量的优先级由向量地址的前后决定。S3C84H5/F84H5 里有 16 个中断向量。

5.1.3 中断源(SOURCES)

中断源可以是任何产生中断的外围，也可以是外部管脚或者计数器的溢出等。多个中断源可以共用一个中断向量地址。S3C84H5/F84H5 的中断结构中有 16 个中断向量，16 个中断源，这意味着每个中断源都有自己的 中断向量

当中断服务程序开始之后，必须清除相应的中断标志位，如果不是硬件自动清零，就必须软件手动清零。标志位类型是由中断源的标志位产生/清除机制决定的。

5.1.4 中断类型

之前介绍的 S3C8- 系列中断结构的三个组成部分 — 中断级，中断向量和中断源 — 充分使用了芯片的中断逻辑，一起决定了芯片的中断结构。中断结构一共有三种可能的组合，分别称为 Type1，2，3。 三者之间的区别在于分配给每个中断级的中断向量和中断源的个数不同。（*图 5-1*）

- Type 1: 一个中断级 (IRQn) + 一个中断向量 (V_1) + 一个中断源 (S_1)
- Type 2: 一个中断级(IRQn) + 一个中断向量 (V_1) + 多个中断源(S_1 - S_n)
- Type 3: 一个中断级(IRQn) + 多个中断向量(V_1 - V_n) + 多个中断源 (S_1 - S_n , S_{n+1} - S_{n+m})

S3C84H5/F84H5 只使用了上述两种 Type 。

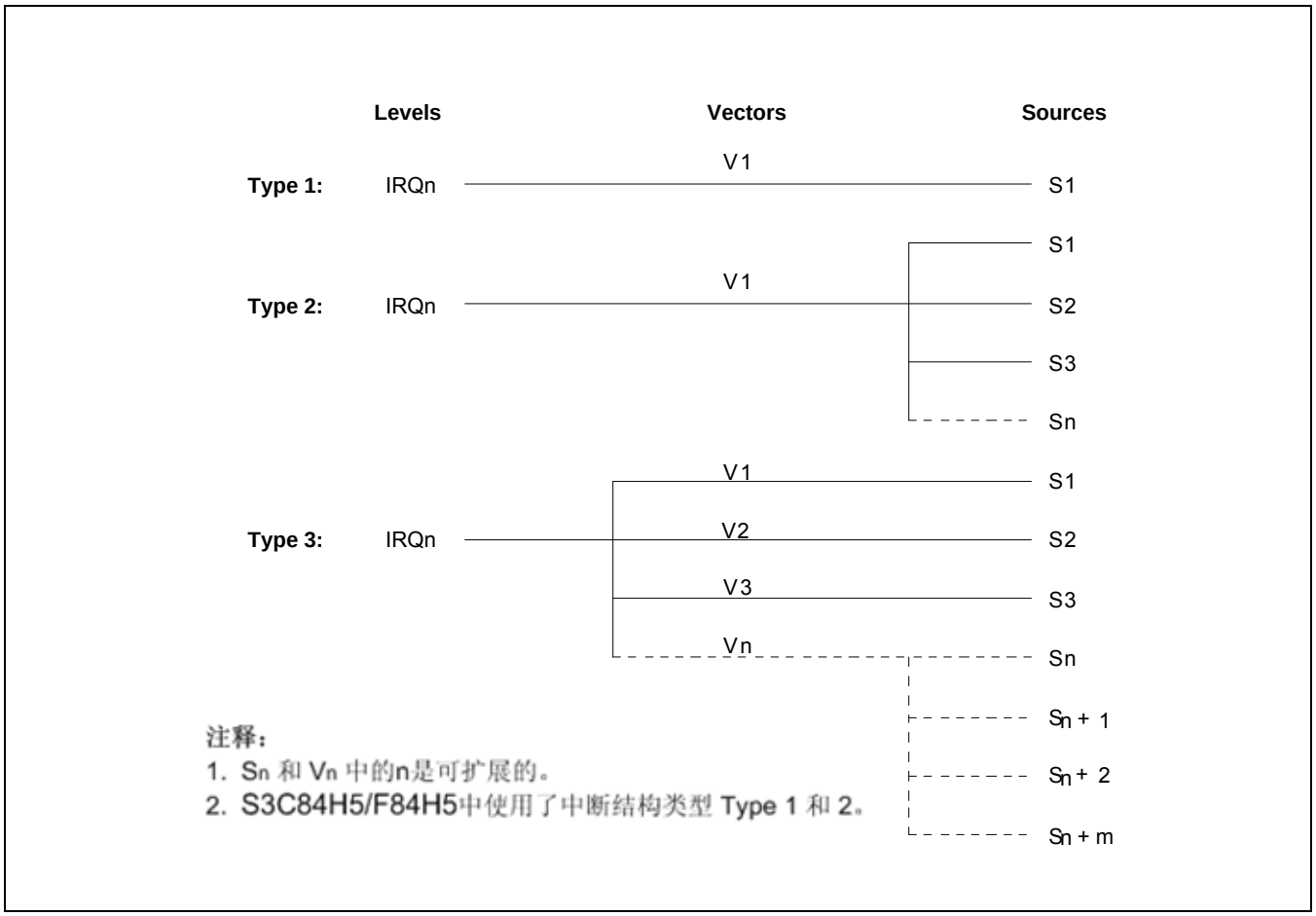


图 5-1 S3C8-系列的中断类型

5.1.5 S3C84H5/F84H5 中断结构

S3C84H5/F84H5 支持 16 个中断源，每一个中断源都有唯一的中断向量地址。在这款芯片中，CPU 可以响应 8 个中断级（*图 5-2*）。

多个中断级同时发出中断请求时，中断优先级寄存器（IPR）中的值决定了哪一个竞争中的中断级先被 CPU响应。如果同一个中断级中的多个中断源同时发出中断请求，中断向量地址最小的那个中断源拥有最高优先级，首先被响应（同一中断级内中断源优先级顺序由硬件决定）。

当 CPU 响应某个中断请求以后，中断处理就开始了。此时所有的其他中断都处于禁止状态，PC 和 FLAGS 的值被压入栈。从中断向量地址中获取中断服务程序的起始地址（16 位地址由中断向量地址处的 8 位和向量地址之后的 8位数据一起构成），程序跳转而后开始执行中断服务程序。

Levels	Vectors	Sources	Reset(Clear)
IRQ0	BEH	Timer B underflow	H/W
IRQ1	C0H	Timer A match/capture	H/W, S/W
	C2H	Timer A overflow	H/W, S/W
IRQ2	C4H	Timer 1(0) match/capture	H/W, S/W
	C6H	Timer 1(0) overflow	H/W, S/W
	C8H	Timer 1(1) match/capture	H/W, S/W
	CAH	Timer 1(1) overflow	H/W, S/W
IRQ3	CEH	P1.0 external interrupt	S/W
	D0H	P1.1 external interrupt	S/W
	D2H	P1.2 external interrupt	S/W
	D4H	P1.3 external interrupt	S/W
IRQ4	D6H	Watch timer	S/W
IRQ5	D8H	SIO receive/transmit	S/W
IRQ6	DAH	PWM overflow interrupt	S/W
IRQ7	DCH	UART data receive	S/W
	DEH	UART data transmit	S/W

注释:

1. 在一个给定的中断级中，向量地址越小，中断优先级越高。
   例如, IRQ5中，DCH 比 DEH 优先级高。同一优先级中各个中断源的优先级顺序由芯片设计者决定。
2. 外部中断由上升沿或下降沿触发，触发方式由相应的控制寄存器决定。

图 5-2 S3C84I8X/F84I8X/C84I9X/F84I9X中断结构

5.1.5.1 中断向量地址

S3C84H5/F84H5所有的中断向量地址都位于16K字节程序存储空间0H–3FFFH的向量地址区(*图 5-3*)。

没有被用作中断向量的地址空间可以当作普通的程序存储空间来用。此时要小心千万不能与中断向量区交叠。

默认的程序复位地址为 ROM 区 0100H。

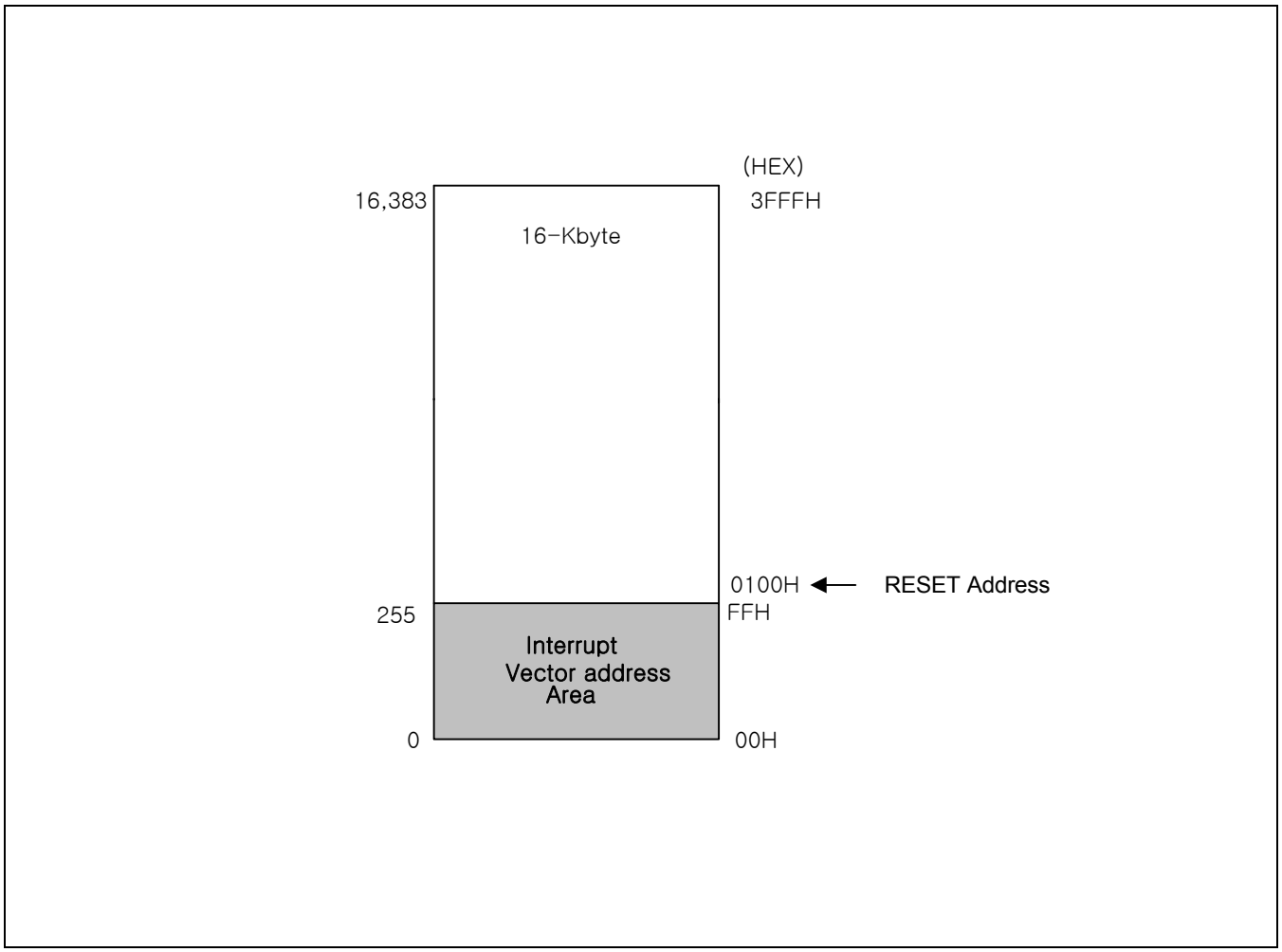


图 5-3 ROM 中断向量地址区

表 5-1 中断向量

向量地址		中断源	要求		复位/清零	
十进制	十进制		中断级	优先级	H/W	S/W
256	100H	Basic timer (WDT) 溢出	nRESET	-	√	
222	DEH	UART 发送中断	IRQ7	1		√
220	DCH	UART 接收中断		0		√
218	DAH	PWM 溢出中断	IRQ6	-		√
216	D8H	SIO 接收 / 发送	IRQ5	-		√
214	D6H	Watch timer 中断	IRQ4	-		√
212	D4H	P1.3 外部中断	IRQ3	3		√
210	D2H	P1.2 外部中断		2		√
208	D0H	P1.1 外部中断		1		√
206	CEH	P1.0 外部中断		0		√
202	CAH	Timer 1(1) 溢出	IRQ2	3	√	√
200	C8H	Timer 1(1) 匹配/捕获		2	√	√
198	C6H	Timer 1(0) 溢出		1	√	√
196	C4H	Timer 1(0) 匹配/捕获		0	√	√
194	C2H	Timer A溢出	IRQ1	1	√	√
192	C0H	Timer A 匹配/捕获		0	√	√
190	BEH	Timer B 溢出	IRQ0	-	√	

注释:

1. 中断优先级是逆序排列的。"0" 是优先级最高， "1" 次之，以此排列。
2. 同一个中断级里有多个中断源，地址低的优先级高于地址高的。给定中断级的优先级由硬件决定。

5.1.5.2 使能/禁止中断的指令(EI, DI)

执行使能中断指令 (EI) 使能全局中断结构。在随后的操作中，可以执行 DI 禁止中断）指令禁止全局中断处理。EI 和 DI 指令会改变寄存器 SYM 的第 0 位状态。

注释: 在复位操作之后，为了使能中断处理，应该在初始化程序中打开中断。 在程序中，你可以任意时间执行 DI (禁止中断)指令来禁止全局中断。EI 和 DI 指令可以改变 SYM 寄存器的第 0 位。

5.1.6 系统级中断控制寄存器

除了特定中断源的控制寄存器外，4 个系统级控制寄存器也参与中断处理控制

- 中断屏蔽寄存器，IMR，使能 (解除屏蔽) 或者 禁止 (屏蔽) 中断级。
- 中断优先级控制寄存器，IPR， 控制各个中断级之间的相对优先级。
- 中断请求寄存器，IRQ，包含每个中断级的中断标志位 (不同于中断源的标志位)。
- 系统模式控制寄存器，SYM，使能或者禁止全局中断处理 (SYM 的设置还可以使能快速中断并在硬件支持的情况下控制外部接口).

表 5-2 中断控制寄存器概述

控制寄存器	ID	R/W	功能描述
中断屏蔽寄存器	IMR	R/W	IMR寄存器中的位设置可以使能或禁止 IRQ0-IRQ7 中任意一个中断级的中断处理。
中断优先级控制寄存器	IPR	R/W	控制各个中断级之间的相对优先级。S3C84H5/F84H5 中的 8 个中断级被分作 3 个组：A，B 和C 组。A组包括 IRQ0 和 IRQ1，B组包括IRQ2，IRQ3 和 IRQ4，C 组包括 IRQ5，IRQ6 和 IRQ7。
中断请求寄存器	IRQ	R	包含每个中断级的中断标志位
系统模式控制寄存器	SYM	R/W	使能或者禁止快速中断处理和动态全局中断处理。

注释: 在改变 IMR 寄存器的内容前，必须禁止所有的中断。建议使用 DI 指令。

5.1.7 中断处理控制要点

中断处理控制可以通过两种方式来完成：全局或者特定中断级和中断源控制。

- 全局中的使能/禁止 (通过 EI 和 DI 指令)
- 中断级使能/禁止 (通过寄存器 IMR)
- 中断级优先级设置 (通过寄存器 IPR)
- 中断源使能/禁止（通过相应的外围控制寄存器)

注释: 在写中断应用程序时，一定要包含必要的寄存器卷地址（寄存器指针）信息

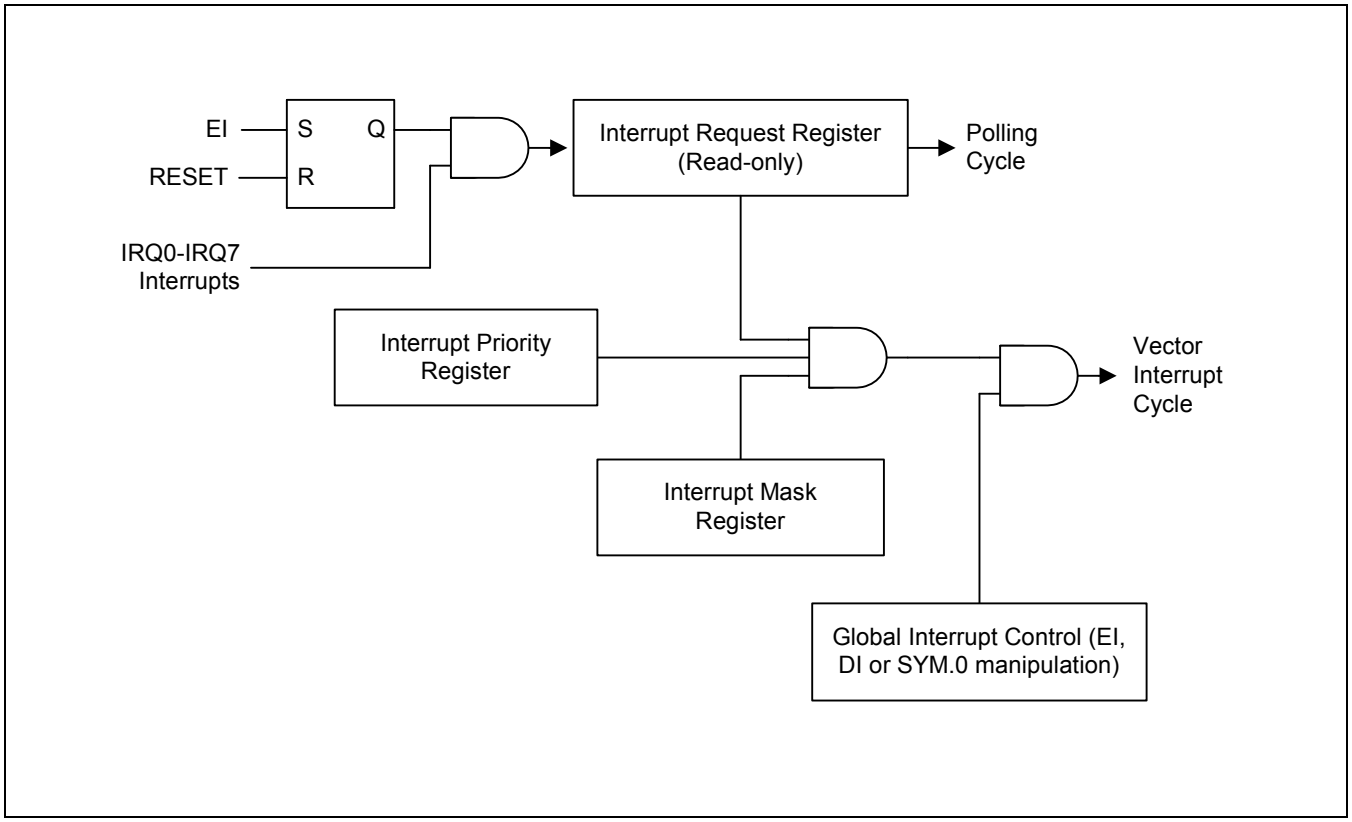


图 5-4 中断功能框图

5.1.8 外围中断控制寄存器

每一个中断源都受控于一个或多个外围控制寄存器（*表 5-3*）

表 5-3 中断源控制寄存器和数据寄存器

中断源	中断级	寄存器	寄存器地址
Timer A 匹配/捕获中断 Timer A 溢出中断	IRQ0	TACON TADATA TACNT TINTPND	E4H, BANK1 E6H, BANK1 EDH, BANK1 F1H, BANK1
Timer A 溢出中断 Timer A 匹配/捕获中断		TINTPND TACON TADATA TACNT	E0H, bank 1 E1H, bank 1 E2H, bank 1 E3H, bank 1
Timer 1(0) 匹配/捕获中断 Timer 1(0) 溢出中断 Timer 1(1) 匹配/捕获中断 Timer 1(1) 溢出中断	IRQ2	T1DATAH0,T1DATAL0 T1DATAH1,T1DATAL1 T1CON0, T1CON1 T1CNTH0, T1CNTL0 T1CNTH1, T1CNTL1 TINTPND	E4H, E5H, bank 1 E6H, E7H, bank 1 E8H, E9H, bank 1 EAH, EBH, bank 1 ECH, EDH, bank 1 E0H, bank 1
P1.0 外部中断 P1.1 外部中断 P1.2 外部中断 P1.3 外部中断	IRQ3	P1CONL P1INT P1INTPND	E9H, bank 0 EBH, bank 0 EAH, bank 0
Watch timer 中断	IRQ4	WTCON	F8H, bank 1
SIO 接收 / 发送	IRQ5	SIOCON,SIODATA	F1H,F2H bank 1
PWM 溢出中断	IRQ6	PWMCON PWMDATAH,PWMDATAL	F5H, bank 1 F3H,F4H bank 1
UART 接收 / 发送	IRQ7	UARTCON UDATA, UARTPND BRDATAH, BRDATAL	F6H, bank 0 F5H, F4H, bank 0 EEH, EFH, bank 1

注释: 如果中断在 IMR 里没有被屏蔽，DI 指令以后应该写中断标志位和使能位。

5.1.9 系统模式控制寄存器(SYM)

系统模式控制寄存器 SYM (地址：DEH，Set 1) 可以用来使能/禁止全局中断处理，控制快速中断处理。(*图 5-5*）

复位清零 SYM.1 和 SYM.0 。

EI 和 DI 指令分别使能和禁止中断处理，可以通过查询 SYM.0 得到当前状态。在复位操作之后，为了使能中断处理，应该在初始化程序中用 EI 指令打开中断。尽管可以通过直接向 SYM.3 位写“1”或“0”来使能或禁止中断，但 还是建议用 EI 指令或 DI 指令。

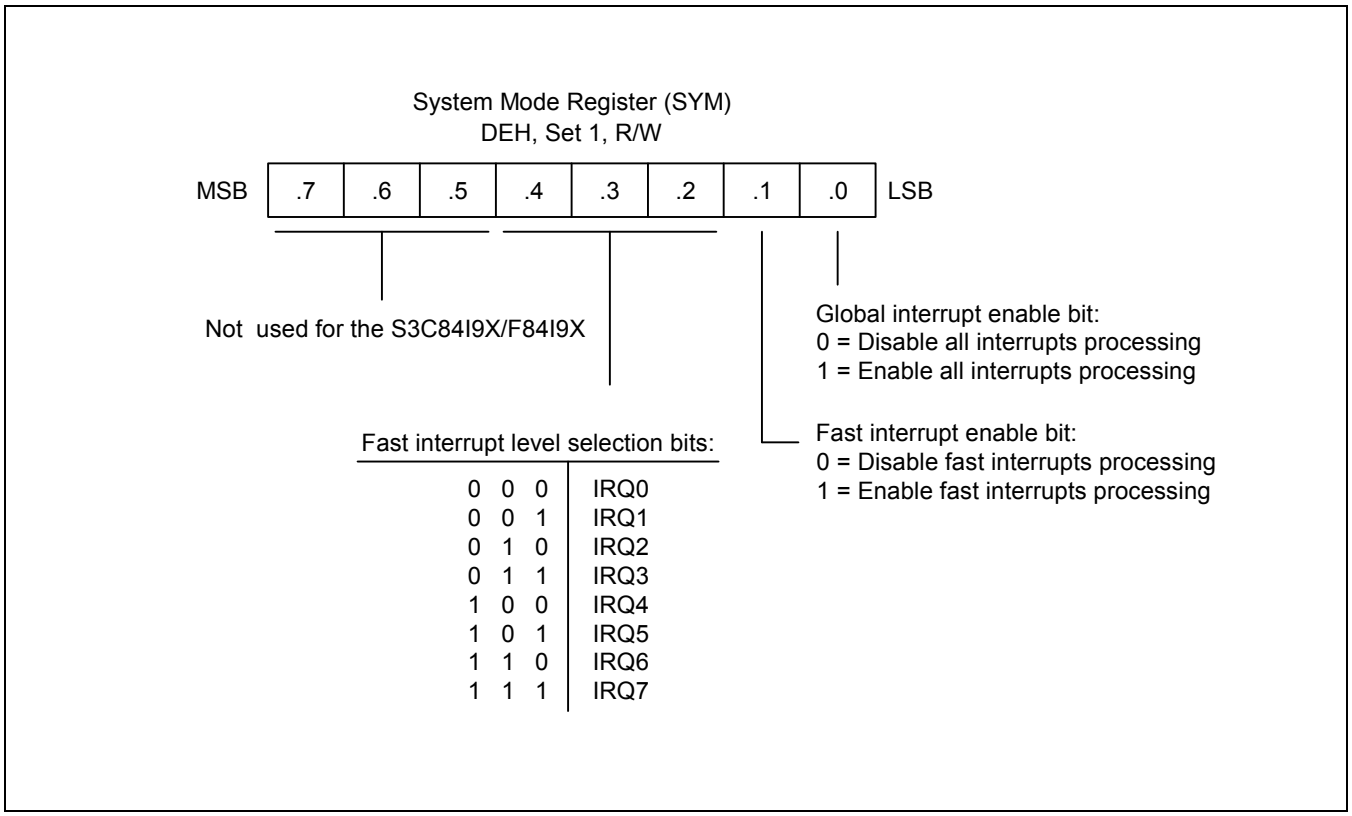


图 5-5 系统模式控制寄存器 (SYM)

5.1.10 中断屏蔽寄存器(IMR)

中断屏蔽寄存器 IMR (地址：DDH，Set 1) 可以用来使能或禁止各个中断级的中断处理。复位后，IMR 的所有位都是不确定的，所以必须在初始化程序中根据应用设置该寄存器。

IMR 中的每一位都对应一个中断级，第 1 位对应于 IRQ1，第 2 位对应于 IRQ2，以此类推。当 IMR 中的某位被清为“0”后，与之对应的中断级的中断处理就被禁止（屏蔽）了。当设置为“1”时，相应中断级的中断处理就被使能（解除屏蔽）了。

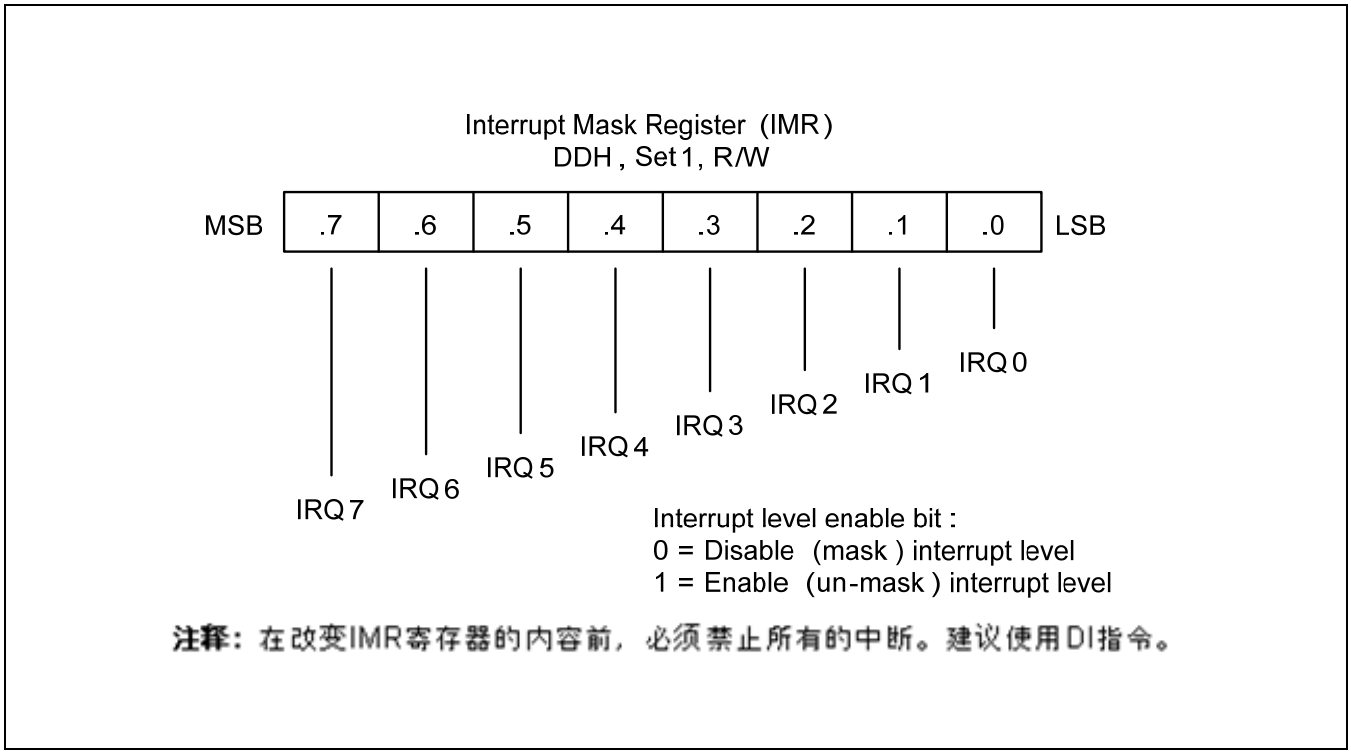


图 5-6 中断屏蔽寄存器 (IMR)

5.1.11 中断优先级控制寄存器(IPR)

中断优先级控制寄存器 IPR (地址：FFH，Set 1，Bank 0) 可以用来设置 MCU 中断结构中各个中断级的相对优先级。复位后，IPR 的所有位都是不确定的，所以必须在初始化程序中根据应用设置该寄存器。

当多个中断源同时发出中断请求时，优先级最高的那个中断源首先被 CPU 响应。 如果同一中断级中的多个中断源同时发出中断请求，中断向量地址最小的那个最先被响应。

为了实现灵活的中断优先级控制，8 个中断级被分为 3 个组。(*图 5-7*):

- Group A IRQ0, IRQ1
- Group B IRQ2, IRQ3, IRQ4
- Group C IRQ5, IRQ6, IRQ7

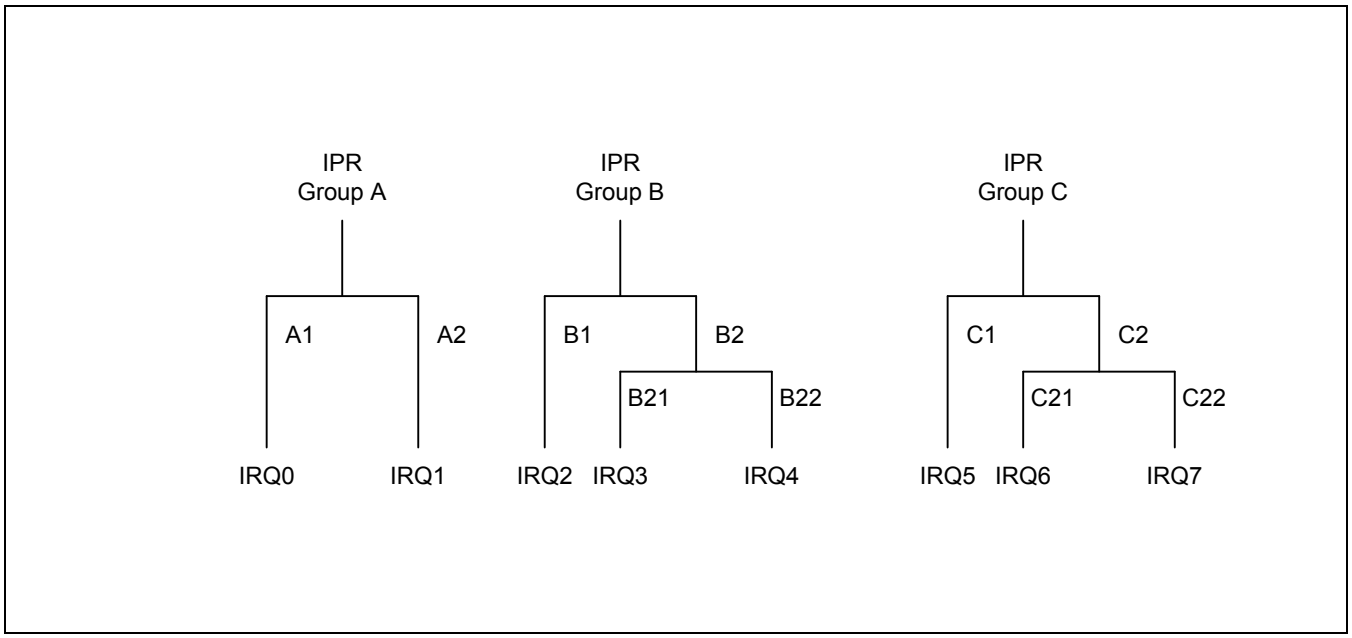


图 5-7 中断请求优先级组

如 *图 5-8* 所示，IPR.7，IPR.4 和 IPR.1 控制中断 ABC 组的相对优先级。例如，当这 3 位设置为 “001B” 时，这 3个中断组的优先级顺序为 B > C > A；设置为“101B”时，这 3 个中断组的优先级顺序为C > B > A。

IPR 其他位控制功能描述如下：

- IPR.5 控制中断 C 组内中断级的相对优先级。
- 中断 C 组包含一个由 IPQ6 和 IPQ7 组成的子组。子组的优先级由 IPR.6 的值决定。
- IPR.0 控制 IRQ0 和 IRQ1 的相对优先级。

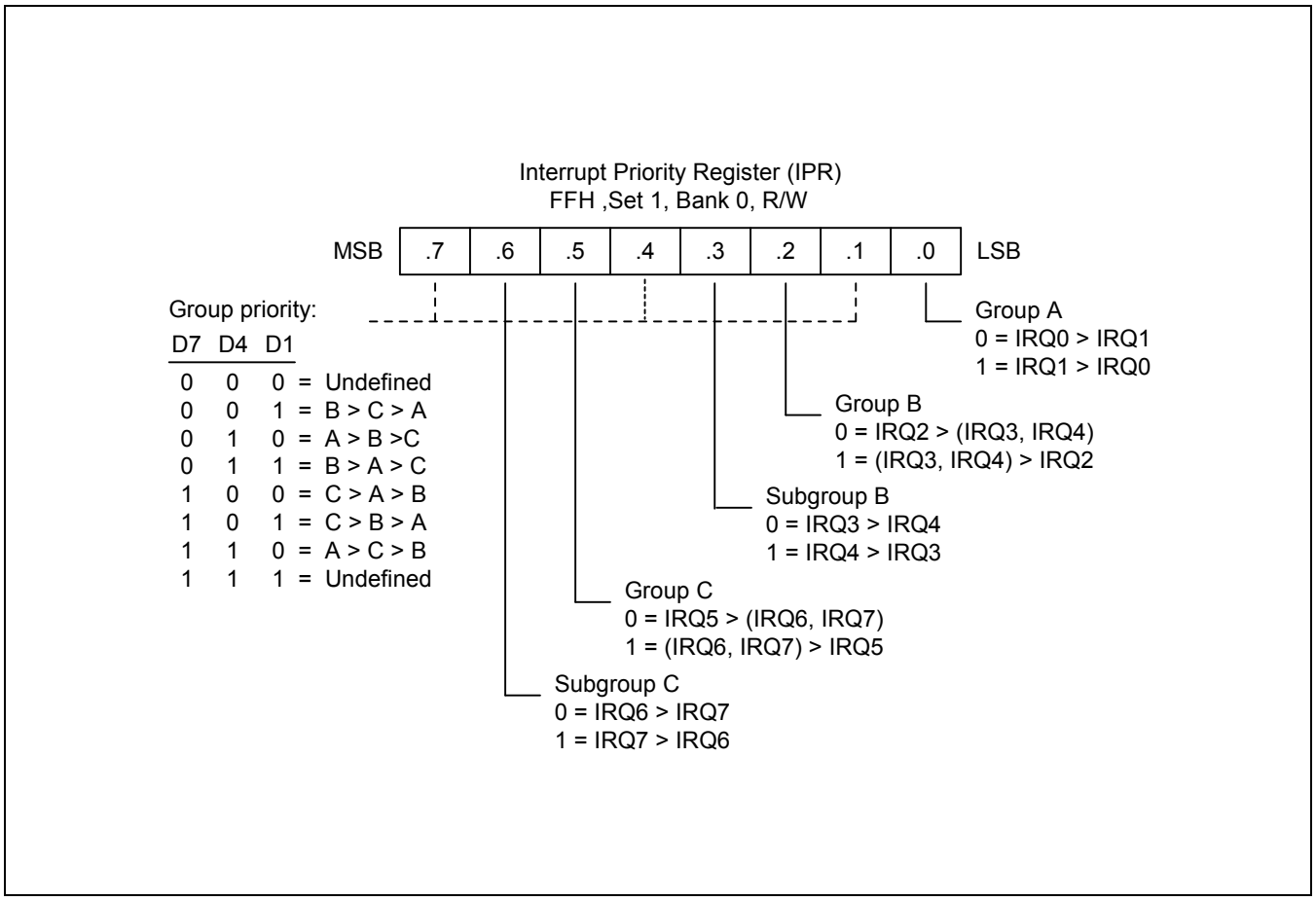


图 5-8 中断优先级寄存器 (IPR)

5.1.12 中断请求寄存器(IRQ)

可以通过查询中断请求寄存器 IRQ (地址：DCH，Set 1) 的每一位监视所有中断级的中断请求情况。 寄存器中的每一位都对应于一个序号相同的中断级：第 0 位对应于 IRQ0，第 1 位对应于 IRQ1，以此类推。 如果读到“0”，说明目前该中断级没有中断发生，反之，读到“1”则说明该中断级中的某个中断源发出了中断请求。

寄存器 IRQ 是只读的。复位后，所有位清零。

即使是执行 DI 指令后，即全局中断处理禁止的情况下，仍然可以查询 IRQ 的内容。 此时如果有中断发生，CPU 并不会做出任何响应。但是仍然可以通过查询 IRQ 确定中断发生情况。

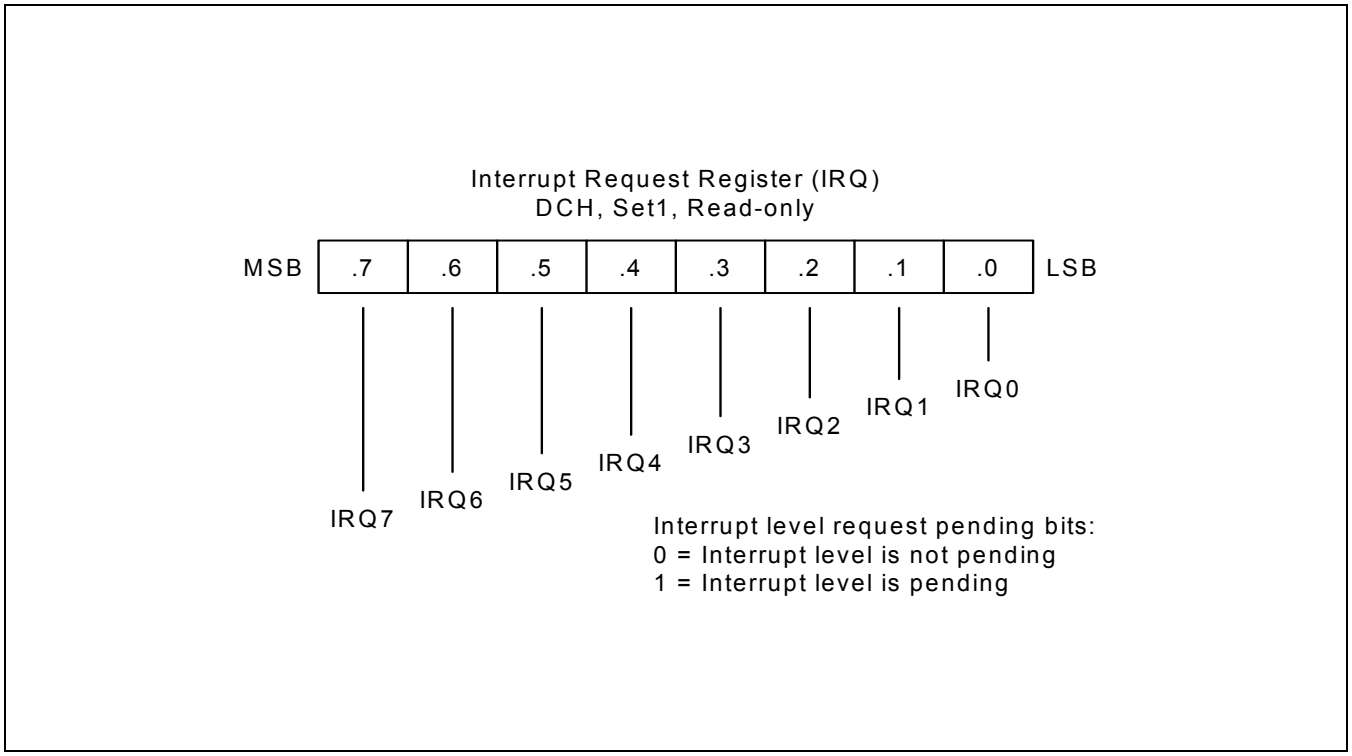


图 5-9 中断请求寄存器 (IRQ)

5.1.13 中断标志位类型

5.1.13.1 概述

有两种类型的中断标志位： 一种会在中断服务程序执行完毕后硬件自动清零，一种必须在中断服务程序中软件清零。

5.1.13.2 硬件自动清零的标志位

对于硬件自动清零的标志位来说，当有中断请求时，中断逻辑置高相应的标志位。然后触发 IRQ 脉冲告知 CPU 有一个中断正等待处理。CPU 随后发送 IACK 给中断源确认接受请求，然后执行服务程序，最后清楚标志位。

在 S3C84H5/F84H5中，Timer B溢出中断(IRQ0)就属于这一类中断。

5.1.13.3 需在中断服务程序里手动清零的标志位

这第二类标志位必须用软件清零。服务程序必须在中断返回（IRET）前清除标志位。可以通过向位于相应中断源的控制寄存器的中断标志位写“0”来实现。

在 S3C84H5/F84H5中， IRQ3, IRQ4, IRQ5, IRQ6和IRQ7就属于这一类中断。

5.1.14 中断响应的步骤

中断请求的查询和中断服务的步骤如下：

1. 某一个中断源通过置位中断标志位发出中断请求。
2. CPU 查询到有中断源已被挂起。
3. CPU 检查中断源中断级。
4. CPU 发出信号确认接受该中断请求。
5. 中断逻辑已经决定了该中断源的中断向量地址。
6. 开始跳转执行中断服务程序，清除中断标志位（通过硬件或者软件方式）。
7. 该中断服务程序结束后，CPU 继续查询中断请求。

5.1.15 中断服务程序

在以下条件全部满足的时候，CPU才会接受一个中断请求：

- 全局中断处理使能 (EI)
- 该中断源所处的中断级处于使能状态(寄存器 IMR)
- 如果有 CPU 同时接到多个中断请求，该中断源所处的中断级必须具有最高优先级
- 该中断源处于使能状态 (外围控制寄存器)

随后，CPU 开始中断处理并完成下面操作：

1. 清零 SYM.0 ，禁止所有后来的中断。.
2. 把 PC 值和系统标志寄存器（FLAGS）压入堆栈
3. 跳转到中断向量，获取中断服务程序地址。
4. 跳转执行中断服务程序

执行完一个中断服务程序后，中断返回指令IRET会将堆栈中的 PC 和 FLAGS 重新装载入当前寄存器，同时置位 SYM.0，使能 CPU 响应之后的中断请求。

5.1.16 中断向量地址的生成

中断向量位于 ROM (00H–FFH)，内容是相应中断服务程序的起始地址。 向量化的中断处理流程如下：

1. PC 低字节压入堆栈
2. PC 高字节压入堆栈
3. 把 FLAGS 寄存器的值压入堆栈
4. 从中断向量地址中取出中断服务程序高字节地址
5. 从中断向量地址中取出中断服务程序低字节地址
6. 跳转执行 16 位中断向量地址所确定的中断服务程序

注释: 16 位向量地址的起始地址始终是位于 ROM 地址空间 00H ~ FFH 中的某个偶数地址。

5.1.17 中断嵌套

在一个低优先级中断请求的处理过程中，可以实现一个高优先级中断的嵌套。为了实现这个功能，必须代码实现如下步骤：

1. 将 IMR 压入堆栈
2. 重新赋值 IMR，除了希望嵌套的那个高优先级中断外，屏蔽所有其他中断级
3. 执行 EI 使能全局中断 (进中断后自动 DI ，EI 后使得那个高优先级中断一旦发生就可以被响应)
4. 在低优先级中断服务处理程序的最后，执行DI ，将之前入栈的 IMR 内容重新装载入当前 IMR，恢复进入中断前的状态
5. 执行中断返回指令 IRET。

根据不同的应用，可能可以简化上述步骤

6 指令集

6.1 概述

SAM8RC 指令集支持对大容量寄存器卷的操作，一共包含 78 条指令，具有强大的数据处理能力。
指令集特性如下：

- 实现了所有 8 位算术和逻辑操作，包括乘法和除法
- 没有专门的输入/输出指令（输入/输出控制寄存器和数据寄存器直接映射到寄存器卷中）
- 十进制调整，包括 BCD 操作
- 16 位(字)数据可自增或自减
- 灵活的位寻址、循环和移位指令

6.1.1 数据类型

SAM8 CPU 可以实现位操作、字节操作、BCD 数字操作和双字节操作。寄存器的每个位可以被置 1、清 0、取反和测试。一个字节的各位按从 7 到 0 编号，其中 0 位是最低位 (在最右边)。

6.1.2 寄存器访问

为访问寄存器，应指定寄存器卷中地址为 0 – 255 之间 的 8 位地址或工作寄存器的 4 位地址。工作寄存器中，寄存器对可以访问 16 位程序存储空间和数据存储空间。关于寄存器访问的详细描述，请参考第 2 章“地址空间”。

6.1.3 寻址模式

有 7 种寻址模式：寄存器寻址(R)，间接寄存器寻址(IR)，偏址寻址(X)，直接寻址(DA)，相对地址寻址(RA)，立即数寻址(IM)和间接寻址(IA)。有关寻址模式的详细描述，请参考第 3 章“寻址模式”。

表 6-1 指令集简介

助记符	操作数	指令介绍
数据传送类指令		
CLR	dst	清零
LD	dst,src	传送数据
LDB	dst,src	传送位数据
LDE	dst,src	传送数据（访问外部数据存储空间）
LDC	dst,src	传送数据（访问程序存储空间）
LDED	dst,src	传送数据后地址减 1（访问外部数据存储空间）
LDCD	dst,src	传送数据后地址减 1（访问程序存储空间）
LDEI	dst,src	传送数据后地址加 1（访问外部数据存储空间）
LDCI	dst,src	传送数据后地址加 1（访问程序存储空间）
LDEPD	dst,src	传送数据前地址减 1（访问外部数据存储空间）
LDCPD	dst,src	传送数据前地址减 1（访问程序存储空间）
LDEPI	dst,src	传送数据前地址加 1（访问外部数据存储空间）
LDCPI	dst,src	传送数据前地址加 1（访问程序存储空间）
LDW	dst,src	传送字数据
POP	dst	出栈
POPUD	dst,src	弹出用户栈（减 1）
POPUI	dst,src	弹出用户栈（加 1）
PUSH	src	压栈
PUSHUD	dst,src	压入用户栈（减 1）
PUSHUI	dst,src	压入用户栈（加 1）
算术操作类指令		
ADC	dst,src	带进位加法
ADD	dst,src	不带进位加法
CP	dst,src	比较指令
DA	dst	十进制调整
DEC	dst	字节减 1
DECW	dst	字减 1
DIV	dst,src	除法指令
INC	dst	字节加 1
INCW	dst	字加 1
MULT	dst,src	乘法指令
SBC	dst,src	带借位减法
SUB	dst,src	不带借位减法
逻辑操作类指令		

助记符	操作数	指令介绍
AND	dst,src	逻辑与
COM	dst	取反
OR	dst,src	逻辑或
XOR	dst,src	逻辑异或
程序控制指令		
BTJRF	dst,src	位测试，如果为 0 跳转
BTJRT	dst,src	位测试，如果为 1 跳转
CALL	dst	调用子程序
CPIJE	dst,src	比较，如果相等跳转
CPIJNE	dst,src	比较，如果不等跳转
DJNZ	r,dst	寄存器减 1，不为 0 跳转
ENTER		进入
EXIT		跳出
IRET		中断返回
JP	cc,dst	有条件跳转
JP	dst	无条件跳转
JR	cc,dst	有条件相对跳转
NEXT		Next 指令
RET		子程序返回
WFI		等待中断
位操作指令		
BAND	dst,src	位与
BCP	dst,src	位比较
BITC	dst	位取反
BITR	dst	位清零
BITS	dst	位置 1
BOR	dst,src	位或
BXOR	dst,src	位异或
TCM	dst,src	取反后位测试
TM	dst,src	位测试指令
循环和移位指令		
RL	dst	循环左移
RLC	dst	带进位循环左移
RR	dst	循环右移
RRC	dst	带进位循环右移
SRA	dst	算术右移

助记符	操作数	指令介绍
SWAP	dst	交换
CPU控制指令		
CCF		进位标志取反
DI		屏蔽全局中断
EI		使能全局中断
IDLE		进入 IDLE 模式
NOP		空操作
RCF		进位标志清零
SB0		选择寄存器块 bank 0
SB1		选择寄存器块 bank 1
SCF		进位标志置 1
SRP	src	设置寄存器指针
SRP0	src	设置寄存器指针 0
SRP1	src	设置寄存器指针 1
STOP		进入 STOP 模式

注释: LDE, LDED, LDEI, LDEPP 和 LDEPI 指令可用来读/写 64 K 字节的外部数据存储空间。

6.2 标志寄存器(FLAGS)

8 位标志寄存器 FLAGS 描述当前 CPU 的操作状态。其中的 4 位（FLAGS.4-FLAGS.7）可以用于测试和条件转移指令； 另外两个标志位 FLAGS.3 和标志位 FLAGS.2 用于 BCD 算术操作。

还有一个标志位 FLAGS.1 用于指示快速中断处理；FLAGS.0 表示当前正在寻址的 Bank 地址状态，是 Bank 0 还是Bank 1。

只要结果不影响到状态位，FLAGS 寄存器可以通过指令（如 LOAD 指令）置 1 或者清 0。逻辑和算术类操作指令，例如与、或、异或、加法和减法操作，会影响到标志位寄存器。例如，AND 指令会根据结果改变 Zero, Sign 和 Overflow（Z、S、O）标志。如果 AND 指令使用 FLAGS 作为目标寄存器，将会同时发生两次 FLAGS 寄存器的写操作，造成不可预知的后果。

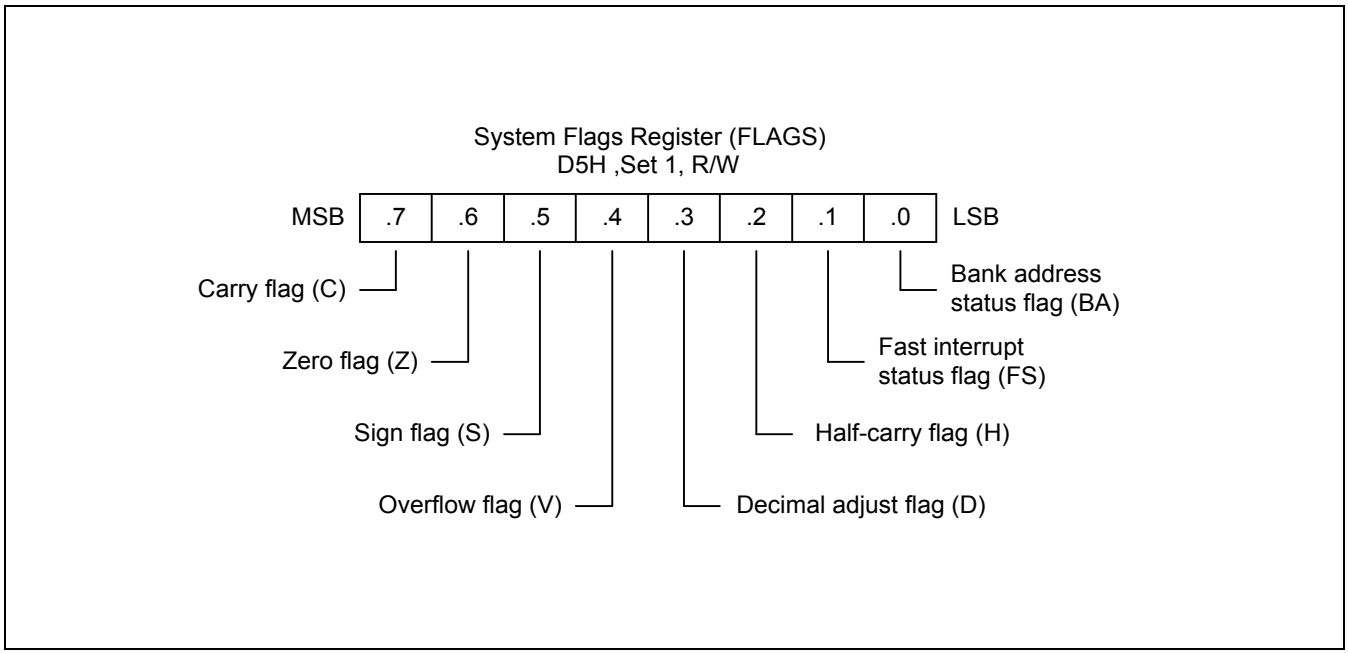


图 6-1 系统标志寄存器（FLAGS）

6.2.1 标志位描述

6.2.1.1 C 进(借) 位标志(FLAGS.7)

如果算术操作后，最高位产生进位或借位，则此标志位为 1。移位、循环移位操作之后，此位保存最后移出的那一位值。指令可以对此位置 1、清 0、取反操作。

6.2.1.2 Z 零标志位(FLAGS.6)

如果算术、逻辑操作的结果为 0 ，则此标志位为 1。位测试指令、移位指令、循环移位指令都会影响此标志位，如果结果为逻辑 0，则此标志位为 1。

6.2.1.3 S 符号标志位(FLAGS.5)

算术、逻辑、循环、移位操作之后，操作结果最高位的状态反映在符号位。逻辑 0 表示操作结果是正数，逻辑 1 表示操作数结果是负数。

6.2.1.4 V 溢出标志(FLAGS.4)

当操作结果大于 + 127 或 小于 – 128 时，溢出标志将会置 1。逻辑操作之后，它将会被清 0。

6.2.1.5 D 十进制调整标志(FLAGS.3)

DA 标志用于指明 BCD 操作中何种类型的指令是最后被执行的，因此随后的十进制调整指令可以正确执行。 编程者通常不能访问 DA 位，也不能用来做测试的条件码 。

6.2.1.6 H 半字节进(借) 位标志(FLAGS.2)

当加法操作时第 3 位产生进位或减法操作时向第 4 位发生借位，H标志将被置 1。通常用在DA指令中，将前一次的加法或减法结果（二进制码）转化为十进制结果（BCD）。程序通常不会直接用到 H 标志 。

6.2.1.7 FIS 快速中断状态标志(FLAGS.1)

在快速中断执行期间，FIS 位被置 1；快速中断返回时，FIS 位被清 0。当 FIS 被置 1 时，将禁止所有中断，直到 IRET 指令被执行后，才重新打开快速中断。

6.2.1.8 BA 寄存器块地址标志(FLAGS.0)

BA 标志表明内部寄存器卷 Set 1 中哪个寄存器块被选中，是 bank 0 还是 bank 1。当执行 SB0 指令，BA标志被清零（选择 bank 0）；当执行 SB1 指令，BA 标志被置 1（选择 bank 1）。

6.2.2 指令集符号

表 6-2 标志位符号

标志位	描述
C	进（借）位标志
Z	零标志
S	符号标志
V	溢出标志
D	十进制调整标志
H	半字节进（借）位标志
0	清为逻辑 0
1	置为逻辑 1
*	根据相应操作置 1 或清 0
-	不受影响
x	不确定

表 6-3 指令集符号

符号	描述
dst	目的操作数
src	源操作数
@	间接寄存器地址前缀
PC	程序计数器
IP	指令指针
FLAGS	标志寄存器(D5H)
RP	寄存器指针
#	立即数或寄存器访问的前缀
H	十六进制后缀
D	十进制后缀
B	二进制后追
opc	指令代码

表 6-4 指令符号的定义

符号	描述	实际操作范围
cc	条件码	参考表格 *表 6-7*
r	工作寄存器	Rn (n = 0-15)
rb	工作寄存器的位 b	Rn.b (n = 0-15, b = 0-7)
r0	工作寄存器的位 0，最低位	Rn (n = 0-15)
rr	工作寄存器对	RRp (p = 0, 2, 4, ..., 14)
R	寄存器或者工作寄存器	reg or Rn (reg = 0–255, n = 0-15)
Rb	寄存器或者工作寄存器的位 b	reg.b (reg = 0–255, b = 0-7)
RR	寄存器对或工作寄存器对	reg or RRp (reg = 0-254，只能是偶数 p = 0, 2, ..., 14)
IA	间接寻址模式	addr (addr = 0-254，只能是偶数)
Ir	间接工作寄存器寻址	@Rn (n = 0-15)
IR	间接工作寄存器寻址或间接寄存器寻址	@Rn or @reg (reg = 0-255, n = 0-15)
Irr	间接工作寄存器对	@RRp (p = 0, 2, ..., 14)
IRR	间接寄存器对或间接工作寄存器对	@RRp or @reg (reg = 0-254，只能是偶数 p = 0, 2, ..., 14)
X	基址寻址模式	#reg [Rn] (reg = 0-255, n = 0-15)
XS	短偏址寻址模式	#addr [RRp] (addr = 范围 -128 to +127, p = 0, 2, ..., 14)
xl	长偏址寻址模式	#addr [RRp] (addr = 范围 0-65535, p = 0, 2, ..., 14)
da	直接寻址模式	addr (addr = 范围 0-65535)
ra	相对地址寻址模式	addr (addr = 在 +127 到 -128 范围内的数字)
im	立即数寻址模式	#data (data = 0-255)
iml	长立即数寻址模式	#data (data = 范围 0-65535)

表 6-5 指令代码快速参考

指令代码对照图									
	LOWER NIBBLE（HEX）								
	-	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,Ir2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0–Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,Ir2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,Ir2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0–Rb
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,Ir2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,Ir2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0–Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,Ir2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,Ir2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0–Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,Ir2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b
B	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2
B	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1
L	A	INCW RR1	INCW IR1	CP r1,r2	CP r1,Ir2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, Irr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,Ir2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, Irr2, xL
	C	RRC R1	RRC IR1	CPIJE Ir,r2,RA	LDC r1,Irr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, Ir2
H	D	SRA R1	SRA IR1	CPIJNE Irr,r2,RA	LDC r2,Irr1	CALL IA1		LD IR1,IM	LD Ir1, r2
E	E	RR R1	RR IR1	LDCD r1,Irr2	LDCI r1,Irr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,Irr1	LDCPI r2,Irr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, Irr1, xs

表 6-6 指令代码快速参考（续）

指令代码对照图									
LOWER NIBBLE（HEX）									
	-	8	9	A	B	C	D	E	F
U	0	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NEXT
P	1	↓	↓	↓	↓	↓	↓	↓	ENTER
P	2								EXIT
E	3								WFI
R	4								SB0
	5								SB1
N	6								IDLE
I	7	↓	↓	↓	↓	↓	↓	↓	STOP
B	8								DI
B	9								EI
L	A								RET
E	B								IRET
	C								RCF
H	D	↓	↓	↓	↓	↓	↓	↓	SCF
E	E								CCF
X	F	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NOP

6.3 条件码

条件转移指令通常包含 4 位条件转移操作判断(cc)。这些条件转移操作判断的结果将决定程序的跳转方向。
例如，比较操作后的“相等”条件转移，只有在两个操作数相等时该条件转移操作才会跳转。条件码列举在表格 *表 6-7*。

C、Z、S、V 等标志位用作条件转移判断位。指令将会根据这些标志位决定跳转方向。

表 6-7 条件码

二进制	助记符	描述	标志位设置
0000	F	逻辑假	-
1000	T	逻辑真	-
0111 (1)	C	有进位或借位	C = 1
1111 (1)	NC	无进位或借位	C = 0
0110 (1)	Z	结果为 0	Z = 1
1110 (1)	NZ	结果不为 0	Z = 0
1101	PL	正数	S = 0
0101	MI	负数	S = 1
0100	OV	溢出	V = 1
1100	NOV	没有溢出	V = 0
0110 (1)	EQ	相等	Z = 1
1110 (1)	NE	不相等	Z = 0
1001	GE	大于等于	(S XOR V) = 0
0001	LT	小于	(S XOR V) = 1
1010	GT	大于	(Z OR (S XOR V)) = 0
0010	LE	小于等于	(Z OR (S XOR V)) = 1
1111 (1)	UGE	无符号大于等于	C = 0
0111 (1)	ULT	无符号小于	C = 1
1011	UGT	无符号大于	(C = 0 AND Z = 0) = 1
0011	ULE	无符号小于等于	(C OR Z) = 1

注释:

1. 一次算术操作的结果可能同时影响两个标志位。例如，Z 符号位被置起时，Z、EQ 都为真。但是 ADD 指令操作之后，可能会用到 Z ；而 CP 指令操作之后，EQ 可能被用到。
2. 如果操作数涉及到无符号数，必须使用 UGE, ULT, UGT, ULE 等条件代码。

6.3.1 指令集描述

本章详细介绍了 S3F8-系列单片机的指令操作，同时给出了具体的编程实例。为便于参阅和快速查找，在介绍指令时采用了统一的格式。对每条指令，采用了如下的描述方法：

- 指令名称（标号）
- 指令全称
- 源操作数/目的操作数的格式
- 具体指令的解释
- 每条指令操作的具体描述
- 每条指令对标志寄存器的影响
- 指令格式、执行周期和访问模式的详细介绍
- 每条指令的编程实例

6.3.1.1 ADC - 带进位加法(Add with Carry)

ADC dst,src

操作: dst ← dst + src + c

目的操作数加上源操作数和 C 位，所得结果保存在目的操作数。源操作数不受影响。
在多字节加法中，该指令允许把低字节的进位加到高字节的加法运算中 。

标志位:

C: 如果加法运算中产生进位，则此位置 1；否则，清 0
Z: 如果运算结果为 0，则该位置 1；否则，清 0
S: 如果运算结果为负，则该位置 1；否则，清 0
V: 如果运算结果产生溢出，该位置 1；否则，清 0
D: 总是被清 0
H: 如果运算结果的低4位有进位，该位置 1；否则，清 0

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	12	r	r
		6	13	r	lr
opc | src | dst	3	6	14	R	R
		6	15	R	IR
opc | dst | src	3	6	16	R	IM

编程实例:

假设: R1 = 10H, R2 = 03H, C flag =“1”, 寄存器 01H = 20H, 寄存器 02H = 03H, 和寄存器 03H = 0AH:

```
ADC  R1,R2       → R1 = 14H, R2 = 03H
ADC  R1,@R2      → R1 = 1BH, R2 = 03H
ADC  01H,02H     → 寄存器 01H = 24H, 寄存器 02H = 03H
ADC  01H,@02H    → 寄存器 01H = 2BH, 寄存器 02H = 03H
ADC  01H,#11H    → 寄存器 01H = 32H
```

在第一个例子中, 目的寄存器R1内容为 10H, 进位标志位为 1, 源寄存器 R2 为 03H。语句“ADC R1,R2”把03H和进位位(“1”)累加到目的数 10H, 结果为 14H, 保存在 R1。

6.3.1.2 ADD - 加法(Add)

ADD dst, src

操作: dst ← dst + src

源操作数和目的操作数相加，结果保存在目的操作数，源操作数不受影响 。

标志位:

C: 如果结果的最高位有进位，被置 1；否则，被清 0
Z: 如果结果为 0，被置 1；否则，被清 0
S: 如果结果是负数，被置 1；否则，被清 0
V: 如果有溢出，被置1，也就是说，两个操作数符号相同，运算结果是相反的符号；否则，被清0
D: 总是被清 0
H: 如果结果的低4位有进位，被置 1；否则，被清 0

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	02	r	r
		6	03	r	lr
opc | src | dst	3	6	04	R	R
		6	05	R	IR
opc | dst | src	3	6	06	R	IM

编程实例:

```
假设： R1 = 12H, R2 = 03H, 寄存器 01H = 21H, 寄存器 02H = 03H, 寄存器 03H = 0AH:

ADD  R1,R2       → R1 = 15H, R2 = 03H
ADD  R1,@R2      → R1 = 1CH, R2 = 03H
ADD  01H,02H     → 寄存器 01H = 24H, 寄存器 02H = 03H
ADD  01H,@02H    → 寄存器 01H = 2BH, 寄存器 02H = 03H
ADD  01H,#25H    → 寄存器 01H = 46H

在第一个例子中，目的工作寄存器 R1 内容为12H，源工作寄存器 R2 内容为 03H。语句“ADD R1,R2”执行 03H+12H，结果为 15H，保存在寄存器 R1。
```

6.3.1.3 AND - 逻辑与(Logical AND)

AND dst,src

操作: dst ← dst AND src

源操作数与目的操作数执行逻辑与操作，结果保存在目的操作数。
只有当两个操作数的对应位都为1时，结果的相应位才是 1；否则，该位为 0。
源操作数不受影响。

标志位:

C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 如果结果是负数，被置 1；否则，被清 0
V: 总是被清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	52	r	r
		6	53	r	lr
opc | src | dst	3	6	54	R	R
		6	55	R	IR
opc | dst | src	3	6	56	R	IM

编程实例:

```
假设：R1 = 12H, R2 = 03H, 寄存器 01H = 21H, 寄存器 02H = 03H, 寄存器 03H = 0AH:

AND  R1,R2       → R1 = 02H, R2 = 03H
AND  R1,@R2      → R1 = 02H, R2 = 03H
AND  01H,02H     → 寄存器 01H = 01H, 寄存器 02H = 03H
AND  01H,@02H    → 寄存器 01H = 00H, 寄存器 02H = 03H
AND  01H,#25H    → 寄存器 01H = 21H

在第一个例子中，目的操作数 R1为 12H，源操作数 R2 为 03H，指令“AND R1,R2”对 03H 和12H
进行逻辑与操作，结果为 02H，保存在寄存器 R1。
```

6.3.1.4 BAND - 位与(Bit AND)

BAND dst,src.b

BAND dst.b,src

操作: dst(0) ← dst(0) AND src(b)
or
dst(b) ← dst(b) AND src(0)

源操作数（或者目的操作数）的特定位与目的操作数（或者源操作数）的最低位进行逻辑与操作，结果保存在目的操作数的特定位。目的操作数的其他位不受影响。
源操作数不受影响 。

标志位:
C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 总是被清零
V: 不确定
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
opc | dst | b | 0 | src	3	6	67	r0	Rb
opc | src | b | 1 | dst	3	6	67	Rb	r0

注释: 在 3 字节指令各式的第二个字节中，目的或者源操作数地址是 4 位，位地址是 3 位 。

编程实例:

```
假设：R1 = 07H 和寄存器 01H = 05H:

BAND R1,01H.1     → R1 = 06H, 寄存器 01H = 05H
BAND 01H.1,R1     → 寄存器 01H = 05H, R1 = 07H

在第一个例子中, 源寄存器 01H 的内容是 05H, 目的寄存器 R1 的内容是 07H, 指令“BAND R1,01H.1”
将源寄存器的位 1 和目的寄存器 R1 的位 0 进行逻辑与操作, 结果为 06H, 保存在寄存器 R1。
```

6.3.1.5 BCP - 位比较(Bit Compare)

BCP dst,src.b

操作: dst(0) – src(b)

源操作数的特定位与目的操作数的最低位做比较，如果相等，零标志位被置 1，否则被清 0。两个操作数都不受影响。

标志位:

C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 总是被清零
V: 不确定
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | b | 0 src	3	6	17	r0	Rb

注释: 在 3 字节指令各式的第二个字节中，目的操作数地址是 4 位，位地址是 3 位。

编程实例:

假设：R1 = 07H 和寄存器 01H = 01H:

```
BCP  R1,01H.1    → R1 = 07H, 寄存器 01H = 01H
```

如果目的寄存器 R1 内容为 07H，源寄存器 01H 的内容为 01H，指令“BCP R1,01H.1”比较源寄存器(01H)的位 1 和目的寄存器 R1 的位 0。因为两个位是不相等的，所以标志位寄存器的 Z 标志位被清零。

6.3.1.6 BITC - 位反(Bit Complement)

BITC dst.b

操作: dst(b) ← NOT dst(b)

这个指令对目的操作数的特定位取反而不影响其他位。

标志位:
C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 总是被清零
V: 不确定
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst | b | 0	2	4	57	rb

注释: 在指令的第二个字节中，目的操作数地址是 4 位，位地址是 3 位。

编程实例:

```
假设：R1 = 07H

BITC R1.1  → R1 = 05H

工作寄存器 R1 内容为 07H，指令“BITC R1.1“对 R1 的位 1 取反，并将结果(05H)保存在 R1。
因为结果不是“0”，所以标志位寄存器的 Z 标志被清零。
```

6.3.1.7 BITR - 位清零(Bit Reset)

BITR dst.b

操作: dst(b) ← 0

BITR 指令将目的操作数的特定位清零，而不影响其它的位。

标志位: 没有任何标志位受影响

格式:

		字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc	dst | b | 0	2	4	77	rb

注释: 在指令的第二个字节中，目的操作数地址是 4 位，位地址是 3 位。

编程实例:

```
假设：R1 = 07H:

BITR R1.1  → R1 = 05H

工作寄存器 R1 的内容是 07H, 指令“BITR R1.1”对目标寄存器 R1 的位 1 清零, 结果为 05H。
```

6.3.1.8 BITS - 位置 1(Bit Set)

BITS dst.b

操作: dst(b) ← 1

BITS 指令将目的操作数的特定位设置为 1，而不影响其它的位。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst | b | 1	2	4	77	rb

注释: 在指令的第二个字节中，目的操作数地址是 4 位，位地址是 3 位 。

编程实例:

```
假设： R1 = 07H:

BITS R1.3  → R1 = 0FH

工作寄存器 R1 的内容是 07H, 指令“BITS R1.3”将目标寄存器 R1 的位 3 置为 1, 结果为 0FH。
```

6.3.1.9 BOR - 位或(Bit OR)

BOR dst,src.b

BOR dst.b,src

操作: dst(0) ← dst(0) OR src(b)
或
dst(b) ← dst(b) OR src(0)

源操作数（或者目的操作数）的特定位与目的操作数（源操作数）的最低位进行逻辑或，运算结果保存在目的操作数的特定位，目的操作数的其它位不受影响。源操作数不受影响。

标志位:
C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 总是被清零
V: 不确定
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | b | 0 | src	3	6	07	r0	Rb
opc | src | b | 1 | dst	3	6	07	Rb	r0

注释: 在 3 字节指令格式的第二个字节中，目的操作数地址是 4 位，位地址是 3 位。

编程实例:

```
假设： R1 = 07H 和寄存器 01H = 03H:

BOR  R1, 01 H.1          → R1 = 07H, 寄存器  01H = 03H
BOR  01H.2, R1           → 寄存器  01H = 07H, R1 = 07H

在第一个例子里面, 目的寄存器 R1 的内容为 07H, 源寄存器 01H 的内容是 03H, 指令“BOR R1,01H.1”
将寄存器 01H 的位 1 与 R1 的位 0 进行逻辑或运算, 结果(还是 07H )保存在 R1 中。

在第二个例子里面, 目的寄存器 01H 的内容为 03H, 源寄存器 R1 的内容是 07H, 指令“BOR 01H.2,R2”
将 01H 的位 2 与寄存器 R1 的位 0 进行逻辑或运算, 结果(07H)保存在寄存器 01H 中。
```

6.3.1.10 BTJRF - 位测试，若为假相对跳转(Bit Test, Jump Relative on False)

BTJRF dst,src.b

操作: 如果目标操作数的位 b 为“0”，那么 PC ← PC + dst

测试源原操作数的特定位，如果为“0”，则将相对地址累加到程序计数器(PC)，并从新的 PC 地址开始执行程序；否则，执行 BTJRF 后面的指令 。

标志位: 没有标志受影响。

格式:

(注释 1)			字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
opc	src | b | 0	dst	3	10	37	RA	rb

注释: 在 3 字节指令各式的第二个字节中，目的操作数地址是 4 位，位地址是 3 位 。

编程实例:

假设：R1 = 07H:

```
BTJRF      SKIP,R1.3    → PC 跳转到 SKIP 地址处
```

如果工作寄存器 R1 的内容是 07H，指令“BTJRF SKIP,R1.3”测试 R1 的位 3。 因其为“0”，相对地址将累加到 PC，然后 PC 跳转到 SKIP 地址处。（记住地址跳转的范围必须在 +127 至 -128之间）

6.3.1.11 BTJRT - 位测试，若为真，相对跳转(Bit Test, Jump Relative on True)

BTJRT dst,src.b

操作: 如果目标操作数的位 b 为“1”，那么 PC ← PC + dst

测试源操作数的特定位，如果为“1”，则将相对地址累加到程序计数器(PC)，并从新的 PC 地址开始执行程序；如果为“0”，执行 BTJRT 后面的指令 。

标志位: 没有标志位受影响。

格式:

(注释 1)			字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
opc	src | b | 1	dst	3	10	37	RA	rb

注释: 在 3 字节指令各式的第二个字节中，目的操作数地址是 4 位，位地址是 3 位。

编程实例:

假设：R1 = 07H:

```
BTJRT      SKIP,R1.1
```

工作寄存器 R1 的内容是 07H，指令“BTJRT SKIP,R1.1”测试寄存器 R1 的位 1。因该位为“1”，相对地址将被叠加到 PC，并跳转到新的 PC 地址 SKIP 处。（记住跳转范围必须在 +127至 -128 之间）

6.3.1.12 BXOR - 位异或(Bit XOR)

BXOR dst,src.b

BXOR dst.b,src

操作: dst(0) ← dst(0) XOR src(b)
或
dst(b) ← dst(b) XOR src(0)

源操作数（或者目的操作数）的特定位与目的操作数（源操作数）的最低位进行逻辑异或运算。结果保存在目的寄存器的特定位中。其他位不受影响。源操作数不受影响 。

标志位:

C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 总是被清零
V: 不确定
D: 不受影响
H: 不受影响

格式:

			字节数	时钟周期	指令代码	寻址模式	
					(Hex)	dst	src
opc	dst | b | 0	src	3	6	27	r0	Rb
opc	src | b | 1	dst	3	6	27	Rb	r0

注释: 在 3 字节指令各式的第二个字节中，目的操作数地址是 4 位，位地址是 3 位 。

编程实例:

假设： R1 = 07H (00000111B) 和寄存器 01H = 03H (00000011B):

```
BXOR R1,01H.1    → R1 = 06H, 寄存器 01H = 03H
BXOR 01H.2,R1    → 寄存器 01H = 07H, R1 = 07H
```

在第一个例子中，目的寄存器 R1 的内容是 07H，源寄存器 01H 的内容是 03H，指令“BXOR R1,01H.1”将源寄存器 01H 的位 1 和 R1 的位 0 进行逻辑异或，结果保存在 R1 的位 0，将 R1 的值从 07H 改写为 06H。源寄存器 01H 的值不受影响。

6.3.1.13 CALL - 程序调用(Call Procedure)

CALL dst

操作:

SP ← SP - 1
@SP ← PCL
SP ← SP - 1
@SP ← PCH
PC ← dst

PC 的当前内容被压入堆栈，也就是紧跟 CALL 指令之后的指令地址。 然后，指定的目标地址被送给PC，指向子程序的第一条指令地址。在子程序末尾，用返回指令 RET 返回到原来的程序流程，继续执行主程序。RET 指令的执行，将 PC 值从堆栈顶部弹出 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	3	14	F6	DA
opc | dst	2	12	F4	IRR
opc | dst	2	14	D4	IA

编程实例:

假设： R0 = 35H, R1 = 21H, PC = 1A47H, 和SP = 0002H:

```
CALL 3521H →     SP = 0000H
                      (存储器 0000H = 1AH, 0001H = 4AH, 4AH 是指令后面的地址)
CALL @RR0  →     SP = 0000H (0000H = 1AH, 0001H = 49H)
CALL #40H  →     SP = 0000H (0000H = 1AH, 0001H = 49H)
```

在第一个例子中，如果 PC 值为 1A47H，堆栈指针内容为 0002H，指令“CALL 3521H”将当前的PC 数值压入堆栈顶，堆栈指针现在指向 0000H，然后 PC 装载入 521H，从子程序的第一条指令地址开始顺序执行。

如果 PC 和 SP 的内容与第一个例子相同，指令“CALL @RP0”运行的结果基本相同，除了堆栈0001H 的内容是 49H（因为是 2 字节指令），然后 PC 装载数值 3521H，并顺序执行指令。如果PC 和堆栈指针的内 容与第一个例子相同，如果程序抵制 0040H 的内容为 35H，41H 的内容为21H，指令“CALL #40H”产生 和第二个例子相同的结果。

6.3.1.14 CCF - 进位标志位取反(Complement Carry Flag)

CCF

操作: C ← NOT C

进位标志 C 取反。如果C = “1”，进位标志变成 0；如果C = “0”，进位标志变成 1。

标志位: **C:** 取反.

其他的标志位不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	EF

编程实例:

假设： 进位标志C = “0”：

```
CCF
```

如果进位标志 C = “0”，指令 CCF 对该标志为取反，在标志位寄存器中，该位从“0”变成“1”。

6.3.1.15 CLR - 清零(Clear)

CLR dst

操作: dst ← "0"

目的操作数被清零。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	B0	R
		4	B1	IR

编程实例:

假设： 寄存器 00H = 4FH, 寄存器 01H = 02H, 和寄存器 02H = 5EH:

```
CLR  00H   → 寄存器    00H   =    00H
CLR  @01H  → 寄存器    01H   =    02H,  寄存器      02H   =    00H
```

在寄存器寻址模式中，指令“CLR 00H”把目的寄存器 00H 的内容清零。在第二个例子中，指令“CLR @01H”使用间接寄存器寻址模式，把寄存器 02H 清零。

6.3.1.16 COM - 取反(Complement)

COM dst

操作: dst ← NOT dst

对目标地址的内容取反，所有的“1”变成“0”，所有的“0”变成“1”。

标志位:
- **C:** 不受影响
- **Z:** 如果结果为 0，被置 1；否则，被清 0
- **S:** 如果结果是负数，被置 1；否则，被清 0
- **V:** 总是被清 0
- **D:** 不受影响
- **H:** 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	60	R
		4	61	IR

编程实例:

假设： R1 = 07H 和寄存器 07H = 0F1H:

```
COM  R1    → R1 = 0F8H
COM  @R1   → R1 = 07H,      寄存器 07H = 0EH
```

在第一个例子中，目的寄存器 R1 的内容是 07H，指令“COM R1”对R1的所有位取反，所有的逻辑“1”变成逻辑“0”，所有的逻辑“0”变成逻辑“1”，结果为 0F8H。

在第二个例子中，应用间接寄存器寻址模式对目的寄存器 07H 的内容取反，结果为 0EH。

6.3.1.17 CP - 比较(Compare)

CP dst,src

操作: dst - src

源操作数和目的操作数作比较(相减)，根据结果设置适当的标志位。
源操作数和目的操作数均不受影响。

标志位:

C: 如果源操作数大于目的操作数，被置 1；否则，被清 0
Z: 如果结果为 0，被置 1；否则，被清 0
S: 如果结果是负数，被置 1；否则，被清 0
V: 总是被清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
opc | dst | src	2	4	A2	r	r
		6	A3	r	lr
opc | src | dst	3	6	A4	R	R
		6	A5	R	IR
opc | dst | src	3	6	A6	R	IM

编程实例:

```
1.   假设： R1 = 02H 和 R2 = 03H:

     CP    R1,R2 →     对 C 和 S 标志位置 1
```

目的寄存器 R1 的内容为 02H，源寄存器 R2 的内容为 03H，指令“CP R1,R2”把 R1 减去 R2，因为产生了借位，结果是负数，所以 C 和 S 被置 1。

```
2.   假设： R1 = 05H 和 R2 = 0AH:

          CP    R1,R2
          JP    UGE,SKIP
          INC   R1
     SKIP LD    R3,R1
```

在这个例子中，目的寄存器 R1 的内容是 05H，小于源寄存器 R2 的内容 0AH。指令“CP R1,R2”使得 C=“1”，JP 指令没有跳转到 SKIP 处。当执行完指令“LD R3,R1”，寄存器 R3的内容为 06H。

6.3.1.18 CPIJE - 比较，增加一，若相等跳转(Compare, Increment, and Jump on Equal)

CPIJE dst,src,RA

操作: If dst - src = "0", PC ← PC + RA
Ir ← Ir + 1

源操作数与目的操作数作比较 (相减)。若结果为“0”，则相对地址被叠加到 PC 上，从 PC 指向的新地址开始执行程序。否则，继续执行 CPIJE 后面的指令。无论哪种情况下，在执行下一条指令前，源指针增加 1。

标志位: 没有标志位受影响

格式:

				字节数	时钟周期	指令代码	寻址模式	
						(Hex)	dst	src
opc	src	dst	RA	3	12	C2	r	Ir

注释: 执行时间是 18 个时钟周期（跳转）或者 16 个时钟周期（无跳转）

编程实例:

假设： R1 = 02H, R2 = 03H, 和寄存器 03H = 02H:

```
CPIJE      R1,@R2,SKIP → R2 = 04H, PC 跳转到地址 SKIP 处
```

在这个例子中，工作寄存器 R1 的内容是 02H，工作寄存器 R2 的内容是 03H，寄存器 03H 的内容是 02H，指令“CPIJE R1,@R2,SKIP”比较@ R2 的内容 02H 和 02H，因为比较的结果是相等的，跳转到新的 PC 地址 SKIP 处。源寄存器 R2 加 1 变成 04H。

记住 CPIJE 指令跳转的地址范围须介于 +127 至 -128 之间。

6.3.1.19 CPIJNE - 比较，增加一，不等跳转(Compare, Increment, Jump on Non-Equal)

CPIJNE dst,src,RA

操作: If dst – src "0", PC ← PC + RA
Ir ← Ir + 1

源操作数与目的操作数作比较(相减)，如果结果不为“0”，则将相对地址叠加到 PC 上，从 PC 指向的新地址开始执行程序。否则，继续执行 CPIJE 后面的指令。无论在哪种情况下，在执行下一条指令前，源指针增加 1。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc src dst RA	3	12	D2	r	Ir

注释: 执行时间是 18 个时钟周期（跳转）或者 16 个时钟周期（无跳转）

编程实例:

```
假设： R1 = 02H, R2 = 03H, 和寄存器 03H = 04H:

CPIJNE     R1,@R2,SKIP → R2 = 04H, PC 跳转到地址 SKIP

在这个例子中，工作寄存器 R1 的内容是 02H，工作寄存器 R2 的内容是 03H，寄存器 03H 的内容是 04H，
指令“CPIJE R1,@R2,SKIP”比较 @R2 的内容 04H 和 02H， 因为比较的结果是不相等的，跳转到新 的
PC 地址 SKIP 处。源寄存器 R2 加 1 变成 04H。

记住 CPIJNE 指令跳转的地址范围须介于 +127 至 -128 之间。
```

6.3.1.20 DA - 十进制调整(Decimal Adjust)

DA dst

操作: dst ← DA dst

在加法或者减法运算后，将结果调整为 2 个 4 位 BCD 码。对于加法（ADD,ADC）或者减法 SUB,SBC）下面的表格指明了操作如何进行。

指令	DA前的进位标志	位7-4（十六进制）	DA前H标志	位3-0（十六进制）	要增加的数字	DA后的进位标志
	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
ADD	0	A-F	0	0-9	60	1
ADC	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
	0	0-9	0	0-9	00 = - 00	0
SUB	0	0-8	1	6-F	FA = - 06	0
SBC	1	7-F	0	0-9	A0 = - 60	1
	1	6-F	1	6-F	9A = - 66	1

标志位:

C: 如果发生进位，被置 1；否则，被清 0
Z: 如果结果为 0，被置 1；否则，被清 0
S: 如果结果是负数，被置 1；否则，被清 0
V: 没有定义
D: 不受影响
H: 不受影响

格式:

opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	4	40	R
	4	41	IR

编程实例:

假设： 工作寄存器R0的内容是15（BCD）， 工作寄存器 R1 的内容是 27（BCD）， 地址 27H 的内容是 46（BCD）：

```
ADD   R1,R0        ;     C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = C, R1 ← 3CH

DA    R1           ;     R1 ← 3CH + 06
```

如果用 BCD 数值 15 和 27 进行加法运算，结果是 27。然而结果是不对的，当加法的时候，使用的是标准的二进制运算：

```
            0 0 0 1    0 1 0 1       15
      +     0 0 1 0    0 1 1 1 27
            0 0 1 1    1 1 0 0 =     3CH
```

DA 指令调整了运算结果，进而得到正确的结果：

```
            0 0 1 1    1 1 0 0
      +     0 0 0 0    0 1 1 0
            0 1 0 0    0 0 1 0 =     42
```

假设同样的数值，指令：

```
SUB   27H,R0       ;     C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = 1
DA    @R1          ;     @R1 ← 31–0
```

执行的结果为 31（BCD），保存在地址 27H（@R1）。

6.3.1.21 DEC - 字节减 1(Decrement)

DEC dst

操作: dst ← dst - 1

目的操作数的内容减 1

标志位:
C: 不受影响
Z: 如果结果为 0，被置 1；否则，被清 0
S: 如果结果是负数，被置 1；否则，被清 0
V: 如果结果溢出，被置 1；否则，被清 0
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	00	R
		4	01	IR

编程实例:

假设： R1 = 03H 和寄存器 03H = 10H:

```
DEC  R1    → R1 = 02H
DEC  @R1   → 寄存器 03H = 0FH
```

在第一个例子中，如果工作寄存器 R1 的内容是 03H，值令“DEC R1”将该 16 进制数减 1，结果为 02H。
在第二个例子中，“DEC @R1”将数值 10H 减 1，得到 0FH，保存在地址 03H 。

6.3.1.22 DECW - 字减 1(Decrement Word)

DECW dst

操作: dst ← dst - 1

目的地址(须为偶数)的内容减 1，把操作数视为 16 位数据。

标志位:
- **C:** 不受影响
- **Z:** 如果结果为 0，被置 1；否则，被清 0
- **S:** 如果结果是负数，被置 1；否则，被清 0
- **V:** 如果有溢出，被置 1；否则，被清 0
- **D:** 不受影响
- **H:** 不受影响

格式:

opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	8	80	RR
	8	81	IR

编程实例:

```
假设： R0 = 12H, R1 = 34H, R2 = 30H, 寄存器 30H = 0FH, 和寄存器 31H = 21H:

DECW RR0   → R0 = 12H, R1 = 33H
DECW @R2   → 寄存器 30H = 0FH, 寄存器 31H = 20H

在第一个例子中, 目的寄存器 R0 内容是 12H, 寄存器 R1 内容 34H, 指令“DECW RR0”把R1, R0 当作 16
位数, 减 1 以后, 得到的结果为 33H。
```

注释: 当 DECW 指令与 Z 标志一起使用时，可能导致系统错误。为避免这个问题，推荐按照如下方法来 使用 DECW 指 令：

```
LOOP:   DECW    RR0
        LD      R2,R1
        OR      R2,R0
        JR      NZ,LOOP
```

6.3.1.23 DI - 屏蔽全局中断(Disable Interrupts)

DI

操作: SYM (0) ← 0

系统模式控制寄存器的位 0，SYM.0 被清零，将禁止全局中断。中断请求仍然会置起相应的中断悬挂标志，但 CPU 不会响应中断服务程序，因为中断处理被屏蔽了。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	8F

编程实例:

假设：SYM = 01H:

```
DI
```

如果 SYM 寄存器的值是 01H，指令 DI 将清零 SYM.0，屏蔽所有的中断处理。

在改变 IMR、中断标志位、中断源控制寄存器之前，确保 DI 状态。

6.3.1.24 DIV - 无符号除法(Unsigned Divide)

DIV dst,src

操作: dst ÷ src
dst (UPPER) ← REMAINDER
dst (LOWER) ← QUOTIENT

目的操作数（16位）除以源操作数（8位），商（8位）保存在目的操作数的低字节，
余数（8位）则保存在目的操作数的高字节。如果商 ≥ 2^8，
保存在目的操作数高低字节的商和余数是不正确的。所有的操作数都是无符号整数。

标志位:
C: 如果V标志被设置并且商在 2^8 和 2^9 -1范围之间，被置 1；否则，被清 0。
Z: 如果除数或者商 = “0”，被置 1；否则，被清 0。
S: 如果商的最高位 = “1”，被置 1；否则，被清 0。
V: 如果商 ≥ 2^8 或者除数 = “0”，被置 1；否则，被清 0。
D: 不受影响
H: 不受影响。

格式:

opc	src	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
3	26/10*	94	RR	R
	26/10*	95	RR	IR
	26/10*	96	RR	IM

注释: 如果除数为 0，运行时间需要 10 个时钟周期；其他情况下，执行时间为 26 个时钟周期。

编程实例:

```
假设： R0 = 10H, R1 = 03H, R2 = 40H, 寄存器 40H = 80H:

DIV  RR0,R2      → R0 = 03H, R1 = 40H
DIV  RR0,@R2     → R0 = 03H, R1 = 20H
DIV  RR0,#20H    → R0 = 03H, R1 = 80H

在第一个例子中，目的寄存器 RR0 的内容是 10H（R0）和 03H（R1），寄存器 R2 的内容是40H。指令
“DIV RR0,R2”将 16 位 RR0 被除数除以 8 位除数 R2，除法运算后，R0 内容为03H，R1 的内容为
40H。8 位余数保存在目的寄存器 RR0 的上半部分(R0)，商则保存在下半部分(R1)。
```

6.3.1.25 DJNZ - 减 1，如果非零，跳转(Decrement and Jump if Non-Zero)

DJNZ r,dst

操作: r ← r - 1
If r ≠ 0, PC ← PC + dst

作为计数器的工作寄存器值首先减 1，如果结果不为“0”，相对地址被累加到 PC，然后程序跳转到新的地址执行。相对地址的范围是 +127 至 -128，PC 的初始值是紧跟在 DJNZ 指令后面的指令地址。

注释： 使用 DJNZ 指令的时候，被用来做计数器的工作寄存器必须通过 SRP,SRP0,SRP1 指令设置为 0C0H~0C1H 中的一个。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
r | opc dst	2	8 (jump taken) 8 (no jump)	rA r = 0 to F	RA

编程实例:

假设: R1 = 02H 和 LOOP 是相对地址的地址标志符：

```
LOOP SRP   #0C0H
DJNZ R1,LOOP
```

指令 DJNZ 经常被用在循环程序中。大多数情况下，用地址标号而非数字相对地址来做目的操作数。 在这个例子中，工作寄存器 R1 内容是 02H，LOOP 是相对地址的标号。

指令“DJNZ R1, LOOP”首先将寄存器 R1 减 1，得到结果 01H，因为减 1 后 R1 的内容非“0”，程序将跳转到标号 LOOP 指向的地址执行。

6.3.1.26 EI - 使能全局中断(Enable Interrupts)

EI

操作: SYM (0) ← 1

EI 指令对系统模式寄存器(SYM)的最低位 SYM.0 置 1，这将允许响应中断服务程序(假定该中断具有最高优先级)。如果中断处理被屏蔽（通过执行 DI) 时，又发生中断，可通过执行EI指令，来执行中断服务程序 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
Opc	1	4	9F

编程实例:

假设：SYM = 00H:

```
EI
```

如果 SYM 寄存器的内容是 00H，也就是说，全局中断被屏蔽，执行指令“EI”将把SYM寄存器设置为 01H，使能所有中断（SYM.0 是全局中断的使能位）。

6.3.1.27 ENTER - 进入(Enter)

ENTER

操作:

SP ← SP - 2
@SP ← IP
IP ← PC
PC ← @IP
IP ← IP + 2

这条指令在执行线性代码语言时非常有用。指令指针的内容压入堆栈，然后把程序计数器(PC)的值写入指令指针，程序存储器中指令指针所指向的字数据载入PC，并且指令指针的值增加 2。

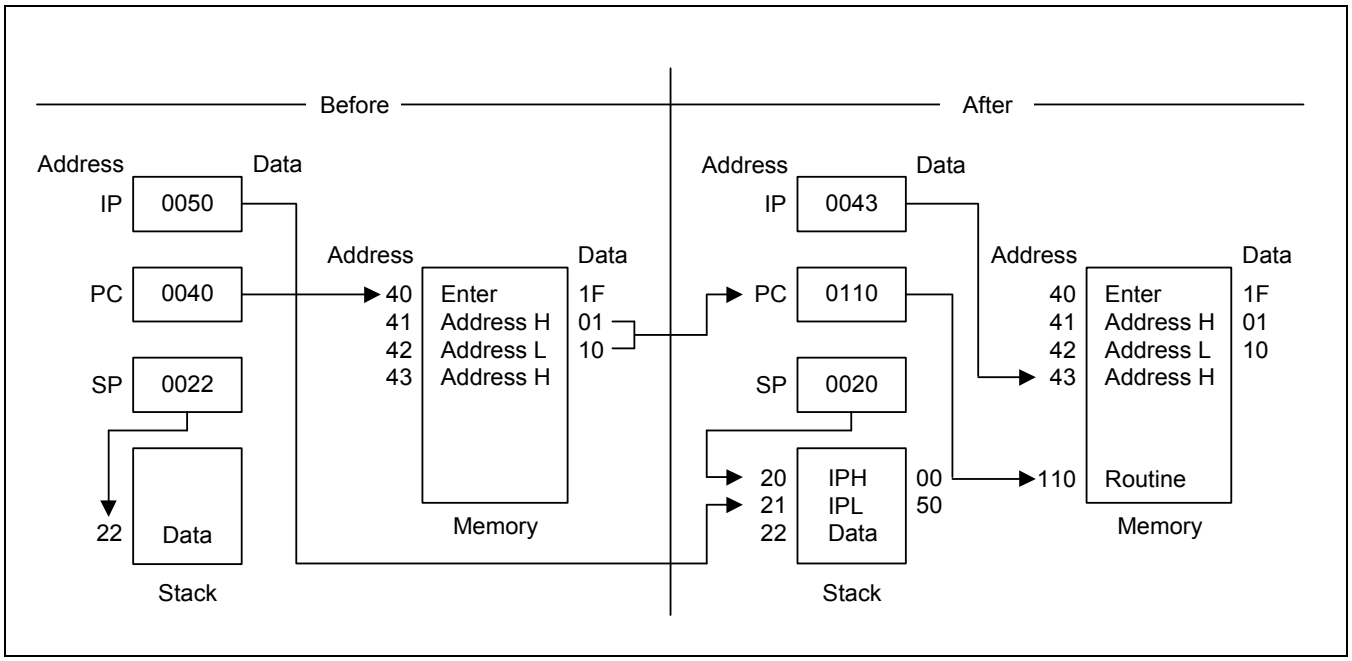
标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex) (Hex)
opc	1	14	1F

编程实例:

下图给出了一个如何使用 ENTER 指令的例子。



6.3.1.28 EXIT - 退出(Exit)

EXIT

操作:

IP ← @SP
SP ← SP + 2
PC ← @IP
IP ← IP + 2

这条指令在执行线性代码语言时非常有用。被压入堆栈的值弹出并载入指令指针，程序存储器中 被指令指针指向的字数据载入程序计数器，并且指令指针的值增加 2。

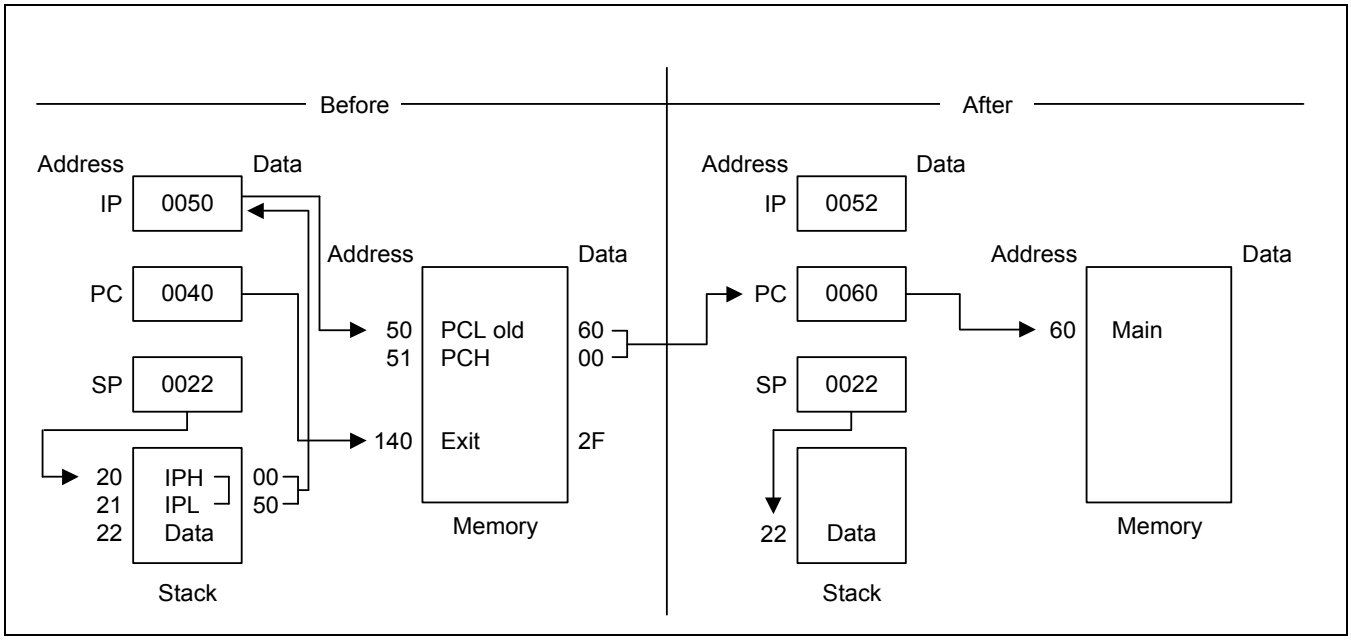
标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	14 (内部堆栈) 16 (外部堆栈)	2F

编程实例:

下图给出了一个如何使用 EXIT 指令的例子。



6.3.1.29 IDLE - 空闲指令(Idle Operation)

IDLE

操作: (见描述)

DLE 指令将停止 CPU 时钟但允许系统时钟继续工作。IDLE 模式可以被中断请求（IRQ）或者是外部复位操作唤醒。
在应用程序中，IDLE 指令后必须立即执行至少 3 个 NOP 指令。这是为了保证在下一条指令执行之前，系统时钟有足够的时间间隔来稳定时钟信号。如果在 IDLE 指令后没有 3 个或更多个的 NOP 指令，内部总线的悬浮状态将导致漏电流的产生。。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc	1	4	6F	-	-

编程实例:

```
指令
IDLE ; 停止 CPU 时钟但不停系统时钟
NOP
NOP
NOP
```

6.3.1.30 INC - 加 1(Increment)

INC dst

操作: dst ← dst + 1

目标操作数的内容增加1

标志位:
C: 没有影响
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果为负数则置 1；否则清零
V: 如果发生溢出则置 1；否则清零
D: 没有影响
H: 没有影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
dst | opc	1	4	rE r = 0 到 F	r
opc | dst	2	4	20	R
		4	21	IR

编程实例:

假如: R0 = 1BH, 寄存器 00H = 0CH, 寄存器 1BH = 0FH:

```
INC  R0    → R0 = 1CH
INC  00H   → 寄存器 00H = 0DH
INC  @R0   → R0 = 1BH, 寄存器 01H = 10H
```

在第一个例子中, 如果目标工作寄存器R0的值为1BH, 那么语句“INC R0”将 R0 的值变为1CH。

第二个例子演示了 INC 指令对寄存器 00H 产生的效果, 假定寄存器的值为 0CH。

第三个例子中, INC 指令用在间接寄存器（IR）寻址模式中, 将寄存器 1BH 的值由 0FH 变为10H。

6.3.1.31 INCW - 字加 1(Increment Word)

INCW dst

操作: dst ← dst + 1

目标操作数（必须是偶地址）中的一个字节和下一地址中的字节数据合在一起作为一个 16 位数据，整个 16 位的数据增加 1

标志位:

C: 没有影响
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果为负数则置 1；否则清零
V: 如果发生溢出则置 1；否则清零
D: 没有影响
H: 没有影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc dst	2	8	A0	RR
		8	A1	IR

编程实例:

```
假如: R0  = 1AH, R1  = 02H, 寄存器 02H  = 0FH, 寄存器 03H  = 0FFH:

INCW RR0    → R0  = 1AH, R1  = 03H
INCW @R1    → 寄存器 02H  = 10H, 寄存器 03H  = 00H
```

在第一个例子中，工作寄存器对 RR0 的内容是 1AH(R0) 和 02H(R1)。语句“INCW RR0”将16 位的目标数据加 1，使寄存器 R1 的值变为 03H。第二个例子中，语句“INCW @R1”用间接寄存器（IR）寻址模式将寄存器 03H 的值从 0FFH 变为 00H，寄存器 02H 的值从 0FH 变为 10H。

注释: 如果 Zero（Z）标志位（标志位.6）与 INCW 指令一起使用，可能会发生冲突而导致系统错误。为了避免这个问题，建议按下面的方法使用 INCW 指令：

```
LOOP:   INCW    RR0
        LD      R2,R1
        OR      R2,R0
        JR      NZ,LOOP
```

6.3.1.32 IRET - 中断返回(Interrupt Return)

IRET IRET (正常) IRET (快速)

操作:

标志位	←	@SP	PC	↔	IP
SP	←	SP + 1	标志位	←	标志位'
PC	←	@SP	FIS	←	0
SP	←	SP + 2			
SYM(0)	←	1			

该指令用在中断服务程序的末尾。该指令将恢复标志寄存器和程序计数器的值，同时重新使能全局中断。只有在快速中断状态位（FIS，标志位寄存器位 1，0D5H）被清零时（=“0”），才执行“正常IRET”。快速中断产生时，对于在中断服务程序开始时被置 1 的 FIS 位，IRET 会将其清零 。

标志位: 所有标志位恢复到初始状态（即中断发生以前的状态）

格式:

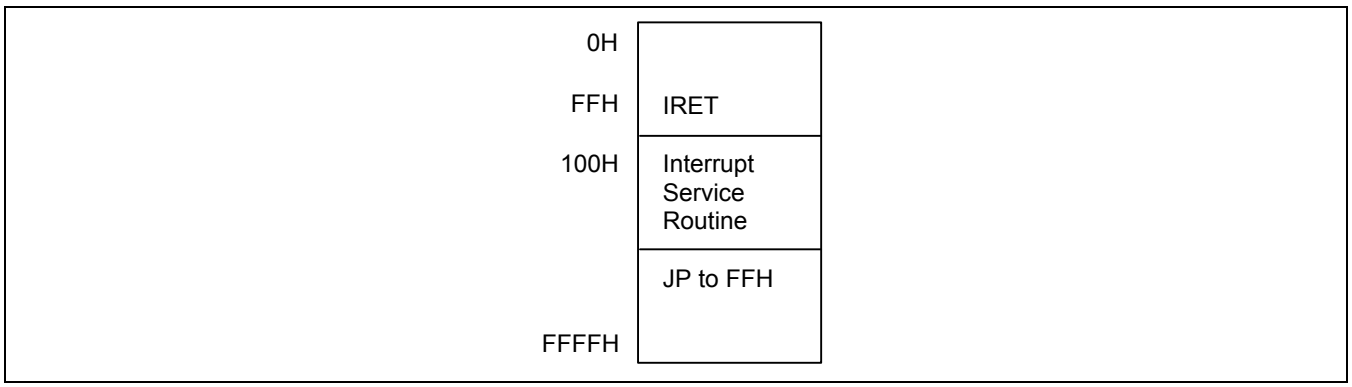
IRET (正常)	字节数	时钟周期	指令代码 (Hex)
Opc	1	10 (内部堆栈) 12 (外部堆栈)	BF

IRET (快速)	字节数	时钟周期	指令代码 (Hex)
opc	1	6	BF

编程实例:

在下图中，中断使能之前，在主程序中将指令指针初始化为 100H。中断发生时，程序计数器和指令指针的内容相交换，这使得 PC 跳转到地址 100H 而IP则保存返回地址。中断服务程序的最后一条指令通常是跳转到 地址 FFH 处的 IRET。

这将使指令指针“重新”被赋值为 100H，且程序计数器跳回到主程序。现在，下一个中断可以发生了，IP 寄存器的值仍然为 100H 。



注释: 上面快速中断的例子中，如果最后的指令不是跳转到 IRET，那么必须注意最后两条指令的顺序。在 IRET 指令之后，不可紧跟用于中断状态清除的指令（例如 IPR 寄存器清 0）。

6.3.1.33 JP - 跳转(Jump)

JP cc,dst (条件跳转)

JP dst (无条件跳转)

操作: 如果 cc 为真，PC ← dst

如果转移条件为真，那么条件跳转指令把系统控制交给目标地址；
否则执行紧跟 JP 指令之后的指令。无条件跳转指令只是简单的用目标地址替换 PC 的内容，然后程序控制交给由 PC 指定的语句。

标志位: 没有标志位受影响

格式: (1)

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
(2) cc | opc dst	3	8	ccD cc = 0 to F	DA
opc dst	2	8	30	IRR

注释:

1. 3 字节格式用于条件跳转，2 字节格式用于无条件跳转
2. 在 3 字节指令格式（条件跳转）的第一字节中，条件码和指令代码各占 4 位

编程实例:

假如: 进位标志 (C) = "1", 寄存器 00 = 01H, 寄存器 01 = 20H:

```
JP   C,LABEL_W   → LABEL_W = 1000H, PC = 1000H
JP   @00H        → PC = 0120H
```

第一个例子是条件跳转 JP。假定进位标志为 "1"，指令 "JP C,LABEL_W" 将 PC 的值替换为1000H 并且跳转到该地址。如果标志位没有被置 1，那么程序将立即转到紧跟 JP 后的那条指令。

第二个例子为无条件跳转 JP。指令 "JP @00" 将 PC 的内容替换为寄存器对 00H 和 01H 的值，亦即 0120H。

6.3.1.34 JR - 相对跳转指令(Jump Relative)

JR cc,dst

操作: 如果 cc 为真，PC ← PC + dst

如果转移条件为真，那么程序计数器值加上相对地址，并把程序控制转到该地址处的指令；否则执行紧跟 JR 指令后的那条指令（见本章：转移条件码列表）。

相对地址的范围是 -128 ~ +127，并且程序计数器的初值被认为是紧跟JR指令之后的第一条指令地址。

标志位: 没有标志位受影响

格式:

(1)

cc | opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	6	ccB cc = 0 到 F	RA

注释: 2 字节指令格式中的第一个字节中，条件码和指令代码各占 4 位

编程实例:

假如： 进位标志位 =“1”并且 LABEL_X = 1FF7H:

```
JR   C,LABEL_X   → PC = 1FF7H
```

如果进位标志位为“1”（也就是，转移条件为真），指令“JR C,LABEL_X”会将程序控制交给当前 PC 指向的地址；否则，执行紧跟 JR 后的程序指令 。

6.3.1.35 LD - 传送数据(Load)

LD dst,src

操作: dst ← src

源操作数的内容将赋给目标操作数。源操作数的内容不受影响。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
dst | opc / src	2	4	rC	r	IM
		4	r8	r	R
src | opc / dst	2	4	r9	R	r
			r = 0 到 F		
opc / dst | src	2	4	C7	r	Ir
		4	D7	Ir	r
opc / src / dst	3	6	E4	R	R
		6	E5	R	IR
opc / dst / src	3	6	E6	R	IM
		6	D6	IR	IM
opc / src / dst	3	6	F5	IR	R
opc / dst | src / x	3	6	87	r	x [r]
opc / src | dst / x	3	6	97	x [r]	r

编程实例:

```
假如: R0  = 01H, R1  = 0AH, 寄存器 00H  = 01H, 寄存器 01H  = 20H,
寄存器 02H  = 02H, LOOP = 30H, 寄存器 3AH  = 0FFH:

LD   R0,#10H          →     R0  =  10H
LD   R0,01H           →     R0  =  20H, 寄存器 01H  =  20H
LD   01H,R0           →     寄存器 01H  =  01H, R0  =  01H
LD   R1,@R0           →     R1  =  20H, R0  =  01H
LD   @R0,R1           →     R0  =  01H, R1  =  0AH, 寄存器 01H  =  0AH
LD   00H,01H          →     寄存器 00H  =  20H, 寄存器 01H  =  20H
LD   02H,@00H         →     寄存器 02H  =  20H, 寄存器 00H  =  01H
LD   00H,#0AH         →     寄存器 00H  =  0AH
LD   @00H,#10H        →     寄存器 00H  =  01H, 寄存器 01H  =  10H
LD   @00H,02H         →     寄存器 00H  =  01H, 寄存器 01H  =  02, 寄存器 02H  =  02H
LD   R0,#LOOP[R1]     →     R0  =  0FFH, R1  =  0AH
LD   #LOOP[R0],R1     →     寄存器 31H  =  0AH, R0  =  01H, R1  =  0AH
```

6.3.1.36 LDB - 传送位数据(Load Bit)

LDB dst,src.b

LDB dst.b,src

操作: dst(0) ← src(b)
或
dst(b) ← src(0)

源操作数的指定位载入目标操作数的最低位，或者源操作数的最低位载入目标操作数的指定位。目标操作数的其它位都不受影响，源操作数也不受影响。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | b | 0 | src	3	6	47	r0	Rb
opc | src | b | 1 | dst	3	6	47	Rb	r0

注释: 指令格式的第二个字节中，目标（或源）地址占 4 位，位地址“b”占 3 位，LSB 地址值占 1 位

编程实例:

假如：R0 = 06H，通用寄存器 00H = 05H：

```
LDB  R0,00H.2    → R0 = 07H, 寄存器 00H = 05H
LDB  00H.0,R0    → R0 = 06H, 寄存器 00H = 04H
```

第一个例子中，目标工作寄存器 R0 的值为 06H，源寄存器 00H 的值为 05H。指令“ R0,00h.2”将寄存器 00H 中的第二位（bit 2）载入寄存器 R0 的最低位，R0 的值变为 07H。

第二个例子中，目标寄存器是 00H。指令“LD 00H.0,R0”将工作寄存器 R0 的最低位载入目标寄存器 00H 的指定位（bit 0），寄存器 00H 的值变为 04H。

6.3.1.37 LDC/LDE - 传送程序/外部数据存储器数据(Load Memory)

LDC/LDE dst,src

操作: dst ← src

这条指令从程序或外部数据存储器中装载一个字节数据到工作寄存器，或者相反。
源操作数不受影响。指令 LDC 用作程序存储器，LDE 用作外部数据存储器。对于程序存储器，编译器将“Irr”或“rr”的值编译成偶地址，而对于外部数据存储器则编译成奇地址 。

标志位: 没有标志位受影响

格式:

		字节数	时钟周期	指令代码	寻址模式	
				(Hex)	dst	src
1.	opc | dst | src	2	10	C3	r	Irr
2.	opc | src | dst	2	10	D3	Irr	r
3.	opc | dst | src | XS	3	12	E7	r	XS [rr]
4.	opc | src | dst | XS	3	12	F7	XS [rr]	r
5.	opc | dst | src | XL_L | XL_H	4	14	A7	r	XL [rr]
6.	opc | src | dst | XL_L | XL_H	4	14	B7	XL [rr]	r
7.	opc | dst | 0000 | DA_L | DA_H	4	14	A7	r	DA
8.	opc | src | 0000 | DA_L | DA_H	4	14	B7	DA	r
9.	opc | dst | 0001 | DA_L | DA_H	4	14	A7	r	DA
10.	opc | src | 0001 | DA_L | DA_H	4	14	B7	DA	r

注释:

1. 格式 5 和 6 的源操作数 [src] 或工作寄存器对 [rr] 不能使用工作寄存器对 0-1
2. 格式 3 和 4 的目标地址“XS[rr]”和源地址“XS[rr]”均为一个字节
3. 格式 3 和 4 的目标地址“XL[rr]”和源地址“XL[rr]”均为两个字节
4. 格式 7 和 8 的 DA 和 r 源操作数值用于访问程序存储器，格式 9 和 10 则用于访问外部数据存储器
5. LDE 指令可用于读/写 64K 字节的外部数据存储器。

编程实例:

```
假如： R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H; 程序存储器空间中
0103H = 4FH, 0104H = 1A, 0105H = 6DH, 1104H = 88H. 外部数据存储空间中
0103H = 5FH, 0104H = 2AH, 0105H = 7DH, 1104H = 98H:

LDC         R0,@RR2             ; R0 ← 程序存储器地址 0104H 的内容
                                ; R0 = 1AH, R2 = 01H, R3 = 04H
LDE         R0,@RR2             ; R0 ← 外部数据存储器地址 0104H 的内容
                                ; R0 = 2AH, R2 = 01H, R3 = 04H
LDC (注释)  @RR2,R0             ; 11H (R0 的内容) 载入程序存储器地址 0104H (RR2)
                                ; 工作寄存器 R0, R2, R3 → 没有变化
LDE         @RR2,R0             ; 11H (R0 的内容) 载入外部数据存储器地址 0104H (RR2)
                                ; 工作寄存器 R0, R2, R3 → 没有变化
LDC         R0,#01H[RR2]        ; R0 ← 程序存储器地址 0105H 的内容
                                ; (01H + RR2),
                                ; R0 = 6DH, R2 = 01H, R3 = 04H
LDE         R0,#01H[RR2]        ; R0 ← 外部数据存储器地址 0105H 的内容
                                ; (01H + RR2), R0 = 7DH, R2 = 01H, R3 = 04H
LDC (注释)  #01H[RR2],R0        ; 11H (R0 的内容) 载入程序存储器地址
                                ; 0105H (01H + 0104H)
LDE         #01H[RR2],R0        ; 11H (R0 的内容) 载入外部数据存储器地址
                                ; 0105H (01H + 0104H)
LDC         R0,#1000H[RR2]      ; R0 ← 程序存储器地址 1104H 的内容
                                ; (1000H + 0104H), R0 = 88H, R2 = 01H, R3 = 04H
LDE         R0,#1000H[RR2]      ; R0 ← 外部数据存储器地址 1104H 的内容
                                ; (1000H + 0104H), R0 = 98H, R2 = 01H, R3 = 04H
LDC         R0,1104H            ; R0 ← 程序存储器地址 1104H 的内容, R0 = 88H
LDE         R0,1104H            ; R0 ← 外部数据存储器地址 1104H 的内容
                                ; R0 = 98H
LDC (注释)  1105H,R0            ; 11H (R0 的内容) 载入程序存储器地址
                                ; 1105H, (1105H) ← 11H

LDE         1105H,R0            ; 11H (R0 的内容) 载入外部数据存储器地址
                                ; 1105H, (1105H) ← 11H
```

注释: 掩膜 ROM 类型的器件不支持 LDC/LDE 指令。

6.3.1.38 LDCD/LDED - 传送数据之后地址减 1(Load Memory and Decrement)

LDCD/LDED dst,src

操作: dst ← src
rr ← rr - 1

该指令用于用户栈或在程序/数据存储器与寄存器卷之间批量传送数据。
存储器的地址由工作寄存器对指定。源地址的内容载入目标地址，然后存储器地址自动减1，
源操作数的内容不变。

LDCD 对应程序存储器，而 LDED 对应外部数据存储器。对于程序存储器，
编译器将“Irr”或“rr”的值编译成偶地址， 对于数据存储器则编译成奇地址 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	10	E2	r	Irr

编程实例:

```
假如： R6 = 10H, R7 = 33H, R8 = 12H, 程序存储器地址 1033H = 0CDH, 外部数据存储地址
1033H = 0DDH:

LDCD R8,@RR6      ; 0CDH (程序存储器 1033H 的内容) 载入R8
                  ; RR6 减1
                  ; R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 - 1)

LDED R8,@RR6      ; 0DDH (数据存储器地址 1033H 的内容) 载入R8
                  ; RR6 减1 (RR6 ← RR6 - 1)
                  ; R8 = 0DDH, R6 = 10H, R7 = 32H
```

6.3.1.39 LDCI/LDEI - 传送数据后地址加 1(Load Memory and Increment)

LDCI/LDEI dst,src

操作: dst ← src
rr ← rr + 1

该指令用于用户栈或在程序/数据存储器与寄存器卷之间批量传送数据。存储器的地址由工作寄存器对指定。源地址的内容载入目标地址，然后存储器地址自动加 1，源操作数的内容不变。

LDCI 对应程序存储器，而 LDEI 对应外部数据存储器。 对于程序存储器，
编译器将“Irr”或“rr”的值编译成偶地址，对于数据存储器则编译成奇地址。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	10	E3	r	Irr

编程实例:

```
假如： R6  = 10H, R7  = 33H, R8  = 12H, 程序存储器地址 1033H  = 0CDH, 1034H  = 0C5H;
      外部数据存储器地址 1033H  = 0DDH, 1034H  = 0D5H:

LDCI R8,@RR6      ; 0CDH (程序存储器地址 1033H 的内容) 载入R8
                  ; RR6 加1 (RR6  ←  RR6 + 1)
                  ; R8  =  0CDH, R6  =  10H, R7  =  34H

LDEI R8,@RR6      ; 0DDH (数据存储器地址 1033H 的内容) 载入R8
                  ; RR6 加1 (RR6  ←  RR6 + 1)
                  ; R8  =  0DDH, R6  =  10H, R7  =  34H
```

6.3.1.40 LDCPD/LDEPD - 传送数据前地址减 1(Load Memory with Pre-Decrement)

LDCPD/LDEPD dst,src

操作: rr ← rr – 1
dst ← src

该指令用于用户栈或在程序/数据存储器和寄存器卷之间批量传送数据。存储器的地址由工作寄存器对指定并首先减 1。之后源地址的内容送入目标地址，源操作数的内容不变。

LDCPD 对应程序存储器，LDEPD 对应外部数据存储器。对于程序存储器，编译器将“Irr”或“rr”的值编译成偶地址，而对于数据存储器则编译成奇地址 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | src | dst	2	14	F2	Irr	r

编程实例:

```
假如： R0 = 77H, R6 = 30H, R7 = 00H:

LDCPD      @RR6,R0      ; (RR6 ← RR6 - 1)
                        ; 77H (R0 的内容) 载入程序存储器地址 2FFFH (3000H - 1H)
                        ; R0 = 77H, R6 = 2FH, R7 = 0FFH

LDEPD      @RR6,R0      ; (RR6 ← RR6 - 1)
                        ; 77H (R0 的内容) 载入外部数据存储器地址 2FFFH (3000H - 1H)
                        ; R0 = 77H, R6 = 2FH, R7 = 0FFH
```

6.3.1.41 LDCPI/LDEPI - 传送数据前地址加 1(Load Memory with Pre-Increment)

LDCPI/LDEPI dst,src

操作: rr ← rr + 1
dst ← src

该指令用于用户栈或在程序/数据存储器与寄存器卷之间批量传送数据。存储器的地址由工作寄存器对指定并首先加 1。之后源地址的内容载入目标地址，源操作数的内容不变。

LDCPI 对应程序存储器，LDEPI 对应外部数据存储器。对于程序存储器，编译器将“Irr”或“rr”的值编译成偶地址，而对于数据存储器则编译成奇地址 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | src | dst	2	14	F3	Irr	r

编程实例:

```
假如： R0 = 7FH, R6 = 21H, R7 = 0FFH:

LDCPI      @RR6,R0      ; (RR6 ← RR6 + 1)
                        ; 7FH (R0 的内容) 载入程序存储器地址 2200H (21FFH + 1H)
                        ; R0 = 7FH, R6 = 22H, R7 = 00H

LDEPI      @RR6,R0      ; (RR6 ← RR6 + 1)
                        ; 7FH (R0 的内容) 载入外部数据存储器地址 2200H (21FFH + 1H)
                        ; R0 = 7FH, R6 = 22H, R7 = 00H
```

6.3.1.42 LDW - 传送字数据(Load Word)

LDW dst,src

操作: dst ← src

源操作数的内容（字）载入目标操作数。源操作数内容不变。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | src | dst	3	8	C4	RR	RR
		8	C5	RR	IR
opc | dst | src	4	8	C6	RR	IML

编程实例:

假如： R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, 寄存器 00H = 1AH,
寄存器 01H = 02H, 寄存器 02H = 03H, 寄存器 03H = 0FH:

```
LDW         RR6,RR4      →     R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH
LDW         00H,02H      →     寄存器 00H = 03H, 寄存器 01H = 0FH,
                               寄存器 02H = 03H, 寄存器 03H = 0FH
LDW         RR2,@R7      →     R2 = 03H, R3 = 0FH,
LDW         04H,@01H     →     寄存器 04H = 03H, 寄存器 05H = 0FH
LDW         RR6,#1234H   →     R6 = 12H, R7 = 34H
LDW         02H,#0FEDH   →     寄存器 02H = 0FH, 寄存器 03H = 0EDH
```

第二个例子中，请注意指令“LDW 00H,02H”将源寄存器 02H 和 03H 中的内容载入目标寄存器00H和01H，使通用寄存器 00H 中的值变为 03H，01H 中的值变为 0FH。

其它的例子演示了如何通过不用的寻址模式和格式来使用 LDW 指令。

6.3.1.43 MULT - 无符号数乘法(Unisigned Multiply)

MULT dst,src

操作: dst ← dst × src

8 位目标操作数（寄存器对中的偶地址寄存器）与源操作数（8位）相乘，乘积（16位）保存在目标地址指定的寄存器对中。两个操作数都为无符号整型数据。

标志位:

C: 如果结果 >255 则置 1；否则清零
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果的最高为“1”则置1；否则清零
V: 清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | src | dst	3	22	84	RR	R
		22	85	RR	IR
		22	86	RR	IM

编程实例:

```
假如： 寄存器 00H = 20H, 寄存器 01H = 03H, 寄存器02H = 09H, 寄存器 03H = 06H:

MULT 00H, 02H    → 寄存器 00H = 01H, 寄存器 01H = 20H, 寄存器 02H = 09H
MULT 00H, @01H   → 寄存器 00H = 00H, 寄存器 01H = 0C0H
MULT 00H, #30H   → 寄存器 00H = 06H, 寄存器 01H = 00H

第一个例子中，指令“MULT 00H,02H”将 8 位目标操作数（寄存器对 00H，01H 中的 00H）与源寄存器
02H 中的操作数（09H）相乘。16 位的乘积，0120H，保存在寄存器对 00H，01H中。
```

6.3.1.44 NEXT - Next 指令

NEXT

操作: PC ← @ IP

IP ← IP + 2

NEXT 指令在执行线性代码语言时非常有用。程序存储器中指令指针所指向的字送入 PC，并且指令指针增加 2。

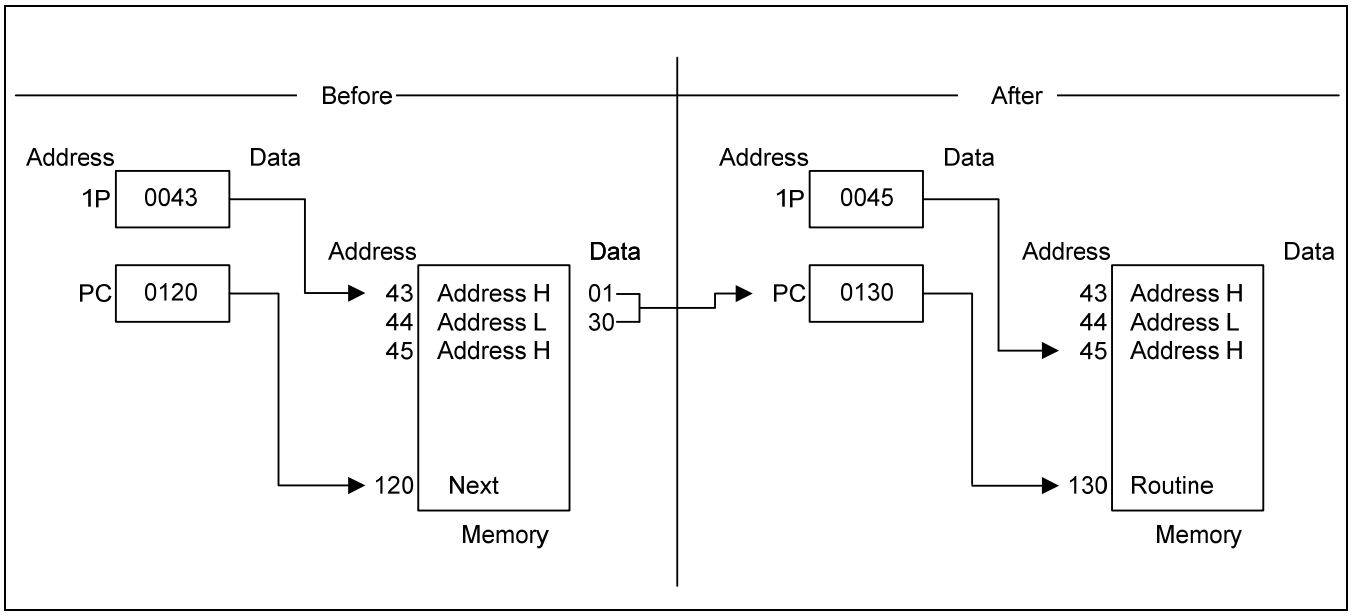
标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	10	0F

编程实例:

下图是一个如何使用 NEXT 指令的例子。



6.3.1.45 NOP - 空操作(No Operation)

NOP

操作: CPU 执行这条指令时，不做任何操作。通常用顺序执行一个或多个 NOP 指令来实现一定时长的延时。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	FF

编程实例:

在程序中执行 NOP 指令时，没有任何操作发生，只是一个指令执行时间的延时。

6.3.1.46 OR - 逻辑或(Logical OR)

OR dst,src

操作: dst ← dst OR src

源操作数与目标操作数进行逻辑或，结果存放在目标操作数。源操作数的值不受影响。
只要两个操作数中任一个的相应位为“1”，那么或的结果就是“1”，否则为“0”。

标志位:

C: 不受影响
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 总是清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	42	r	r
		6	43	r	lr
opc | src | dst	3	6	44	R	R
		6	45	R	IR
opc | dst | src	3	6	46	R	IM

编程实例:

假如：R0 = 15H, R1 = 2AH, R2 = 01H, 寄存器 00H = 08H, 寄存器 01H = 37H, 寄存器 08H = 8AH:

```
OR   R0,R1       →    R0 = 3FH, R1 = 2AH
OR   R0,@R2      →    R0 = 37H, R2 = 01H, 寄存器 01H = 37H
OR   00H,01H     →    寄存器 00H = 3FH, 寄存器 01H = 37H
OR   01H,@00H    →    寄存器 00H = 08H, 寄存器 01H = 0BFH
OR   00H,#02H    →    寄存器 00H = 0AH
```

第一个例子中，如果工作寄存器 R0 的值是 15H，寄存器 R1 的值是 2AH，指令“OR R0,R1”将寄存器 R0 和 R1 的内容进行逻辑或，并将结果（3FH）存在目标寄存器 R0。

其他的例子演示了如何通过不同的寻址模式和格式来使用逻辑或指令。

6.3.1.47 POP - 出栈(Pop from Stack)

POP dst

操作: dst ← @SP
SP ← SP + 1

堆栈指针指定地址的内容被装入目标操作数，然后堆栈指针加1。

标志位: 没有标志位受影响

格式:

opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	8	50	R
	8	51	IR

编程实例:

假如：寄存器 00H = 01H, 寄存器 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, 堆栈寄存器 0FBH = 55H:

```
POP  00H   →    寄存器 00H = 55H, SP = 00FCH
POP  @00H  →    寄存器 00H = 01H, 寄存器 01H = 55H, SP = 00FCH
```

第一个例子中，通用寄存器 00H 的值为 01H。指令“POP 00H”将地址 00FBH 中的值（55H）送入目标寄存器 00H，然后堆栈指针加 1。寄存器 00H 的值变为 55H，并且 SP 指向地址 00FCH。

6.3.1.48 POPUD - 弹出用户栈(自减)(Pop User Stack(Drementing))

POPUD dst,src

操作: dst ← src
IR ← IR – 1

该指令用于寄存器卷中的用户自定义栈。用户栈指针指定地址的内容被载入目标寄存器，然后用户栈指针减 1。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc src dst	3	8	92	R	IR

编程实例:

假如： 寄存器 00H = 42H (用户栈指针寄存器), 寄存器 42H = 6FH, 寄存器 02H = 70H:

POPUD 02H,@00H → 寄存器 00H = 41H, 寄存器 02H = 6FH, 寄存器 42H = 6FH

如果通用寄存器 00H 的值为42H, 寄存器 42H 的值为 6FH, 那么指令“POPUD 02H,@00H”将寄存器 42H 的内容载入目标寄存器 02H, 然后用户栈指针减 1, 变成 41H。

6.3.1.49 POPUI - 弹出用户栈(自增)(Pop User Stack(Incrementing))

POPUI dst,src

操作: dst ← src
IR ← IR + 1

该指令用于寄存器卷中的用户自定义栈。用户栈指针指定地址的内容载入目标寄存器，然后用户栈指针加1。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc src dst	3	8	93	R	IR

编程实例:

假如： 寄存器 00H = 01H, 寄存器 01H = 70H:

```
POPUI    02H,@00H    →    寄存器 00H = 02H, 寄存器 01H = 70H, 寄存器 02H = 70H
```

如果通用寄存器 00H 的值为 01H，寄存器 01H 的值为 70H，那么语句“POPUI 02H,@00H”将值70H载入目标通用寄存器 02H，然后用户栈指针（寄存器00H）加1，从 01H 变为 02H。

6.3.1.50 PUSH - 压栈(Push to Stack)

PUSH src

操作: SP ← SP – 1
@SP ← src

PUSH 指令先将栈指针减 1，然后再将源操作数的内容载入减 1 后的栈地址。此操作将在栈顶置入新的数值。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc src	2	8 (内部时钟)	70	R
		8 (外部时钟)		
		8 (内部时钟)		
		8 (外部时钟)	71	IR

编程实例:

```
假如： 寄存器 40H = 4FH, 寄存器 4FH = 0AAH, SPH = 00H, SPL = 00H:

PUSH 40H    →    寄存器 40H = 4FH, 堆栈寄存器 0FFH = 4FH,
                 SPH = 0FFH, SPL = 0FFH
PUSH @40H   →    寄存器 40H = 4FH, 寄存器 4FH = 0AAH,
                 堆栈寄存器 0FFH = 0AAH, SPH = 0FFH, SPL = 0FFH

第一个例子中，如果栈指针包含的内容为 0000H，通用寄存器 40H 的内容为 4FH，语句“PUSH 40H”使栈
指针从 0000H 减为 0FFFFH，然后将寄存器 40H 的值载入地址 0FFFFH 中， 即将这个新的数值加入栈顶。
```

6.3.1.51 PUSHUD - 压入用户栈(自减) Push User Stack(Decrementing)

PUSHUD dst,src

操作: IR ← IR - 1
dst ← src

该指令用于寄存器卷中的用户自定义栈。
PUSHUD 先将用户栈指针减 1，然后将源操作数的内容载入减1后的栈指针指向的寄存器。

标志位: 没有标志位受影响

格式:

			字节数	时钟周期	指令代码	寻址模式	
					(Hex)	dst	src
opc	dst	src	3	8	82	IR	R

编程实例:

假如： 寄存器 00H = 03H, 寄存器 01H = 05H, 寄存器 02H = 1AH:

```
PUSHUD    @00H,01H    →    寄存器 00H = 02H, 寄存器 01H = 05H, 寄存器 02H = 05H
```

如果用户栈指针（例如寄存器 00H）的值为 03H，语句“PUSHUD @00H,01H”使用户栈指针减1，变为 02H。而寄存器 01H 的值，05H，则载入减 1 后的用户栈指针所指向的寄存器。

6.3.1.52 PUSHUI - 压入用户栈(自增) Push User Stack(Incrementing)

PUSHUI dst,src

操作: IR ← IR + 1

dst ← src

该指令用于寄存器卷中的用户自定义栈。PUSHUI 先将用户栈指针加 1，然后将源操作数的内容载入加1后的栈指针指向的寄存器。

标志位: 没有标志位受影响

格式:

			字节数	时钟周期	指令代码 (Hex)	寻址模式 dst	寻址模式 src
opc	dst	src	3	8	83	IR	R

编程实例:

假如： 寄存器 00H = 03H, 寄存器 01H = 05H, 寄存器 04H = 2AH:

```
PUSHUI    @00H,01H    →    寄存器 00H = 04H, 寄存器 01H = 05H, 寄存器 04H = 05H
```

如果用户栈指针（例如寄存器 00H）的值为 03H，语句“PUSHUI @00H,01H”使用户栈指针加1，变为 04H。而寄存器 01H 的值，05H，则载入加 1 后的用户栈指针所指向的寄存器。

6.3.1.53 RCF - C 清 0(Reset Carry Flag)

RCF RCF

操作: C ← 0

进/借位标志位被无条件清零。

标志位: **C:** 清除为“0”.

其他标志位都不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	CF

编程实例:

假如：C =“1”或 “0”：

指令 RCF 将进/借位标志位（C）清零。

6.3.1.54 RET - 子程序返回(Return)

RET

操作: PC ← @SP
SP ← SP + 2

RET 指令通常被用来从 CALL 指令所调用的子程序中返回。栈指针指向的地址内容被弹出到程序计数器中，下一条要执行的指令地址由程序计数器的新值指定 。

标志位: 没有标志位被影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	8 (内部堆栈) 10 (外部堆栈)	AF

编程实例:

假如： SP = 00FCH, (SP) = 101AH, PC = 1234:

```
RET          →     PC = 101AH, SP = 00FEH
```

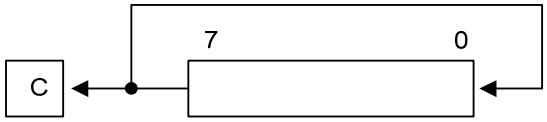
指令“RET”将堆栈指针 00FCH 中的内容（10H）弹出到程序计数器的高字节中，00FDH 中的内容（1AH）弹出到 PC 的低字节，地址 101AH 中的指令被执行。堆栈指针现在指向地址 00FEH 。

6.3.1.55 RL - 左移(Rotate Left)

RL dst

操作: C ← dst (7)
dst (0) ← dst (7)
dst (n + 1) ← dst (n), n = 0-6

目标操作数的内容左移一位，原来的第 7 位（bit 7）移到第0位（LSB）和C标志位中，如下图示：



标志位:
C: 如果从最高位（bit 7）移出的数为“1”，则置 1
Z: 如果结果为“0”则置1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 如果发生溢出则置 1；否则清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	90	R
		4	91	IR

编程实例:

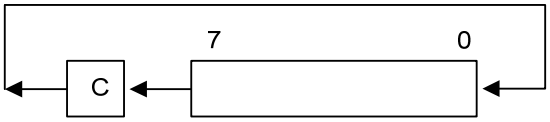
```
假如： 寄存器 00H = 0AAH, 寄存器 01H = 02H, 寄存器 02H = 17H:

RL   00H   → 寄存器 00H = 55H, C ="1"
RL   @01H  → 寄存器 01H = 02H, 寄存器 02H = 2EH, C ="0"

第一个例子中, 如果通用寄存器 00H 的值为 0AAH（10101010B）, 语句“RL 00H”将 0AAH 左移一位,
变为 55H（01010101B）, 并将进位和溢出标志置 1。
```

6.3.1.56 RLC - 带进位左移(Rotate Left Through Carry)

RLC dst

操作: dst (0) ← C
C ← dst (7)
dst (n + 1) ← dst (n), n = 0-6

目标操作数的内容通过 C 标志位左移一位，原来的第 7 位（bit 7）移到 C 标志位中，原来的 C 标志位则移至第 0 位（LSB）。



标志位:

C: 如果从最高位（bit 7）移出的为“1”则置 1
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 如果发生溢出，也就是目标操作数的符号在移位中发生改变，则置 1；否则清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	10	R
		4	11	IR

编程实例:

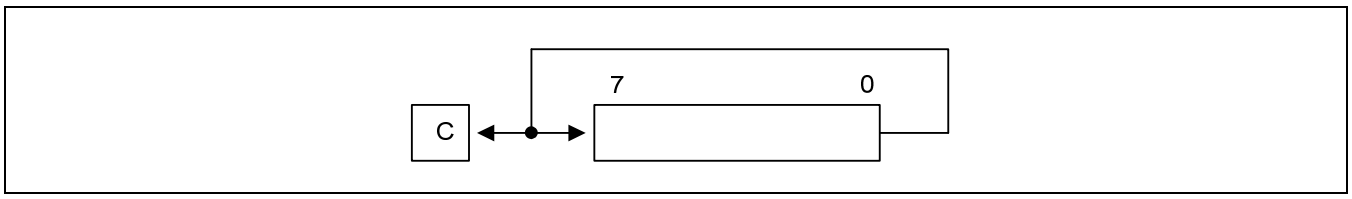
```
假如： 寄存器 00H = 0AAH, 寄存器 01H = 02H, 寄存器 02H = 17H, C ="0":

RLC  00H   →     寄存器 00H = 54H, C = "1"
RLC  @01H  →     寄存器 01H = 02H, 寄存器 02H = 2EH, C = "0"

第一个例子中，如果通用寄存器的值为 0AAH（10101010B），语句“RLC 00H”将 0AAH 左移一位。 原来的第 7 位（bit 7）将进位标志置 1，C 标志位原来的值则移至寄存器 00H 的第 0 位（LSB）， 使寄存器的值变为 55H（01010101B）。寄存器 00H 的最高位 MSB 重新置 C 标志位为“1”， 并且将溢出标志位置 1。
```

6.3.1.57 RR - 右移(Rotate Right)

RR dst

操作: C ← dst (0)
dst (7) ← dst (0)
dst (n) ← dst (n + 1), n = 0-6

目标操作数的内容右移一位，原来的第 0 位（LSB）移到第 7 位（MSB）和 C 标志位中。



标志位:
C: 如果从第 0 位（LSB）移出的为“1”，则置 1
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 如果发生溢出，也就是目标操作数的符号在移位中发生改变，则置 1；否则清零
D: 不受影响
H: 不受影响

格式:

opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	4	E0	R
	4	E1	IR

编程实例:

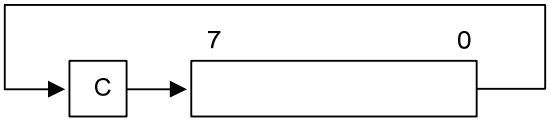
```
假如： 寄存器 00H = 31H, 寄存器 01H = 02H, 寄存器 02H = 17H:

RR   00H   → 寄存器 00H = 98H, C ="1"
RR   @01H  → 寄存器 01H = 02H, 寄存器 02H = 8BH, C ="1"

第一个例子中，如果通用寄存器 00H 的值为 31H（00110001B），语句“RR 00H”将 00H 的值右移一位，
原来第 0 位移至第 7 位，目标寄存器的值变为 98H（10011000B）。原来的第 0 位将 C 标志位置“1”， 同
时符号标志和溢出标志也被置 “1”。
```

6.3.1.58 RRC - 带进位右移(Rotate Right Through Carry)

RRC dst

操作: dst (7) ← C
C ← dst (0)
dst (n) ← dst (n + 1), n = 0-6

目标操作数的内容通过 C 标志位右移一位，原来的第 0 位（LSB）移到C标志位中，原来的 C 标志位则移至第 7 位（MSB）。



标志位:
C: 如果从最低位（bit 0）移出的为“1”，则置 1
Z: 如果结果为“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 如果发生溢出，也就是说目标操作数的符号在移位中发生改变，则置 1；否则清零
D: 不受影响
H: 不受影响

格式:

opc	dst

字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
2	4	C0	R
	4	C1	IR

编程实例:

```
假如： 寄存器 00H = 55H, 寄存器 01H = 02H, 寄存器 02H = 17H, C = "0":

RRC  00H   → 寄存器 00H = 2AH, C ="1"
RRC  @01H  → 寄存器 01H = 02H, 寄存器 02H = 0BH, C ="1"

第一个例子中，如果通用寄存器 00H 的值为 55H（01010101B），语句“RRC 00H”将 00H 的值右移一位，原来第 0 位（“1”）移至进位标志位，原来的 C 标志位（“1”）移至第 7 位，将目标寄存器 00H 的变 为 2AH（00101010B）。符号标志和溢出标志都被清零。
```

6.3.1.59 SB0 - 选择 Bank 0(Select Bank 0)

SB0

操作: BANK ← 0

SB0 指令将标志位寄存器(FLAGS)中的 bank 地址标志（FLAGS.0）清零，在寄存器卷的 set 1区域选择 bank 0 。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	4F

编程实例:

```
语句 SB0 将 FLAGS.0 清零，选择 bank 0 中的寄存器进行寻址。
```

6.3.1.60 SB1 - 选择 Bank 1(Select Bank 1)

SB1

操作: BANK ← 1

SB1 指令将标志位寄存器(FLAGS)中的 bank 地址标志（FLAGS.0）置 1，在寄存器卷的 set 1区域中选择 bank 1 。

注释：某些 S3F8- 系列的单片机没有 Bank 1。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	5F

编程实例:

```
指令 SB1 将 FLAGS.0 清零，选择 bank 1 的寄存器进行寻址 (如果存在 bank 1)。
```

6.3.1.61 SBC - 带进位减法(Subtract with Carry)

SBC dst,src

操作: dst ←dst - src - c

目标操作数减去源操作数和当前 C 标志位的值，结果存在目标操作数中。
源操作数不受影响。减法操作是通过将源操作数的补码加到目标操作数来完成的。
在多次进行的精确运算中，指令允许低阶的操作数向高阶的操作数“借位”。

标志位:

C: 如果借位操作发生（src>dst）则置 1；否则清零
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果为负则置 1；否则清零
V: 如果发生溢出，也就是说如果操作数符号相反而差值与源操作数符号相同，则置1；否则清零
D: 总是被置 1
H: 如果从结果低4位的最高位有进位则清零；否则置 1，表示有“借位”发生。

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	32	r	r
		6	33	r	lr
opc | src | dst	3	6	34	R	R
		6	35	R	IR
opc | dst | src	3	6	36	R	IM

编程实例:

```
假如： R1 = 10H, R2 = 03H, C ="1", 寄存器 01H = 20H, 寄存器 02H = 03H, 寄存器
03H = 0AH:

SBC  R1,R2       → R1 = 0CH, R2 = 03H
SBC  R1,@R2      → R1 = 05H, R2 = 03H, 寄存器 03H = 0AH
SBC  01H,02H     → 寄存器 01H = 1CH, 寄存器 02H = 03H
SBC  01H,@02H    → 寄存器 01H = 15H, 寄存器 02H = 03H, 寄存器 03H = 0AH
SBC  01H,#8AH    → 寄存器 01H = 95H; C, S, V = "1"

第一个例子中，如果工作寄存器 R1 的值为 10H，R2 的值为 03H，语句“SBC R1,R2”从目标操作数
（10H）中减去源操作数（03H）和 C 标志的值（“1”），然后将结果（0CH）存放在寄存器 R1 中。
```

6.3.1.62 SCF - C 置 1(Set Carry Flag)

SCF

操作: C ← 1

将进位标志位（C）无条件置 1。

标志位: **C:** 置为“1”

其它标志位都不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4	DF

编程实例:

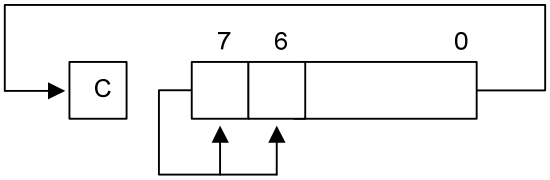
```
语句 SCF 将进位标志置 1。
```

6.3.1.63 SRA - 算术右移(Shift Right Arithmetic)

SRA dst

操作: dst (7) ← dst (7)
C ← dst (0)
dst (n) ← dst (n + 1), n = 0-6

将目标操作数算数右移一位，第 0 位（LSB）移至 C 标志，第 7 位（bit 7，符号位）不改变并且移到第 6 位。



标志位:

C: 如果从第 0 位（LSB）移出的是“1”则置 1；否则清零
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果是负数则置 1；否则清零
V: 总是清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	D0	R
		4	D1	IR

编程实例:

```
假如： 寄存器 00H = 9AH, 寄存器 02H = 03H, 寄存器 03H = 0BCH, C = “1”:

SRA  00H   → 寄存器 00H = 0CD, C = “0”
SRA  @02H  → 寄存器 02H = 03H, 寄存器 03H = 0DEH, C = “0”

第一个例子中，如果通用寄存器 00H 的值为 9AH（10011010B），语句“SRA 00H”将寄存器 00H
右移一位，第 0 位（“0”）清除 C 标志，第7位（“1”）移至第 6 位（第 7 位保持不变），目标寄存器 00H
的值变为 0CDH（11001101B）。
```

6.3.1.64 SRP/SRP0/SRP1 - 设置寄存器指针(Set Register Pointer)

SRP/SRP0/SRP1 src

操作:

如果 src (1) = 1 且 src (0) = 0 那么:	RP0 (3-7)	←	src (3-7)
如果 src (1) = 0 且 src (0) = 1 那么:	RP1 (3-7)	←	src (3-7)
如果 src (1) = 0 且 src (0) = 0 那么:	RP0 (4-7)	←	src (4-7),
	RP0 (3)	←	0
	RP1 (4-7)	←	src (4-7),
	RP1 (3)	←	1

源操作数的第 1 位和第 0 位（LSB）决定写入 2 个寄存器指针中的 1 个还是 2 个一起写入。如果两个寄存器指针都被选择，那么选择的寄存器指针的 3 到 7 位被写入，然后 RP0.3 清零，RP1.3 置 1。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 src
opc src	2	4	31	IM

编程实例:

语句

```
SRP  #40H
```

将地址为 0D6H 的寄存器指针 0（RP0）设定在 40H，地址为 0D7H 的寄存器指针 1（RP1）设定在 48H。

语句“SRP0 #50H”将 RP0 设为 50H，而语句“SRP1 #68H”将 RP1 设为 68H。

6.3.1.65 STOP - Stop 操作(Stop Operation)

STOP

操作: STOP 指令同时停止 CPU 时钟和系统时钟，使单片机进入 STOP 模式。在 STOP 模式下，CPU 寄存器、外设寄存器、I/O 口的控制和数据寄存器内容都保持不变。STOP 模式可以被外部的复位操作或者外部中断唤醒。对于复位操作，RESET 引脚的低电平必须保持足够长的时间，以保证晶振稳定所需的时间间隔。

在应用程序中，STOP 指令后必须跟有 3 个 NOP 指令，以保证在下条指令执行之前有足够长的时间间隔来稳定晶振。如果 STOP 后没有使用 3 个或者多个 NOP 指令，内部总线的悬浮状态将会产生漏电流。

标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc	1	4	7F	-	-

编程实例:

```
语句

STOP ;  停止所有单片机操作
NOP
NOP
NOP
```

注释: 在执行 STOP 指令之前，必须设置 STPCON 寄存器值为“10100101b”，否则 STOP 指令不会执行。

6.3.1.66 SUB - 减法(Subtract)

SUB dst,src

操作: dst ← dst - src

目标操作数减去源操作数，结果存放在目标操作数中。源操作数的内容不受影响。
减法操作是将源操作数的补码加到目标操作数来完成的。

标志位:

C: 如果“借位”发生则置 1；否则清零
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果是负数则置 1；否则清零
V: 如果发生溢出，也就是如果操作数符号相反而结果与源操作数的符号相同，则置 1；否则清零
D: 总是被置 1
H: 如果从结果低 4 位的最高位有进位则清零；否则置 1，表示有“借位”发生。

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	22	r	r
		6	23	r	lr
opc | src | dst	3	6	24	R	R
		6	25	R	IR
opc | dst | src	3	6	26	R	IM

编程实例:

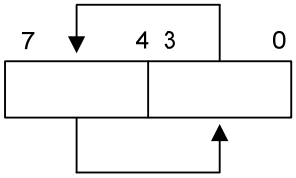
```
假如：R1 = 12H, R2 = 03H, 寄存器 01H = 21H, 寄存器 02H = 03H, 寄存器 03H = 0AH:

SUB  R1,R2      → R1 = 0FH, R2 = 03H
SUB  R1,@R2     → R1 = 08H, R2 = 03H
SUB  01H,02H    → 寄存器 01H = 1EH, 寄存器 02H = 03H
SUB  01H,@02H   → 寄存器 01H = 17H, 寄存器 02H = 03H
SUB  01H,#90H   → 寄存器 01H = 91H; C, S, V = “1”
SUB  01H,#65H   → 寄存器 01H = 0BCH; C 和 S = “1”, V = “0”

在第一个例子中，如果工作寄存器 R1 的值为12H，R2 为 03H，语句“SUB R1,R2”将源操作数（03H）
从目标操作数（12H）中减去，并将结果（0FH）存在目标寄存器 R1 中。
```

6.3.1.67 SWAP - 交换(Swap Nibbles)

SWAP dst

操作: dst (0 - 3) ↔ dst (4 - 7)

目标操作数的低四位和高四位互相交换。



标志位:

C: 没有定义
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 没有定义
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码 (Hex)	寻址模式 dst
opc | dst	2	4	F0	R
		4	F1	IR

编程实例:

```
假如： 寄存器 00H = 3EH, 寄存器 02H = 03H, 寄存器 03H = 0A4H:

SWAP 00H    →    寄存器 00H = 0E3H
SWAP @02H   →    寄存器 02H = 03H, 寄存器 03H = 4AH

第一个例子中，如果通用寄存器 00H 的值为 3EH（00111110B），语句“SWAP 00H”
将其高四位和低四位交换，使寄存器 00H 的值变为 0E3H（11100011B）。
```

6.3.1.68 TCM - 取反后位测试(Test Complement Under Mask)

TCM dst,src

操作: (NOT dst) AND src

该指令用于测试目标操作数中特定位是否为逻辑1。被测试的位通过将源操作数中相应位设为“1”（屏蔽）来指定。TCM 语句将目标操作数取反，然后与源操作数进行与操作。
通过检查零标志位（Z）来确定结果。目标操作数和源操作数不受影响 。

标志位:

C: 不受影响
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 总是清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	62	r	r
		6	63	r	lr
opc | src | dst	3	6	64	R	R
		6	65	R	IR
opc | dst | src	3	6	66	R	IM

编程实例:

```
假如： R0 = 0C7H, R1 = 02H, R2 = 12H, 寄存器 00H = 2BH, 寄存器 01H = 02H, 寄存器
02H = 23H:

TCM  R0,R1       → R0 = 0C7H, R1 = 02H, Z = “1”
TCM  R0,@R1      → R0 = 0C7H, R1 = 02H, 寄存器 02H = 23H, Z = “0”
TCM  00H,01H     → 寄存器 00H = 2BH, 寄存器 01H = 02H, Z = “1”
TCM  00H,@01H    → 寄存器 00H = 2BH, 寄存器 01H = 02H,
                 → 寄存器 02H = 23H, Z = “1”
TCM  00H,#34     → 寄存器 00H = 2BH, Z = “0”

第一个例子中，如果工作寄存器 R0 的值为 0C7H（11000111B），寄存器 R1 为 02H（00000010B），
语句“TCM R0,R1”测试目标寄存器的第 1 位（bit 1）是否为“1”。由于屏蔽值与对应的测试位一致， Z
标志被置 1，可用来确定 TCM 操作的结果。
```

6.3.1.69 TM - 位测试(Test Under Mask)

TM dst,src

操作: dst AND src

该指令用于测试目标操作数中特定位是否为逻辑 0。被测试的位通过将源操作数中相应位设为“1”（屏蔽）来指定，然后目标操作数与源操作数进行与操作。通过检查零标志位（Z）来确定结果。目标操作数和源操作数不受影响。

标志位:

C: 不受影响
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果的第 7 位为“1”则置 1；否则清零
V: 总是清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	72	r	r
		6	73	r	lr
opc | src | dst	3	6	74	R	R
		6	75	R	IR
opc | dst | src	3	6	76	R	IM

编程实例:

```
假如： R0 = 0C7H, R1 = 02H, R2 = 18H, 寄存器 00H = 2BH, 寄存器 01H = 02H, 寄存器
02H = 23H:

TM   R0,R1       → R0 = 0C7H, R1 = 02H, Z ="0"
TM   R0,@R1      → R0 = 0C7H, R1 = 02H, 寄存器 02H = 23H, Z ="0"
TM   00H,01H     → 寄存器 00H = 2BH, 寄存器 01H = 02H, Z ="0"
TM   00H,@01H    → 寄存器 00H = 2BH, 寄存器 01H = 02H,
                 → 寄存器 02H = 23H, Z ="0"
TM   00H,#54H    → 寄存器 00H = 2BH, Z ="1"

第一个例子中，如果工作寄存器 R0 的值为 0C7H（11000111B），寄存器 R1 的值为
02H（00000010B），语句“TM R0,R1”测试目标寄存器的第1位是否为“0”。 由于屏蔽值与对应的的测试位不匹配，Z 标志被清零，可以用来确定 TM 操作的结果。
```

6.3.1.70 WFI - 等待中断(Wait for Interrupt)

WFI

操作: 在等待状态时，CPU 被暂停直到中断发生，只有 DMA 传送仍然可以工作。
WFI 状态可以由内部中断，包括快速中断唤醒。

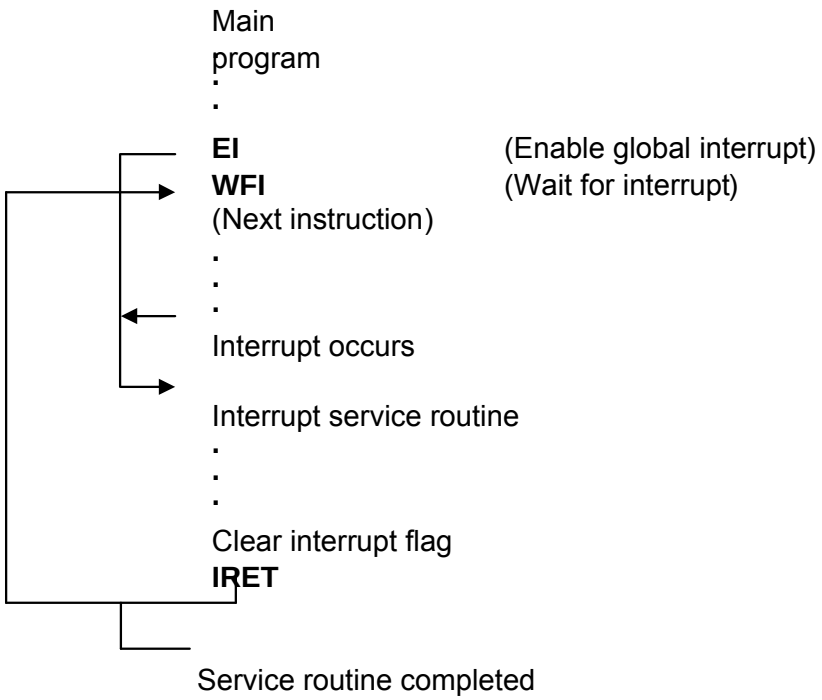
标志位: 没有标志位受影响

格式:

	字节数	时钟周期	指令代码 (Hex)
opc	1	4n	3F
			(n = 1, 2, 3, …)

编程实例:

以下实例程序的结构表示了“WFI”指令执行后的一系列操作：



6.3.1.71 XOR - 逻辑异或(Logical Exclusive OR)

XOR dst,src

操作: dst ← dst XOR src

源操作数与目标操作数进行逻辑异或，结果存在目标操作数中。
如果两个操作数中对应的位不相同，那么异或的结果为“1”；否则为“0”。

标志位:

C: 不受影响
Z: 如果结果是“0”则置 1；否则清零
S: 如果结果的第 7 位（bit 7）为“1”则置 1；否则清零
V: 总是清零
D: 不受影响
H: 不受影响

格式:

	字节数	时钟周期	指令代码	寻址模式	
			(Hex)	dst	src
opc | dst | src	2	4	B2	r	r
		6	B3	r	lr
opc | src | dst	3	6	B4	R	R
		6	B5	R	IR
opc | dst | src	3	6	B6	R	IM

编程实例:

```
假如： R0 = 0C7H, R1 = 02H, R2 = 18H, 寄存器 00H = 2BH, 寄存器 01H = 02H, 寄存器
02H = 23H:

XOR  R0,R1       → R0 = 0C5H, R1 = 02H
XOR  R0,@R1      → R0 = 0E4H, R1 = 02H, 寄存器 02H = 23H
XOR  00H,01H     → 寄存器 00H = 29H, 寄存器 01H = 02H
XOR  00H,@01H    → 寄存器 00H = 08H, 寄存器 01H = 02H, 寄存器 02H = 23H
XOR  00H,#54H    → 寄存器 00H = 7FH

第一个例子中，如果工作寄存器 R0 的值为 0C7H，R1 的值为 02H，语句“XOR R0,R2”将R1和R0
的值进行逻辑异或，并将结果（0C5H）存在目标寄存器 R0 中。
```

7 时钟电路

7.1 概述

S3C84H5/F84H5 主时钟，连外部晶振支持1 MHz 到10 MHz频率。CPU 最大时钟频率为 10 MHz. X_{IN} 和 X_{OUT} 管脚连外部晶振或者片上时钟电路。 Watch timer 的副时钟，外部连晶振频率范围 30kHz～35kHz. XT_{IN} 和 XT_{OUT}管脚连外部晶振或者片上时钟电路。

7.1.1 系统时钟电路

系统时钟电路包含以下部分：

- 外部晶振或陶振 (或者外部时钟源)
- 振荡器停止和唤醒功能
- 可编程的 CPU 时钟分频器 (fxx 1, 2, 8, 或 16 分频)
- 系统时钟控制寄存器 CLKCON
- 振荡器控制寄存器 OSCCON 和 STOP 控制寄存器 STPCON

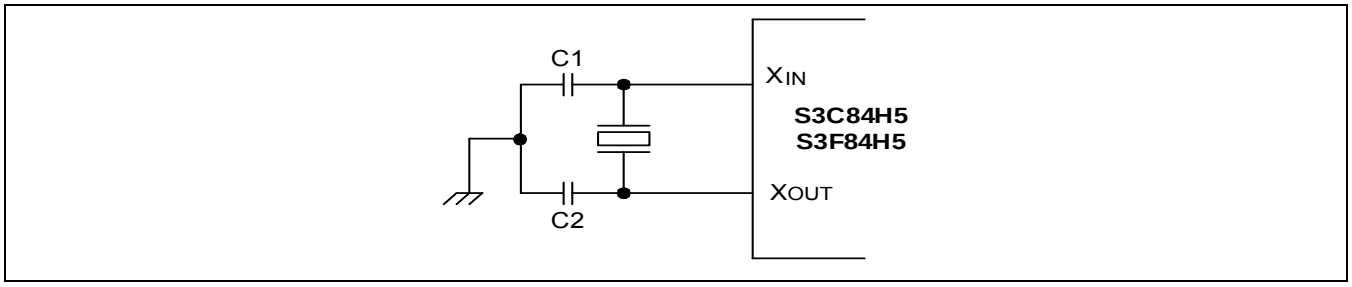


图 7-1 主振荡电路（石英/陶瓷晶振）

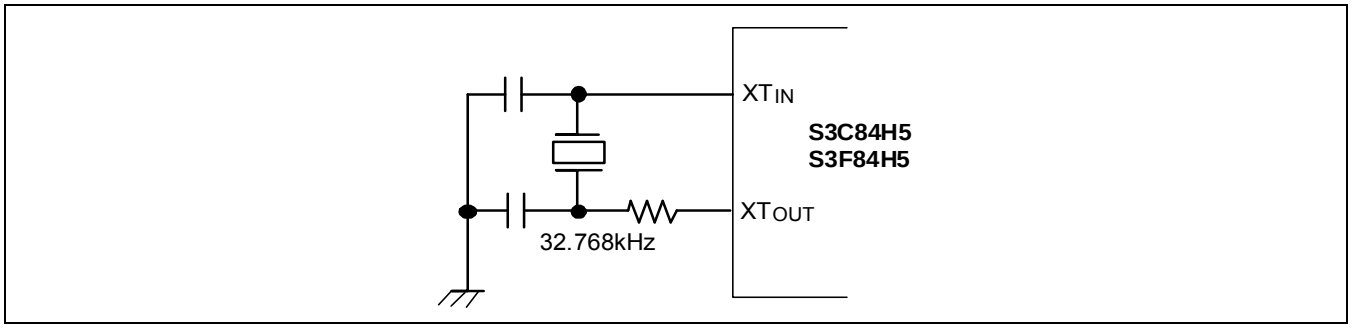


图 7-2 副振荡电路 (石英晶振)

7.1.2 省电模式下时钟电路状态

有两种省电模式，STOP 模式和 IDLE 模式，分别对时钟振荡产生如下影响：

- 在STOP模式下，主振荡器停止工作，CPU 和外设停止。寄存器卷中的值和当前系统寄存器的值都保持原值。在复位操作或具有 RC 延迟噪音滤波器的外部中断被触发下，CPU 会退出 STOP 模式，振荡器重新起振。如果是副振荡电路和 Watch Timer工作，内部中断也可以使 CPU 退出 STOP 模式，振荡器重新起振。
- 在 IDLE 模式下，主振荡器进入 CPU 的时钟被切断，但内部中断及Timers 仍然工作。 复位或中断（外部中断或内部中断）都可以使 CPU 退出 IDLE 模式。

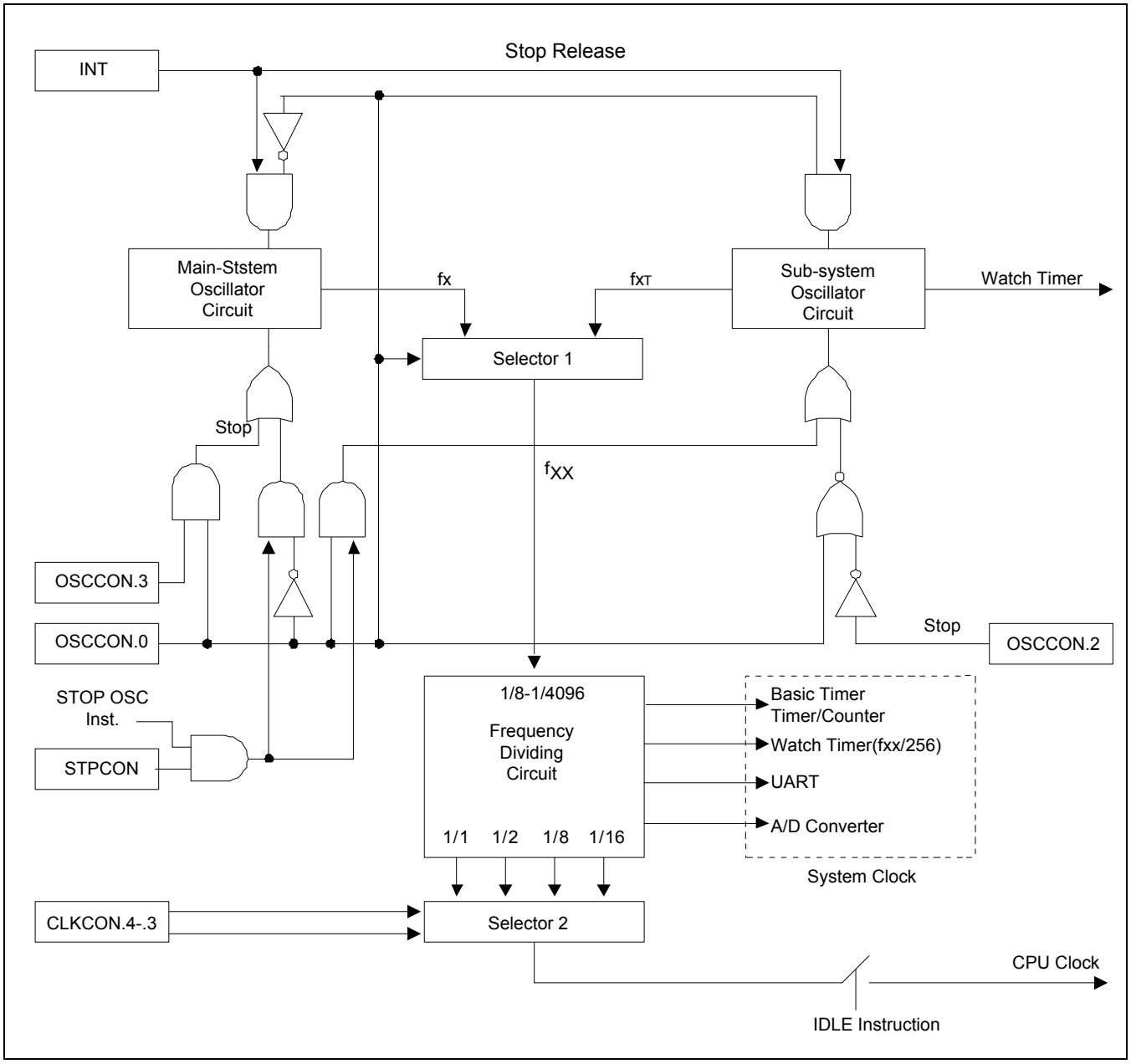


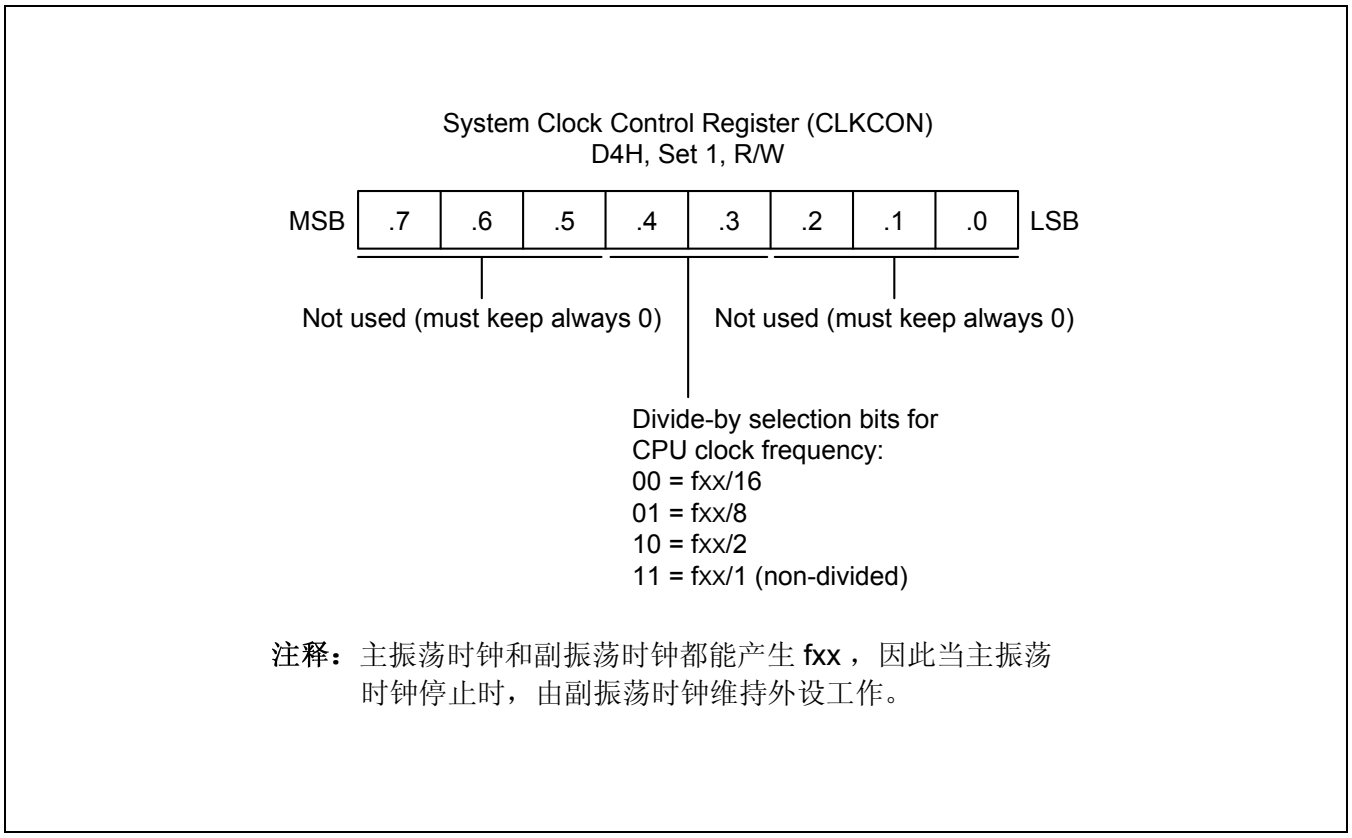
图 7-3 系统时钟电路原理图

7.1.3 系统时钟控制寄存器(CLKCON)

系统时钟控制寄存器 CLKCON 的地址是 D4H，Set 1，可读/写，有如下功能：

- 系统时钟分频系数选择：不分频、2 分频、8 分频或16 分频

复位后，选择 f_{OSC} /16（最慢时钟速度）作为 CPU 时钟。如果需要，用户可提高 CPU 时钟速度至 f_{OSC}，f_{OSC}/2 或 f_{OSC} /8。



注释： 主振荡时钟和副振荡时钟都能产生 fxx，因此当主振荡时钟停止时，由副振荡时钟维持外设工作。

图 7-4 系统时钟控制寄存器 (CLKCON)

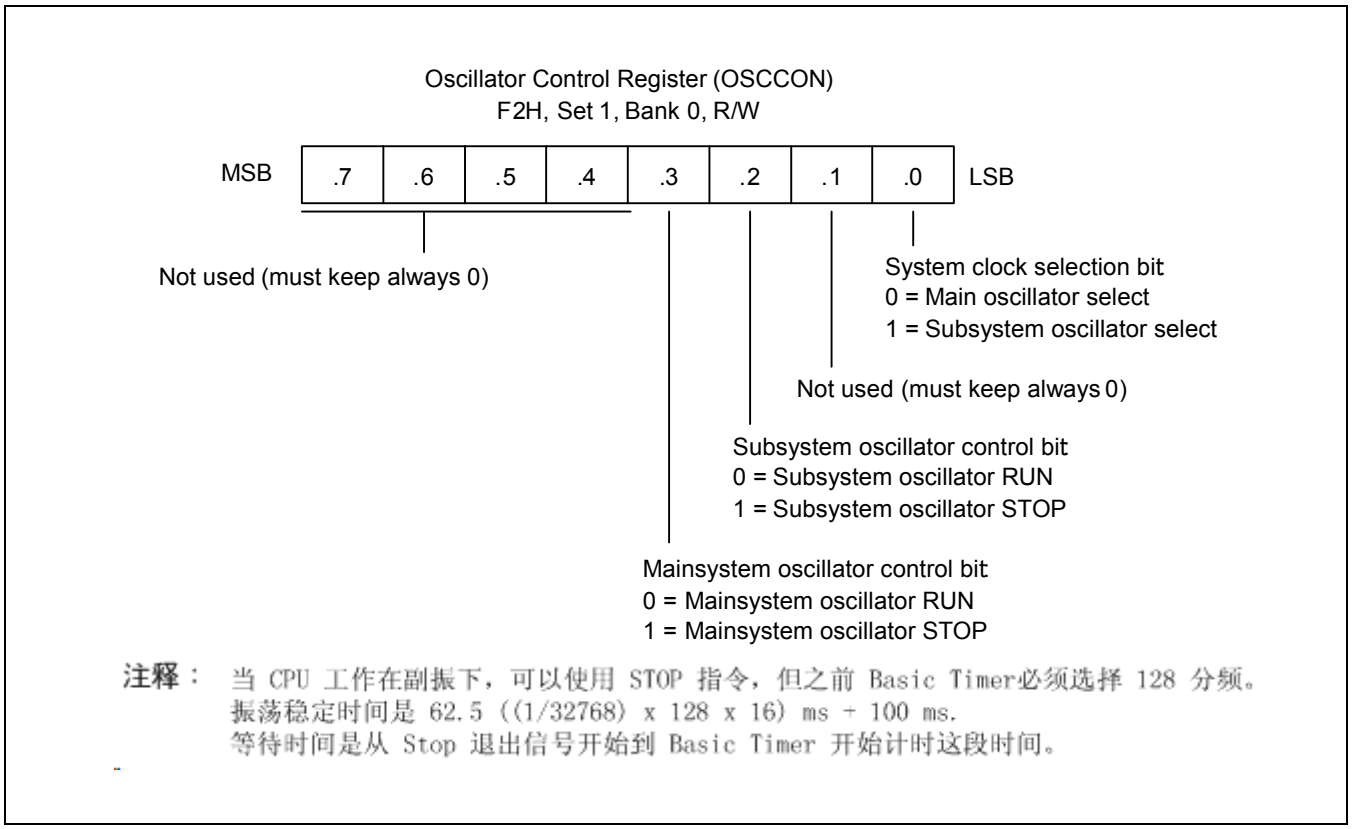


图 7-5 振荡器控制寄存器 (OSCCON)

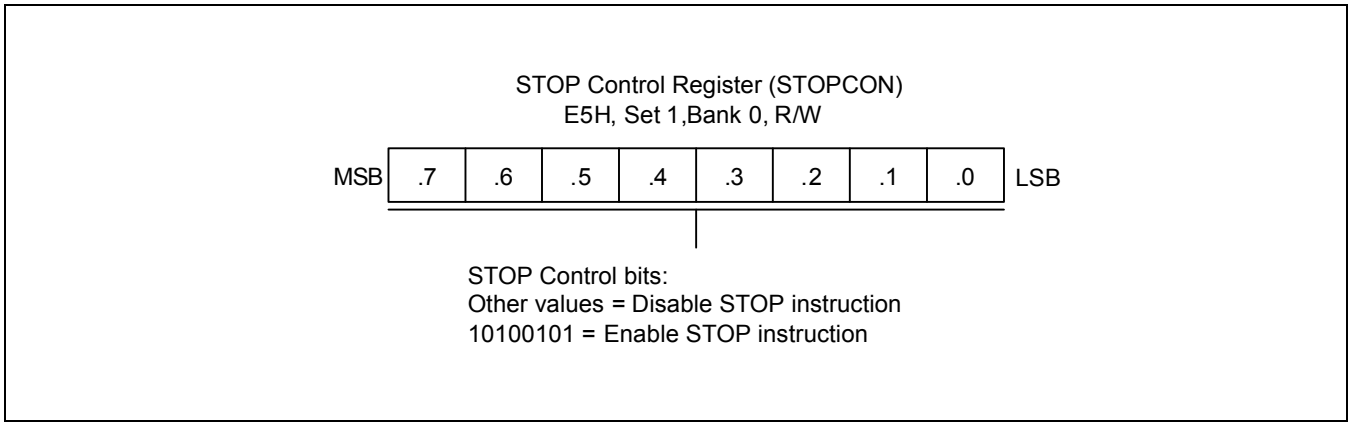


图 7-6 STOP 控制寄存器 (STOPCON)

8 复位和省电模式

8.1 系统复位

8.1.1 概述

如果采用外部上电复位，当 V_{DD} 管脚为高电平并且 nRESET 管脚为低电平时，复位信号通过施 密特触发电路与 CPU 时钟同步，继而复位系统。复位操作使得 S3C84H5/F84H5 处于已知的操作状态。

接电源后，nRESET 管脚必须维持低电平一段时间，以允许内部 CPU 时钟振荡达到稳定。 复位操作时最小的振荡稳定等待时间约为 1ms 。

在正常工作模式下，V_{DD} 和 RESET 都为高电平。复位发生时， RESET 管脚被拉低，发生复位。 所有的系统和外设寄存器都变为复位后由硬件决定的默认值。

总之，复位发生时的执行顺序为：

- 禁止所有中断
- 允许看门狗功能（Basic Timer）
- 设置 P0-P3 口为输入模式
- 外围控制寄存器和数据寄存器恢复为硬件决定的默认值。
- 程序计数器（PC）装入 ROM 复位地址 0100H.
- 当振荡稳定后，调入 ROM 中复位地址第一和第二个单元的指令并执行。

8.1.2 普通模式下的 复位操作

在普通模式(掩膜 ROM)下，TEST 管脚连到 V_{SS}。 复位可以访问到 16K 字节的片上 ROM.

注释: 如果希望改变振荡稳定的等待时间，可以在进入 STOP 状态前，将 Basic Timer 控制寄存器 BTCON 设置合适的值。如果不想使用 Basic Timer 看门狗功能（Basic Timer 计数器溢出产生复位），也可以写“1010B”到 BTCON 的高四位来禁止 Basic Timer。

8.1.3 硬件复位值

表 8-1 至表 表 8-3 列出了 CPU、系统寄存器、外围控制寄存器和外围数据寄存器复位后的值。
以下标号用作表示复位值：

- “1”或“0”代表复位后此位为逻辑 1 或逻辑 0。
- “x”代表复位后此位值不确定。
- “-”代表此位没有用到或未映射，但读此位为 0。

表 8-1 S3C84H5/F84H5 复位后 Set1 的寄存器值

寄存器名字	助记标号	地址		复位值 (位)							
		Dec	Hex	7	6	5	4	3	2	1	0
Timer B 控制寄存器	TBCON	208	D0H	0	0	0	0	0	0	0	0
Timer B 数据寄存器（高字节）	TBDATAH	209	D1H	1	1	1	1	1	1	1	1
Timer B 数据寄存器（低字节）	TBDATAL	210	D2H	1	1	1	1	1	1	1	1
Basic Timer 控制寄存器	BTCON	211	D3H	0	0	0	0	0	0	0	0
系统时钟控制寄存器	CLKCON	212	D4H	0	0	0	0	0	0	0	0
系统标志寄存器	FLAGS	213	D5H	x	x	x	x	x	x	0	0
寄存器指针 0	RP0	214	D6H	1	1	0	0	0	-	-	-
寄存器指针 1	RP1	215	D7H	1	1	0	0	1	-	-	-
堆栈指针（高字节）	SPH	216	D8H	x	x	x	x	x	x	x	x
堆栈指针（低字节）	SPL	217	D9H	x	x	x	x	x	x	x	x
指令指针（高字节）	IPH	218	DAH	x	x	x	x	x	x	x	x
指令指针（低字节）	IPL	219	DBH	x	x	x	x	x	x	x	x
中断请求寄存器	IRQ	220	DCH	0	0	0	0	0	0	0	0
中断屏蔽寄存器	IMR	221	DDH	x	x	x	x	x	x	x	x
系统模式寄存器	SYM	222	DEH	0	0	0	x	x	x	0	0
寄存器页指针	PP	223	DFH	0	0	0	0	0	0	0	0

表 8-2 S3C84H5/F84H5 复位后 Set1, Bank0 的寄存器值

寄存器名字	助记标号	地址		复位值 (位)							
		Dec	Hex	7	6	5	4	3	2	1	0
P0 口数据寄存器	P0	224	E0H	0	0	0	0	0	0	0	0
P1 口数据寄存器	P1	225	E1H	0	0	0	0	0	0	0	0
P2 口数据寄存器	P2	226	E2H	0	0	0	0	0	0	0	0
P3 口数据寄存器	P3	227	E3H	0	0	0	0	0	0	0	0
E4H 地址空间未映射											
STOP 控制寄存器	STOPCON	229	E5H	0	0	0	0	0	0	0	0
P0 口控制寄存器	P0CON	230	E6H	0	0	0	0	0	0	0	0
FBH 地址空间未映射											
P1 口控制寄存器（高字节）	P1CONH	232	E8H	0	0	0	0	0	0	0	0
P1 口控制寄存器（低字节）	P1CONL	233	E9H	0	0	0	0	0	0	0	0
P1 口中断标志位寄存器	P1INTPND	234	EAH	0	0	0	0	0	0	0	0
P1 口中断控制寄存器	P1INT	235	EBH	0	0	0	0	0	0	0	0
P2 口控制寄存器（高字节）	P2CONH	234	ECH	0	0	0	0	0	0	0	0
P2 口控制寄存器（低字节）	P2CONL	235	EDH	0	0	0	0	0	0	0	0
EEH 地址空间未映射											
P3 口控制寄存器（低字节）	P3CONL	239	EFH	0	0	0	0	0	0	0	0
F0H-F1H 地址空间未映射											
振荡器控制寄存器	OSCCON	242	F2H	0	0	0	0	0	0	0	0
FBH 地址空间未映射											
UART 中断标志位寄存器	UARTPND	244	F4H	0	0	0	0	0	0	0	0
UART 数据寄存器	UDATA	245	F5H	1	1	1	1	1	1	1	1
UART 控制寄存器	UARTCON	246	F6H	0	0	0	0	0	0	0	0
A/D 转换控制寄存器	ADCON	247	F7H	0	0	0	0	0	0	0	0
A/D 转换数据寄存器（高字节）	ADDATAH	248	F8H	0	0	0	0	0	0	0	0
A/D 转换数据寄存器（低字节）	ADDATAL	249	F9H	0	0	0	0	0	0	0	0
P2 口上拉电阻使能寄存器	P2PUR	250	FAH	0	0	0	0	0	0	0	0
FBH 地址空间未映射											
FCH 地址空间保留											
Basic Timer 计数器寄存器	BTCNT	253	FDH	0	0	0	0	0	0	0	0
FEH 地址空间保留											
中断优先级寄存器	IPR	255	FFH	x	x	x	x	x	x	x	x

表 8-3 S3C84H5/F84H5 复位后 Set1, Bank1 的寄存器值

寄存器名字	助记标号	地址		复位值 (位)							
		Dec	Hex	7	6	5	4	3	2	1	0
Timer A, 1 中断标志位寄存器	TINTPND	224	E0H	0	0	0	0	0	0	0	0
Timer A 控制寄存器	TACON	225	E1H	0	0	0	0	0	0	0	0
Timer A 数据寄存器	TADATA	226	E2H	1	1	1	1	1	1	1	1
Timer A 计数器寄存器	TACNT	227	E3H	0	0	0	0	0	0	0	0
Timer 1(0) 数据寄存器（高字节）	T1DATAH0	228	E4H	1	1	1	1	1	1	1	1
Timer 1(0) 数据寄存器（低字节）	T1DATAL0	229	E5H	1	1	1	1	1	1	1	1
Timer 1(1) 数据寄存器（高字节）	T1DATAH1	230	E6H	1	1	1	1	1	1	1	1
Timer 1(1) 数据寄存器（低字节）	T1DATAL1	231	E7H	1	1	1	1	1	1	1	1
Timer 1(0) 控制寄存器	T1CON0	232	E8H	0	0	0	0	0	0	0	0
Timer 1(1) 控制寄存器	T1CON1	233	E9H	0	0	0	0	0	0	0	0
Timer 1(0) 计数器寄存器（高字节）	T1CNTH0	234	EAH	0	0	0	0	0	0	0	0
Timer 1(0) 计数器寄存器（低字节）	T1CNTL0	235	EBH	0	0	0	0	0	0	0	0
Timer 1(1) 计数器寄存器（高字节）	T1CNTH1	236	ECH	0	0	0	0	0	0	0	0
Timer 1(1) 计数器寄存器（低字节）	T1CNTL1	237	EDH	0	0	0	0	0	0	0	0
UART 波特率数据寄存器 (高字节)	BRDATAH	238	EEH	1	1	1	1	1	1	1	1
UART波特率数据寄存器(低字节)	BRDATAL	239	EFH	1	1	1	1	1	1	1	1
SIO 预分频寄存器	SIOPS	240	F0H	0	0	0	0	0	0	0	0
SIO 数据寄存器	SIODATA	241	F1H	0	0	0	0	0	0	0	0
SIO 控制寄存器	SIOCON	242	F2H	0	0	0	0	0	0	0	0
PWM 数据寄存器 (高字节)	PWMDATAH	243	F3H	0	0	0	0	0	0	0	0
PWM 数据寄存器(低字节)	PWMDATAL	244	F4H	0	0	0	0	0	0	0	0
PWM 控制寄存器	PWMCON	245	F5H	0	0	-	0	0	0	0	0
F6H, F7H地址空间未使用											
Watch Timer 控制寄存器	WTCON	248	F8H	0	0	0	0	0	0	0	0
F9H – FFH地址空间未使用											

8.2 省电模式

8.2.1 STOP 模式

STOP 指令（指令码 7FH）将会使系统进入 STOP 模式。在 STOP 模式下，CPU 和所有外围设备将停止工作。也就是说，片上主振荡器会停止；在禁止 LVR（低电平复位）的情况下，芯片消耗电流将小于 3μA。 当时钟冻结时，所有的系统功能停止，但存储在内部寄存器卷中的数据仍然保留。可以通过 2 种方式退出 STOP 模式：nRESET 复位信号或外部中断。

注释: 在使用外部时钟源的情况下，不要使用 STOP 模式，因为 X_{IN} 输入必须内部连到 V_{SS} 来减少漏电流。

8.2.1.1 执行复位操作退出 STOP 模式

芯片复位完成后，复位信号重新恢复到高电平，此时系统退出 STOP 模式，所有的系统和外围控制寄存器恢复为默认值，但数据寄存器区仍然保持复位以前的值。因为 CLKCON.3 和 CLKCON.4 被清除为“00B”，CPU 时钟选择最慢频率（fxx/16）。待振荡稳定后，CPU 调入 ROM 0100H （和 0101H）地址中存储的程序指令， 并执行系统初始化程序。

8.2.1.2 使用外部中断退出 STOP 模式

具有 RC 延迟噪声滤波电路的外部中断可以让系统退出 STOP 模式。在给定的情况下，使用哪种中断推出 STOP 模式由 MCU 当时的内部工作模式决定。S3C84H5/F84H5 中断结构中的外部中断 P1.0-P1.3 (INT0–INT3) 都可以使系统退出 STOP 状态。

请注意以下 STOP 模式退出的情况：

- 利用外部中断退出 STOP 模式时，系统和外围控制寄存器中的值不变。
- 如果使用外部中断退出 STOP 模式，可以对时钟振荡稳定时间进行编程，为了实现这个目的，必须在进入 STOP 模式之前，进行相应的控制和时钟设置。
- 当使用外部中断退出 STOP 模式时，CLKCON.3 和 CLKCON.4 寄存器中的值也保持不变，即使用当前选择的 CPU 的时钟分频。
- 在退出 STOP 模式时，系统处理外部中断服务程序。中断处理完毕后，立即返回执行 STOP 指令的下一条指令。

8.2.1.3 如何进入 STOP 模式

进入 STOP 模式需要两步：

1. 往 STOPCON 寄存器写入合适的值 (10100101B)。
2. 写 STOP 指令 (按照顺序)。

8.2.2 IDLE 模式

IDLE 指令（指令码 6FH）将会使系统进入 IDLE 模式。在 IDLE 模式下，CPU 操作停止，但一些外围仍工作。在IDLE 模式中，进入CPU 的内部时钟被切断，但外围仍有时钟。各端口仍保持进入 IDLE 模式那刻的状态 （输入或输出）。

可以通过 2 种方式退出 IDLE 模式：

1. 执行复位操作。所有的系统和外围控制寄存器复位至默认值，但所有数据寄存器的值保持不变。
   因为 CLKCON.3 和 CLKCON.4 清除为“00B”，复位自动选择慢时钟（f_{xx}/16）。
   如果中断被屏蔽， 则复位是退出 IDLE 模式的唯一方法。

2. 激活使能的中断，退出 IDLE 模式。当用中断退出 IDLE 模式，CLKCON.3 和 CLKCON.4 寄存器的值保持不变，选择当前的 CPU 时钟分频。之后，系统处理中断。中断处理完毕后，返回执行 IDLE 指令的下一条指令。

9 I/O 口

9.1 概述

S3C84H5/F84H5 有 4 个 I/O 口，P0-P3。一共 22 个I/O。可以灵活地设置各个端口以满足应用要求。无需专门的 I/O 指令，CPU 可以通过读写 I/O口 数据寄存器，直接访问这些端口。

表 9-1 表述 S3C84H5/F84H5 I/O 口功能概况。

表 9-1 S3C84H5/F84H5 I/O 口设置概述

I/O 口	设置选项
0	位可编程 I/O 口管脚。可以设定为输入或推挽式输出模式。通过软件设定上拉电阻。 P0 口也可单独作为其他功能使用，P0.0~P0.3 可用作 AD0~AD3。
1	位可编程 I/O 口管脚。可以设定为输入或推挽式输出模式。通过软件设定上拉电阻。 P1 口也可单独作为其他功能使用，P1.0~P1.5 可用作 INT0~INT3,TAOUT,TACK,TACAP,T1OUT1,T1CK1,T1CAP1,AD5,AD6。
2	位可编程 I/O 口管脚。可以设定为输入或推挽式输出模式。通过软件设定上拉电阻。 P2 口也可单独作为其他功能使用，P2.0~P2.7 可用作 ADC4,ADC7,SI,T1CAP0,T1OUT1,T1CK0,SO,SCK,RxD,TxD,TBPWM,PWM。
3	位可编程 I/O 口管脚。可以设定为输入、推挽式输出，或开漏输出模式。通过软件设定上拉电阻。

注释: 调试模式时，必须包含三句额外命令来正确初始化端口。 如果漏掉这几句命令，端口操作会不正确。(*编程实例 9-1*)
写好程序后，编译之前，需去掉这几句命令。

```
.ORG 100H
SB1                          ; 调试时的额外命令
LD          0F7H,#5FH        ; 调试时的额外命令
SB0                          ; 调试时的额外命令
```

9.1.1 端口数据寄存器

表 9-2 列出 S3C84H5/F84H5 4个 I/O 口数据寄存器概述。P0-P3 口数据寄存器的结构如图 *图 9-2* 所示。

表 9-2 端口数据寄存器概述

寄存器名称	标号	十进制	十六进制	位置	R/W
P0 口数据寄存器	P0	224	E0H	Set 1, Bank 0	R/W
P1 口数据寄存器	P1	225	E1H	Set 1, Bank 0	R/W
P2 口数据寄存器	P2	226	E2H	Set 1, Bank 0	R/W
P3 口数据寄存器	P3	227	E3H	Set 1, Bank 0	R/W

9.1.2 P0 口

P0 口为一个 4 位 I/O 口，有以下两种功能：

- 通用数字 I/O 功能
- 复用功能：AD0~AD3

可通过写或读端口数据寄存器 P0 （地址：E0H，Set 1，Bank 0）直接访问 P0 口。

9.1.2.1 P0 口控制寄存器(P0CON)

P0 口有一个 8 位的控制寄存器 P0CON，用于 P0.0-P0.3。复位后，P0CON 寄存器被清为“00H”， 所有管脚为输入模式。可以通过配置此寄存器来设置管脚为输入或输出（推挽式），以及使能其他复用功能。

在对此端口进行编程时，务必记得，任何由 P0 控制寄存器设置的外围复用的 I/O 功能都必须在对应的外设模块中使能。

Port 0 Control Register, Low Byte (P0CON)
E6H, Set1, Bank0, R/W, Reset value="00H"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P0.3/ADC3Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = Alternative function mode;AD3 input

[.5-.4] P0.2/AD2 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = Alternative function mode: ADC2 input

[.3-.2] P0.1/ADC1 Configuration Bits
0 0 = Input mode
0 1 = Input mode
1 0 = Push-pull output mode
1 1 = Alternative function mode: ADC1 input

[.1-.0] P0.0/ADC0 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = Alternative function mode: ADC0 input

图 9-1 P0 口低字节控制寄存器 (P0CON)

9.1.3 P1 口

P1 口为一个 6 位 I/O 口，可单独设置以下两种功能：

- 通用数字 I/O 口功能
- 复用功能：INT0~INT3, TAOUT, TACK, TACAP, T1OUT1,T1CK1,T1CAP1,AD5,AD6

可通过写或读端口数据寄存器 P1（地址：E1H，Set 1，Bank 0）直接访问 P1 口。

9.1.3.1 口控制寄存器(P1CONH, P1CONL)

P1 口的管脚由两个 6 位控制寄存器来配置： P1CONH 用于 P1.4-P1.5，P1CONL 用于 P1.0-P1.3。复位后，P1CONL 和 P1CONH 清为“00H”，所有 I/O 口配置为输入模式。可通过设置控制寄存器来选择输入或输出（推挽式），以及使能复用功能。

在对此端口进行编程时，务必记得，任何由 P1 控制寄存器设置的外围复用的 I/O 功能都必须在对应的外设模块中使能。

9.1.3.2 P1 口 中断使能, 标志位寄存器(P1INT,P1INTPND)

为处理 P1 口管脚上的外部中断，提供 2 个额外控制寄存器：P1口中断控制寄存器 P1INT (地址：EAH，Set 1，Bank 0) 和 P1口中断标志位寄存器 P1INTPND (地址：EBH，Set 1，Bank 0)。

中断服务程序执行时，P1 中断标志位可查询中断标志条件以及清除中断标志条件。 应用程序通过在定期时间内查询 P1INTPND.3-.0 位检测到中断。

当任何 P1 管脚的中断使能位设置为“1”时，管脚上的上升沿或者下降沿将产生一个中断请求，对应的 P1INTPND 位自动设置为“1”，同时 IRQ 级变为低电平通知 CPU 有个中断请求等待处理。
当 CPU 响应该中断请求时，应用程序必须往 P1INTPND 位写“0”来清除中断标志条件。

Port 1 Control Register, High Byte (P1CONH)
E8H, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.4]Not used(must keep always 0)

[.3-.2]P1.5/T1CAP1/AD6 Configuration Bits

0 0 = Input mode: T1CAP1 input
0 1 = Input mode with pull-up: T1CAP1 input
1 0 = Push-pull output mode
1 1 = Alternative function mode: AD6

[.1-.0]P1.4/T1CK1/AD5 Configuration Bits

0 0 = Input mode: T1CK1 input
0 1 = Input mode with pull-up: T1CK1 input
1 0 = Push-pull output mode
1 1 = Alternative function mode: AD5

图 9-2 P1 口高字节控制寄存器 (P1CONH)

Port 1 Control Register, Low Byte (P1CONL)
E9H, Set1, Bank0, R/W, Reset value="00H"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P1.3/T1OUT0/INT3 Configuration Bits

0 0 = Input mode; Interrupt input (INT3)
0 1 = Input mode with pull-up; Interrupt input (INT3)
1 0 = Push-pull output mode
1 1 = Alternative function mode: T1OUT0 output

[.5-.4] P1.2/TACAP/INT2 Configuration Bits

0 0 = Input mode; Interrupt input (INT2); TACAP
0 1 = Input mode with pull-up; Interrupt input (INT2);TACAP
1 0 = Push-pull output mode
1 1 = Alternative function mode: Not used

[.3-.2] P1.1/TACK/BUZ/INT1 Configuration Bits

0 0 = Input mode; Interrupt input (INT1); TACK
0 1 = Input mode with pull-up; Interrupt input (INT1); TACK
1 0 = Push-pull output mode
1 1 = Alternative function mode: BUZ output

[.1-.0] P1.0/TAOUT/INT0 Configuration Bits

0 0 = Input mode; Interrupt input (INT0)
0 1 = Input mode with pull-up; Interrupt input
1 0 = Push-pull output mode
1 1 = Alternative function mode: TAOUT output

图 9-3 P1 口低字节控制寄存器 (P1CONL)

Port 1 Interrupt Pending Register (P1INTPND)
EAH, Set1, Bank0, R/W, Reset value="00H"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.4] Not used for S3C84H5/F84H5

[.3] P1.3/INT3 Interrupt Pending bit
0 = Interrupt request is not pending,pending bit clear when write 0
1 = Interrupt request is pending

[.2] P1.2/INT2 Interrupt Pending bit
0 = Interrupt request is not pending,pending bit clear when write 0
1 = Interrupt request is pending

[.1] P1.1/INT1 Interrupt Pending bit
0 = Interrupt request is not pending,pending bit clear when write 0
1 = Interrupt request is pending

[.0] P1.0/INT0 Interrupt Pending bit
0 = Interrupt request is not pending,pending bit clear when write 0
1 = Interrupt request is pending

图 9-4 P1 口中断标志位寄存器

Port 1 Interrupt Enable Register (P1INT)
EBH, Set1, Bank0, R/W, Reset value="00H"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P1.3's Interrupt Enable/Disable Selection Bit
0X = Disable Interrupt
10 = Enable Interrupt; Falling Edge
11 = Enable Interrupt; Rising Edge

[.5-.4] P1.2s Interrupt Enable/Disable Selection Bit
0X = Disable Interrupt
10 = Enable Interrupt; Falling Edge
11 = Enable Interrupt; Rising Edge

[.3-.2] P1.1's Interrupt Enable/Disable Selection Bit
0X = Disable Interrupt
10 = Enable Interrupt; Falling Edge
11 = Enable Interrupt; Rising Edge

[.1-.0] P1.0's Interrupt Enable/Disable Selection Bit
0X = Disable Interrupt
10 = Enable Interrupt; Falling Edge
11 = Enable Interrupt; Rising Edge

图 9-5 P1 口中断使能寄存器 (P1INT)

9.1.4 P2 口

P2 口是一个 8 位I/O口。可以通过读写 P2 口的数据寄存器 P2（地址：E2H，Set 1，Bank 0）来访问 P2 口。P2 口可以用作输入，输出（推挽式输出）或者其他功能：

- 通用数字 I/O口
- 其它复用功能：ADC4, ADC7, SI, T1CAP0, T1OUT1, T1CK0, TBPWM, PWM

9.1.4.1 P2 口控制寄存器(P2CONH, P2CONL)

端口 2 有两个 8 位控制寄存器：P2CONH 用于 P2.4-P2.7，P2CONL 用于 P2.0-P2.3。复位后，P2CONH 和 P2CONL 清为“00H”，所有 I/O 口配置为输入模式。可通过设置控制寄存器来选择输入或输出（推挽式）， 以及使能复用功能。

在对此端口进行编程时，务必记得，任何由 P2 控制寄存器设置的外围复用的 I/O 功能都必须在对应的外设模块中使能。

9.1.4.2 P2 口上拉电阻控制寄存器(P2PUR)

使用 P2 口上拉电阻控制寄存器 P2PUR（地址：FAH，Set 1，Bank 0）可以单独控制 P2 口的上拉电阻。

Port 2 Control Register, High Byte (P2CONH)
ECH, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P2.7/TxD Configuration Bits
0 0 = Input mode
0 1 = Alternative function mode; Not used
1 0 = Push-pull output mode
1 1 = Alternative function mode; TxD output

[.5-.4] P2.6/RxD Configuration Bits
0 0 = Input mode
0 1 = Alternative function mode; Not used
1 0 = Push-pull output mode
1 1 = Alternative function mode; RxD output

[.3-.2] P2.5/SCK Configuration Bits
0 0 = Input mode
0 1 = Alternative function mode; Not used
1 0 = Push-pull output mode
1 1 = Alternative function mode; SCK output

[.1-.0] P2.4/SO Configuration Bits
0 0 = Input mode
0 1 = Alternative function mode; Not used
1 0 = Push-pull output mode
1 1 = Alternative function mode; SO output

图 9-6 P2 口高字节控制寄存器 (P2CONH)

Port 2 Control Register, Low Byte (P2CONL)
EDH, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P2.3/AD7/SI Configuration Bits
0 0 = Input mode; SI
0 1 = Alternative function mode; Not used
1 0 = Push-pull output mode
1 1 = Alternative function mode; AD7

[.7-.6] P2.2/AD4/T1OUT1 Configuration Bits
0 0 = Input mode
0 1 = Alternative function mode; T1OUT1
1 0 = Push-pull output mode
1 1 = Alternative function mode; AD4

[.7-.6] P2.1/PWM/T1CAP0 Configuration Bits
0 0 = Input mode; T1CAP0
0 1 = Alternative function mode; T1CAP0
1 0 = Push-pull output mode
1 1 = Alternative function mode; PWM

[.7-.6] P2.0/TBPWM/T1CK0 Configuration Bits
0 0 = Input mode;T1CK0
0 1 = Alternative function mode; T1CK0
1 0 = Push-pull output mode
1 1 = Alternative function mode; TBPWM

图 9-7 P2 口低字节控制寄存器 (P2CONL)

Port 2 Pull-up Control Register (P2PUR)
FAH, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7] P2.7/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.6] P2.6/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.5] P2.5/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.4] P2.4/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.3] P2.3/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.2] P2.2/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.1] P2.1/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

[.0] P2.0/Pull-up Resistor Enable/Disable
0 = Pull-up resistor disable
1 = Pull-up resistor enable

图 9-8 P2 口上拉电阻控制寄存器 (P2PUR)

9.1.5 P3 口

P3 口是一个 8 位通用 I/O 口。可以通过读写 P3 口的数据寄存器 P3（地址：E3H，Set 1，Bank 0）来访问 P3 口。P3 口可以用作输入或输出(推挽式输出)。

9.1.5.1 口控制寄存器(P3CONH, P3CONL)

端口 3 的有两个 8 位控制寄存器： P3CONH 用于 P3.4-P3.7，P3CONL 用于 P3.0-P3.3。复位后，P3CONH 和 P3CONL 清为“00H”，所有 I/O 口配置为输入模式。 可通过设置控制寄存器来选择输入或输出（推挽式），以及使能复用功能。

在对此端口进行编程时，务必记得，任何由 P3 控制寄存器设置的外围复用的 I/O 功能都必须在对应的外设模块中使能。

Port 3 Control Register, High Byte (P3CONH)
EEH, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P3.7 Configuration Bits

0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = N-channel open-drain output

[.5-.4] P3.6 Configuration Bits

0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = N-channel open-drain output

[.3-.2] P3.5 Configuration Bits

0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = N-channel open-drain output

[.1-.0] P3.4 Configuration Bits

0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = N-channel open-drain output

图 9-9 P3 口高字节控制寄存器 (P3CONH)

Port 3 Control Register, Low Byte (P3CONL)
EFH, Set1, Bank0, R/W, Reset value="00"

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

[.7-.6] P3.3 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1 = N-channel open-drain output

[.5-.4] P3.2 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1= N-channel open-drain output

[.3-.2] P3.1 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1= N-channel open-drain output

[.1-.0] P3.0 Configuration Bits
0 0 = Input mode
0 1 = Input mode with pull-up
1 0 = Push-pull output mode
1 1= N-channel open-drain output

图 9-10 P3 口低字节控制寄存器 (P3CONL)

编程实例 9-1 使用 Timer A

```
                ORG             0000h

                VECTOR          0C0h,TAMC_INT
                VECTOR          0C2h,TAOV_INT

                ORG             0100h

INITIAL:
                LD              SYM,#00h            ; 禁止全局中断/快速中断 → SYM
                LD              IMR,#00000010b      ; 使能 IRQ1 中断
                LD              SPH,#00000000b      ; 设置堆栈区
                LD              SPL,#00000000b
                LD              BTCON,#10100011b    ; 禁止看门口

                LD              P1CONL,#0ABH        ; 使能TAOUT 输出
                SB1
                LD              TADATA,#80h
                LD              TACON,#01001010b    ; 匹配中断使能
                                                    ; 6.55 ms 间隔 (10 MHz 外部时钟)
                SB0

                EI

MAIN:
                •
                •
                MAIN ROUTINE
                •
                •
                JR      T,MAIN

TAMC_INT:
                •
                •
                Interrupt service routine
                •
                •
                IRET

TAOV_INT:
                •
                Interrupt service routine
                •
                IRET

                .END
```

编程实例 9-2 使用I/O口

```
                ORG     0100h

INITIAL:
                SB1                             ; 调试用的额外命令 (注释)
                LD      0F7H,#5FH               ; 调试用的额外命令 (注释)
                SB0                             ; 调试用的额外命令 (注释)

                DI
                LD      SPL,#00000000b
                LD      BTCON,#10100011b        ; 禁止看门狗
                LD      CLKCON,#18H

                LD      P0CON,#0AAH             ; P0 口设置为推挽式输出
                LD      P1CONH,#0AAH            ; P1 口设置为推挽式输出
                LD      P1CONL,#0AAH            ; P1 口设置为推挽式输出
                LD      P2CONH,#0AAH            ; P2 口设置为推挽式输出
                LD      P2CONL,#0AAH            ; P2 口设置为推挽式输出
                LD      P3CONL,#0AAH            ; P3 口设置为推挽式输出

MAIN:
                •
                •
                •
                XOR     P0,#0FH
                XOR     P1,#03FH
                XOR     P2,#0FFH
                XOR     P3,#0FH
                •
                •
                •

                JR      T,MAIN

                .END
```

注释: 调试模式时，为了正确操作端口，初始化端口程序必须包含这几句额外命令。
如果漏掉这些命令，端口则不能正确操作。程序写好以后，编译之前，必须去掉这些命令。

10 BASIC TIMER

10.1 概述

10.1.1 BASIC TIMER(BT)

Basic Timer 主要应用在两个方面：

- 当系统出现异常时，Basic Timer 作为看门狗定时器，自动复位芯片。
- 在复位或退出 STOP 模式后，用于延时以稳定振荡。

Basic Timer 单元包含以下几个部分：

- 时钟分频器（fosc/4096/1024/128）
- 8 位 Basic Timer 计数器 BTCNT (地址：FDH，Set 1， Bank 0，只读)
- Basic Timer 控制寄存器 BTCON (地址：D3H，Set 1，可读/写)

10.1.2 BASIC TIMER 控制寄存器(BTCON)

Basic Timer 控制寄存器 BTCON，用于选择输入时钟频率，清除 BT 计数器和分频器，禁止或使能看门狗功能。

复位后，BTCON 被清零“00H”，该选项将使能看门狗功能，并选择 BT 的时钟为 fxx/4096。
如果想取消看门狗功能，必须将 Basic Timer 控制寄存器的高四位 BTCON.7–BTCON.4 设置为“1010B”。

在正常操作情况下，可以写“1”到 BTCON.1 来清零 8 位 Basic Timer 计数器 BTCNT。
也可以写“1”到BTCON.0 位，来清除 Basic Timer 输入时钟的分频器。

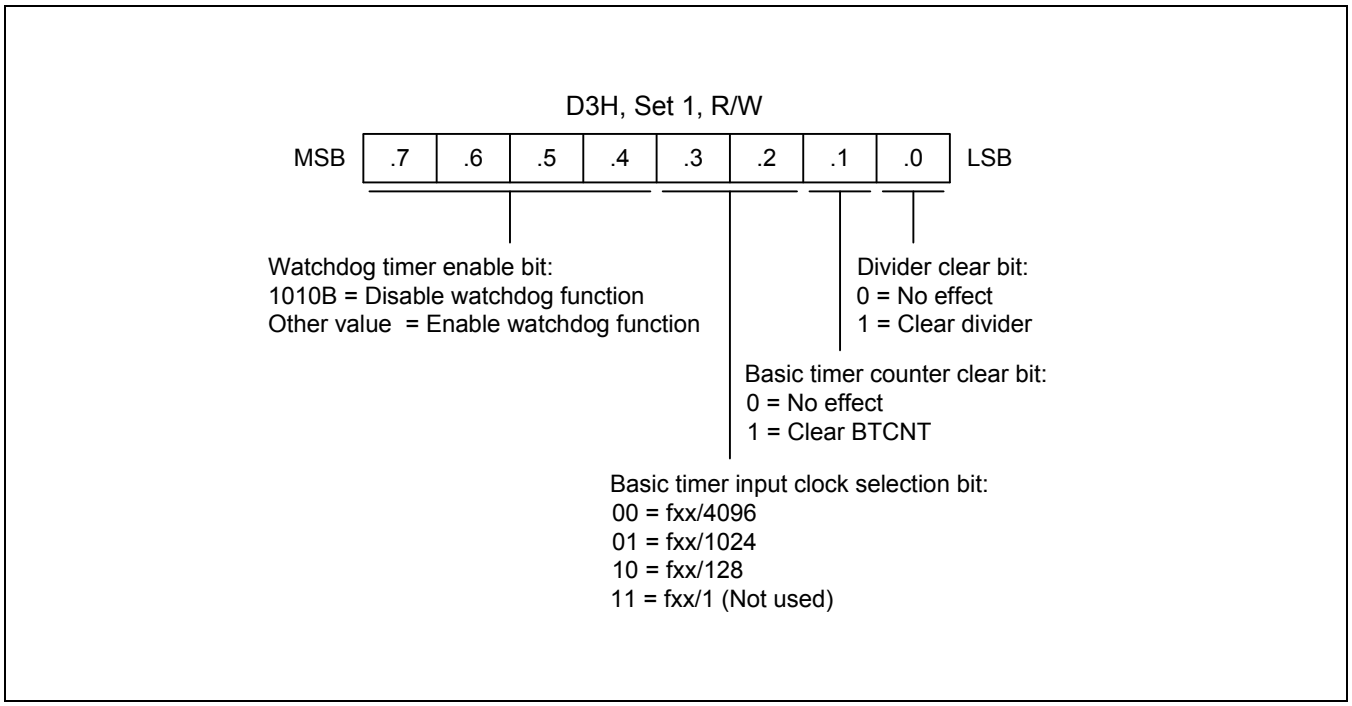


图 10-1 Basic Timer 控制寄存器 (BTCON)

10.1.3 BASIC TIMER 功能描述

10.1.3.1 看门狗定时器功能

写入 BTCON.7 - BTCON.4 位的值如果不是“1010B”，将会允许看门狗功能。当 BTCNT 溢出时，将产生 MCU 复位信号。复位时清 BTCON 为“00H”，允许看门狗功能。同时还复位选择 Basic Timer 的时钟信号为 fxx/4096。

无论什么时候 BT 计数器溢出，都将产生复位信号。在正常情况下，应用程序须防止计数器溢出，为此， 每隔一段时间要清除看门狗计数器 (写“1”到 BTCON.1)。

如果由于电路噪声或其它原因造成系统失灵，Basic Timer 的清零操作将不会执行，看门狗将复位芯片。 换言之，在正常运行期间，必须通过 BTCNT 的清零操作来打破 Basic Timer 的溢出循环。 如果发生异常，复位自动产生。

10.1.3.2 振荡稳定功能

还可以用 Basic Timer 对复位或 STOP 模式退出时的振荡稳定时间进行定时。

在 STOP 模式下，无论复位或者外部中断发生，振荡电路开始运行。然后 BTCNT 寄存器按照 fxx/4096 的频率（复位时）或者按照预设时钟 (对外部中断)开始计数。当 BTCNT.4 被置起时，将产生一个信号说明振荡已经稳定，CPU 开始在稳定的时钟下开始工作。

当系统从 STOP 模式退出时，以下的事件会发生：

1. 上电复位或外部中断产生，系统退出 STOP 状态，振荡电路起振。
2. 如果是复位操作，BTCNT 以 fxx/4096 的时钟频率计数。如果是外部中断，BTCNT 寄存器就按照预先设置的时钟计数。
3. BTCNT.4 位被置起，振荡达到稳定。
4. CPU 开始正常工作。

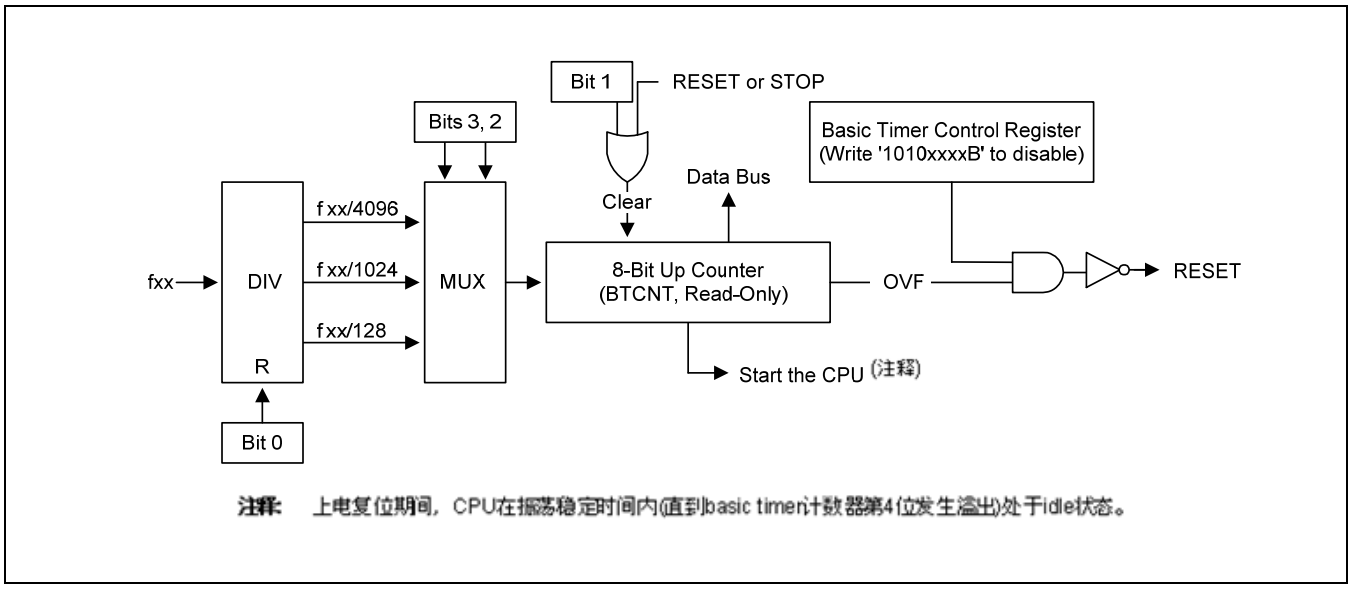


图 10-2 Basic Timer 方框图

11 8 位 TIMER A/B

11.1 8 位 TIMER A

11.1.1 概述

Timer A 是一个 8 位通用 Timer/Counter 。它有三种工作模式，可通过配置 寄存器TACON 来选择其中的一种：

- Interval（定时）模式 (TAOUT 反转输出)
- Capture（捕获）模式，TACAP 上升沿或下降沿触发
- PWM 模式 (TAPWM)

Timer A 包括以下几个功能模块：

- 带多路复用的时钟分频器 (fxx/1024，fxx/256，fxx/64)
- 外部时钟输入管脚 (TACK)
- 一个 8 位计数器(TACNT)，一个 8 位比较器和一个 8 位参考数据寄存器(TADATA)
- I/O 管脚用于捕获输入管脚 (TACAP) 或 PWM 输出或匹配输出 (TAOUT)
- Timer A 溢出中断 (IRQ1，中断向量地址：C2H) 和匹配/捕获中断 (IRQ1，中断向量地址：C0H) 产生
- Timer A 控制寄存器 TACON (地址：Set 1，Bank 1，E1H，可读/写)

11.1.2 功能描述

11.1.2.1 Timer A 中断(IRQ1, 中断向量地址：C0H 和 C2H)

Timer A 模块能产生 2 种中断：Timer A 溢出中断 (TAOVF)，以及 Timer A 匹配/捕获中断 (TAINT)。TAOVF 的中断优先级为 IRQ1，中断向量地址为 C2H。TAINT 的中断优先级也为 IRQ1，但不同的是中断向量地址为 C0H。

当 Timer A 溢出中断被响应时，硬件自动清除中断标志位。同样，Timer A 的匹配/捕获中断被响应以后，也是硬件方式自动清除中断标志位。

11.1.2.2 Interval(定时)模式

Timer A 模块能产生Timer A 匹配中断 (TAINT)。TAINT 的中断优先级为 IRQ1，中断向量地址为 C0H。

当 Timer A 匹配产生时，CPU 响应该中断，硬件自动清除中断标志位。

在定时模式中，当计数器的值与 Timer A 参考数据寄存器 TADATA 写入的值相等时，将产生一个匹配信号，同时 TAOUT 产生反转。这个匹配信号产生一个 Timer A 的 匹配中断(TAINT，中断向量地址：C0H) 并将计数器清零。

举个例子，如果写 10H 到 TADATA ，往 TACON 写 0AH，在计数器将增计数直到值为 10H。当计数器值达到 10H时，Timer A 中断请求产生，计数器的值复位，之后重新计数。

11.1.2.3 PWM 模式

脉冲宽度调制(PWM) 模块可以让用户编程控制 TAOUT 管脚上输出脉冲宽度（持续时间）。 和定时模式一样，当计数器的值和 Timer A 数据寄存器写入的值一致时，产生一个匹配信号。不同的是，在 PWM 模式中，这个匹配信号并不将计数器清零，而是继续计数直到 FFH 溢出，之后从 00H 重新开始向上计数。

虽然在 PWM 模式下可以使用 Timer A 的匹配信号产生一个 Timer A 溢出中断，但这些中断在 PWM 类型的应用中并不普遍。当参考数据值小于或者等于(≤)计数器值时，TAOUT 管脚保持低电平； 而当参考数据值大于(>)计数器值时，TAOUT 管脚保持高电平。一个周期为 256 个 tCLK 。

11.1.2.4 Capture(捕获)模式

在捕获模式中，当 TACAP 管脚上检测到一个信号沿时，就会把当前计数器的值载入 Timer A 数据寄存器中。 可以选择上升沿或者下降沿触发。

Timer A 提供了捕获输入源：TACAP 管脚上的信号沿。通过设置 P1 口低字节控制寄存器 P1CON（地址：E9H，Set 1，Bank 0）的 Timer A 捕获输入选择位来选择捕获输入。 当 P1CON.5-.4 位为“00B”或“01B”时, 选择作为 TACAP 输入或普通输入；当 P1CON.5-.4 位为“10B”时，选择作为普通推挽式输出。

Timer A 的两种中断均可用在捕获模式：当计数器溢出时可以产生 Timer A 溢出中断；而当计数器值载入到 Timer A 数据寄存器时可以产生 Timer A 匹配/捕获中断。

通过读取 TADATA 中的捕获数据值，并为 Timer A 假定一个特定的时钟频率，就可以计算出 TACAP 管脚上输入信号的脉冲宽度（持续时间）。

11.1.3 TIMER A 控制寄存器 (TACON)

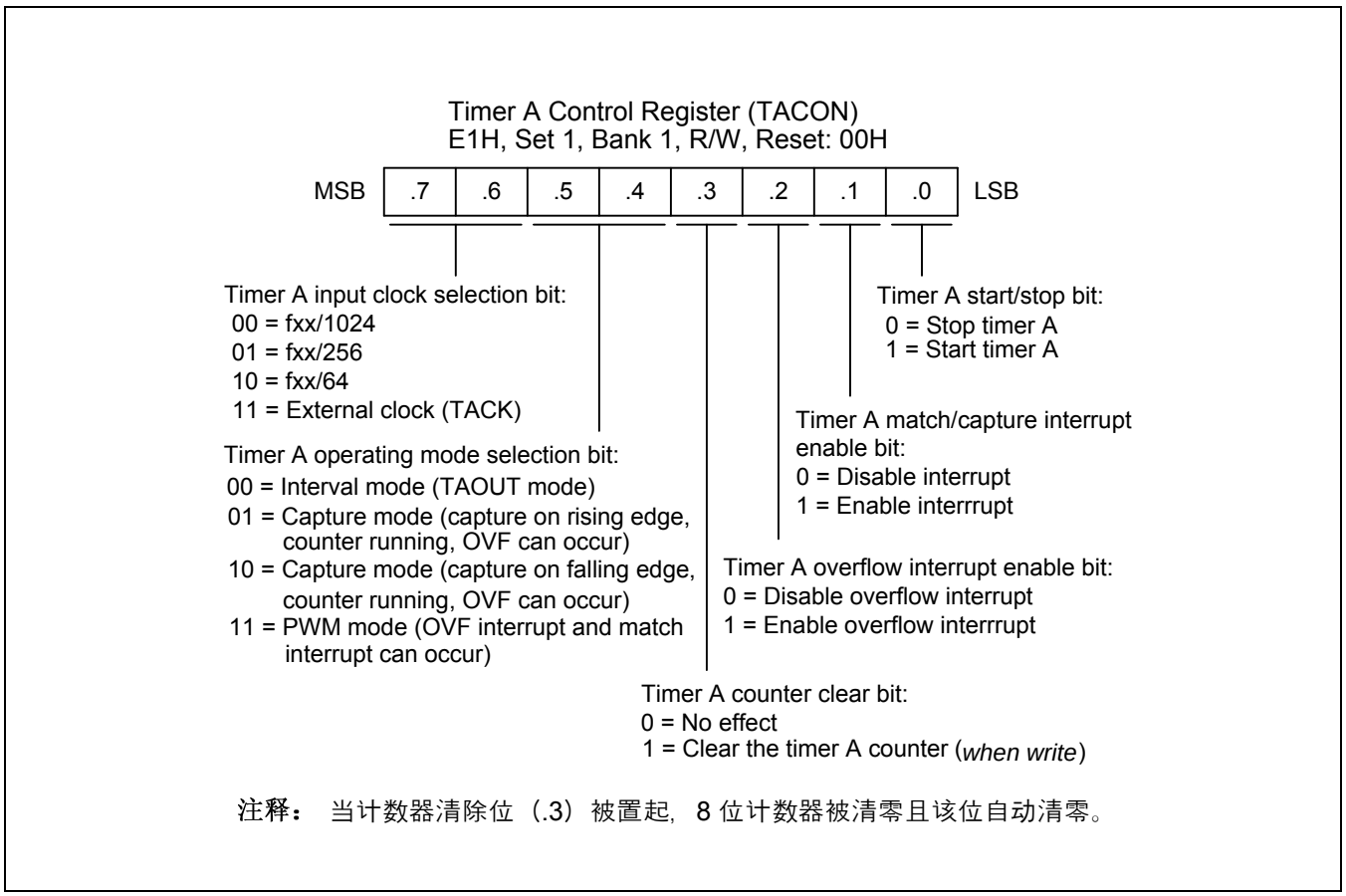
Timer A 控制寄存器 TACON 可以：

- 选择 Timer A 工作模式 (定时模式，捕获模式和 PWM 模式)
- 选择 Timer A 输入时钟频率
- 将 Timer A 计数器 TACNT 清零
- 使能 Timer A 溢出中断或 Timer A 匹配/捕获中断
- 清除 Timer A 匹配/捕获中断标志位

TACON 地址为：E1H，Set 1，Bank 1。可读/写，支持寄存器寻址模式。复位时，将清除 TACON 至 “00H”，将设置 Timer A 为定时模式，选择 fxx/1024 作为输入时钟频率，并且禁止所有的 Timer A 中断。
正常工作时，任何时刻都可通过往 TACON.3 位写“1”来将 Timer A 计数器清零。

Timer A 的溢出中断 (TAOVF) 的中断优先级为 IRQ1，中断向量地址为 C2H。当 Timer A 溢出中断发生后，CPU 响应此中断，硬件将自动清除中断标志位。

往 TACON.1 位写“1”可使能 Timer A 的匹配/捕获中断 (IRQ1，中断向量地址：C0H)。为了产生精确的定时间隔，应往 TACON.3 和 .0 位写“1”来清除计数器以及中断标志位。执行中断服务程序时，中断标志位可以通过软件的方法清除，即写“0”到中断标志位(TINTPND.0 或 TINTPND.1 位)。也可以硬件清零。



注释： 当计数器清除位 （.3）被置起，8 位计数器被清零且该位自动清零。

图 11-1 Timer A 控制寄存器 (TACON)

11.1.4 模块框图

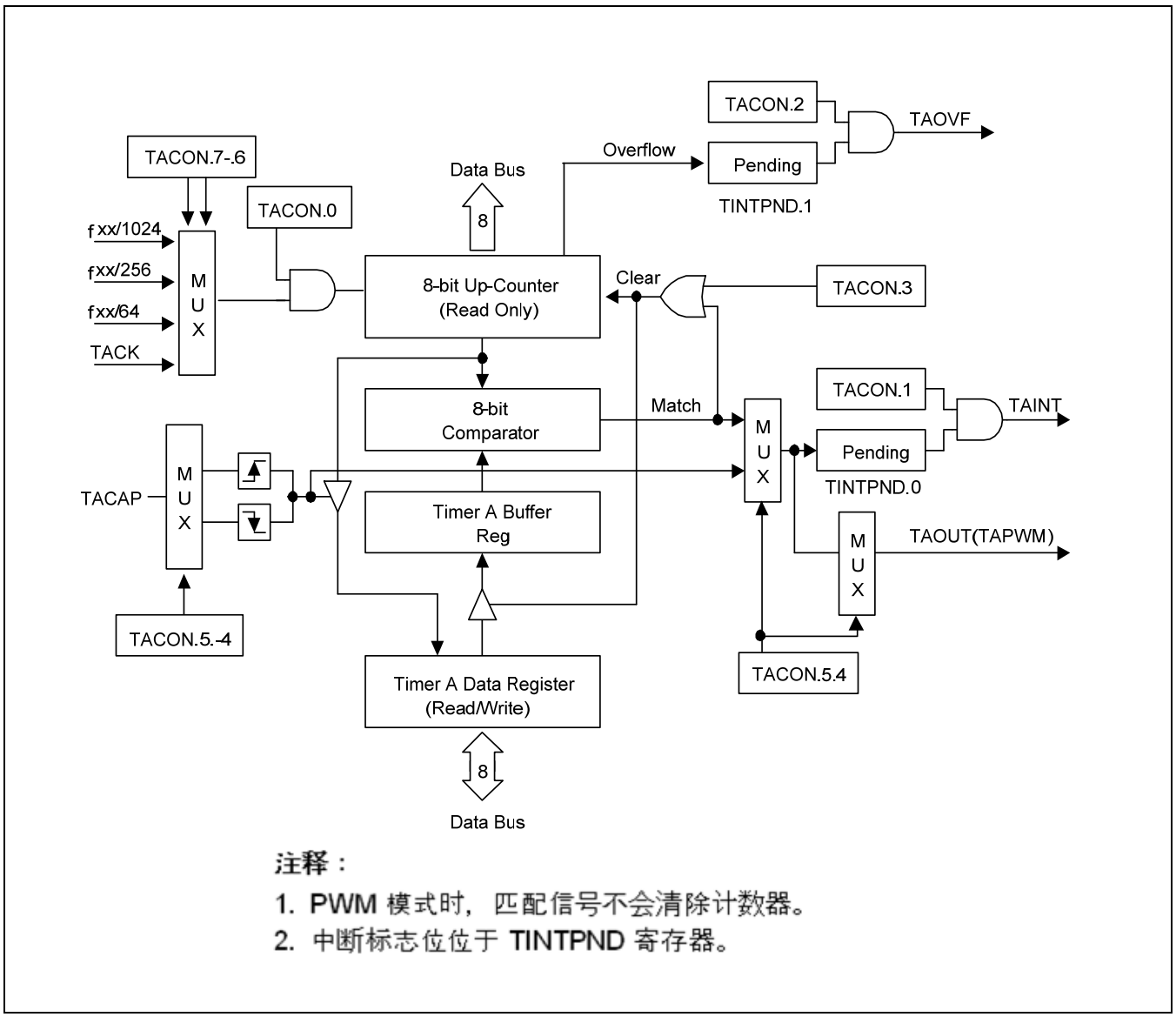


图 11-2 Timer A 功能模块框图

11.2 8 位 TIMER B

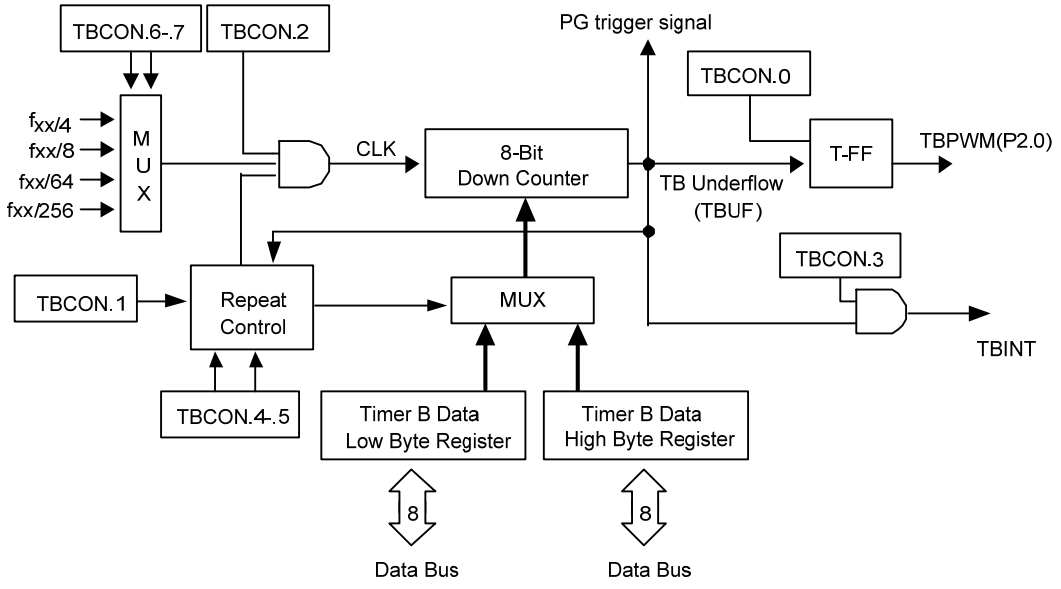
11.2.1 概述

T S3C84H5/F84H5 单片机有一个 8 位 Timer B，用于产生遥控器信号的载波频率。 另外，它也可用作可编程蜂鸣信号发生器，能发出从 200Hz 到 20KHz 可变频率的声音。这些可变频率可用于产生曲调。

Timer B 有 2 个功能：

- 作为普通定时时钟，在编程的时间间隔内产生 Timer B 中断
- 在 P2.0 管脚处产生一个可编程载波脉冲用于遥控器信号

11.2.2 模块框图



注释： 当 TBCON.5-.4位设置为 “10B” 时，TBDATAL 寄存器的值在 Timer B 开始工作时载入 8 位计数器。之后,计数器下溢发生时，TBDATAH 寄存器的值载入 8 位计数器。无论如何，只要下一次借位发生，TBDATAL 寄存器的值载入 8 位计数器中。用户通过设置 P2CONL.1-.0 为 “11B” 来获得 TBPWM 作为载波输出。

图 11-3 简化的Timer B 功能框图

11.2.3 TIMER B 控制寄存器 (TBCON)

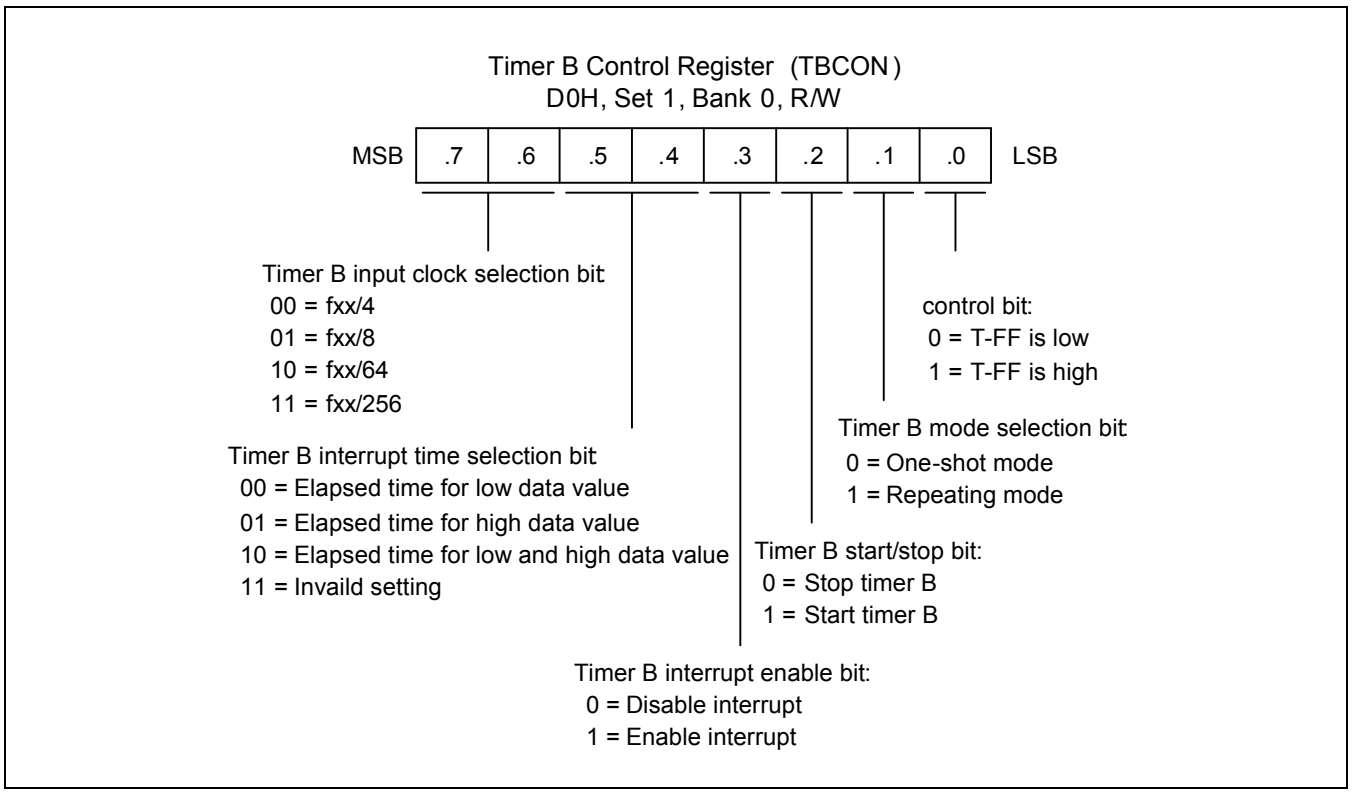


图 11-4 Timer B 控制寄存器 (TBCON)

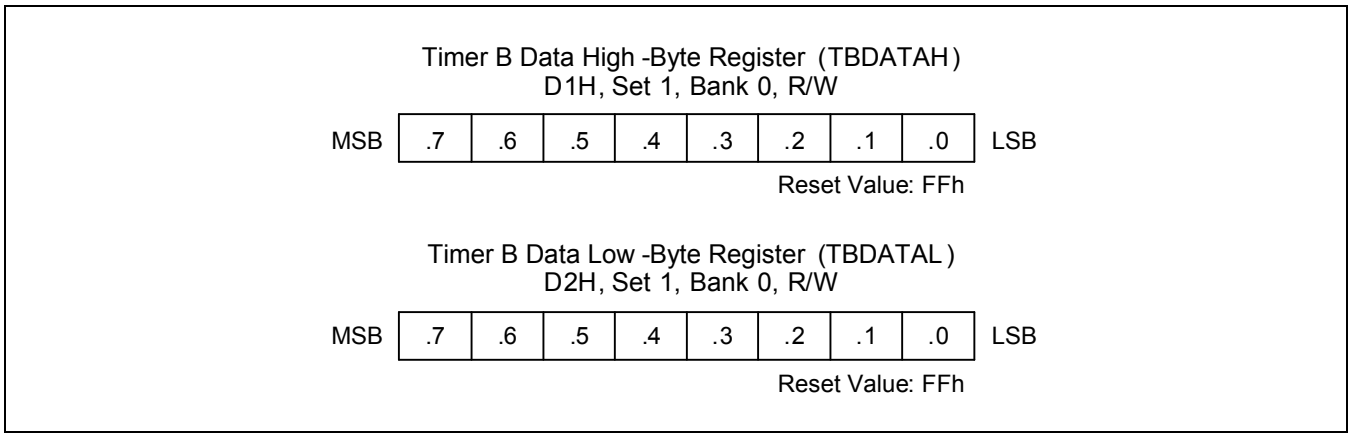
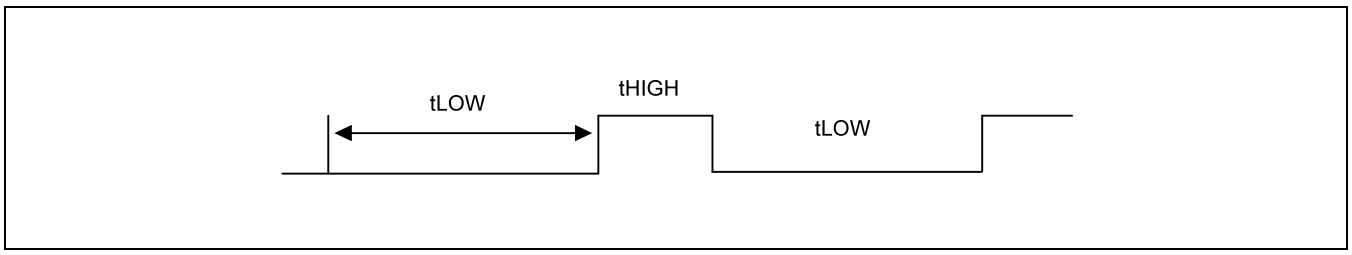


图 11-5 Timer B 数据寄存器 (TBDATAH, TBDATAL)

11.2.4 TIMER B 脉冲宽度计算



为产生如上所示重复的波形，需要由低脉宽时间 t_{LOW} 和高脉宽时间 t_{HIGH} 。

当 T-FF = 0 时，

t_{LOW} = (TBDATAL + 1) x 1/fx，0H < TBDATAL < 100H，此时 fx = 已选的时钟。

t_{HIGH} = (TBDATAH + 1) x 1/fx，0H < TBDATAH < 100H，此时 fx = 已选的时钟。

当 T-FF = 1 时，

t_{LOW} = (TBDATAH + 1) x 1/fx，0H < TBDATAH < 100H，此时 fx = 已选的时钟。

t_{HIGH} = (TBDATAL + 1) x 1/fx，0H < TBDATAL < 100H，此时 fx = 已选的时钟。

为获得 t_{LOW} = 24 us 以及 t_{HIGH} = 15 us。 f_{OSC} = 4 MHz，fx = 4 MHz/4 = 1 MHz

当 T-FF = 0 时，

t_{LOW} = 24 us = (TBDATAL + 1) /fx = (TBDATAL + 1) x 1us，TBDATAL = 23。

t_{HIGH} = 15 us = (TBDATAH + 1) /fx = (TBDATAH + 1) x 1us，TBDATAH = 14。

当 T-FF = 1 时，

t_{HIGH} = 15 us = (TBDATAL + 1) /fx = (TBDATAL + 1) x 1us，TBDATAL = 14。

t_{LOW} = 24 us = (TBDATAH + 1) /fx = (TBDATAH + 1) x 1us，TBDATAH = 23。

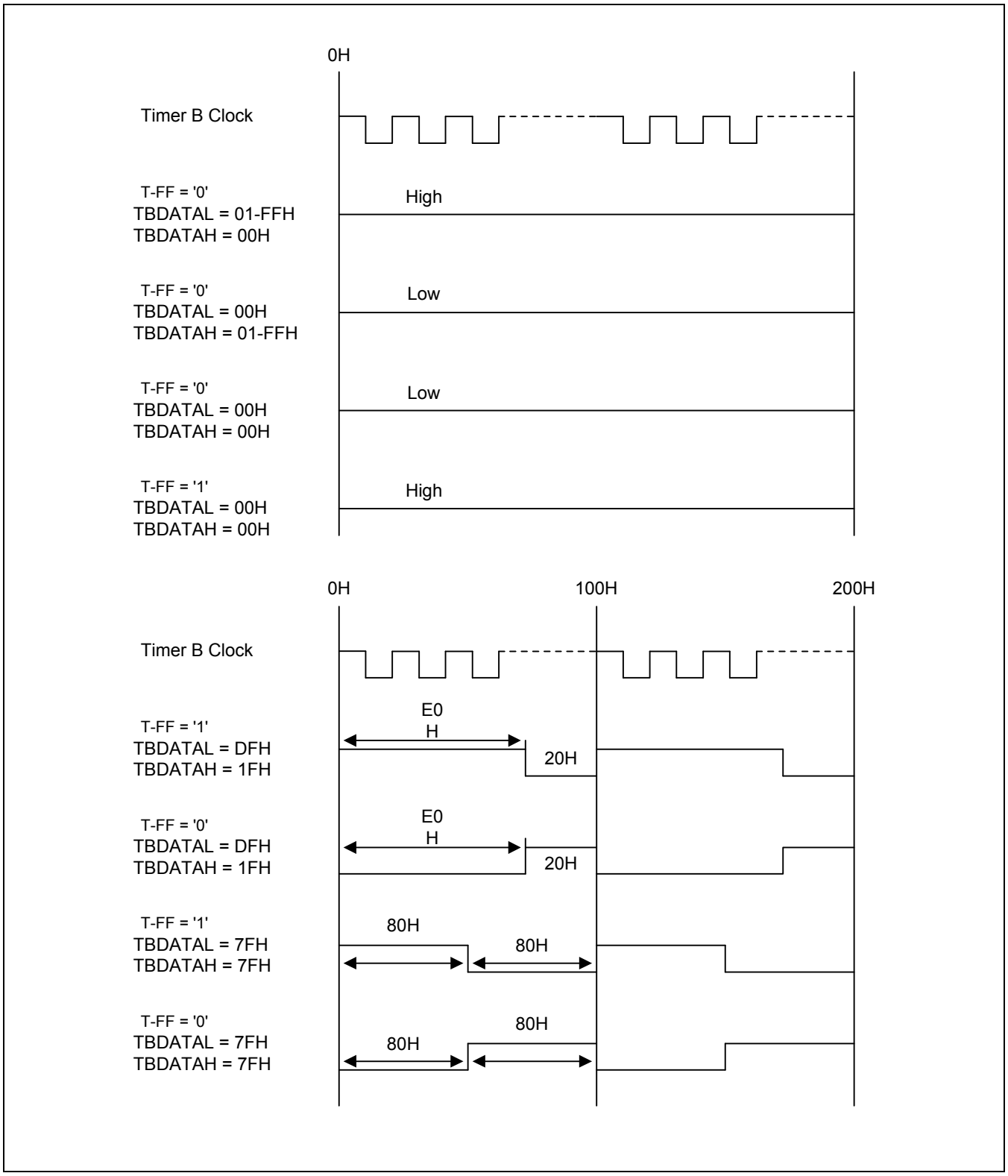
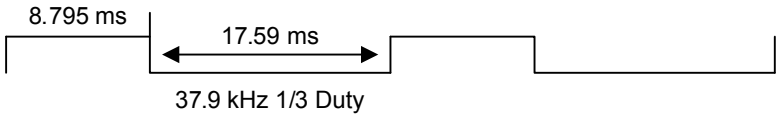


图 11-6 Timer B 在重复模式下输出触发器波形

编程实例 11-1 如何在 P2.0 处产生 38 kHz, 1/3 占空比信号

本例中设置 Timer B 为重复模式，并且设置系统时钟为 Timer B 时钟源，以及设置 TBDATAH 和 TBDATAL 获得一个 38 kHz，1/3 占空比得载波频率。该编程参量如下：

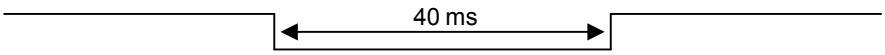


- Timer B 用于重复模式
- 时钟频率为 16 MHz (0.0625 μs)，fx=fxx/4=4MHz (0.25 μs)
- TBDATAH = 8.795 μs/0.25 μs = 35.18，TBDATAL = 17.59 μs/0.25 μs = 70.36
- 设置 P2.0 为 TBPWM 模式

```
                ORG     0100H                   ; 复位地址
START           DI
                •
                •
                •
                LD      TBDATAL,#(35-1)         ; 设置 8.75 μs
                LD      TBDATAH,#(70-1)         ; 设置 17.5 μs
                LD      TBCON,#00100111B        ; 时钟源 ← fxx/4
                                                ; 禁止 Timer B 中断
                                                ; 选择 Timer B 为重复模式
                                                ; 启动 Timer B 工作
                                                ; 设置 Timer B 输出触发 (T-FF) 高
                                                ;
                LD      P2CONL,#03H             ; 设置 P2.0 为 TBPWM 模式
                                                ; 该指令在 P2.0 处产生 38 kHz，1/3 占空比脉冲信号
                •
                •
                •
```

编程实例 11-2 在 P2.0 处产生一个脉冲信号

本例中设置 Timer B 为单触发模式，并且设置系统时钟为 Timer B 时钟源，设置合适的 TBDATAH 和 TBDATAL 以获得一个 40 s 宽度脉冲。参数如下：



- Timer B 用于单触发模式
- 时钟频率为 4 MHz (fx=1/4 clock = 1 μs)
- TBDATAH = 40 μs / 1 μs = 40，TBDATAL = 1
- 设置 P2.0 为 TBPWM 模式

```
                ORG     0100H               ; 复位地址
START           DI
                •
                •
                •
                LD      TBDATAH,# (40-1)    ; 设置 40 μs
                LD      TBDATAL,# 1         ; 设置为除 00H 外任何值
                LD      TBCON,#00010001B    ; 时钟源 ← fxx/4
                                            ; 禁止 Timer B 中断
                                            ; 设置 Timer B 为单触发模式
                                            ; 停止 Timer B 工作
                                            ; 设置 Timer B 输出触发 (T-FF) 高
                LD      P2CONL,#03H         ; 设置 P2.0 为 TBPWM 模式
                •
                •
PULSE_OUT:      LD      TBCON,#00000101B    ; 启动 Timer B 工作
                                            ; 为在此刻获得该脉冲
                •                           ; 该指令执行后，在脉冲下降沿开始前需要 0.75μs
                •
                •
```

编程实例 11-3 使用 Timer A

```
                ORG             0000h

                VECTOR          0C0h,TAMC_INT
                VECTOR          0C2h,TAOV_INT

                ORG             0100h

INITIAL:
                LD              SYM,#00h            ; 禁止全局/快速中断 → SYM
                LD              IMR,#00000010b      ; 使能 IRQ1 中断
                LD              SPL,#00000000b
                LD              BTCON,#10100011b    ; 禁止看门狗

                LD              P1CONL,#0ABH        ; 使能 TAOUT 输出
                SB1
                LD              TADATA,#80h
                LD              TACON,#01001010b    ; 使能匹配中断
                                                    ; 持续时间 6.55 ms (10 MHz x'tal)

                SB0

                EI

MAIN:
                •
                •
                MAIN ROUTINE
                •
                •

                JR      T,MAIN

TAMC_INT:
                •
                •
                Interrupt service routine
                •
                •
                IRET

TAOV_INT:
                •
                Interrupt service routine
                •
                IRET

                .END
```

编程实例 11-4 使用 Timer B

```
                ORG             0000h

                VECTOR          0BEh,TBUN_INT

                ORG             0100h

INITIAL:
                LD              SYM,#00h                ; 禁止全局/快速中断
                LD              IMR,#00000001b          ; 使能 IRQ0 中断
                LD              SPL,#00000000b
                LD              BTCON,#10100011b        ; 禁止看门狗

                LD              P2CONL,#03H             ; 使能 TBPWM 输出

                LD              TBDATAH,#80h
                LD              TBDATAL,#80h
                LD              TBCON,#11101110b        ; 使能中断, fxx/256, 重复模式
                                                        ; 持续时间 6.605ms (10 MHz x'tal)

                EI

MAIN:
                •
                •
                •
                MAIN ROUTINE
                •
                •
                •

                JR              T,MAIN

TBUN_INT:
                •
                •
                •
                Interrupt service routine
                •
                •
                •
                IRET

                .END
```

12 16 位TIMER 1(0,1)

12.1 概述

S3C84H5/F84H5 有2个16位的timer/counter。16位 Timer1(0,1) 是一个16位的通用 timer/counter。Timer1(0,1)有3种工作模式，通过设置T1CON0、T1CON1寄存器可以选择任一工作模式：

- 定时器模式 (T1OUT0、T1OUT1 输出)
- 捕获输入模式 (T1CAP0、T1CAP1的上升沿或下降沿触发)
- PWM 模式 (T1PWM0、T1PWM1)；PWM 输出与T1OUT0、T1OUT1输出管脚复用

Timer 1(0,1) 有如下的功能模块：

- 带多路复用的时钟分频器 (fxx /4096，fxx /256， fxx $_C$/8，fxx /1)
- 外部时钟输入管脚 (T1CK0, T1CK1)
- 16位计数器 (T1CNTH0/L0, T1CNTH1/L1)，16 位比较器以及16位数据寄存器 (T1DATAH0/L0, T1DATAH1/L1)
- 捕获输入管脚 (T1CAP0，T1CAP1)，或者匹配输出管脚 (T1OUT0, T1OUT1)
- 产生Timer 1(0) 溢出中断 (IRQ2, 中断向量地址 C6H) 和匹配/捕获中断 (IRQ2, 中断向量地址 C4H)
- 产生Timer 1(1) 溢出中断 (IRQ2, 中断向量地址 CAH) 和匹配/捕获中断 (IRQ2, 中断向量地址 C8H)
- Timer 1(0) 控制寄存器 T1CON0 (set 1, E8H, Bank 1, 读/写)
- Timer 1(1) 控制寄存器 T1CON1 (set 1, E9H, Bank 1, 读/写)

12.1.1 功能描述

12.1.1.1 Timer 1(0,1) 中断 (IRQ2，中断向量地址 C4H, C6H, C8H 和 CAH)

Timer 1(0) 模块能产生2个中断，Timer1(0) 溢出中断 (T1OVF0) 以及Timer 1(0) 匹配/捕获中断 (T1INT0)。T1OVF0的中断优先级为IRQ2，向量地址为C6H。T1INT0的中断优先级也为IRQ2，不同的是向量地址为C4H。

Timer 1(0) 溢出中断标志在中断响应后由硬件自动清除。Timer 1(0) 匹配/捕获中断T1INT0中断标志也在中断响应后由硬件自动清除。

Timer 1(1) 模块能产生2个中断，Timer 1(1) 溢出中断 (T1OVF1) 以及Timer 1(1) 匹配/捕获中断 (T1INT1)。T1OVF1的中断优先级为IRQ2，向量地址为CAH。T1INT1的中断优先级也为IRQ2，不同的是向量地址为C8H。

Timer 1(1) 溢出中断标志在中断响应后由硬件自动清除。Timer 1(1) 匹配/捕获中断T1INT0中断标志也在中断响应后由硬件自动清除。

12.1.1.2 定时器模式 (匹配)

Timer 1(0) 模块能生成Timer 1(0) 匹配中断 (T1INT0)。T1INT0的中断优先级为IRQ2，中断向量地址为C4H。在定时器模式中，当计数器的值与Timer 1参考数据寄存器T1DATAH0、T1DATAL0的值相等时，生成一个匹配信号并且T1OUT0信号产生反转。匹配信号生成一个Timer 1(0) 匹配中断 (T1INT0，向量地址 C4H) 同时计数器清零。

Timer 1(1) 模块能生成定时器 1(1) 匹配中断 (T1INT1)。T1INT1的中断优先级为IRQ2，中断向量地址为C8H。在定时器模式中，当计数器的值与Timer 1参考数据寄存器T1DATAH1、T1DATAL1的值相等时，生成一个匹配信号并且T1OUT1信号产生反转。匹配信号生成一个定时器 1(1) 匹配中断(T1INT1, 向量地址 C8H) 同时计数器清零。

12.1.1.3 捕获模式

Timer 1(0) 的捕获模式下，一旦管脚T1CAP0上检测到信号边沿就将当前计数器的值载入Timer 1 数据寄存器 (T1DATAH0，T1DATAL0，上升/下降沿)中。用户可以选择上升沿或者下降沿来触发此操作。Timer 1(0) 捕获输入源：管脚T1CAP0上的信号边沿。可以通过设置P0高字节控制寄存器P0CONH (set 1 bank0，E6H) 来选择捕获输入。

Timer 1(0) 的两种中断 (T1OVF0, T1INT0) 均可用于捕获模式，当计数器溢出时产生 Timer 1(0) 溢出中断； 当计数器值载入 Timer 1数据寄存器时产生 Timer 1(0) 捕获中断。

通过读取T1DATAH0 和 T1DATAL0寄存器中的值，并为Timer 1(0) 假设一个特定的时钟频率，就可以计算出管脚 T1CAP0 上输入信号的脉冲宽度（持续时间）。

Timer 1(1) 的捕获模式下，一旦管脚 T1CAP1 上检测到信号边沿就将当前计数器的值载入 Timer 1 数据寄存器 (T1DATAH1, T1DATAL1, 上升/下降沿)。用户可以选择上升沿或者下降沿来触发此操作。Timer 1(1) 提供了捕获输入源：管脚 T1CAP1 上的信号边沿。可以通过设置 P0 低字节控制寄存器 P0CONL (set 1 bank0, E7H) 来选择捕获输入。

Timer 1(1) 的两种中断 (T1OVF1, T1INT1) 均可用于捕获模式，当计数器溢出时产生Timer 1(1) 溢出中断；当计数器值载入定时器1数据寄存器时产生 Timer 1(1) 捕获中断。

通过读取 T1DATAH1 和 T1DATAL1寄存器中的值，并为 Timer 1(1) 假设一个特定的时钟频率， 就可以计算出管脚 T1CAP1 上输入信号的脉冲宽度 (持续时间)。

12.1.1.4 PWM 模式

脉宽调制模块 (PWM) ，用户可以控制管脚 T1OUT0 和 T1OUT1 上输出的脉冲宽度 (持续时间)。 在定时器模式下，当计数器的值与Timer 1(0, 1) 数据寄存器中的值相等时，产生一个匹配信号和匹配中断， 并将计数器清零。然而在 PWM 模式下，虽然会产生匹配中断，但不清零计数器，它会连续工作，直到 FFFFH 时溢出，然后从 0000H 继续向上计数。当溢出发生时，产生溢出中断 (T1OVF0, 1)。

虽然在 PWM 模式下可以使用匹配或者溢出中断，但是这些中断在 PWM 应用中并不普遍。 常用到的功能是，管脚 T1OUT0、T1OUT1 上的脉冲，在参考数据寄存器中的值小于等于计数器的值时保持低电 平；而在参考数据寄存器中的值大于计数器的值时保持高电平。一个脉冲宽度等于tCLK。

12.1.2 TIMER 1(0, 1) 控制寄存器(T1CON0, T1CON1)

Timer 1(0, 1) 控制寄存器 T1CON0，T1CON1可以：

- 选择 Timer 1(0, 1) 工作模式 (定时器模式, 捕获模式, PWM模式)
- 选择 Timer 1(0, 1) 输入时钟频率
- 清零 Timer 1(0, 1) 计数器 T1CNTH0/L0, T1CNTH1/L1
- 使能 Timer 1(0, 1) 溢出中断
- 使能 Timer 1(0, 1) 匹配/捕获中断

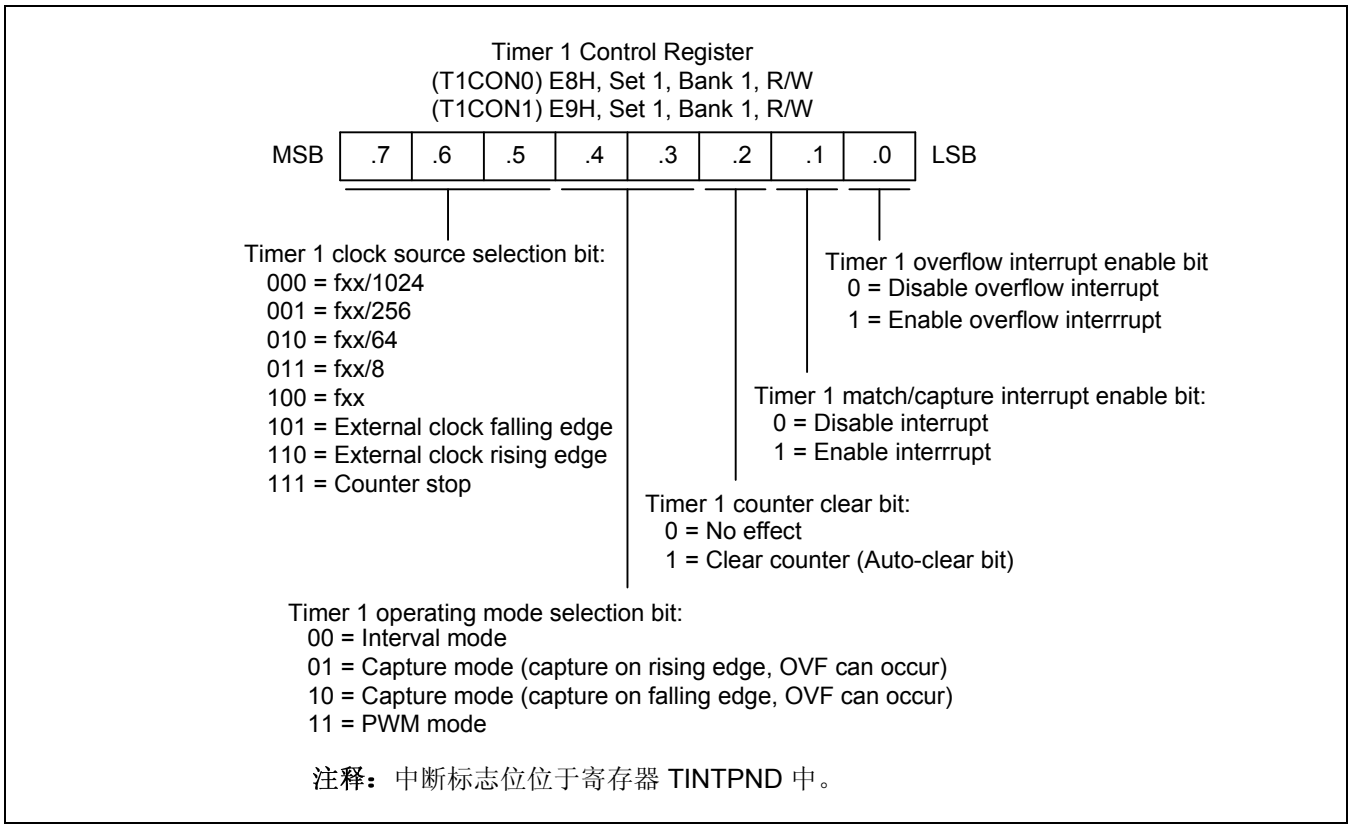
T1CON0 位于 set 1，Bank 1 的地址 E8H，可以通过寄存器寻址模式读/写。T1CON1 位于 set 1，Bank 1 的地址 E9H，可以通过寄存器寻址模式读/写。

复位操作将置 T1CON0 和 T1CON1 为“00H”，这将设置 Timer 1(0, 1) 为定时器模式，选择输入的时钟频率为 fxx/1024，并禁止所有 Timer 1(0, 1) 中断。可以通过设置 T1CON1(0, 1) .7 - .5 为“111B” 来禁止计数器操作。可以通过写“1”到 T0CON.3 随时清l零 Timer 0 计数器。

Timer 1(0) 溢出中断 (T1OVF0) 的中断优先级为 IRQ2，中断向量地址为 C6H。Timer 1(1) 溢出中断 (T1OVF1) 的中断优先级为 IRQ2，中断向量地址为 CAH。

为了产生准确的时间间隔，应该写“1”到 T1CON1(0, 1).2 寄存器并且清零 TINTPED 寄存器中的相应标志位。

当T1INT0、T1INT1 或者 T1OVF0、T1OVF1 禁止时，为了检测匹配/捕获或者溢出中断标志状态，应用程序必须访问 bank1，地址为 E0H 的 TINTPND 寄存器。检测到标志位为“1”时，Timer 1(0, 1) 匹配/捕获或者溢出中断挂起。当中断响应后，可以通过软件对中断标志位写“0”来清除中断标志。如果中断 (匹配/捕获或者溢出) 使能，则中断标志位可由硬件自动清除。



注释：中断标志位位于寄存器 TINTPND 中。

图 12-1 Timer 1(0,1) 控制寄存器 (T1CON0, T1CON1)

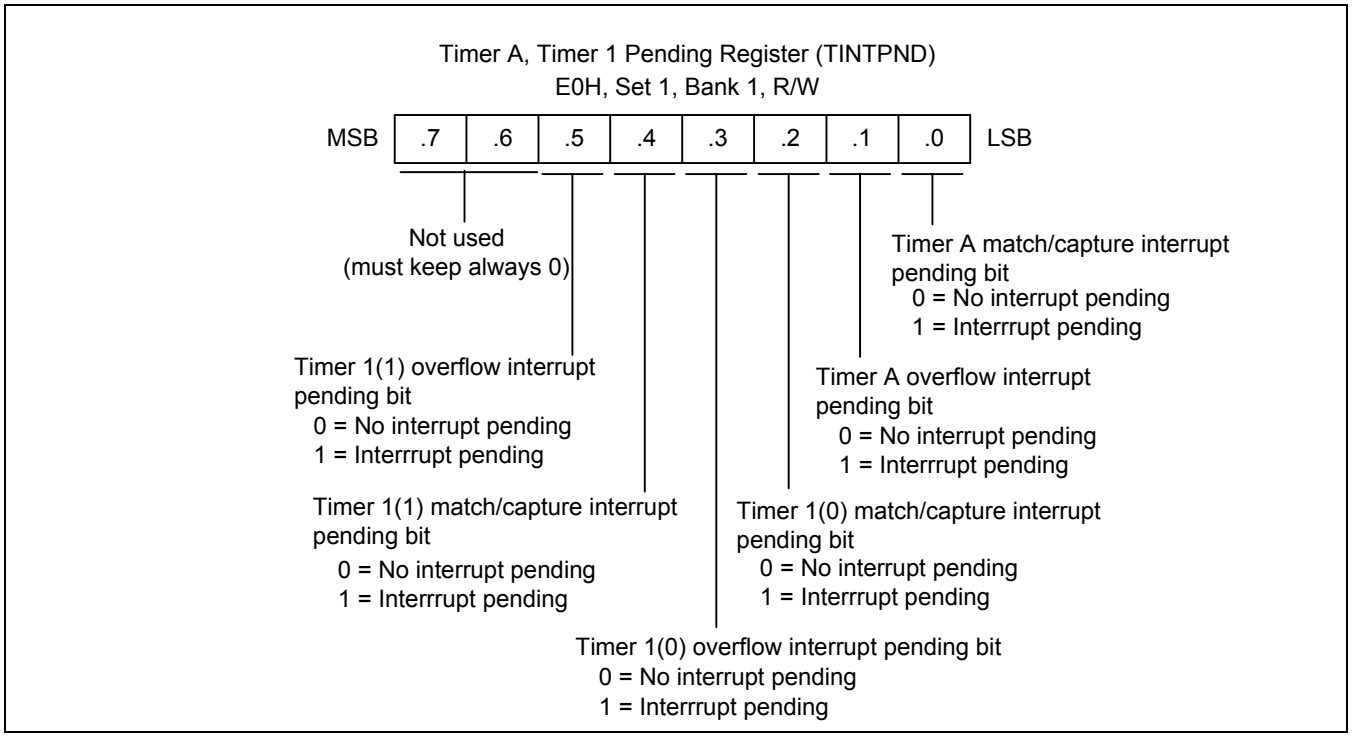
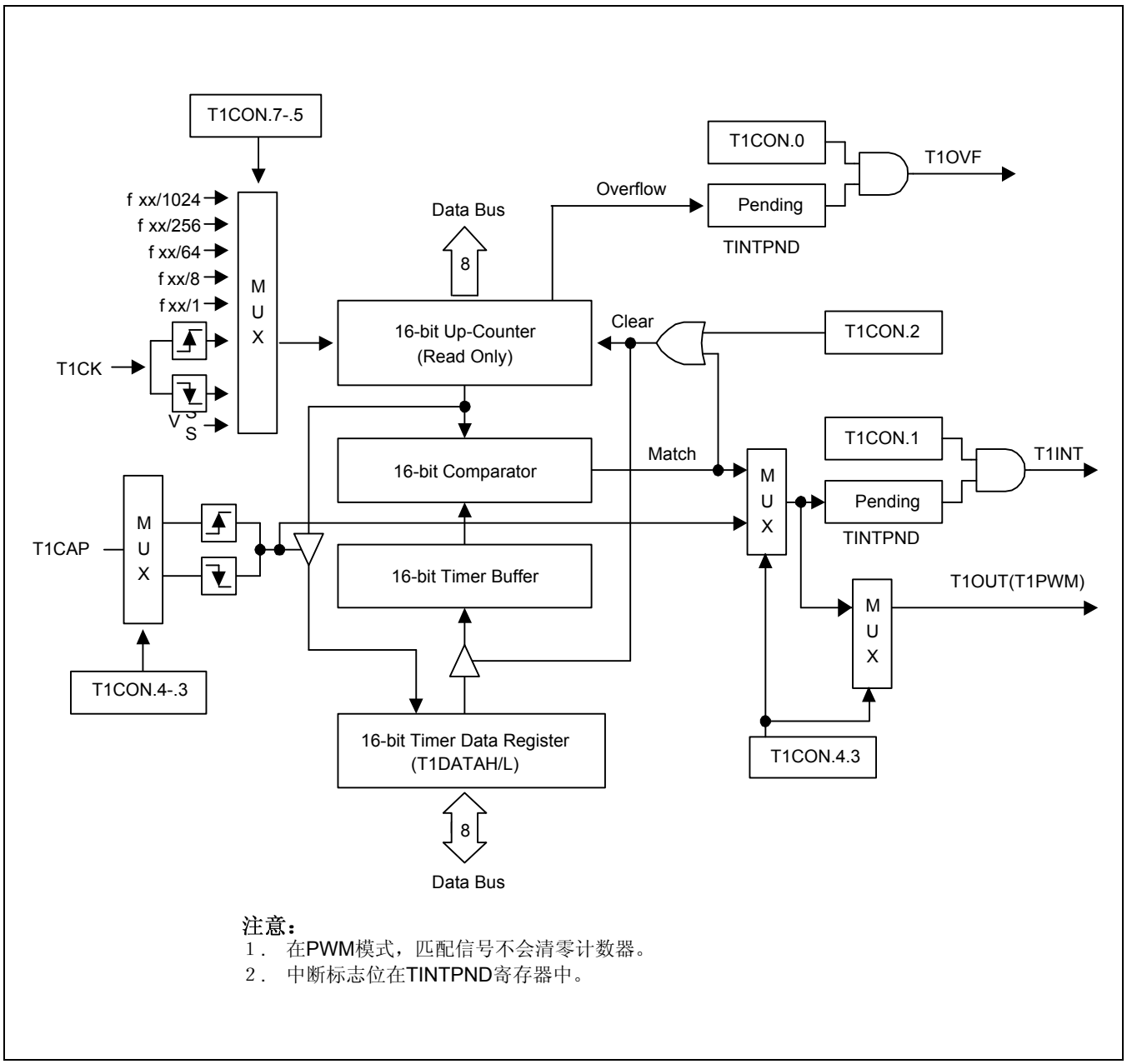


图 12-2 Timer A，Timer 1(0, 1) 中断标志寄存器 (TINTPND)

12.1.3 框图



注意：
1． 在PWM模式，匹配信号不会清零计数器。
2． 中断标志位在TINTPND寄存器中。

图 12-3 Timer 1（0，1）功能框图

编程实例 12-1 使用 Timer 1（0）

```
                ORG             0000h

                VECTOR          0C4h,TIM1_INT

                ORG             0100h

INITIAL:
                LD              SYM,#00h            ; 屏蔽全局/快速中断
                LD              IMR,#00001000b      ; 使能IRQ2中断
                LD              SPH,#00000000b      ; 设置堆栈指针
                LD              SPL,#00000000b
                LD              BTCON,#10100011b    ; 禁止看门狗功能

                SB1
                LD              T1CON0,#01000110b   ; 使能中断, 频率fxx/64, 定时器模式,
                                                    ; 定时时间= 1.536 ms (10 MHz 外部晶振)
                LDW             T1DATAH0,#00F0h     ; T1DATAH0=00h, T1DATAL0=F0h
                SB0

                EI

MAIN:
                •
                •
                •
                MAIN ROUTINE
                •
                •
                •

                JR              T,MIAN

TIM1_INT:
                •
                •
                •
                Interrupt service routine
                •
                •
                •
                IRET

                END
```

13 10 位 PWM(脉宽调制)

13.1 概述

MCU中集成了一个PWM电路。控制寄存器 PWMCON 控制 PWM 电路的运行。

PWM 计数器是一个10位自增计数器。将 PWMCON.2 设为“1”，即使能 PWM并开始计数。如果停止计数器，则该计数器会保留当前计数值。重新开始计数后，又会从原先的计数值开始计数。任何时候置高 PWMCON.3，计数值都会被清零。

通过设置 PWMCON.6-.7，可以选择 PWM 的时钟频率为f_{OSC} /64, f_{OSC} /8, f_{OSC} /2, f_{OSC} /1。

13.2 PWM 功能描述

13.2.1 PWM

10位PWM模块由以下几个部分组成：

- 8位计数器和扩展逻辑电路
- 8位比较数据寄存器 (PWMDATAH .7-.0)
- 2位扩展数据寄存器 (PWMDATAL .1-.0)
- PWM 输出管脚 (P2.1/PWM)

13.2.1.1 PWM 计数器

PWM 模块的基本波形是通过对比高 8位计数器和 PWM 比较数据寄存器(PWMDATAH .7-.0) 中的值来决定的。为了实现更高的精度，可用低 2位的扩展数据寄存器 PWMDATAL 来调制 PWM 基本波形。 通过对比扩展计数器的值和扩展数据寄存器 PWMDATAL 中写入的值，来‘延伸’PWM 输出的某几个特定周期 的占空比。所以扩展数据寄存器又被称为脉冲延伸控制寄存器。

13.2.1.2 PWM 数据比较寄存器和脉冲延伸控制寄存器

PWM数据比较寄存器和脉冲延伸寄存器分别位于Set 1, Bank1中的F3H和F4H单元。

在比较数据寄存器(PWMDATAH）和脉冲延伸控制寄存器(PWMDATAL）中写入合适的值可以获得需要的 10位 PWM 输出。设置 PWMCON.2 为“1”，则启动PWM 计数器或重新开始计数。

复位后禁止所有的PWM 输出。计数器停止,而当前的计数器值被保留。计数器启动后,会从原先计数值开始计数。

13.2.1.3 PWM 时钟频率

PWM 的输出频率由 PWMCON.6-.7 决定，是和 f_{OSC} 同步的。

表 13-1 PWM 控制寄存器和数据寄存器

寄存器名称	标号	地址	功能
PWM数据寄存器	PWMDATAH .7-.0	F3H, Set 1 Bank 1	8位 PWM 基本波形控制寄存器
	PWMDATAL .1-.0	F4H, Set 1 Bank 1	2位延伸波形控制寄存器
PWM控制寄存器	PWMCON	F5H, Set 1 Bank 1	停止/启动PWM 计数器，选择PWM 时钟

13.2.1.4 PWM 功能描述

当 8位计数器的值和存储在比较数据寄存器 PWMDATAH 中的值相匹配时，PWM 输出即切换到低电平。 如果PWMDATAH中的值非零，则 8位计数器的溢出会使 PWM 输出重新切换到高电平，完成一个 PWM 基本周期。由此，写入比较数据寄存器PWMDATAH的值决定了PWM 波形的基本占空比

扩展计数器中的值则会不断与脉冲延伸控制寄存器的低 2位比较。通过扩展逻辑决定延伸特定周期 PWM 的输出占空比。延伸值是固定的一个 PWM 时钟周期（*表 13-2*）。

例如，如果扩展寄存器 PWMDATAL 的值为‘01B’，则第2个周期会比其他3个基本周期多一个PWM时钟周期。如果基本占空比是 50%，则第2个周期的占空比将延伸至约 51%。如果写‘10B’到扩展寄存器，所有奇数周期都会增加一个PWM时钟周期。如果写‘FCH’到扩展寄存器 PWMDATAL，那么所有的奇数个周期都会增加一个 PWM 时钟周期。 如果写‘11B’到PWMDATAL，那么除了第4个周期以外的所有周期都会增加一个 PWM 时钟周期。以此实现在 10位精度下最小可调步长为1个PWM时钟周期，即获得了 10位的输出精度。但是寄存器 PWMDATAH、PWMDATAL 的值仍然可以在基本周期（8位）结束时获得更新。 所以这种数据比较 + 脉冲延伸控制的 PWM 实现模式在提供高精度输出的同时， 保持了快速及时的PWM数据更新频率， 适用于需要高精度控制的场合。

表 13-2 PWM 脉冲延伸控制寄存器 (PWMDATAL .1-.0)

PWMDATAL Bit (Bit1–Bit0)	‘延伸’周期
00	-
01	第2个基本周期
10	第1, 3个基本周期
11	第1, 2, 3个基本周期

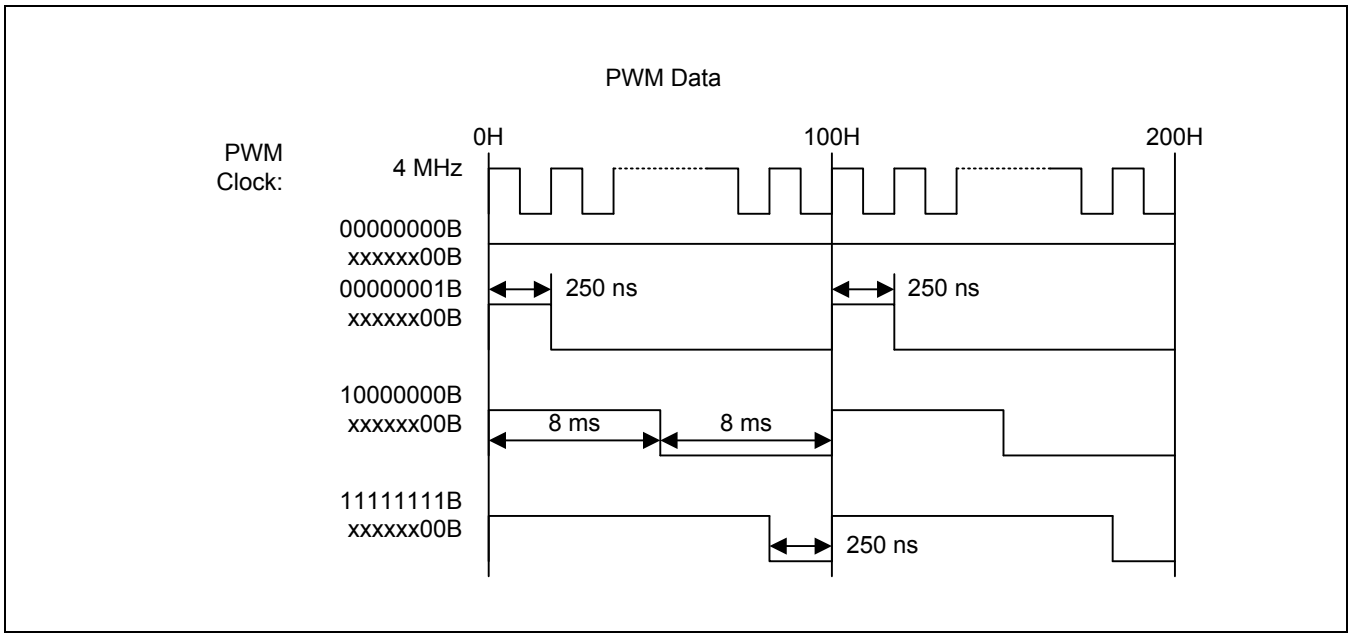


图 13-1 PWM 基本波形（8位）

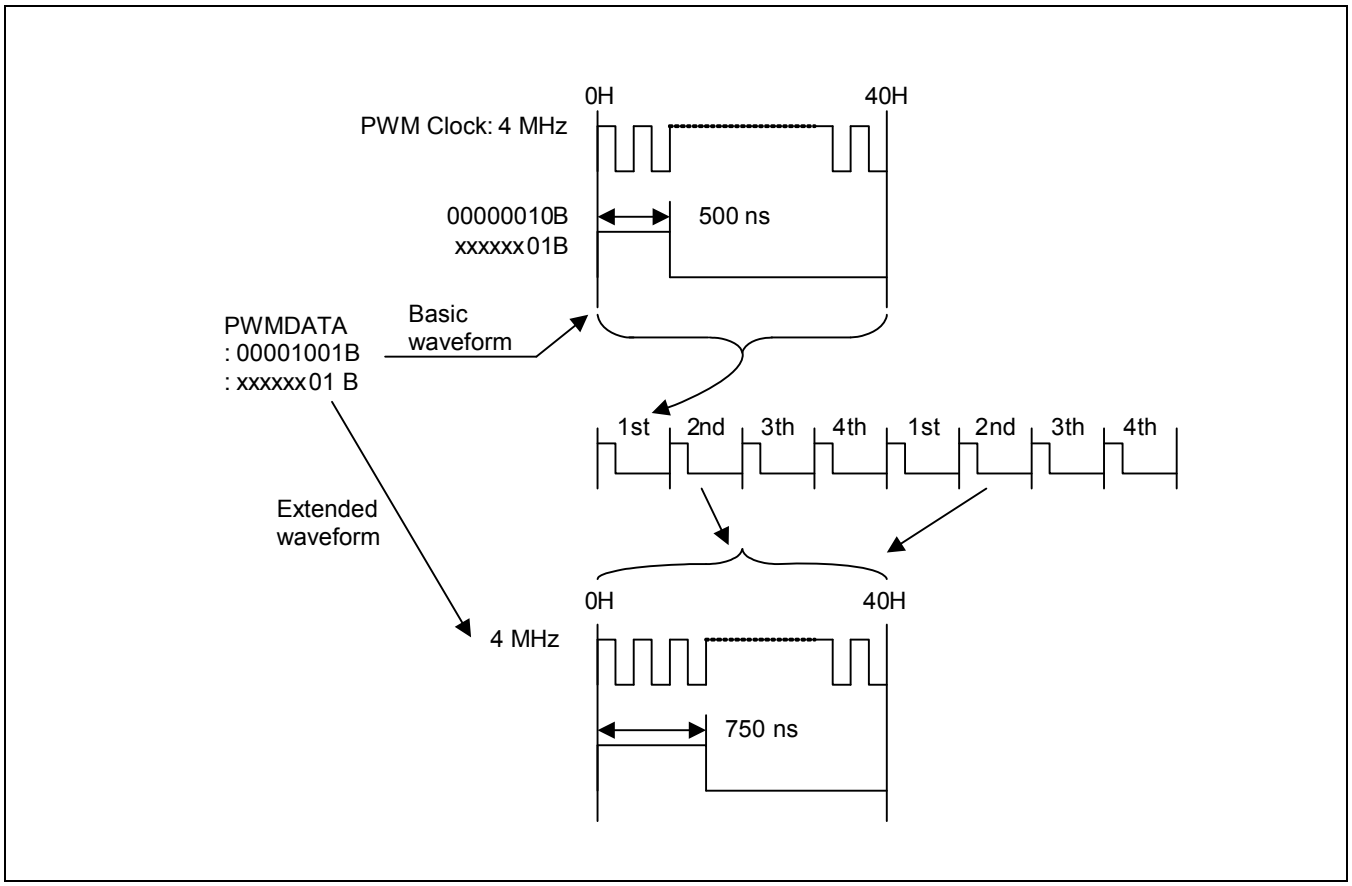


图 13-2 PWM‘延伸’波形（10位）

13.2.2 PWM控制寄存器(PWMCON)

PWM 控制寄存器的地址为 F5H。通过设置 PWMCON 寄存器可以实现以下功能：

- PWM 计数器时钟选择
- PWM 数据更新周期选择
- PWM 计数器清零
- PWM 计数器停止/开始操作
- PWM 计数器溢出（低2位扩展计数器溢出）中断控制

复位时 PWMCON 所有位被清零，禁止 PWM 输出。

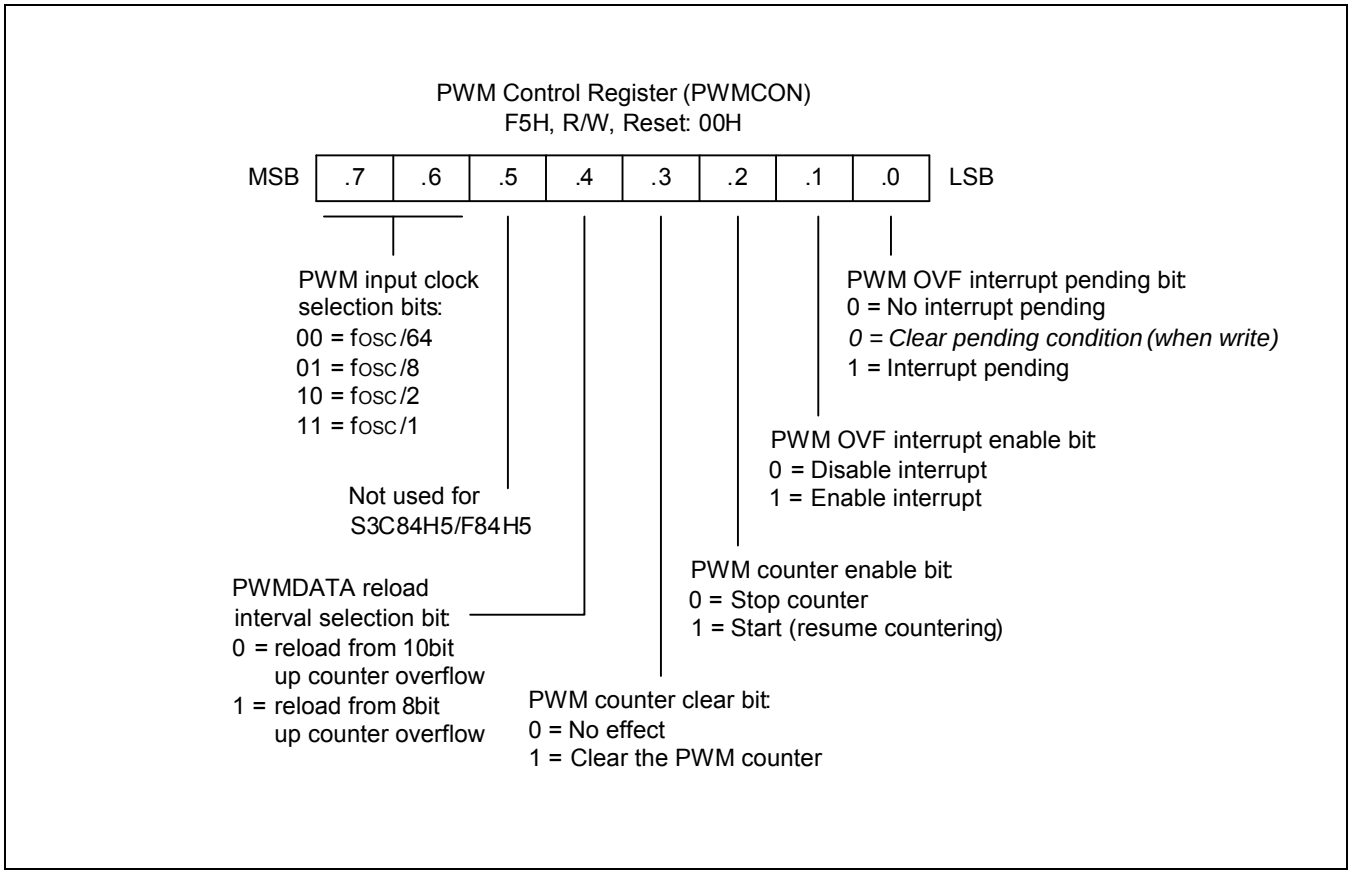


图 13-3 PWM 控制寄存器 (PWMCON)

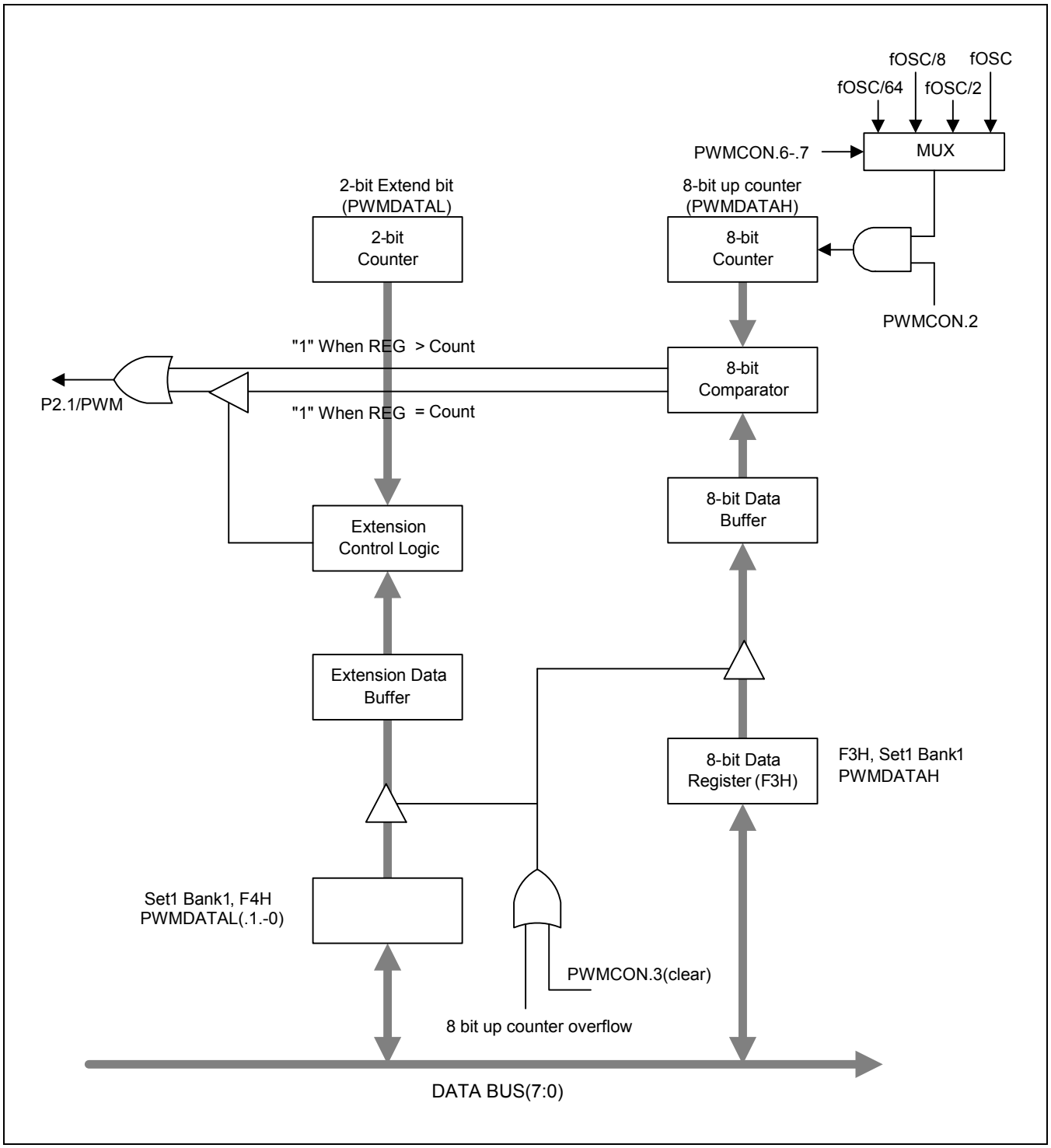


图 13-4 PWM 功能框图

编程实例 13-1 PWM 编程详例

```
;--------------<< 中断向量地址>>

ORG         0000H
            VECTOR      0DAH,INT_PWM

;--------------<< 初始化系统和外围>>

            ORG         0100H
RESET:      DI                              ; 禁止中断
            SB1                             ; 只有在用目标板调试时才需要 (注释)
            LD          0F7H,#5FH           ; 只有在用目标板调试时才需要 (注释)
            SB0                             ; 只有在用目标板调试时才需要 (注释)

            LD          BTCON,#10100011B    ; 禁止看门狗功能

            •
            •
            LD          P2CONL,#00001100B   ; 配置 P2.1 PWM 输出
            LD          PWMCON,#00000110B   ; OSC/64, 使能计数器/中断

            LD          PWMDATAH,,#80H
            LD          PWMDATAL,#0

            EI                              ; 使能全局中断

;--------------<< 主循环>>

MAIN:
            •
            •
            JR          t,MAIN

;--------------<< 中断服务程序>>

INT_PWM:                                    ; PWM中断服务程序
            •
            AND         PWMCON,#11111110B   ; 标志位清零
            IRET
            •
            END
```

注释: 在用目标板调试的时候，必须在初始化代码中加入这 3 条额外的指令，否则端口将不能正常工作。而在调试完成之后，烧写芯片前则必须删除这 3 条指令重新编译再执行烧写。

14 串行输入输出接口

14.1 概述

串行输入输出接口 (SIO) 可以与各种外设进行串行数据通讯。SIO 模块主要包括：

- 8 位控制寄存器 (SIOCON)
- 时钟选择逻辑
- 8 位数据缓冲器 (SIODATA)
- 8 位预分频器
- 3 位串行时钟计数器
- 串行数据输入/输出引脚 (SI, SO)
- 外部时钟输入引脚 (SCK)

SIO 模块可以接收和发送8位数据，其接收、发送速率由寄存器设定。 为了实现灵活的数据传输速率， 可以选择内部或者外部时钟源。

14.1.1 编程流程

SIO 模块的编程遵循以下基本的步骤：

1. 通过设置寄存器 P2CONL/H 寄存器，设置 port 2 中 SIO 对应的 I/O 口为 SO, SCK, SI
2. 设置寄存器 SIOCON，正确的配置 SIO 模块。SIOCON.2 必须设置为 ’1’ 以使能数据的移位。
3. 需要中断时，将中断使能位 (SIOCON.1) 置为 ‘1’。
4. 发送数据时，将要发送的数据写到数据缓冲器 SIODATA，并将 SIOCON.3 设为 ’1’，则开始发送数据。
5. 当数据接收/发送完成后，SIO 中断标志位 (SIOCON.0) 置为’1’，同时产生 SIO 中断请求。

14.1.2 SIO 控制寄存器(SIOCON)

SIO 控制寄存器，SIOCON，地址为F2H，Set 1，Bank 1，设置 SIO 模块的以下功能：

- 时钟源选择 (内部或者外部)
- 中断使能控制
- 移位操作的边沿选择
- 清除 3 位计数器并开始移位操作
- 使能移位操作 (发送)
- 模式选择 (发送/接收或只接收)
- 数据方向选择 (高位在前或低位在前)

复位后，SIOCON 的初始值为 '00H'，默认 SIO 选择内部时钟，工作在仅接收模式下，3位时钟计数器清零。 禁止数据移位操作 (发送) 和中断，选择数据方向为高位在前。

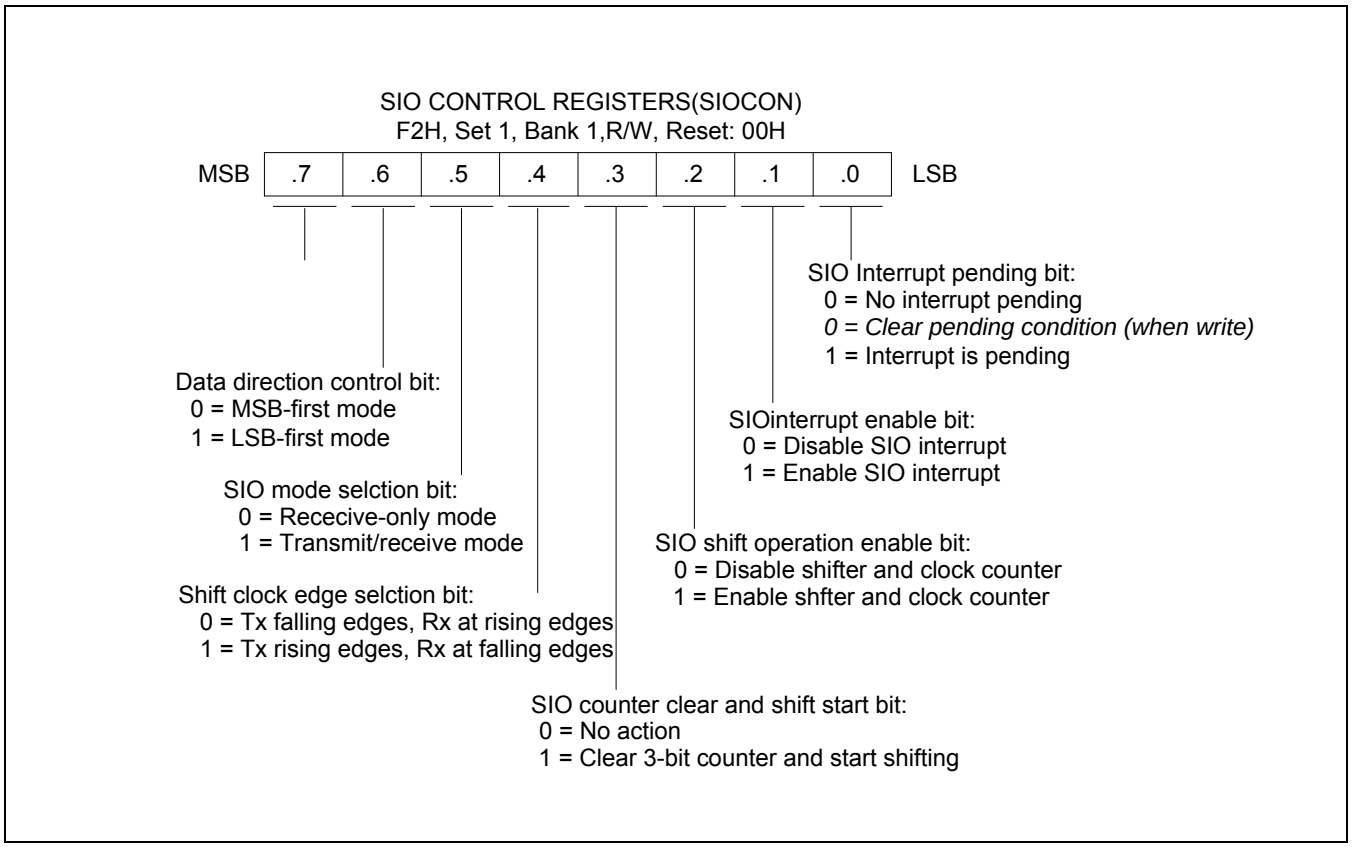


图 14-1 SIO 控制寄存器 (SIOCON)

14.1.3 SIO 预分频寄存器(SIOPS)

SIO 预分频寄存器，SIOPS，地址为 F0H，Set 1，Bank 1。SIOPS 寄存器的值决定了 SIO 数据传输的波特率：

波特率 = 输入时钟频率(Xin)/4 / (预分频值+ 1)， 或外部输入时钟频率。

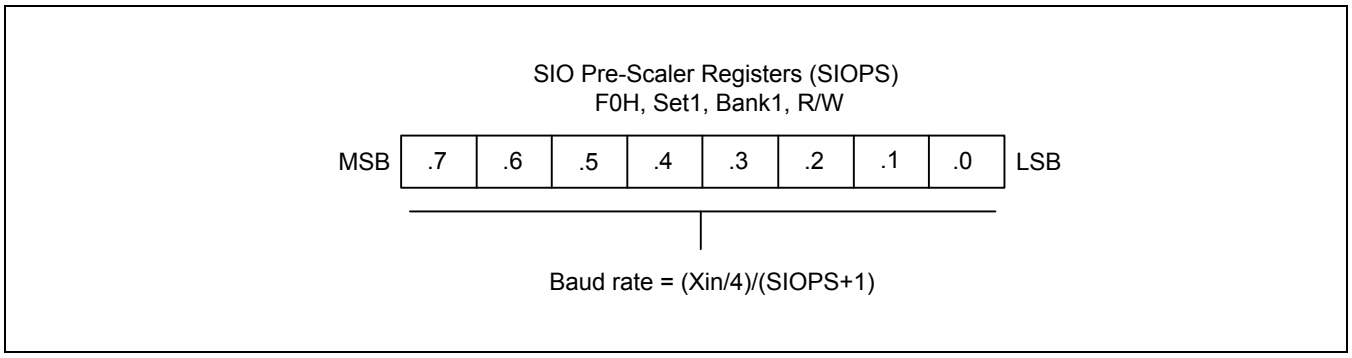


图 14-2 SIO 预分频寄存器 (SIOPS)

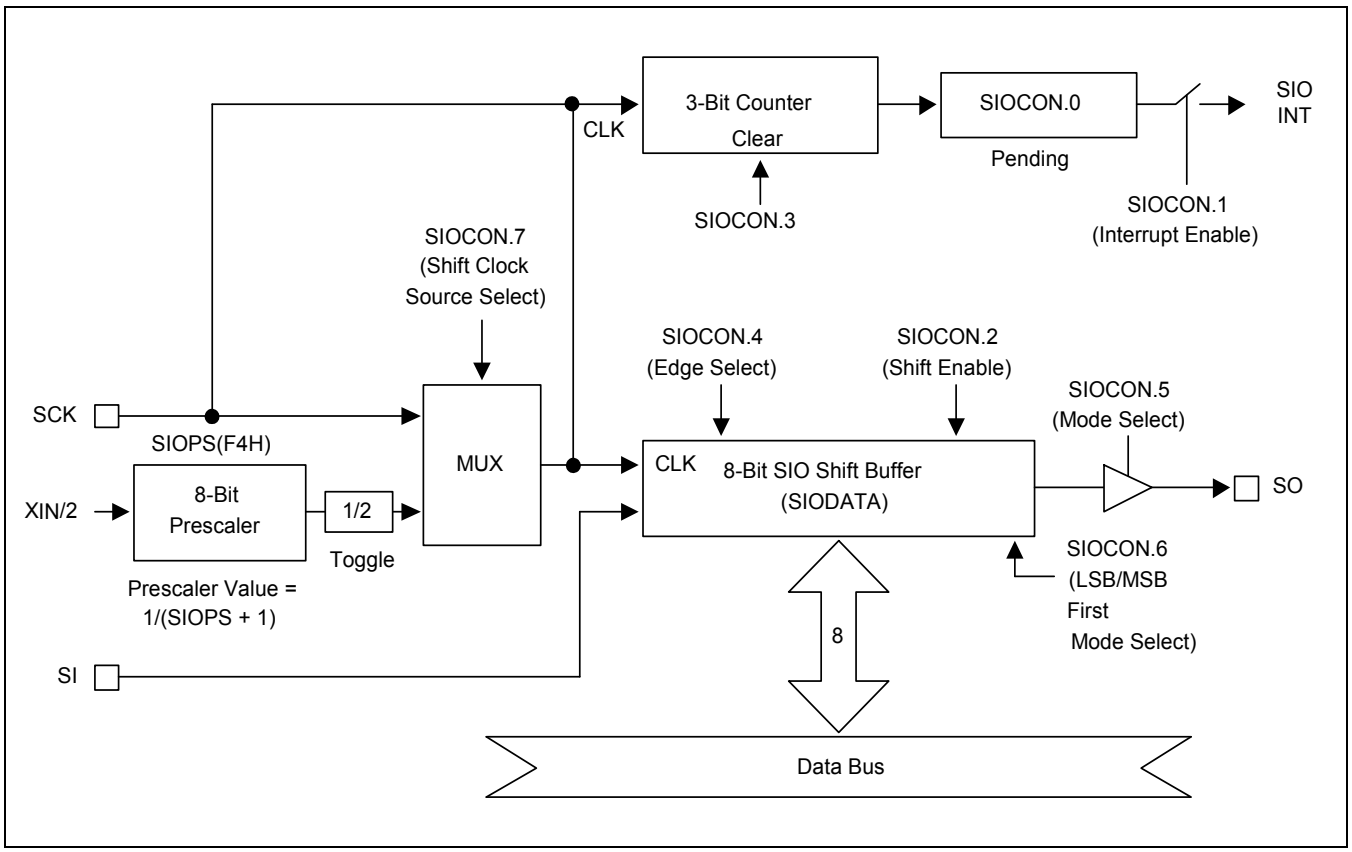


图 14-3 SIO 功能模块框图

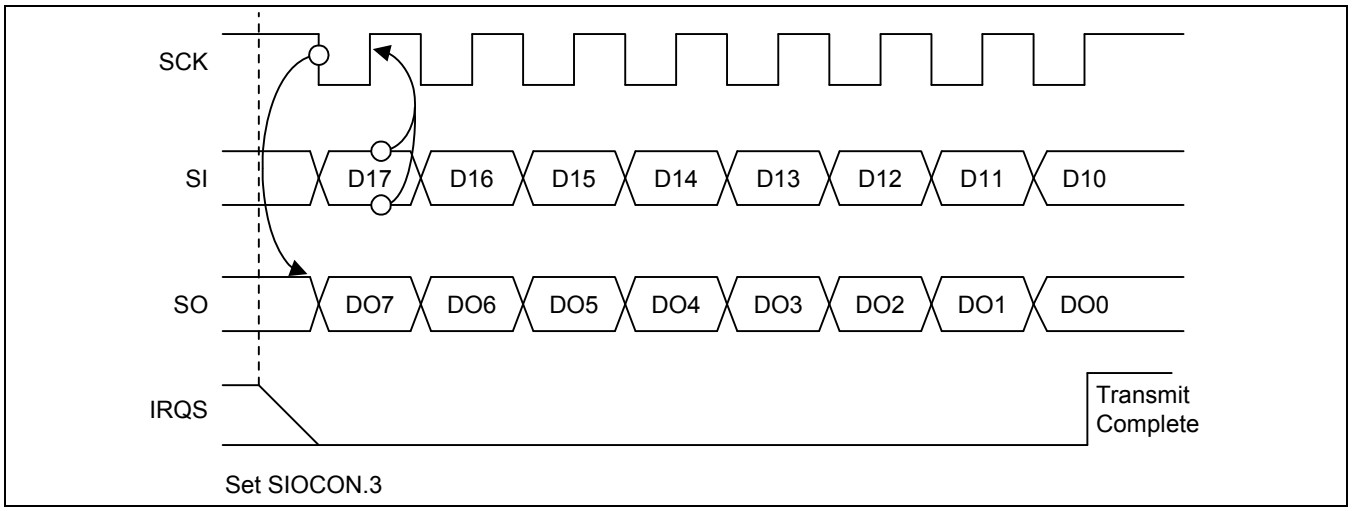


图 14-4 SIO 发送/接收模式 (下降沿发送，SIOCON.4 = 0)

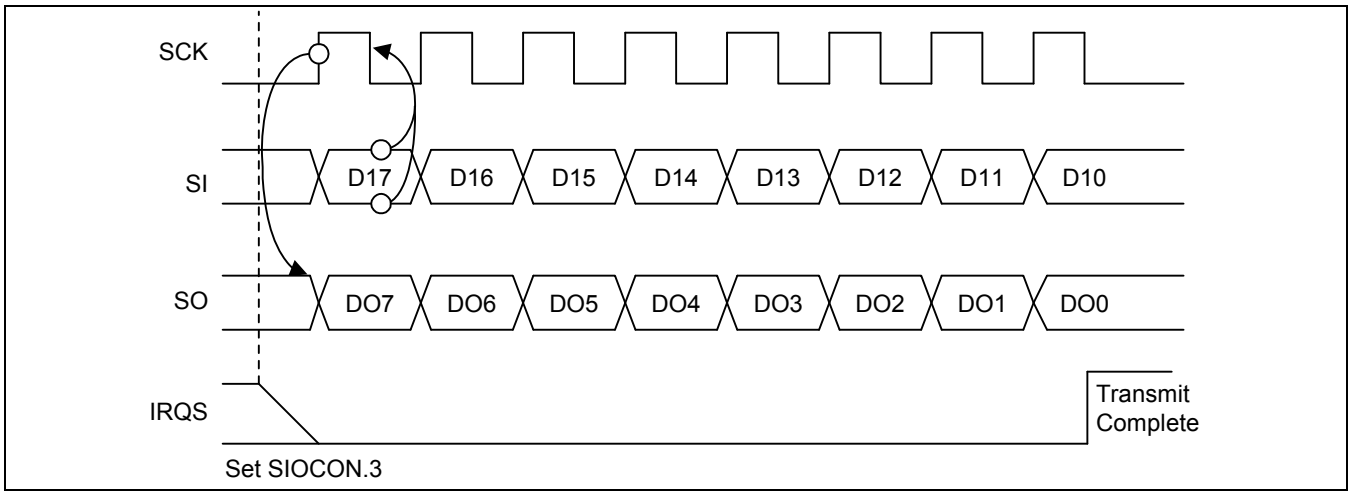


图 14-5 SIO 发送/接收模式 (上升沿发送，SIOCON.4 = 1)

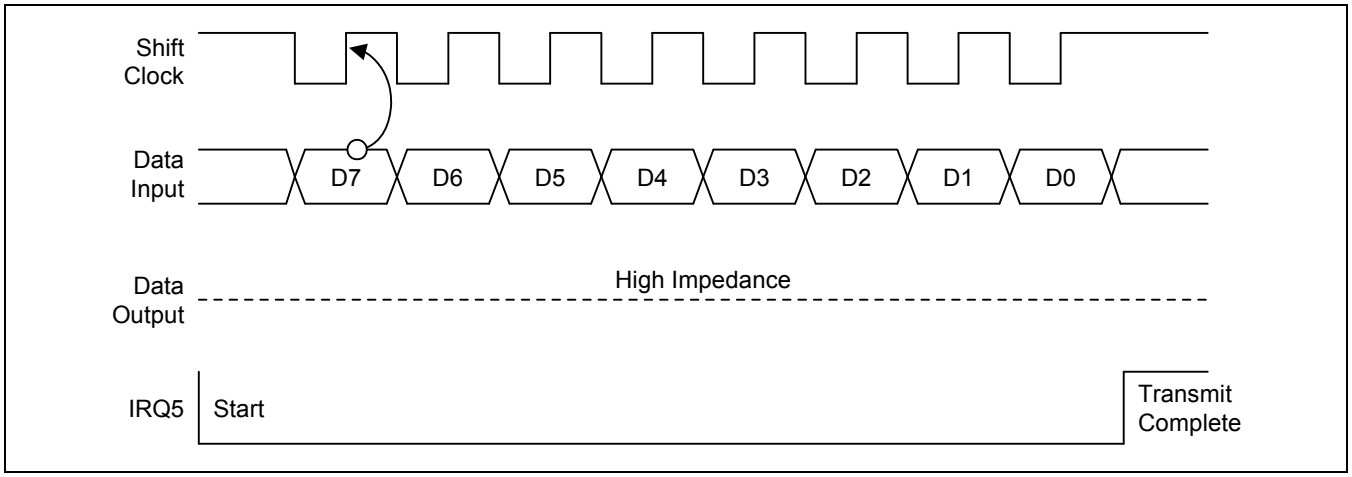


图 14-6 SIO 仅接收模式时序图

编程实例 14-1 SIO

```
                ORG         0000H
                VECTOR      00H, INT_SIO
                ORG         0100H

INITIAL:        DI
                SB1                             ; Debug 模式必需的额外指令 (注释)
                LD          0F7H,#5FH           ; Debug 模式必需的额外指令 (注释)
                SB0                             ; Debug 模式必需的额外指令 (注释)

                LD          BTCON, #10100010B   ; 禁止 Watch-dog
                LD          CLKCON, #00011000B  ; CPU 时钟不分频
                LD          SPL, #00H           ;

                •
                •
                LD          P2CONH, #10101111B  ; SIO 设置
                LD          P2CONL, #00101010B
                •
                •
                LD          SIOCON, #00100110B  ; 使能 SIO 中断
                LD          SIOPS, #20          ; 设置波特率
                •
                •
                EI

MAIN:           •
                •
                •
                CALL        SUB_SIO             ; 数据传输子程序
                •
                •
                •
                JP          MAIN

SUB_SIO:        LD          SIODATA, TRANSBUF   ; 单字节传输
                OR          SIOCON, #00001000B  ; 移位操作开始 (8位数据发送)
                •
                •
                RET

INT_SIO:        AND         SIOCON, #11111110   ; 清中断标志位
                •
                •
                •
                IRET
```

注释: 在 Debug 模式下，必需包括这三个额外的指令以使 I/O 口正常工作。如果没有这三个指令，则 I/O 口不能正常工作。在完成程序的调试后，最后将程序烧到 Main chip 的前，应该将这三个指令删除。

15 UART

15.1 概述

UART 模块是运行模式可设置的全双工串行通讯口。其运行模式包括一个并行模式和两个 UART (通用异步串行接收/发送) 模式：

- 寄存器移位型 I/O口，波特率可设置，为 fxx/(16 × (16位 BRDATA+1))
- 8位 UART 模式， 波特率可设置，为 fxx/(16 × (16位 BRDATA+1))
- 9位 UART 模式，波特率可设置，为 fxx/(16 × (16位 BRDATA+1))

UART 的接收和发送都是通过 UART 数据寄存器 UDATA (地址为 F5H) 来实现的。写 UART 数据寄存器时，数据装载到 UART 的发送缓冲器中；读 UART 数据寄存器时，数据从 UART 的接收缓冲器中读出。 发送缓冲器和接收缓冲器在物理上是分开的。

接收缓冲器为移位寄存器，在上一个接收的数据还未读出前，下一个数据已经开始接收。因此，如果在下一个数据完成接收之前，上一个接收的数据还没有读出的话，则上一个数据会丢失(过载错误)

在所有运行模式下，当任何指令将 UDATA 作为目标地址写入数据时，则立即开始发送。在 mode 0 模式下， 只有当接收中断标志位 (UARTPND.1) 为 "0" 和接收使能位 (UARTCON.4) 为 "1" 时，才开始接收串行数据。
在 mode 1 和 mode 2 模式下，当接收使能位 (UARTCON.4)为 "1" 时，任何时刻只要收到数据起始位 (“0”) 则立即开始接收数据。

15.1.1 编程流程

UART 模块编程的基本步骤为：

1. 通过设置 P2CONH 为相应的值，将 P2.6 和 P2.7 设为UART功能引脚 (RXD (P.6), TXD (P2.7))。
2. 设置 UARTCON 控制寄存器来选择UART的运行模式。
3. 在 mode 2 模式下，需要奇偶校验位时，设置奇偶校验使能位 (UARTPND.5) 为 "1"。
4. 需要中断时，将UART中断使能位 (UARTCON.1 或 UARTCON.0) 设为 "1"。
5. 传送数据时，将需要传送的数据写入到 UDATA，则开始传送。
6. 当发送/接收完一个数据后，UART 标志位 (UARTPND.1 或 UARTPND.0) 被置为 "1"，同时产生一个相应的发送/接收中断。

15.1.2 UART 控制寄存器(UARTCON)

UART 的控制寄存器是 UARTCON，地址为 F6H。主要实现以下功能：

- 运行模式和波特率选择
- 多节点通讯和中断控制
- 串行接收使能/禁止控制
- Mode 2 模式下发送和接收时第 9 位数据位的位置
- Mode 2 模式下接收/发送时奇偶校验位的产生和校验。
- UART 发送和接收中断控制

复位后，UARTCON 的初始值为 ”00H”，因此，如果需要使用 UAR T模块，应设定 UARTCON 为适当的值。

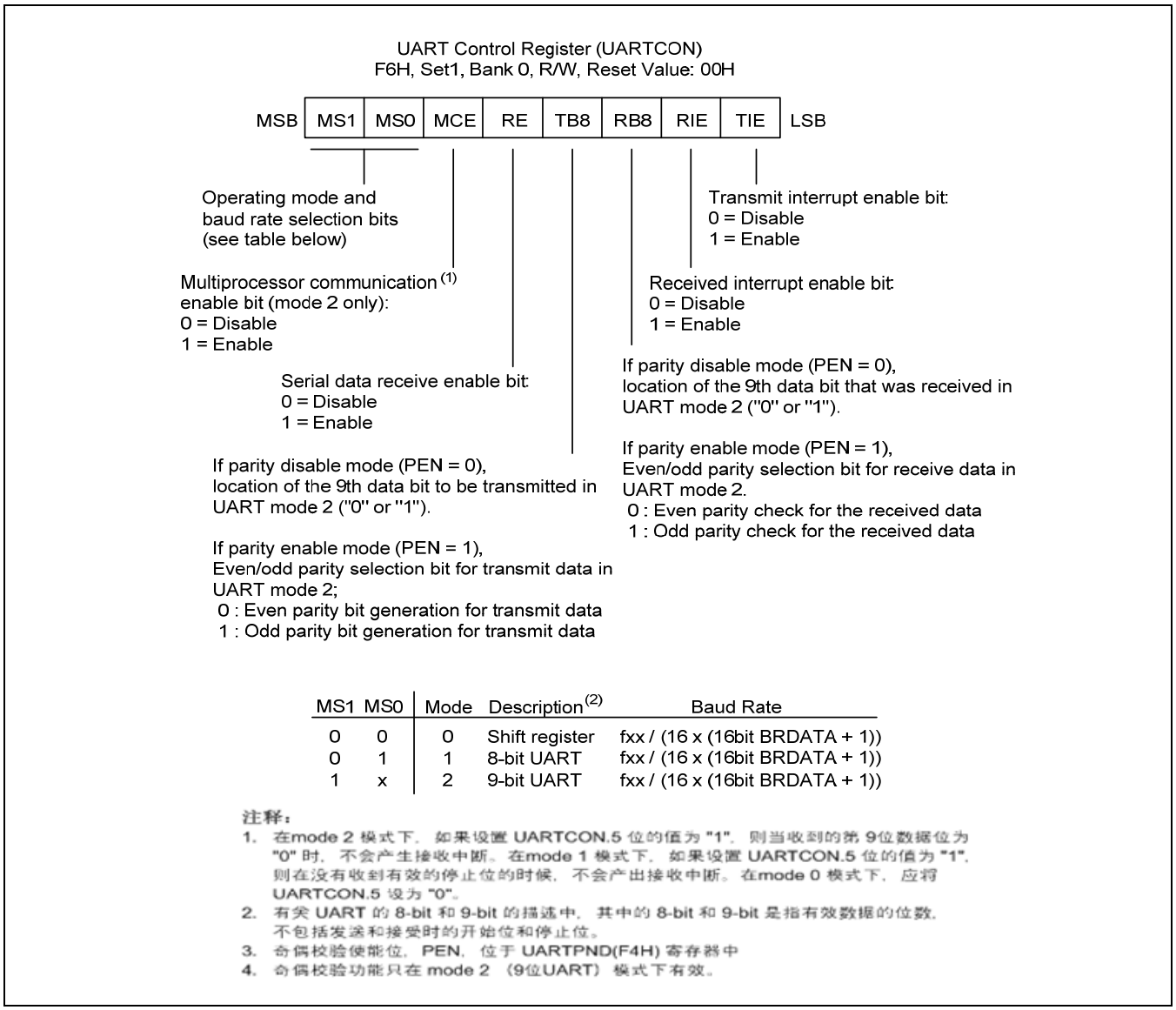


图 15-1 UART 控制寄存器 (UARTCON)

15.1.3 UART 中断标志寄存器(UARTPND)

UART 中断标志寄存器，UARTPND(F4H,Set 1, Bank 0) 包括UART数据发送中断标志位 (UARTPND.0) 和接收中断标志位 (UARTPND.1)。

在 mode 0 模式下，当收到第 8 位数据位后，接收中断标志位 UARTPND.1 被置为 "1"。在 mode 1 或 mode 2 模式下，UARTPND.1 在停止位接收的中间点置 "1"。当 CPU 响应了接收中断后，UARTPND.1 必须在中断服务程序中软件清零。

在UART 的 mode 0 模式下，当发送完第8位数据后，发送中断标志位 UARTPND.0 被置为 "1"。在 mode 1 或 mode 2 模式下，UARTPND.0 在停止位开始发送时置 "1"。当 CPU 响应了发送中断后，UARTPND.0 必须在中断服务程序中软件清零。

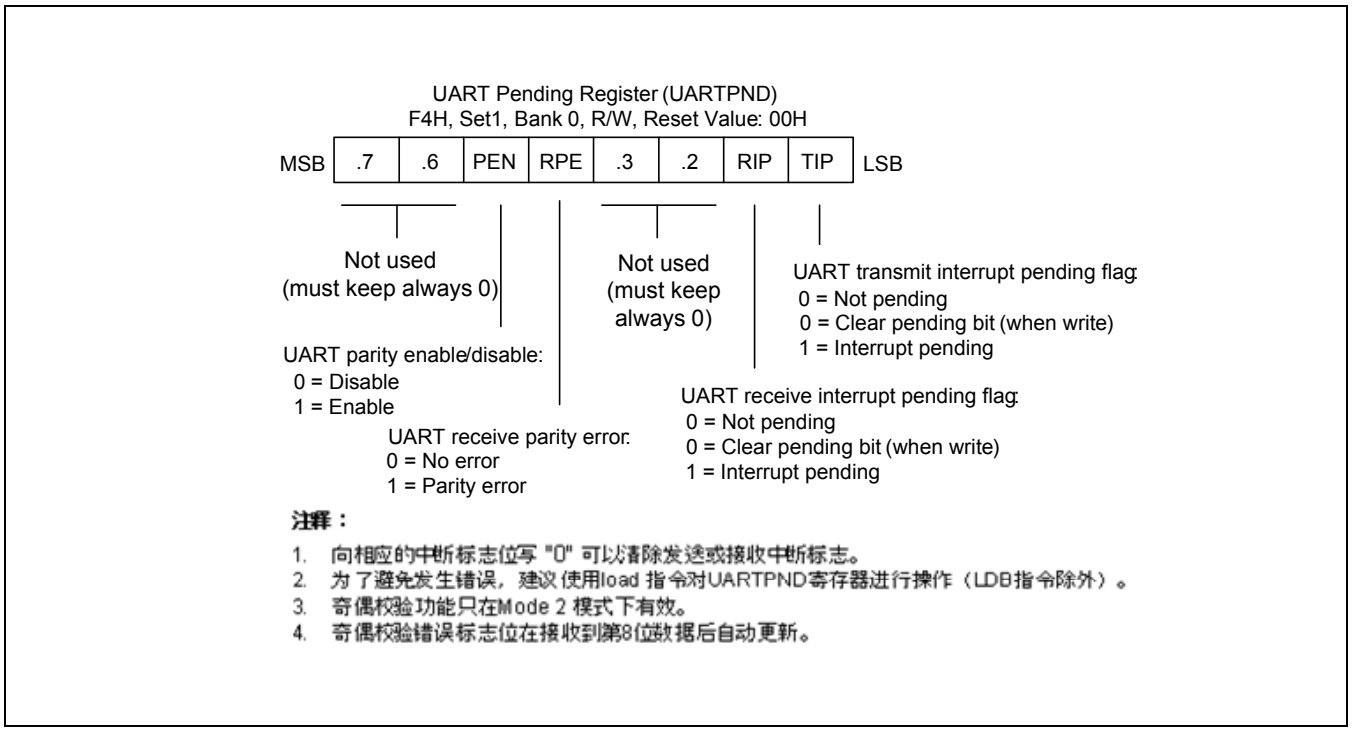


图 15-2 UART 中断标志寄存器 (UARTPND)

在 Mode 2 (9 位 UART 数据)模式下，设置 UARTPND 寄存器中的 PEN 位为 '1' 以使能奇偶校验，则传送的第9位数据即为自动产生的奇偶校验位，而收到的第 9 位数据位则被认为是奇偶校验位以对接收的数据进行校验。

当奇偶校验使能时 (PEN = 1)，UARTCON.3 (TB8) 和 UARTCON.2 (RB8) 为相应的发送和接收的奇偶校验选择位。UARTCON.3 (TB8) 设置发送时是进行偶校验 (TB8 = 0) 还是进行奇校验(TB8 = 1)。UARTCON.2 (RB8) 设置接收时是进行偶校验 (RB8 = 0) 还是进行奇校验 (RB8 = 1)。奇偶校验功能只在 Mode 2 模式下有效。

在 Mode 2 模式下，不需要使用奇偶校验功能时，UARTCON.2 (RB8) 和 UARTCON.3 (TB8) 为普通的控制位，同时 PEN 必须设置为禁止 ("0")，TB8 为将要发送的第9位数据。

接收奇偶校验错误标志位在接收完第 8 位数据位后，自动根据奇偶校验结果设为 '0' 或者 '1'。

15.1.4 UART 数据寄存器(UDATA)

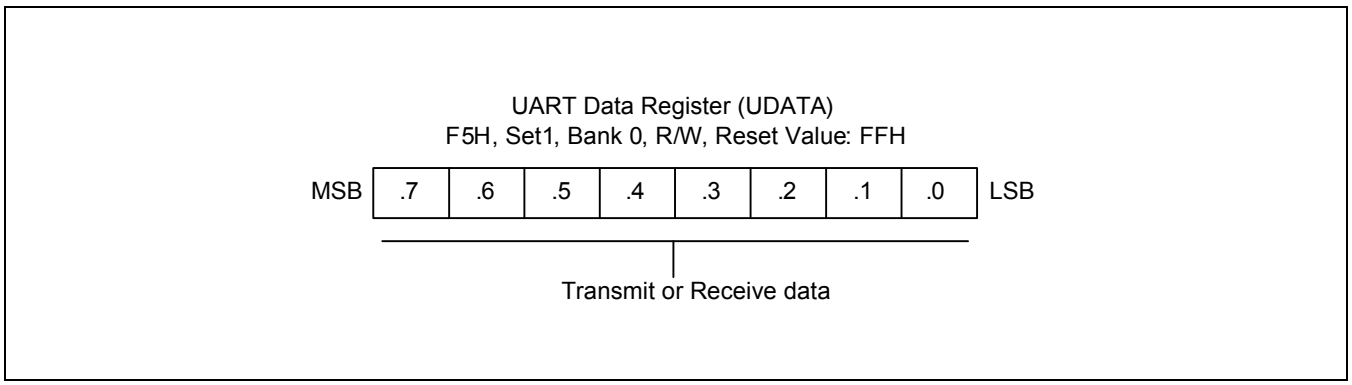


图 15-3 UART 数据寄存器 (UDATA)

15.1.5 UART 波特率数据寄存器(BRDATAH, BRDATAL)

通过设置 UART 波特率数据寄存器(BRDATAH, BRDATAL)，可以设定 UART 的时钟速率(即波特率)。

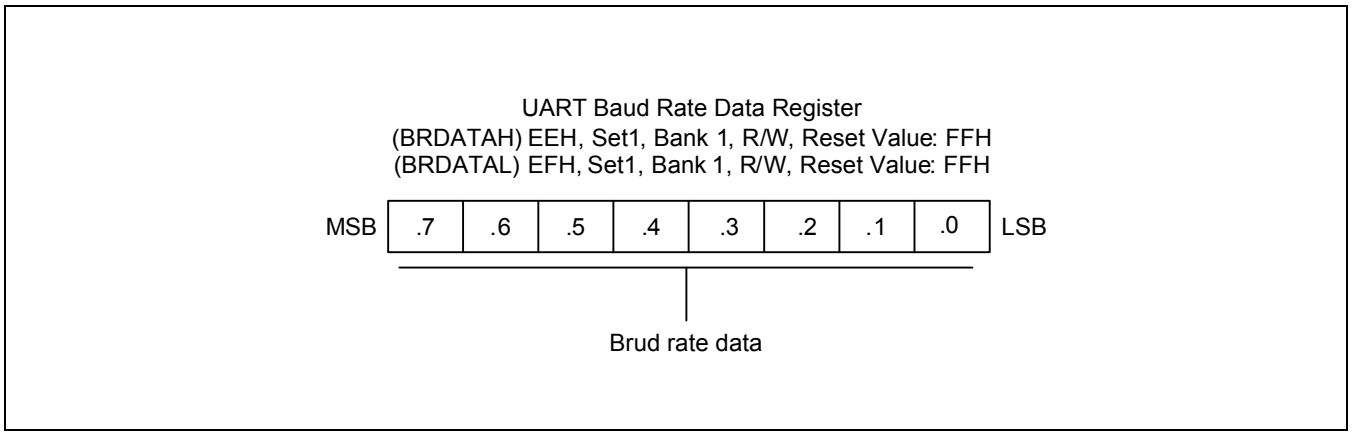


图 15-4 UART 波特率数据寄存器 (BRDATAH, BRDATAL)

15.1.6 波特率计算

波特率由波特率数据寄存器的值决定，对 16-bit 的BRDATA：

- Mode 0 的波特率 = fxx/(16 × (16位 BRDATA + 1))
- Mode 1 的波特率 = fxx/(16 × (16位 BRDATA + 1))
- Mode 2 的波特率 = fxx/(16 × (16位 BRDATA + 1))

表 15-1 16bit BRDATA可以产生的常用波特率表

波特率	晶振频率	BRDATAH		BRDATAL	
		十进制	十六进制	十进制	十六进制
230,400 Hz	11.0592 MHz	0	0H	02	02H
115,200 Hz	11.0592 MHz	0	0H	05	05H
57,600 Hz	11.0592 MHz	0	0H	11	0BH
38,400 Hz	11.0592 MHz	0	0H	17	11H
19,200 Hz	11.0592 MHz	0	0H	35	23H
9,600 Hz	11.0592 MHz	0	0H	71	47H
4,800 Hz	11.0592 MHz	0	0H	143	8FH
76,800 Hz	10 MHz	0	0H	7	7H
38,400 Hz	10 MHz	0	0H	15	FH
19,200 Hz	10 MHz	0	0H	31	1FH
9,600 Hz	10 MHz	0	0H	64	40H
4,800 Hz	10 MHz	0	0H	129	81H
2,400 Hz	10 MHz	1	1H	3	3H
600 Hz	10 MHz	4	4H	16	10H
38,461 Hz	8 MHz	0	0H	12	0CH
12,500 Hz	8 MHz	0	0H	39	27H
19,230 Hz	4 MHz	0	0H	12	0CH
9,615 Hz	4 MHz	0	0H	25	19H

15.2 模块框图

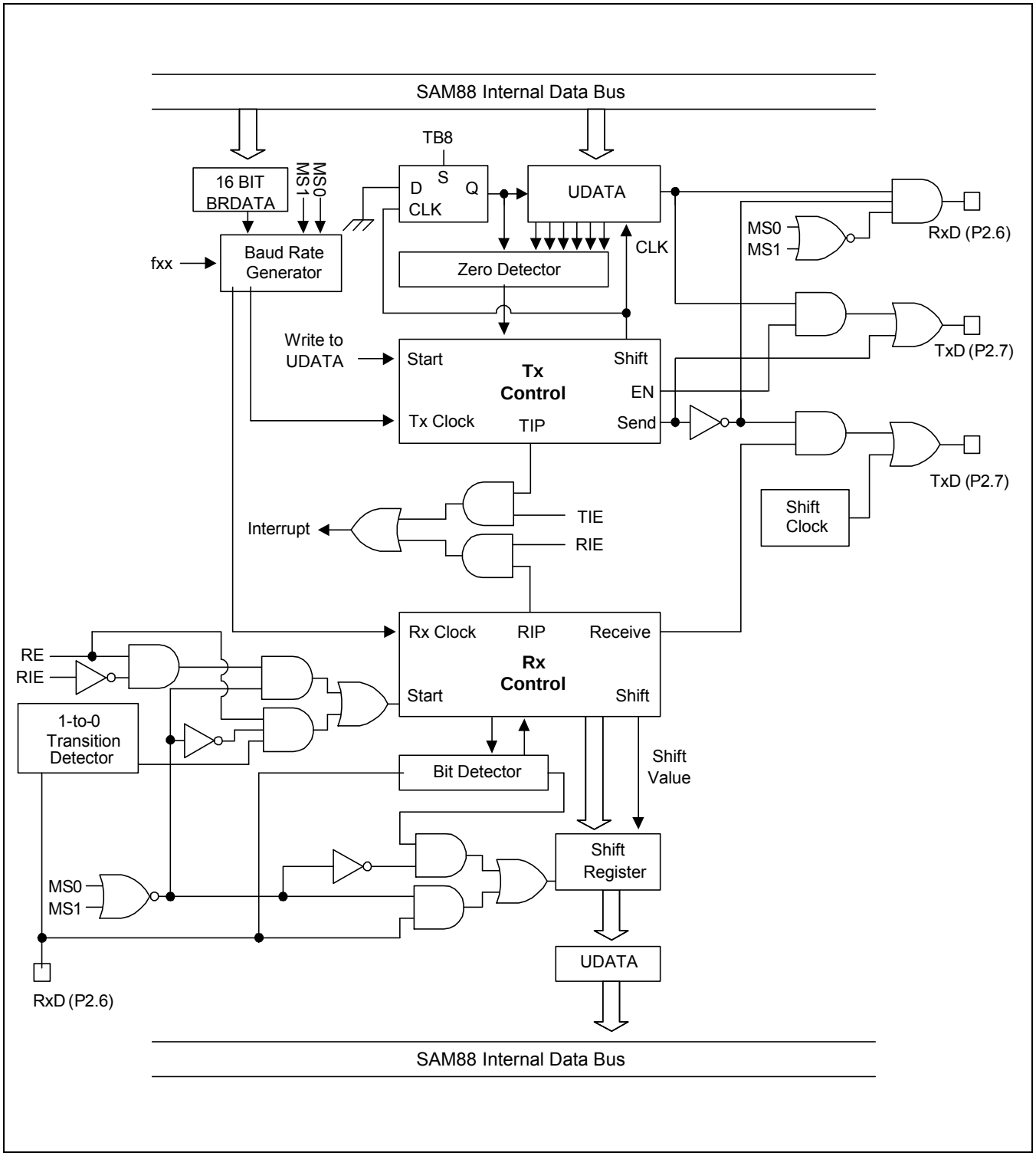


图 15-5 UART 功能模块框图

15.2.1 UART MODE 0 模式功能描述

在mode 0 模式下，UART通过 RxD(P2.6) 接收和发送数据，TxD(P2.7) 输出数据移位时钟信号。 数据接收和发送的长度都是8位，首先发送或接收的是数据的低位(LSB)。

15.2.1.1 Mode 0 模式数据发送流程

1. 设置 UARTCON.6-.7 为 "00B"，选择 mode 0。
2. 将需要发送的数据写入 UDATA (F5H)，数据开始发送。

15.2.1.2 Mode 0 模式数据接收流程

1. 设置 UARTCON.6-.7 为 "00B"，选择 mode 0。
2. 写 "0" 到 UARTPND.1，清除接收中断标志位。
3. 设置 UART 接收使能位 (UARTCON.4) 为 "1"，使能 UART 接收。
4. 时钟移位信号开始输出到 TxD (P2.7)，同时开始从 RxD (P2.6) 读取数据。当 UARTCON.1 置为 "1" 时，产生 UART 接收中断 (向量地址: E4H)。

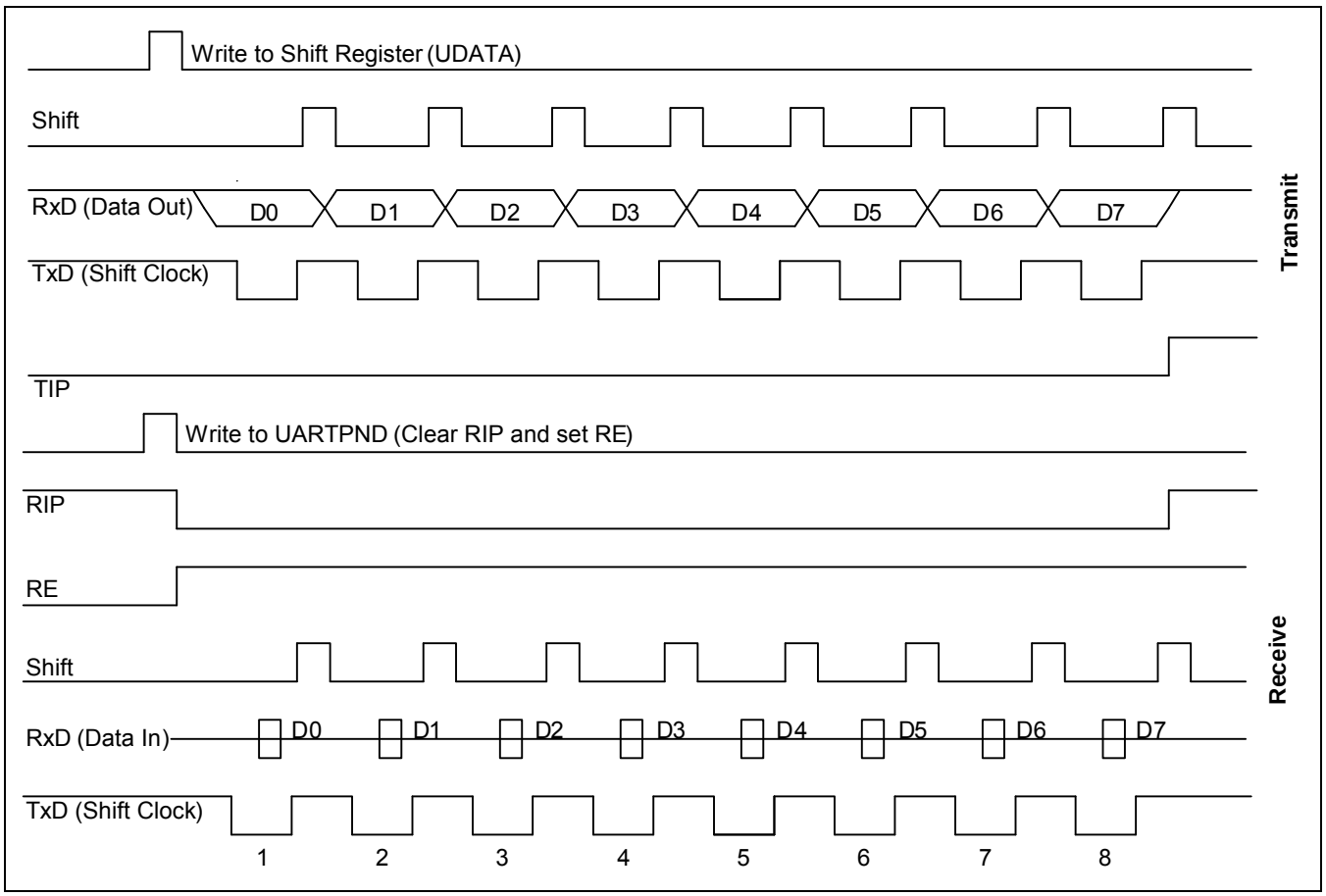


图 15-6 UART Mode 0 模式运行时序图

15.2.2 UART MODE 1 功能描述

在mode 1 模式下，发送 (通过TxD (P2.7)) 或者接收 (通过RxD (P2.6))的数据帧总长度是 10 位。 每一个数据帧由三部分组成：

- 起始位 ("0")
- 8 位有效数据 (低位在前)
- 停止位 ("1")

接收时，接收到的停止位写入到了 UARTCON 寄存器的 RB8 位。Mode 1 模式的波特率是可变的。

15.2.2.1 Mode 1 模式数据发送流程

1. 通过设置 16 位 BRDATA 寄存器设定波特率。
2. 设置 UARTCON.6-.7 为"'01B"，选择 mode 1(8位 UART).
3. 将需要发送的数据写入 UDATA (F5H)，数据开始发送。起始位和停止位由硬件自动产生。

15.2.2.2 Mode 1 模式数据接收流程

1. 通过设置 16 位 BRDATA 寄存器设定波特率。
2. 选择 mode 1 模式并设置 UARTCON 的 RE (接收使能) 位为 "1".
3. 当 RxD (P2.6) 管脚上检测到起始条件(低电平"0")时，UART 模块开始进行串行数据接收操作。

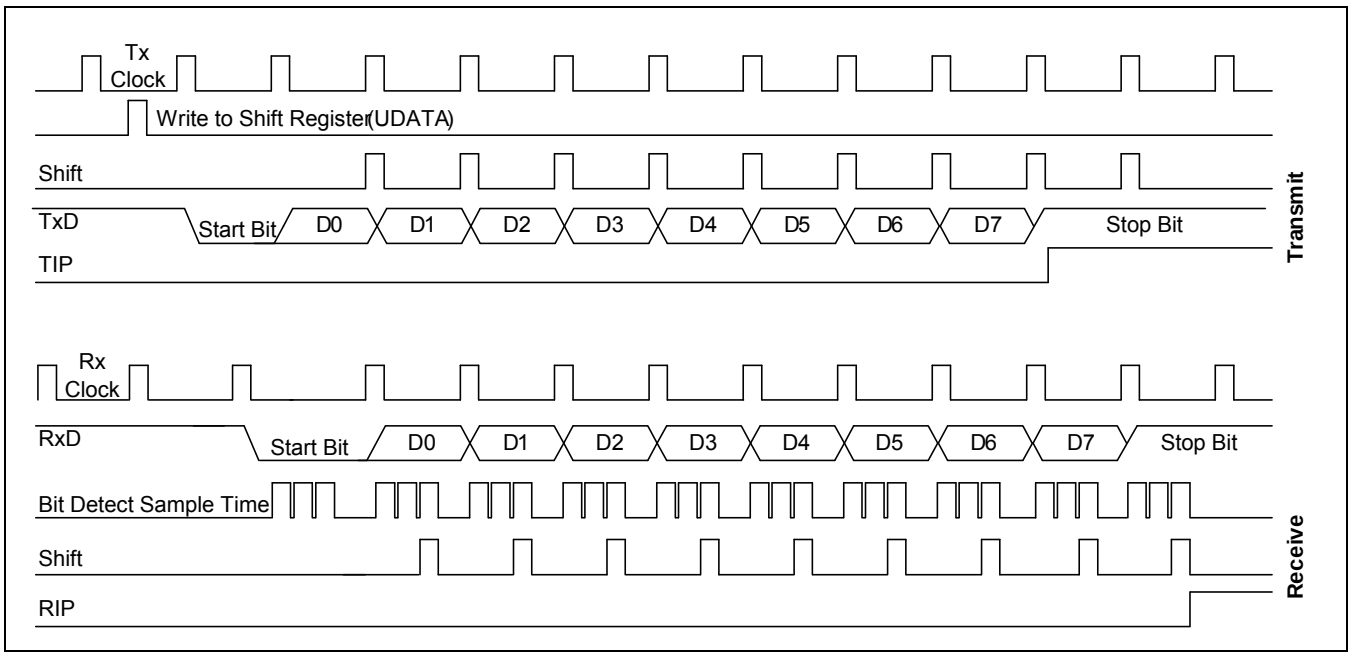


图 15-7 UART Mode 1 模式运行时序图

15.2.3 UART MODE 2 模式功能描述

在mode 2 模式下，发送 (通过TxD) 或者接收 (通过RxD)的数据帧总长度是11位。在mode 2 模式下， 每一个数据帧由三部分组成:

- 起始位("0")
- 8位有效数据 (低位在前)
- 可编程设置的第9位数据位或奇偶校验位
- 停止位 ("1")

15.2.3.1 奇偶校验禁止(PEN = 0)

发送时，将需要发送的第 9位数据的值 "0" 或 "1" 的写入到 TB8 位 (UARTCON0.3)。接收时，接收到的第 9 位数据存放在 RB8 位 (UARTCON0.2)，同时停止位被忽略。 Mode 2 模式的波特率为 fosc/(16 x (16bit BRDATA + 1))。

15.2.3.2 奇偶校验使能(PEN = 0)

发送时，第 9 位发送的数据根据奇偶校验的方式自动产生。接收时，接收到地第 9 位数据被当做由 RB8 确定的奇偶校验方式产生的奇偶校验位。当忽略停止位时， Mode 2 模式的波特率为 fosc/(16 x (16bit BRDATA + 1))。

15.2.3.3 Mode 2 发送数据流程

1. 通过设置 16 位 BRDATA 寄存器设定波特率。

2. 设置 UARTCON.6-.7 为 "10B"，选择 mode 2 (9位 UART)，同时将需要发送的第 9 位数据写入 TB8 (奇偶校验禁止)；当使能奇偶校验时，设置 TB8 选择奇偶校验方式，并设置PEN为 ’1’ 以使能奇偶校验。

3. 将需要发送的数据写入 UDATA (F5H)，数据开始发送.

15.2.3.4 Mode 2 接收流程

1. 通过设置 16 位 BRDATA 寄存器设定波特率。

2. 选择 mode 2 模式并设置 UARTCON 的 RE (接收使能) 位为 "1"。

3. 不使用奇偶校验时，设置 PEN 为 ‘0‘。 使用奇偶校验时， 设置 TB8 选择奇偶校验的方式并设置 PEN 以使能奇偶校验。收到的数据长度为8位有效数据。

4. 当 RxD (P2.6) 管脚上的电平变为低时，开始接收数据。

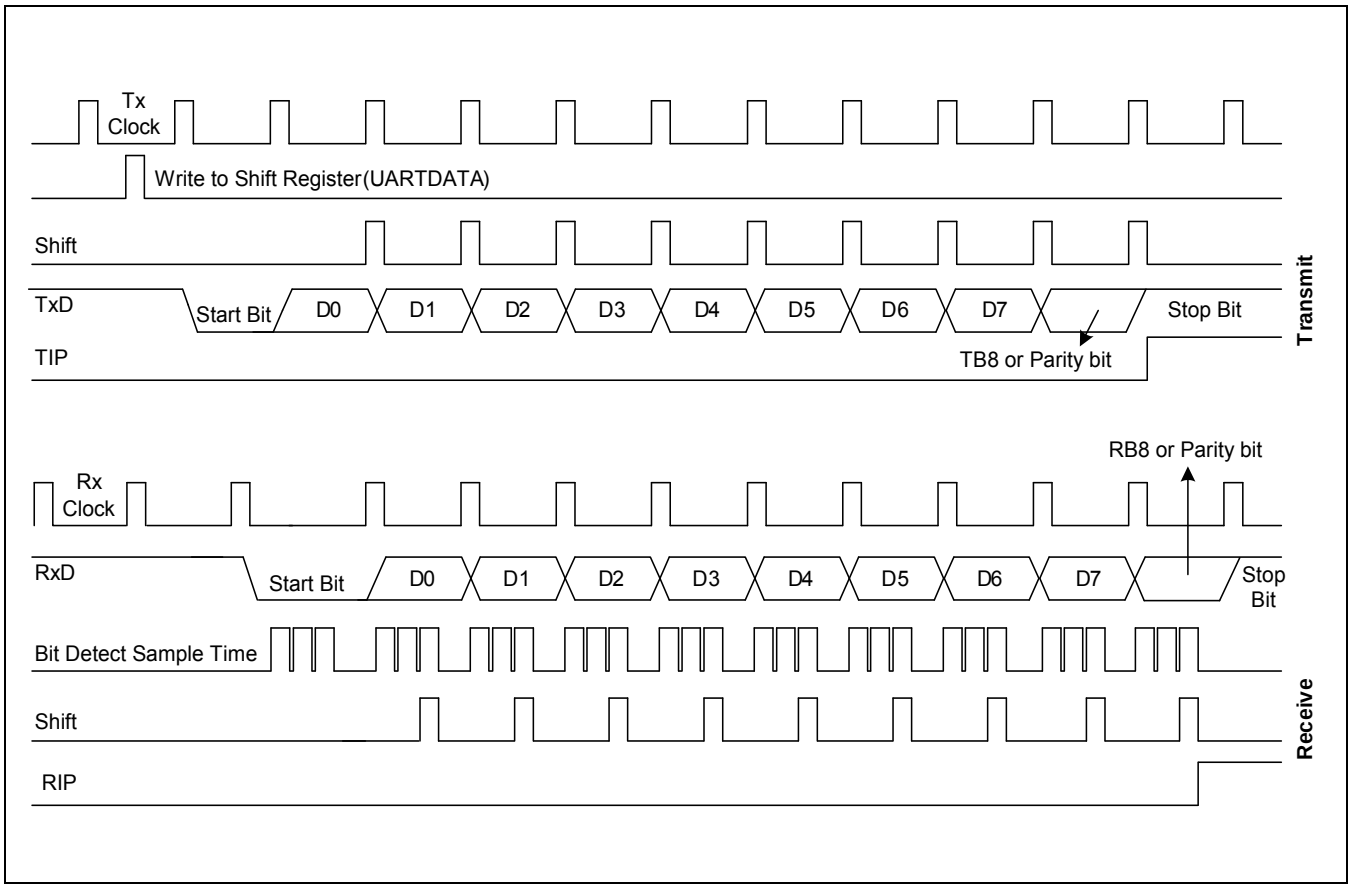


图 15-8 UART Mode 2 模式运行时序图

15.2.4 多节点串行通讯

S3F8 系列的多节点通讯功能的特点是：在一个由多个 S3C84H5/F84H5 组成的串行通信系统中，允许一个 S3C84H5/F84H5 作为主机向另一个特定的从机设备发送串行数据，而不会影响连接在同一个串行线上的其他从机设备。

这个功能只有在 UART 的 mode 2 模式下，禁止奇偶校验时才可以实现。在 mode 2模式下，收到的数据供 9 位，第 9 位写入了 RB8 (UARTCON.2)。接收操作在收到停止位后完成。因此可以编程实现这样的功能： 收到停止位后，只有当 RB8 = "1" 时才产生中断。

为了实现这个功能，需要设置 UARTCON 寄存器中的 MCE 位，当设置 MCE 位为 "1" 时，接收的串行数据帧中的第 9 位数据 = "0" 时不会产生中断。在这种情况下，就可以使用第 9 位来简单的区分地址和数据。

15.2.4.1 主机/从机交互协议的一个例子

当主机希望给串接在同一条串行总线上的很多从机中的一个从机发送数据时，它首先发送地址信息来选中目标从机。注意在这种情况下，一个地址帧和一个数据帧是不同的，地址帧的第 9 位数据为 "1"，而数据帧的第 9 位数据 为 "0"。

地址帧会让所有的从机都产生中断，并被每个从机都接收到，则每个从机就可以通过检查接收到的数据来判断是否被选中。 被选中的目标从机会将 MCE 位清为 "0" 并准备开始接收数据。

没有被选中的从机的 MCE 位继续保持为"1"，保持正常运行，并忽略串行总线上的数据帧。

MCE 位在 mode 0 模式下没有用处。在 mode 1 模式下，它可以用来检测停止位的有效性。在 mode 1 模式下接收数据时，如果设置 MCE 为"1"，只有接收到的停止位有效(为"1")时，才可以产生接收中断。

15.2.4.2 多节点通讯的建立流程

多节点通讯的建立流程如下：

1. 设置所有的串行总线的 S3C84H5/F84H5 设备的 UART 模式为 mode 2。
2. 设置所有从机设备的 MCE 位为"1"。
3. 主机的传输协议为:
   - 第一个字节：地址
     标识目标从机设备地址 (第9位 = "1")
   - 下一个字节：数据
     (第9位 = "0")
4. 由于第 9 位为"1"，所有的从机都会产生中断来接收第一个字节。当目标从机收到第一个字节后，将收到的地址与自己的地址进行比较，确认被选中后，则将MCE位清为 "0" 来准备接受数据。其他的从机继续正常运行。

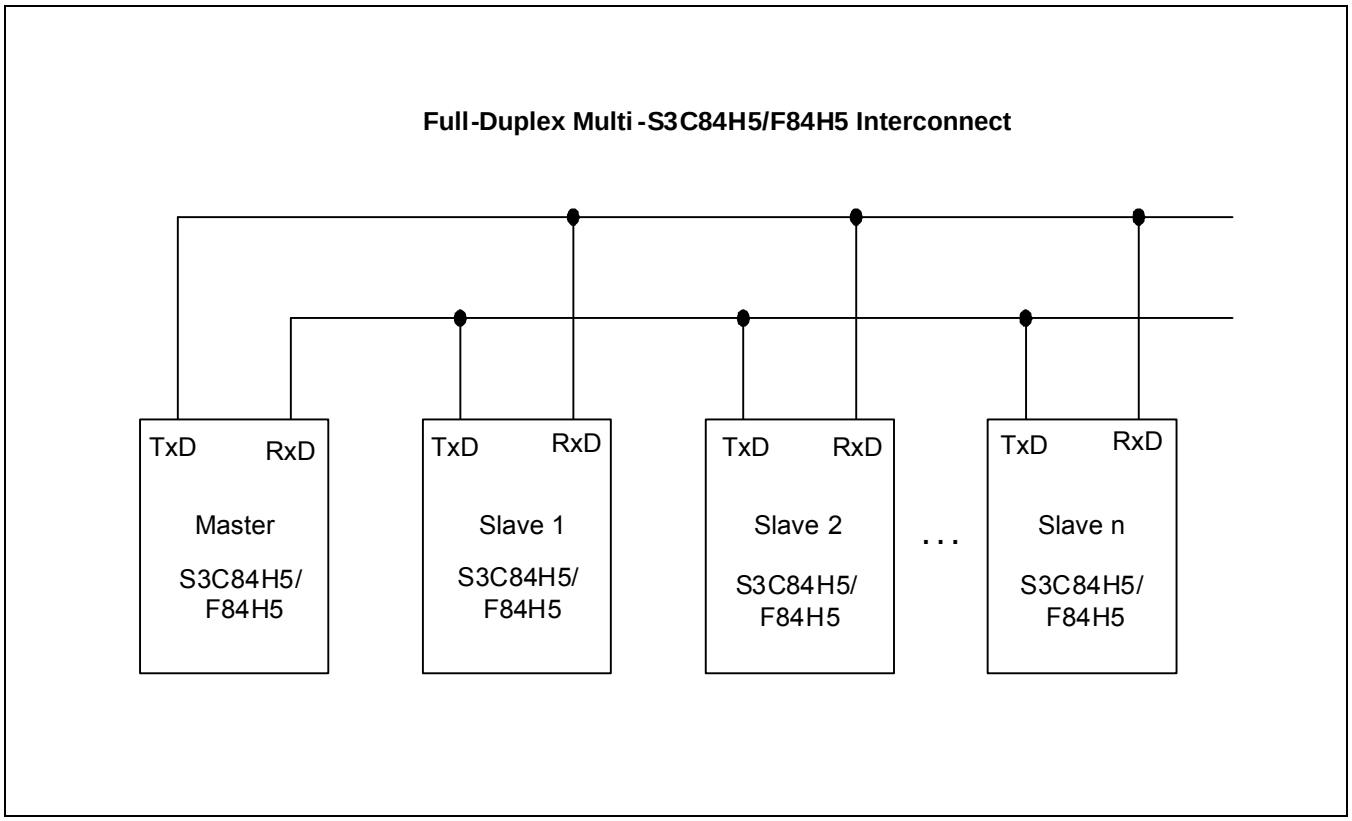


图 15-9 多节点串行通讯连接示例

16 A/D 转换器

16.1 概述

S3C84H5/F84H5 内部集成了一个8 通道10 位 A/D 转换器（ADC），通过逐次逼近逻辑把模拟信号转换为相应的 10 位数字量。模拟输入量的值必须在 V_{DD} 和 V_{SS}之间。A/D 转换器由以下几个部分组成：

- 逐次逼近型模拟比较器
- D/A 转换逻辑(电阻串型)
- AD 转换控制寄存器, ADCON (set 1, bank 0, F7H, 读/写, 但 ADCON.3 位只读)
- 8 路模拟信号输入端(ADC0-ADC7)
- 10 位 A/D 转换结果寄存器(ADDATAH, ADDATAL)

16.2 功能描述

为了开始一次模数转换，首先必须在转换开始前将 P0.0 - P0.3, P1.4 - P1.5, P2.2 - P2.3 配置为模拟输入口，P0.0 - P0.3, P1.4 - P1.5, P2.2 - P2.3 可以复用作普通 I/O 口或模拟输入口。为此， 需要写相应的数据到寄存器 P0CONL, P1CONH, P2CONL (ADC0 – ADC7)。然后需要通过 A/D 控制寄存器（ADCON）选择一路模拟输入通道，同时启动 A/D 转换即置高使能位 ADCON.0。每次只能实现一个通道的10位AD转换。A/D 转换控制寄存器的地址为 F7H，set 1，bank 0。

在一个正常转换过程中，AD 转换逻辑电路先将逐次逼近寄存器的值设为 200H（10 位满量程转换结果的一半）。随后这个寄存器在每次转换时会自动更新。逐次逼近电路每次只能实现一个通道的10位AD转换，但是可以通过操作 ADCON 寄存器的 ADCON.6-.4 动态分时选择多路 A/D 输入。

设置使能位 ADCON.0 位为“1”，则启动 A/D 转换。当 A/D 转换结束时，控制寄存器 ADCON.3 自动置“1”，即 EOC 被置“1”。A/D 转换结果保存于 ADDATAH/L 寄存器，A/D 转换电路随即进入空闲状态。 在开始下一次转换前，上一次 A/D 转换的数据必须被读出。否则，前面的结果将被下一次的转换结果覆盖。

注释: ADC 电路内部没有采样保持机制，所以在每次 A/D 转换期间，要求模拟信号输入端的波动尽可能小，否则转换结果可能会不正确。

16.2.1 A/D 转换控制寄存器(ADCON)

A/D 转换控制寄存器 ADCON 的地址为 F3H，set 1， bank 0。ADCON 是一个8位可读写寄存器，但 EOC 位，即ADCON.3 是只读位。该寄存器可以实现以下3个功能：

- 第 6-4 位选择模拟输入通道(ADC0-ADC7)
- 第 3 位是 A/D 转换结束标志位
- 第 2-1 位选择转换速度
- 第 0 位启动 A/D 转换

一次只能选择一路模拟输入通道。可以通过操作 ADCON 寄存器的 ADCON.6-.4 动态分时选择多路 A/D 输入。

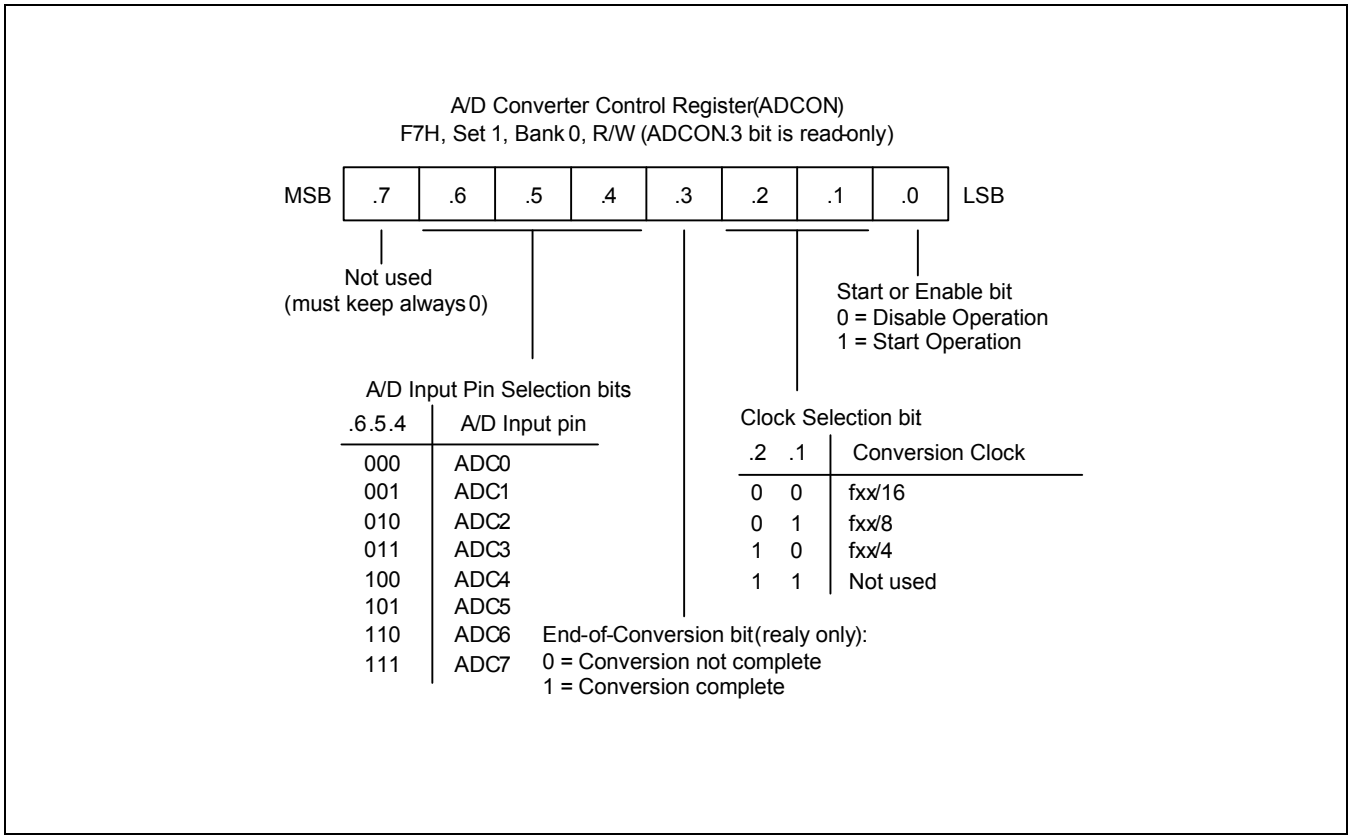


图 16-1 A/D 转换控制寄存器 (ADCON)

Conversion Data Register High Byte (ADDATAH)
F8H, Set 1, Bank 0, Read only

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Conversion Data Register Low Byte (ADDATAL)
F9H, Set 1, Bank 0, Read only

MSB	x	x	x	x	x	x	.1	.0	LSB

图 16-2 A/D 转换数据寄存器 (ADDATAH, ADDATAL)

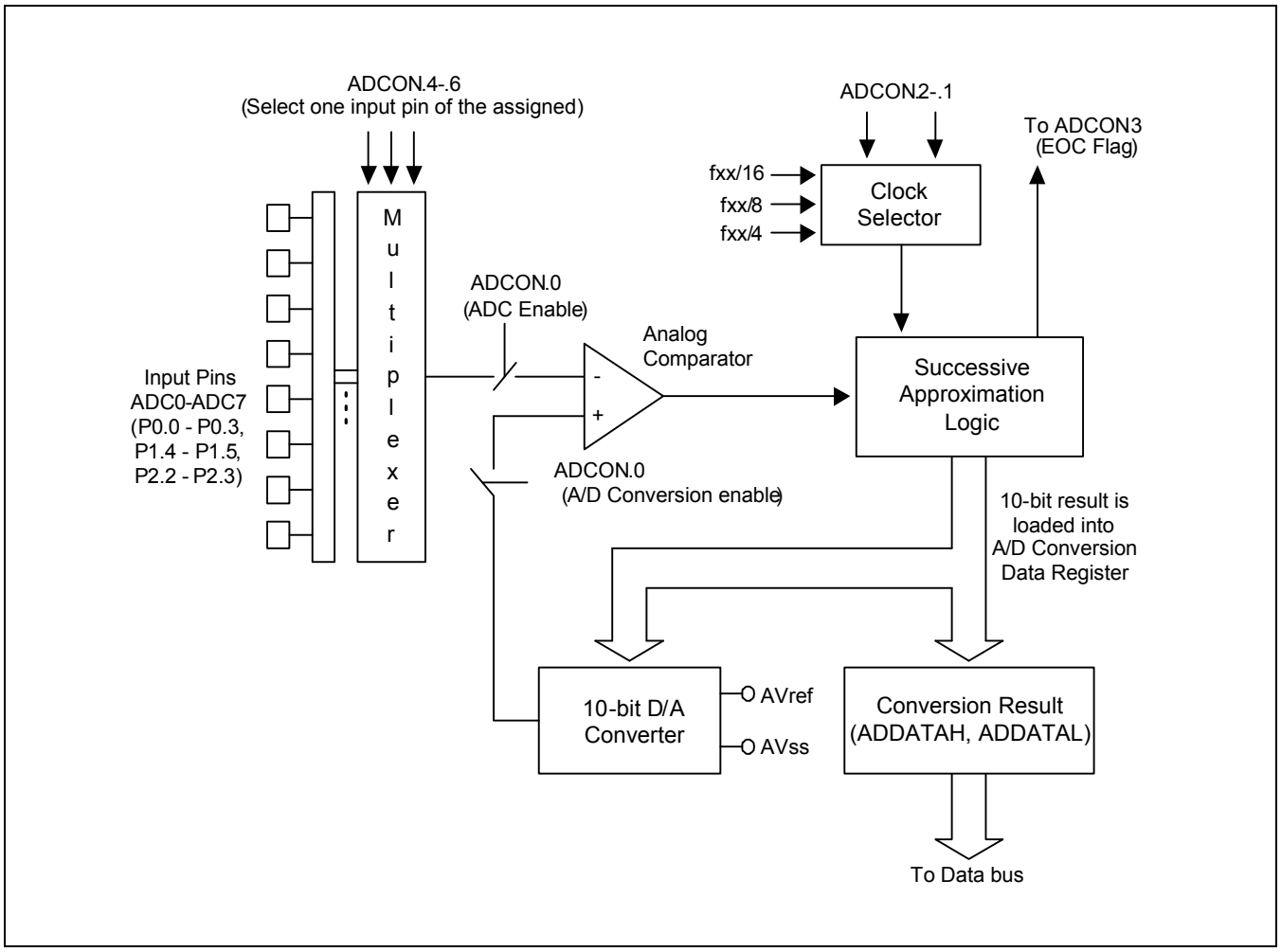


图 16-3 A/D 转换器电路图

16.2.2 内部参考电压

在 A/D 转换中，输入模拟信号要与内部参考电压进行比较。输入模拟信号的电压范围应在 AV_{SS}至 AV_{REF} 之间。($AV_{REF} = V_{DD}$)。

不同的比较电压是由内部电阻网络分压产生的，开始的比较电压一般为 1/2 AV_{REF}。

16.2.3 转换时间

A/D 转换 1 位需要 4 个ADC 时钟周期，建立 A/D 转换需要 10 个时钟周期，因此 A/D 转换完 10 位一共需要 50 个时钟周期。在 10MHZ 频率下，一个时钟周期为400 ns (4/fxx)。则 A/D 转换所需要的时间为：

```
4 时钟周期/位 x 10 位 + 建立时间 (10 个时钟周期) = 50 个时钟周期
50 个时钟周期 x 400 ns = 20 μs at 10 MHz, 1 个时钟周期 = 4/fxx
```

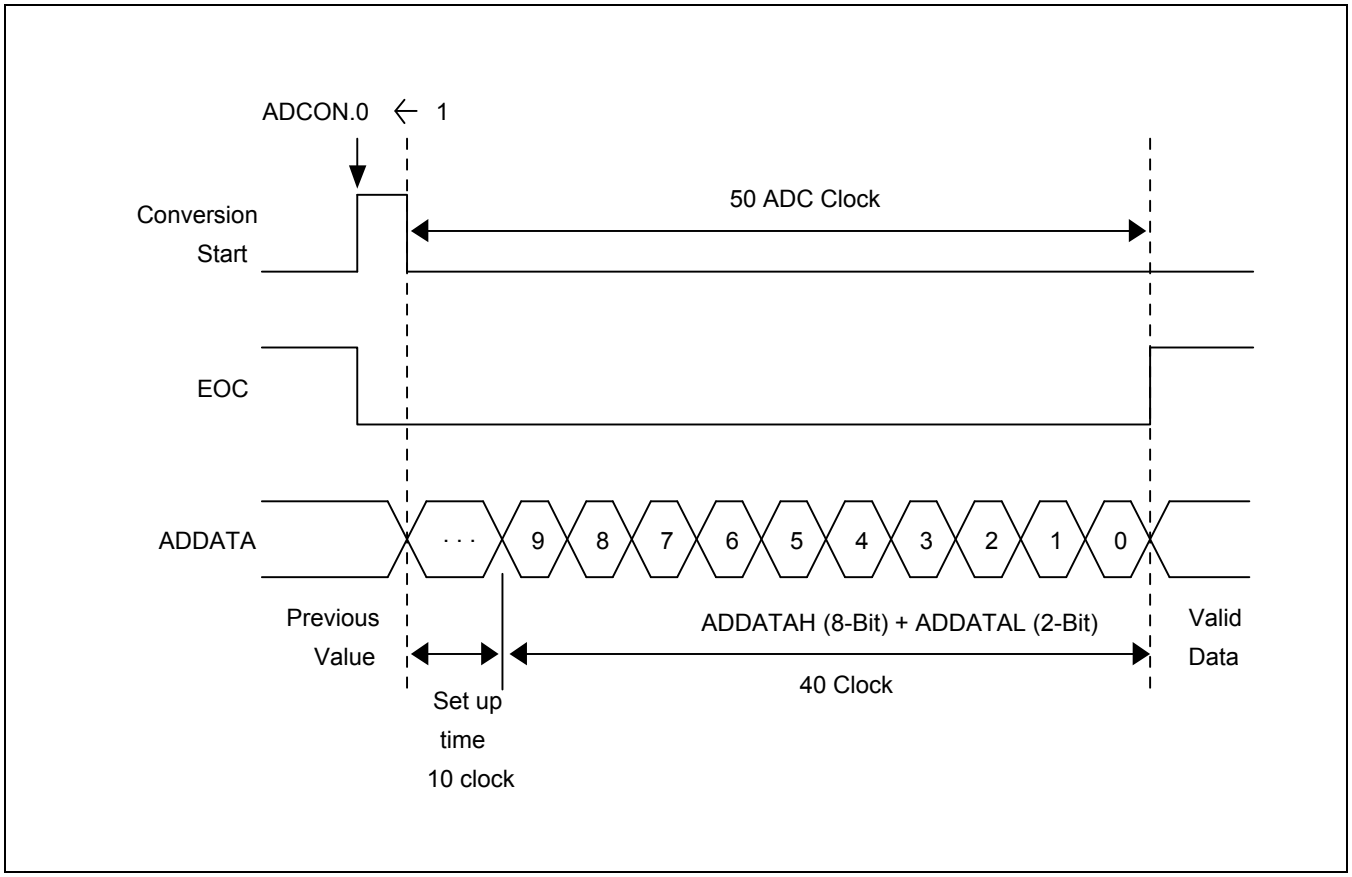
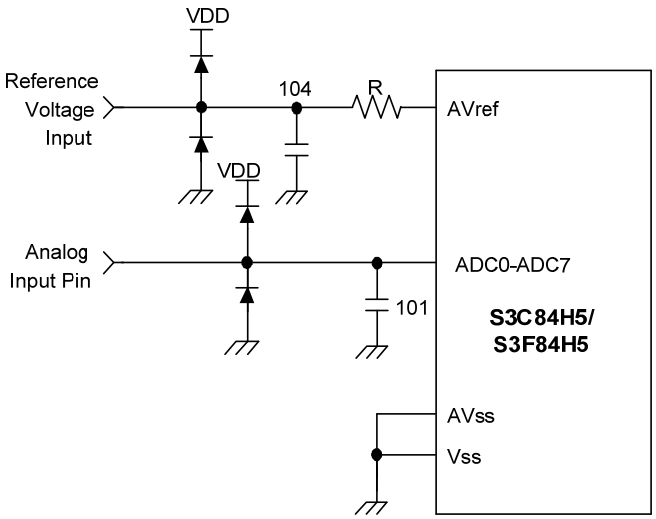


图 16-4 A/D 转换时序图

16.2.4 内部 A/D 转换过程

1. 模拟信号输入应该在AV_{SS}和AV_{REF}之间。
2. 将 P0.0 - P0.3, P1.4 - P1.5, P2.2 - P2.3 配置为模拟输入口，为此，需要写相应的数据到寄存器 P0CONL, P1CONH, P2CONL (ADC0 - ADC7)。
3. 在开始转换之前，通过设置 ADCON 控制寄存器来选定一路模拟通道 (ADC0 - ADC7)。
4. A/D 转换结束后，将置起 EOC 标志位，通过检测此位，可以确定 A/D 转换是否进行完毕。
5. A/D 转换结束后，相应的数据存放在 ADDATAH (8 位) 和 ADDATAL (2 位)寄存器中。转换完毕后，ADC 转换模块进入空闲状态。
6. 转换结果可以从寄存器 ADDATAH 和 ADDATAL 中读出。



注释： 标志 "R"表示阻值为 50～100Ω 的偏置电阻。

图 16-5 推荐高精度 A/D 转换电路

编程实例 16-1 配置 A/D 转换器

```
                •
                SB1                         ; 调试相关指令 (注释)
                LD      0F7H,#5FH           ; 调试相关指令 (注释)
                SB0                         ; 调试相关指令 (注释)
                •
                •
                •
                LD      P0CON, #11111111B   ; 配置 P0.0-P0.3 为 A/D 输入端口
                LD      P1CONH, #00001111B  ; 配置 P1.4-P1.5 为 A/D 输入端口
                LD      P2CONL, #11110000B  ; 配置 P2.2-P2.3 为 A/D 输入端口

                •
                •

                LD      ADCON, #00000001B   ; 选择输入通道 ADC0, fxx, 转换开始
AD0_CHK:        TM      ADCON, #00001000B   ; A/D 转换结束? → EOC 检测
                JR      Z, AD0_CHK          ; 转换未结束

                LD      AD0BUFH, ADDATAH    ; 8 位转换数据
                LD      AD0BUFL, ADDATAL    ; 2 位转换数据
                •
                •

                LD      ADCON, #00110001B   ; 选择输入通道 ADC3, fxx, 转换开始
AD3_CHK:        TM      ADCON, #00001000B   ; A/D 转换结束? → EOC 检测
                JR      Z, AD3_CHK          ; 转换未结束

                LD      AD3BUFH, ADDATAH    ; 8 位转换数据
                LD      AD3BUFL, ADDATAL    ; 2 位转换数据
                •
                •
```

注释: 在调试时，必须在初始化代码中包含这三条相关指令以正确初始化 I/O 口。如果遗漏这些指令，I/O 口将不能正确操作。在完成调试以后，必须删除这些指令。

17 钟表定时器

17.1 概述

钟表定时器的功能包括实时时钟，时间测量以及为系统时钟提供间隔定时。为了启动钟表定时器，将控制寄存器的第 1 位和第6位 (WTCON.1和.6) 设置为 "1"。钟表定时器启动之后经过一段时间，中断标志会被自动置 "1"， 中断请求以 1.955ms, 0.125 s, 0.25s 或 0.5s 的间隔发生。

钟表定时器可以产生 0.5 kHz, 1 kHz, 2 kHz 或 4 kHz 的稳定信号，送到蜂鸣器的输出管脚 (BUZ)。将 WTCON.3 和 WTCON.2 设置为 "11b"，钟表定时器将工作于高速模式，每隔 1.955ms 产生一次中断。 在程序调试时可使用高速模式来对事件定时。

钟表定时器包含以下组成部分：

- 实时时钟和时间测量
- 时钟源可使用主系统时钟或副系统时钟
- 蜂鸣器输出频率发生器
- 高速模式下的时序测试

17.1.1 钟表定时器控制寄存器(WTCON : R/W)

F8H	WTCON.7	WTCON.6	WTCON.5	WTCON.4	WTCON.3	WTCON.2	WTCON.1
RESET	"0"	"0"	"0"	"0"	"0"	"0"	"0"

表 17-1 钟表定时器控制寄存器 (WTCON) : Set1, Bank1, F8H 读/写

位	值		功能	地址
WTCON.7	0		选择 (fx/256) 作为钟表定时器时钟 (fx: 主时钟)	F8H
	1		选择副系统时钟 作为钟表定时器时钟	
WTCON.6	0		禁止钟表定时器中断	
	1		使能钟表定时器中断	
WTCON.5-.4	0	0	0.5 kHz 蜂鸣器 (BUZ) 信号输出	
	0	1	1 kHz 蜂鸣器 (BUZ) 信号输出	
	1	0	2 kHz 蜂鸣器 (BUZ) 信号输出	
	1	1	4 kHz 蜂鸣器 (BUZ) 信号输出	
WTCON.3-.2	0	0	设置钟表定时器中断间隔为 0.5 s.	
	0	1	设置钟表定时器中断间隔为 0.25 s.	
	1	0	设置钟表定时器中断间隔为 0.125 s.	
	1	1	设置钟表定时器中断间隔为 1.955 ms.	
WTCON.1	0		关闭钟表定时器, 清分频设置	
	1		使能钟表定时器	
WTCON.0	0		无中断产生, 清中断标志(写)	
	1		有中断产生	

注释: 假定主系统时钟(fx) 频率为 9.8304 MHz

17.1.2 钟表定时器电路框图

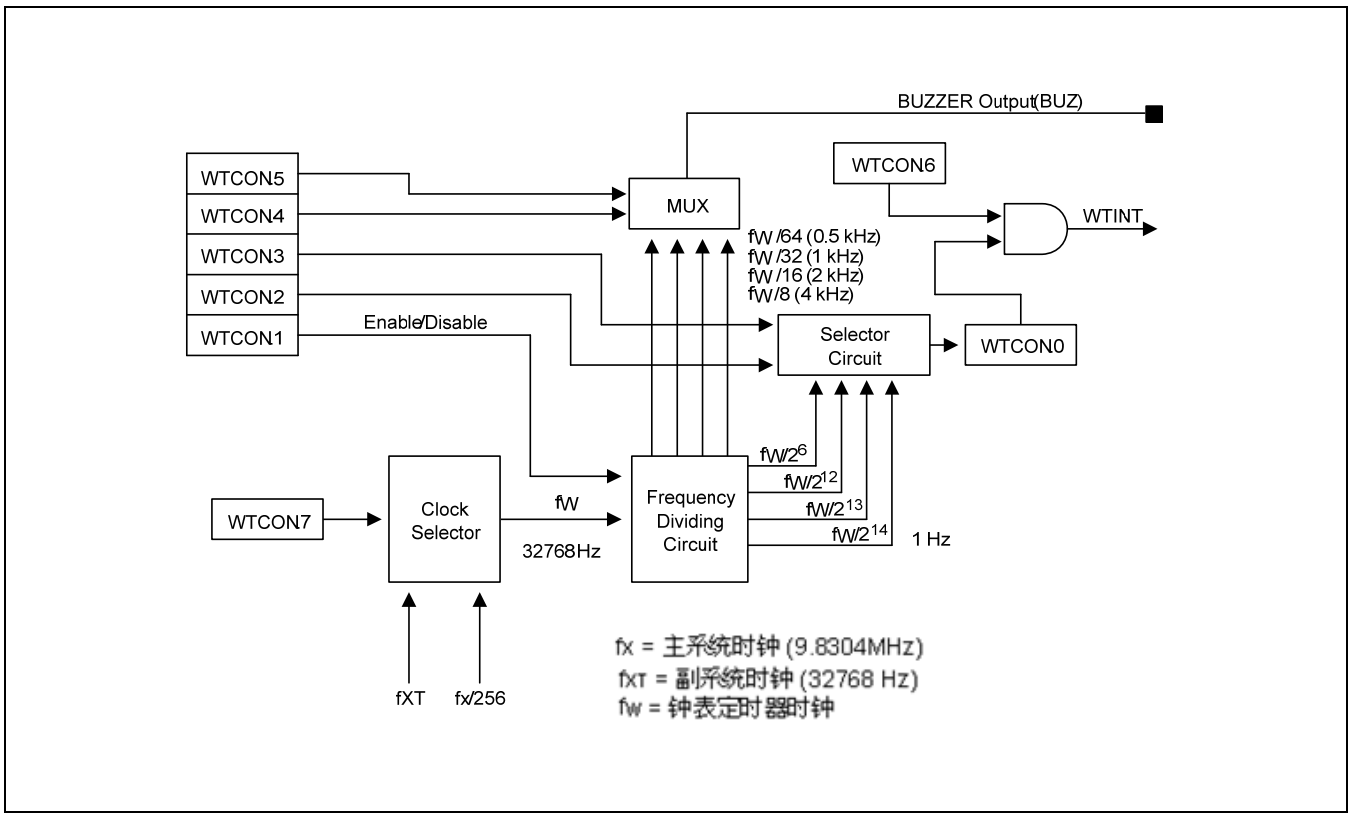


图 17-1 钟表定时器电路框图

编程实例 17-1 使用钟表定时器

```
                ORG             0000h

                VECTOR          0D6h, WT_INT

                ORG             0100h

INITIAL:
                DI
                SB1                             ; 调试模式下的额外指令 (注释)
                LD              0F7H,#5FH       ; 调试模式下的额外指令 (注释)
                SB0                             ; 调试模式下的额外指令 (注释)

                LD              IMR,#00010000b  ; 使能 IRQ3 中断
                LD              SPH,#00000000b  ; 设置堆栈指针
                LD              SPL,#0FFh
                LD              BTCON,#10100011b ; 禁止看门狗功能
                LD              WTCON,#11001110b ; 0.5 kHz 蜂鸣器, 1.955ms 定时中断
                                                 ; 中断使能, (fxt:32,768Hz)
                EI

MAIN:
                •
                •
                •
                MAIN            ROUTINE
                •
                •
                •

                JR              T, MAIN

WT_INT:
                •
                •
                •
                AND             WTCON,#11111110b ; 清中断标志位

                IRET

                .END
```

注释: 调试模式下，为了正确地操作端口，必须在初始化程序中包含这 3 条额外的指令，否则端口操作将无法正常进行。程序完成后可去除这几条指令。

18 低电压复位

18.1 概述

通过设置Smart Option (ROM 中的 3FH.6)，可以选择内部低电压复位 (LVR) 或外部复位。

S3C84H5/F84H5 有 4 种复位方式：

- 外部上电复位
- 拉低外部复位输入管脚电平复位
- 看门狗溢出复位
- 内部低电压 (LVR) 复位

如果采用外部上电复位，当 V_{DD} 管脚为高电平并且复位管脚为低电平时，复位信号通过施密特触发电路与 CPU 时钟同步，继而复位系统。复位操作使得 S3C84H5/F84H5 处于已知的操作状态。为了确保芯片正常开始工作，必须维持复位信号为低电平直到电源 VDD 爬升充分。

接电源后，复位管脚必须维持低电平一段时间，以允许内部 CPU 时钟振荡达到稳定。 最小的振荡稳定等待时间约为 6.55 ms (≅216/fxx, fxx= 10 MHz)。

在正常操作模式下，V_{DD} 和 RESETB 均为高电平。 当 RESETB 管脚电平变低时，将会导致系统复位。 复位后，系统和外围控制寄存器均恢复为默认值 (如表 8-1)。

MCU 带有看门狗功能，以防程序跑飞。使能看门狗后，如果程序没有在额定时间内清除看门狗计数器，计数器计满溢出后，就会启动 MCU 复位操作。

S3C84H5/F84H5 内建一个低电压复位电路，提供 V_{DD} 输入电压跌落检测功能，以防止 MCU 在供电不稳定情况下运行异常。 这个电压检测模块是MCU复位电路的一部分，由一个模拟比较器和参考电压电路组成。 检测的电压值是2.8V。使能片上低电压复位后，当电压低于设定电压时 (典型值为2.8 V)，系统会进入复位状态。 由于这个特性，可在保证应用安全性的同时，为用户省去外部复位电路。只要电压低于设定电压，内部即处于复位状态；而当电压大于设定电压时，系统又会重新开始工作。

在任何操作模式下，例如 STOP，IDLE 和普通 RUN 模式，计算能耗时需要额外地将 LVR 电路的静态电流加入到芯片工作电流中。

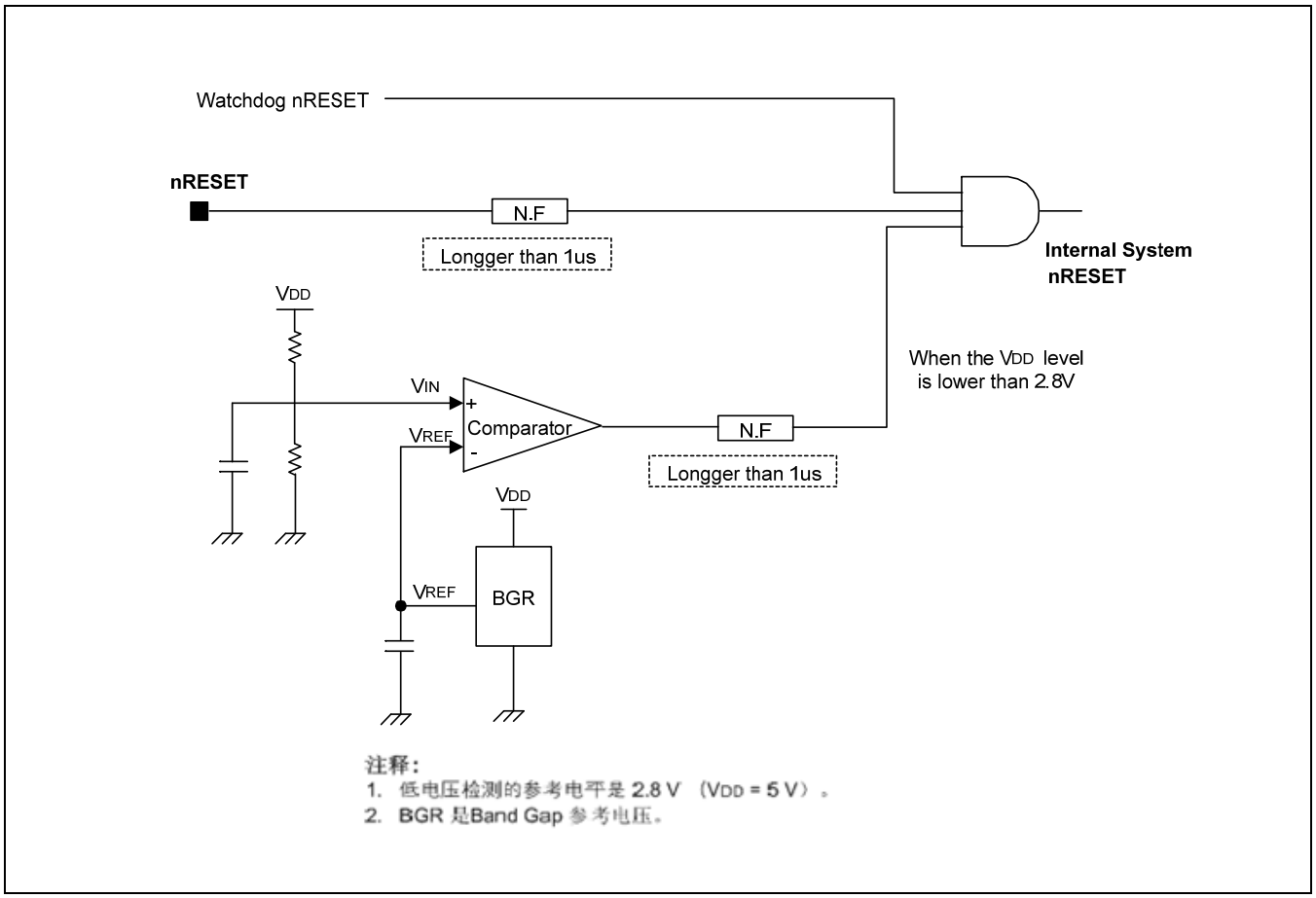


图 18-1 低压复位电路

注释: 如果希望改变振荡稳定的等待时间，可以在进入 STOP 状态前重新设置 Basic Timer 的控制寄存器 BTCON。如果不希望使用 Basic Timer 看门狗功能 (Basic Timer 计数器溢出，会导致系统复位)，可以把 "1010B" 写到 BTCON 的高 4 位，以禁止看门狗工作。

19 嵌入式闪存接口(MTP)

19.1 概述

S3F84H5 单芯片CMOS MCU是S3C84H5的多次可编程版本。区别于S3C84H5的掩模ROM，S3F84H5 片上集成了一个 Half Flash ROM，并可以通过专业的串行协议对其进行访问。Half Flash ROM 的可编程次数高达 100次。

除此之外，S3F84H5和 S3C84H5在功能，DC电器特性和管脚配置上都完全兼容。
因为S3F84H5的可多次编程的特性，可以在S3C84H5调试阶段使用S3F84H5。

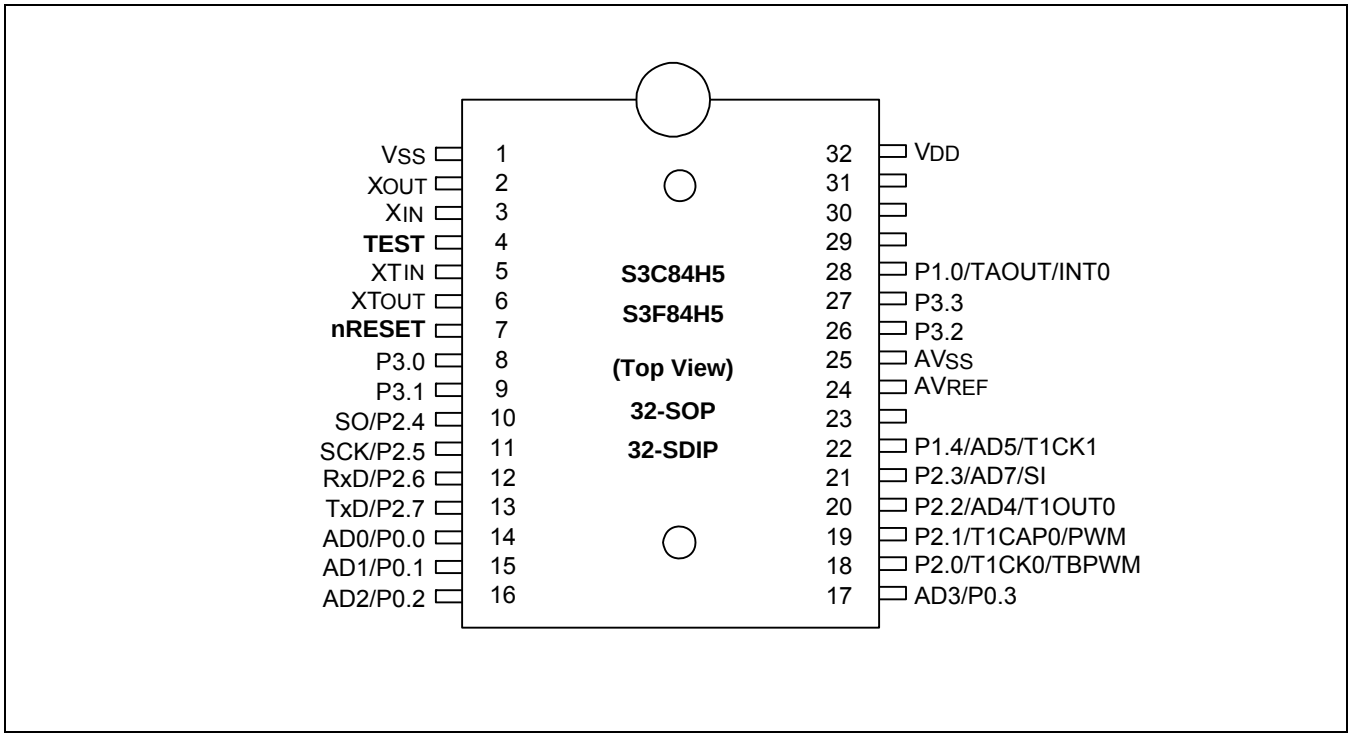


图 19-1 管脚分布图 (32SOP/SDIP)

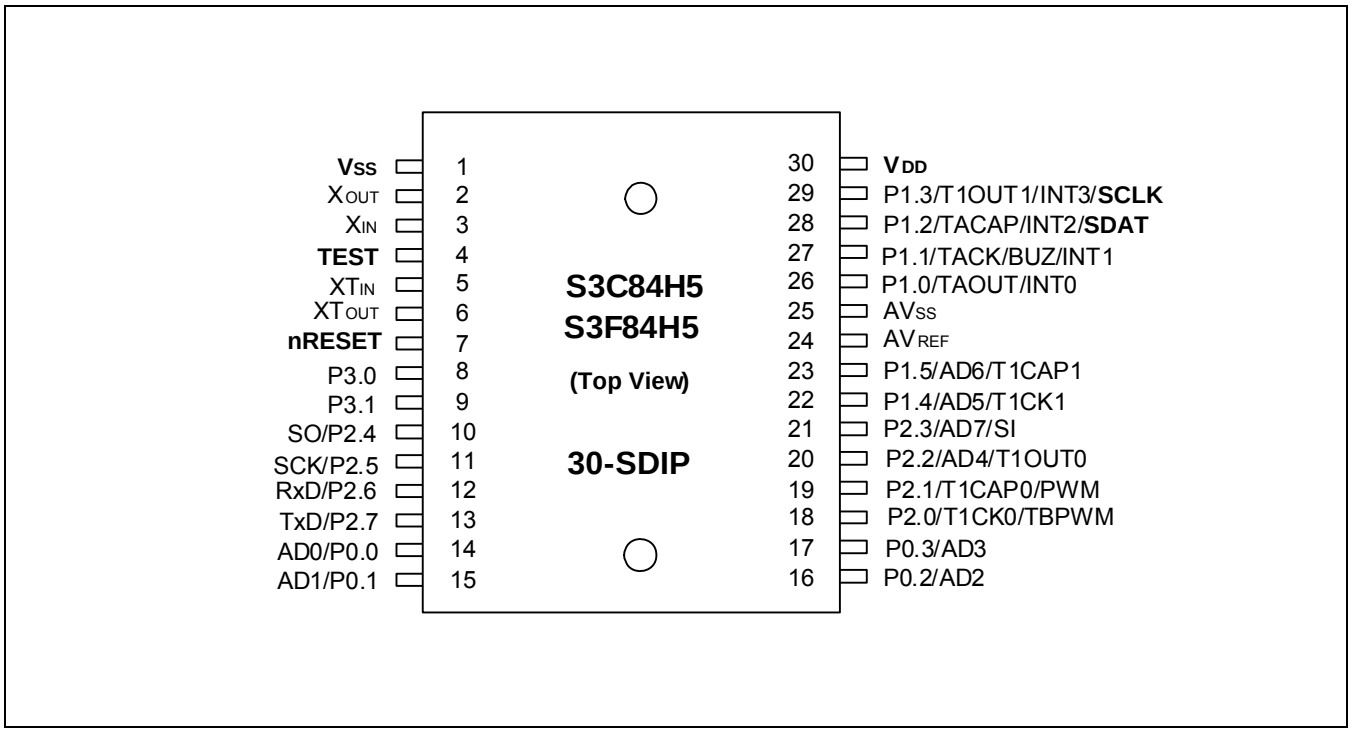


图 19-2 管脚分布图(30SDIP)

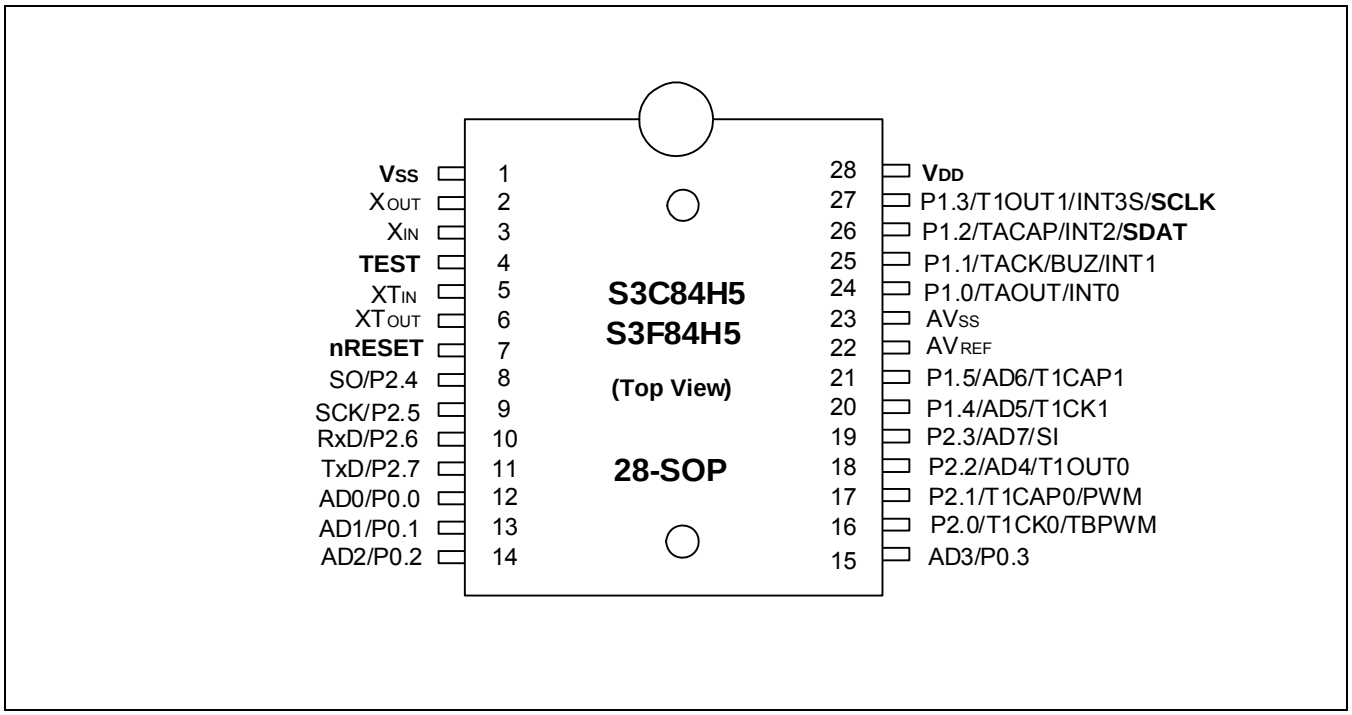


图 19-3 管脚分布图 (28SOP)

表 19-1 闪存读写管脚描述

主芯片管脚名称	编程过程中			
	管脚名称	管脚号	输入/出	功能
P1.2	SDAT	30 (32-pin) 28 (30-pin) 26 (28-pin)	I/O	串行数据管脚（读时为输出脚，写入时为输入脚），管脚可设置为输入和推挽式输出模式。
P1.3	SCLK	31 (32-pin) 29 (30-pin) 27 (28-pin)	I	串行时钟管脚（仅为输入管脚）。
TEST	VPP	4	I	用于Tool模式下flash单元编程的供电管脚 （指示MTP进入编程模式）。 管脚上检测到12.5 V，MTP 即进入编程状态；检测到5V，MTP进入读状态。
nRESET	nRESET	7	I	芯片初始化。
V_{DD}/V_{SS}	V_{DD}/V_{SS}	32/1 (32-pin) 30/1 (30-pin) 28/1 (28-pin)	I	逻辑供电管脚。

表 19-2 S3F84H5 和S3C84H5 特性比较

特性	S3F84H5	S3C84H5
程序存储空间	16 K字节 Flash ROM	16K 字节 mask ROM
工作电压(VDD)	2.8 V 到 5.5 V	2.8 V 到 5.5 V
MTP 编程模式	V_{DD} = 5 V, V_{PP} = 12.5 V	-
管脚配置	32 SDIP/ SOP, 30 SDIP,28 SOP	
EPROM 可编程性	用户可多次编程	工厂编程

20 电气参数

20.1 概述

在本章节，将通过表格和图形来介绍 S3C84H5/F84H5 的电气特性。
按照下面的顺序介绍：

- 芯片极限物理特性
- 输入/输出电容
- 直流电气特性
- 交流电气特性
- 振荡器特性
- 振荡稳定时间
- STOP 模式下数据保持供电电压
- 在模式 0 下 UART 时序特性
- A/D 转换电气特性

表 20-1 芯片极限物理特性

(T_A = 25 °C)

参数	符号	条件	数值	单位
电源电压	V_{DD}	-	- 0.3 to +3.8	V
输入电压	V_{IN}	-	- 0.3 to V_{DD} + 0.3	
输出电压	V_O	所有输出管脚	- 0.3 to V_{DD} + 0.3	
输出高电流	I_{OH}	只有一个输入/输出管脚工作	- 15	mA
		所有的输入/输出管脚都在工作	- 60	
输出低电流	I_{OL}	只有一个输入/输出管脚工作	+ 30	
		所有的输入/输出管都在工作	+ 200	
工作温度	T_A	-	- 25 to + 85	°C
储存温度	T_{STG}	-	- 65 to + 150	°C

表 20-2 输入/出电容

(T_A = -25 °C to + 85 °C，VDD = 0V)

参数	符号	条件	最小值	典型值	最大值	单位
输入电容	C_{IN}	f = 1 MHz; 未被测量的管脚接至 V_{SS}	-	-	10	pF
输出电容	C_{OUT}					
输入/输出电容	C_{IO}					

表 20-3 直流电气特性

(T_A = - 25 °C to + 85 °C, 2.5V to 5.5 V)

参数	符号	条件	最小值	典型值	最大值	单位
工作电压	V_{DD}	Fx = 0 - 8MHz, fxt = 32.8 kHz LVR 关断	2.5	-	5.5	V
		Fx = 0 - 8MHz, fxt = 32.8 kHz LVR 打开	LVR	-	5.5	
		Fx = 0 - 10 MHz	4.5	-	5.5	
输入高电压	V_{IH1}	V_{DD} = 2.5V to 5.5 V 所有端口和 nRESET	0.8 V_{DD}	-	V_{DD}	V
	V_{IH2}	V_{DD} = 2.5V to 5.5 V X_{IN} 和 XT_{IN}	V_{DD} - 0.5	-	-	
输入低电压	V_{IL1}	V_{DD} = 2.5V to 5.5 V 所有端口和nRESET	-	-	0.2V_{DD}	V
	V_{IL2}	V_{DD} = 2.5V to 5.5 V X_{IN} 和 XT_{IN}	-	-	0.4	
输出高电压	V_{OH}	VDD = 5.0 V IOH = - 2 mA 所有端口	VDD - 1.0	-	-	V
输出低电压	V_{OL1}	VDD = 5.0 V, IOL = 16 mA 端口 0 和 4	-	0.4	2.0	V
	V_{OL2}	VDD = 5.0 V, IOL = 4 mA 端口 1, 2 和 3	-	-	-	
输入高漏电流	I_{LIH1}	VIN = VDD 除 ILIH2之外的所有输入管脚	-	-	3	μA
	I_{LIH2}	VIN = VDD XIN, XOUT 和XTIN, XTOUT	-	-	20	
输入低漏电流	I_{LIL1}	VIN = 0 V 除I_{LIL2}之外的所有输入管脚	-	-	- 3	
	I_{LIL2}	VIN = 0 V XIN, XOUT 和 XTIN, XTOUT	-	-	- 20	
输出高漏电流	I_{LOH}	VOUT = VDD 所有输出管脚	-	-	3	
输出低漏电流	I_{LOL}	VOUT = 0 V 所有输出管脚	-	-	- 3	

表 20-4 直流电气参数(续)

(T_A = - 25 °C to + 85 °C, V_{DD} = 2.5 V to 5.5 V)

参数	符号	条件	最小值	典型值	最大值	单位
上拉电阻	R_{P1}	V_{DD} = 5 V; V_{IN} = 0 V, T_A = 25 °C 除nRESET之外的所有输入管脚	25	50	100	kΩ
	R_{P2}	V_{DD} = 5 V; V_{IN} = 0 V, T_A = 25 °C nRESET管脚	150	250	480	
供电电流 (1)	I_{DD1} (2)	V_{DD} = 4.5 V to 5.5 V 运行模式 10 MHz CPU clock	-	5.0	10	mA
		V_{DD} = 2.5 V to 3.3 V 运行模式 4 MHz CPU clock	-	2.5	5.0	
	I_{DD2}	V_{DD} = 4.5 V to 5.5 V IDLE 模式 10 MHz CPU clock	-	2.0	4.0	
		V_{DD} = 2.5 V to 3.3 V IDLE 模式 4 MHz CPU clock	-	1.0	2.0	
	I_{DD3}	Sub Run 模式, 主振荡器停止: V_{DD} = 2.5 V to 3.3 V 32768 Hz	-	400	800	uA
	I_{DD4}	Sub IDLE 模式, 主振荡器停止: V_{DD} = 2.5 V to 3.3 V 32768 Hz	-	300	600	
	I_{DD5} (3)	V_{DD} = 4.5V to 5.5 V, T_A = 25 °C, STOP模式	-	150	400	

注释:

1. 供电电流不包括内部上拉电阻或外部负载消耗的电流。
2. IDD1和IDD2包括子系统振荡器消耗的电流。
3. IDD3和IDD4是主系统时钟停止而使用子系统时钟时消耗的电流。
4. IDD5是主系统和子系统时钟均停止时的电流。
5. 所有电流 (IDD1- IDD4) 包括LVR电路消耗的电流。
6. LVR打开和关断时，IDD5都一样。

表 20-5 A.C. 电气参数

(T_A = - 25 °C to + 85 °C, 2.5V to 5.5 V)

参数	符号	条件	最小值	典型值	最大值	单位
中断输入高低电平脉冲宽度(Ports 2)	t_{INTH}, t_{INTL}	V_{DD} = 5 V	180	-	-	ns
nRESET 输入低电平脉冲宽度	t_{RSL}	输入	1.0	-	-	µs

注释: 用户必须保持输入值比最小值要大。

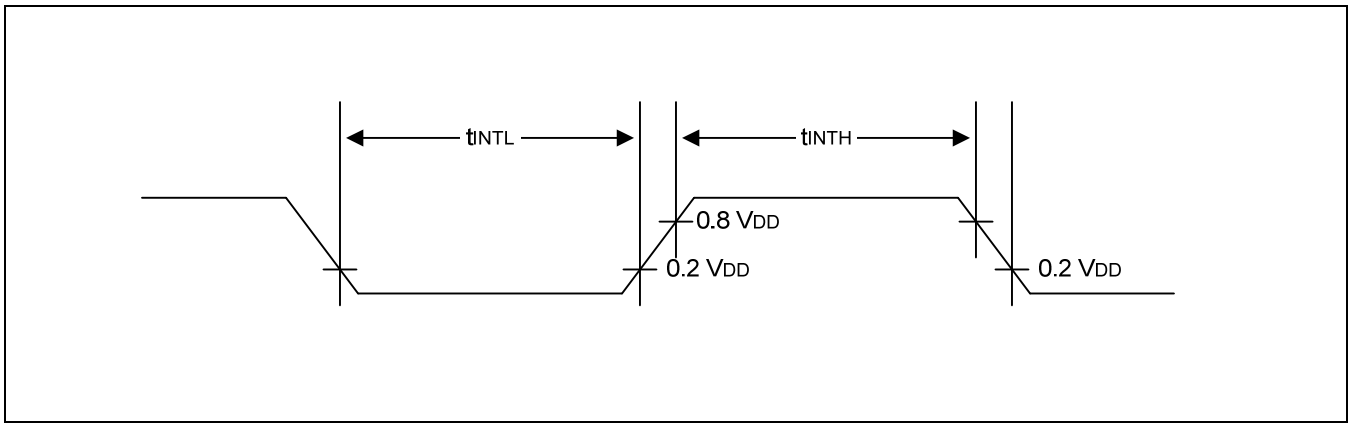


图 20-1 外部中断输入时序 (Ports2)

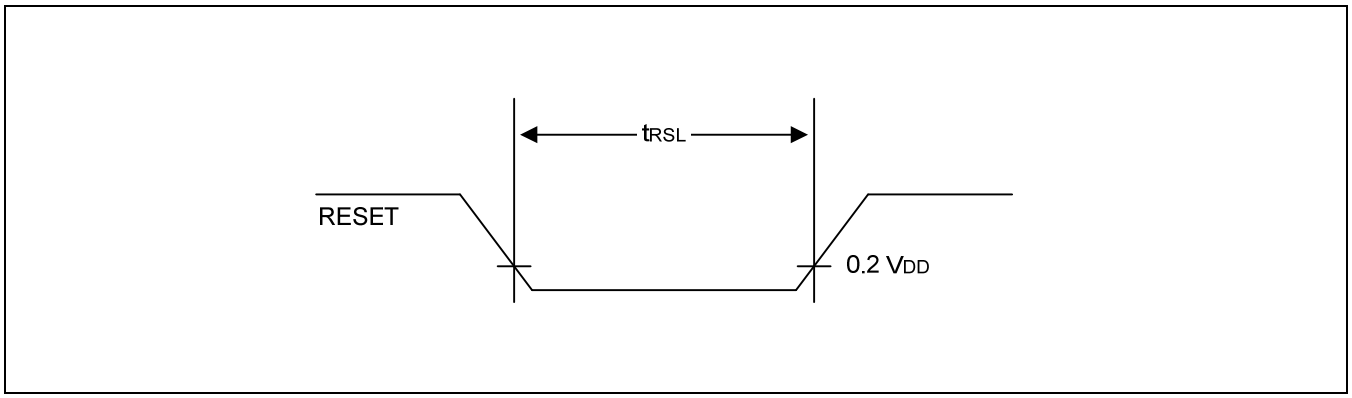


图 20-2 nRESET 输入时序

表 20-6 主振荡器频率 (f_{OSC1})

(T_A = - 25 °C + 85 °C, 2.5V to 5.5V)

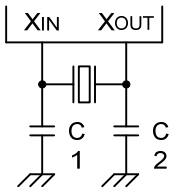
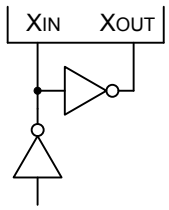
振荡器	时钟电路	测试条件	最小值	典型值	最大值	单位
主晶体振荡器或陶瓷振荡器		V_{DD} = 2.5V to 5.5 V	1	-	10	MHz
外部时钟 (主系统)		V_{DD} = 2.5V to 5.5 V	1	-	10	

表 20-7 主振荡器稳定时间(t_{ST1})

(T_A = - 25 °C + 85 °C, 2.5V to 5.5V)

振荡器	测试条件	最小值	典型值	最大值	单位
主晶体振荡器	f_{OSC} > 400 kHz; 当V_{DD} 等于振荡电路起振的最小值时候， 振荡稳定时间开始计时	-	-	10	ms
陶瓷振荡器		-	-	4	-
外部时钟(主系统)	X_{IN} 输入高电平和低电平宽度 (t_{XH}, t_{XL})	50	-	-	ns
振荡器稳定等待时间	复位释放 t_{WAIT} (1)	-	$2^{16}/f_{OSC}$	-	sec
	中断释放 t_{WAIT} (2)	-	-	-	sec

注释:

1. f_{OSC} 为振荡器频率。
2. 当中断释放时，振荡器稳定等待持续时间 t_{WAIT} 由基本计时器控制寄存器 BTCON 的设置决定。

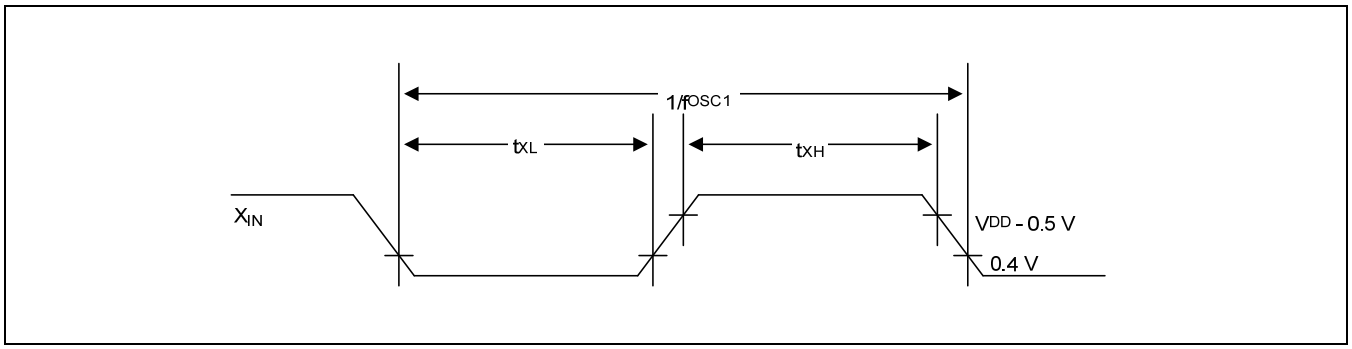


图 20-3 在X_{IN} 的时钟时序测量值

表 20-8 子振荡器频率 (f_{OSC2})

(T_A = - 25 °C + 85 °C, V_{DD} = 2.5 to 5.5V)

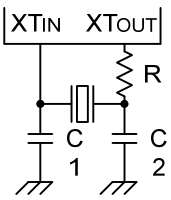
振荡器	时钟电路	测试条件	最小值	典型值	最大值	单位
晶体振荡器		晶体振荡频率 C1 = 100 pF, C2 = 100 pF R = 330 XT_{IN} 和 XT_{OUT} 与 R 和C 焊接起来	32	32.768	34	kHz

表 20-9 子系统振荡器 (晶体) 稳定时间 (t_{ST2})

(T_A = 25 °C)

测试条件	最小值	典型值	最大值	单位
V_{DD} = 4.5 V to 5.5V	-	800	1600	ms
V_{DD} = 2.5 to 3.3V	-		10	s

注释: 振荡稳定时间 (t_{ST2})是当 STOP 模式被中断释放以后振荡器回到正常振荡所需要的时间。

表 20-10 STOP 模式数据保持供电电压

(T_A = - 25 °C to + 85 °C, 2.5V to 5.5 V)

参数	符号	条件	最小值	典型值	最大值	单位
保持数据所需供电电压	V_{DDDR}	STOP模式	2.5	-	5.5	V
保持数据所需供电电流	I_{DDDR}	STOP模式, V_{DDDR} = 2.5 V	-	-	8	µA

注释: 供电电流不包括内部上拉电阻电流或外部输出加载电流。

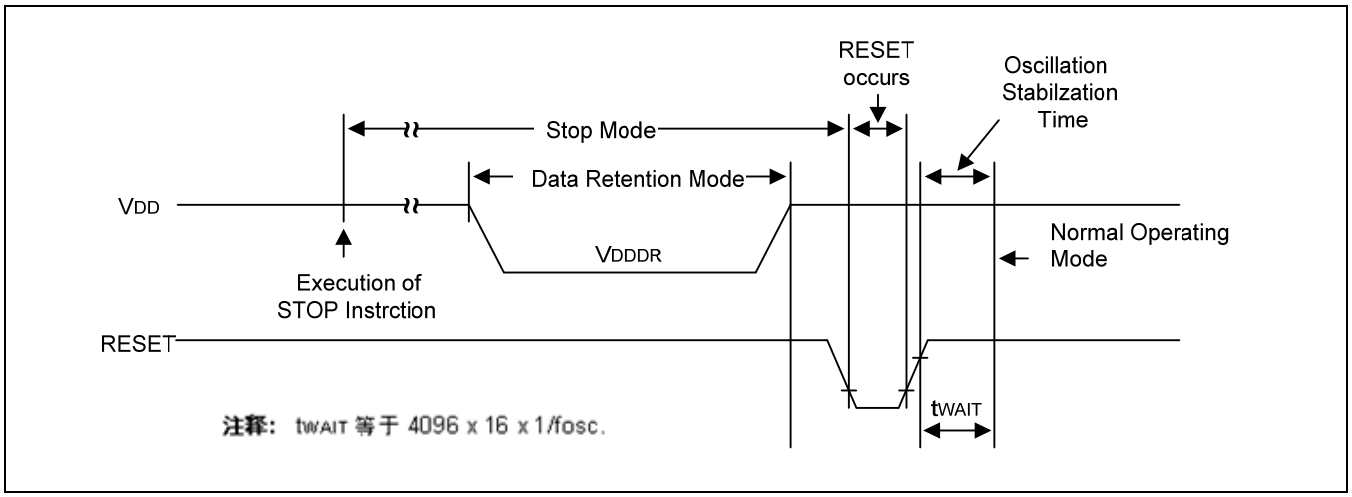


图 20-4 复位使系统退出 Stop 模式时序图

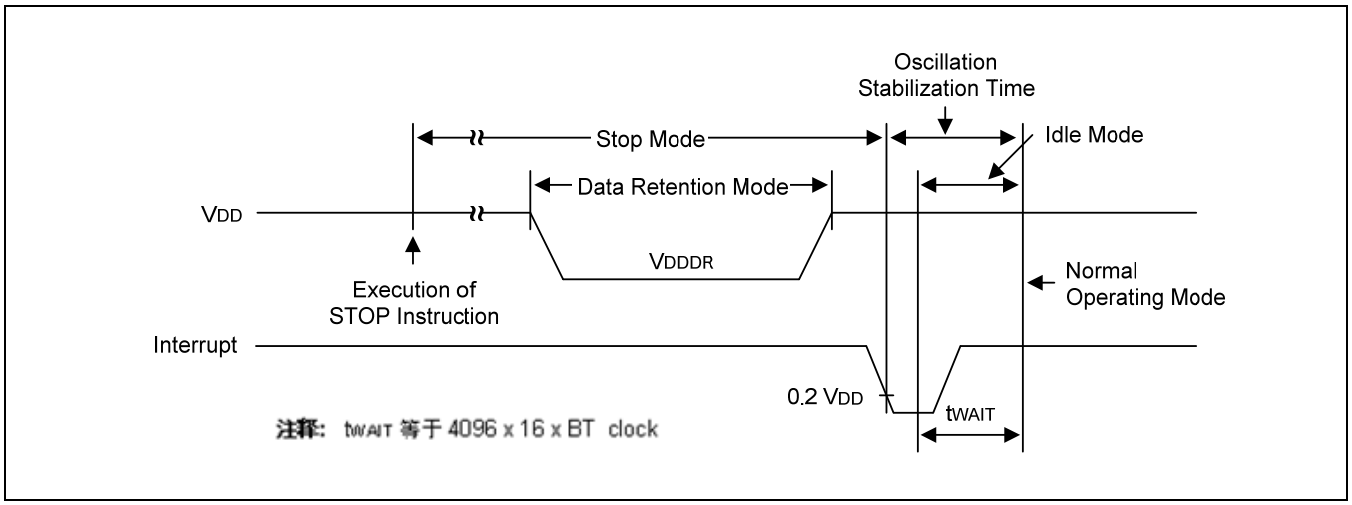


图 20-5 中断使系统退出 Stop 模式（主系统）时序图

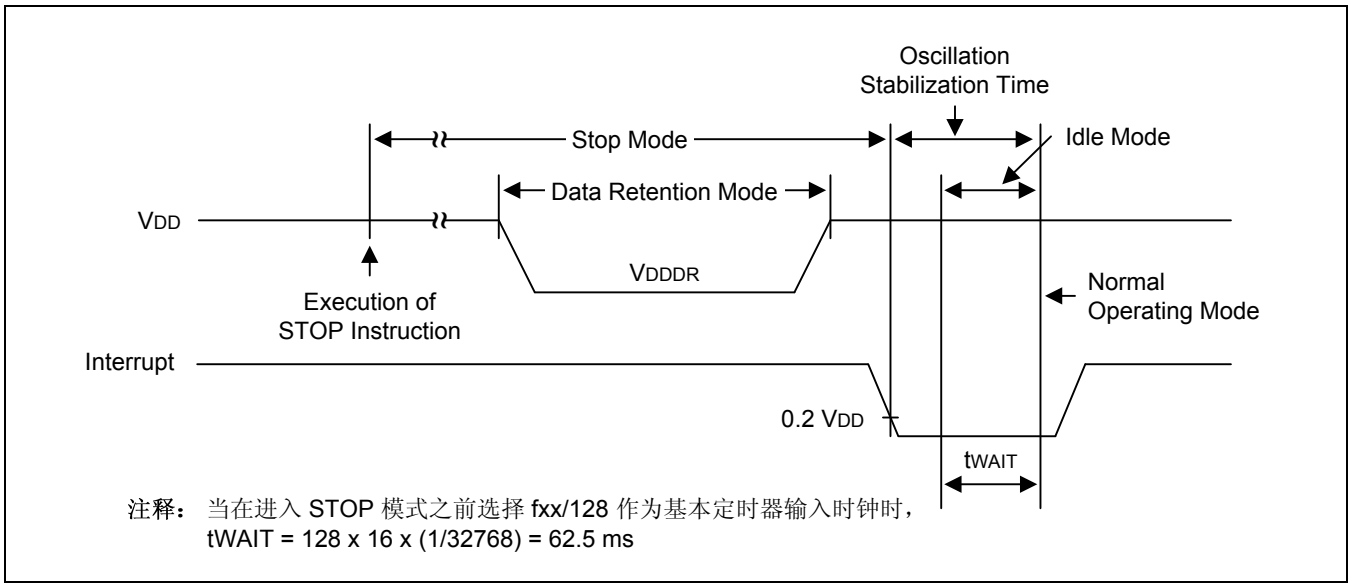


图 20-6 中断使系统退出 Stop 模式（子系统）时序图

表 20-11 Mode0 UART 时序特性(10 MHz)

(T_A = - 25 °C to + 85 °C, 2.5V to 5.5 V, 负载电容 = 80 pF)

参数	符号	最小值	典型值	最大值	单位
串口时钟周期	t_{SCK}	500	t_{CPU} 6	700	ns
输出数据启动到时钟上升沿	t_{S1}	300	t_{CPU} 5	-	
时钟上升沿到输入数据有效	t_{S2}	-	-	300	
时钟上升沿后输出数据保持	t_{H1}	t_{CPU} - 50	t_{CPU}	-	
时钟上升沿后输入数据保持	t_{H2}	0	-	-	
串口高低电平宽度	t_{HIGH}, t_{LOW}	200	t_{CPU} 3	400	

注释:

1. 所有时间都是以纳秒为单位，假定CPU时钟频率为10-MHz 。
2. tCPU为CPU 时钟周期。

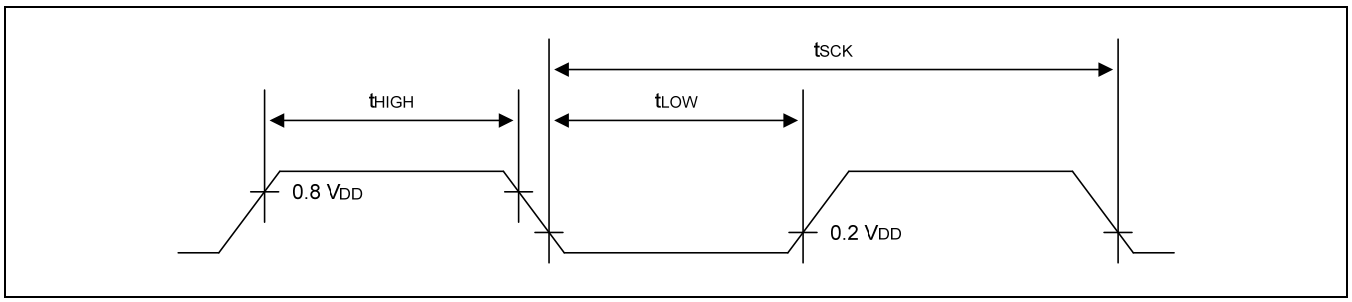


图 20-7 UART时序特性波形

表 20-12 A/D 转换电气特性

(T_A = - 25 °C to + 85 °C, 2.5V to 5.5 V, Avref = V_{DD}, V_{SS} = 0V)

参数	符号	条件	最小值	典型值	最大值	单位
精度		-	-	10	-	bit
总误差		V_{DD} = 5.12 V	-	-	± 3	LSB
积分线性误差	ILE	CPU clock = 10 MHz AV_{REF} = 5.12 V	-	-	± 2	-
微分线性误差	DLE	AV_{SS} = 0 V	-	-	± 1	-
最高点偏离误差	EOT		-	± 1	± 3	-
最低点偏离误差	EOB		-	± 0.5	± 2	-
转换时间 (1)	t_{CON}	10-bit 转换 50 x 4/f_{OSC} (3), f_{OSC} = 10 MHz	20	-	-	μs
模拟输入电压	V_{IAN}	-	AV_{SS}	-	AV_{REF}	V
模拟输入阻抗	R_{AN}	-	2	1000	-	M
模拟基准电压	AV_{REF}	-	2.5	-	V_{DD}	V
模拟地	AV_{SS}	-	V_{SS}	-	V_{SS} + 0.3	-
模拟输入电流	I_{ADIN}	AV_{REF} = V_{DD} = 5 V 转换时间 = 20 μs	-	-	10	μA
模拟块电流 (2)	I_{ADC}	AV_{REF} = V_{DD} = 5 V 转换时间= 20 μs	-	1	3	mA
		AV_{REF} = V_{DD} = 3 V 转换时间= 20 μs		0.5	1.5	
		AV_{REF} = V_{DD} = 5 V 省电模式		100	500	

注释:

1. "转换时间" 是从转换开始直到结束所用时间。
2. I_{ADC}是A/D转换时的工作电流。
3. f_{OSC}是主振荡器时钟频率。
4. AVref 必须链接到VDD。

表 20-13 LVR (低电压复位) 电路特性

(T_A = 25 °C)

参数	符号	条件	最小值	典型值	最大值	单位
LVR 电平	V_{LVR}	LVR由 smart option开启 T_A = 25 °C	2.5	2.8	3.1	V

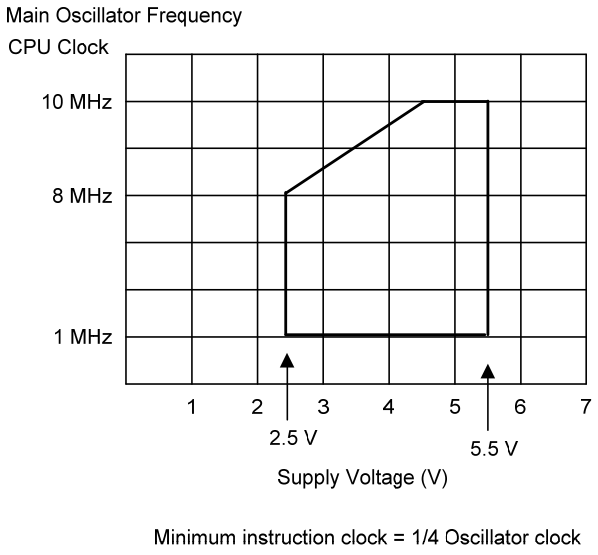


图 20-8 工作电压范围

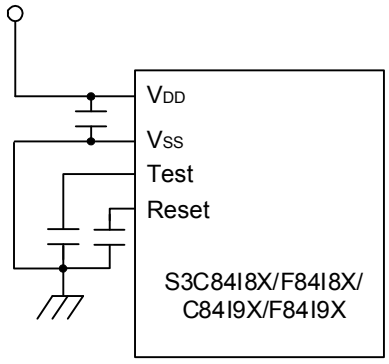


图 20-9 改进 EFT 特性的电路图

21 械尺寸

21.1 概述

S3F84H5 目前可用的封装有 30-SDIP-400, 32-SOP-450A ， 32-SDIP-400和28-SOP-375 。
封装尺寸见 *图 21-1* 和 *图 21-2*。

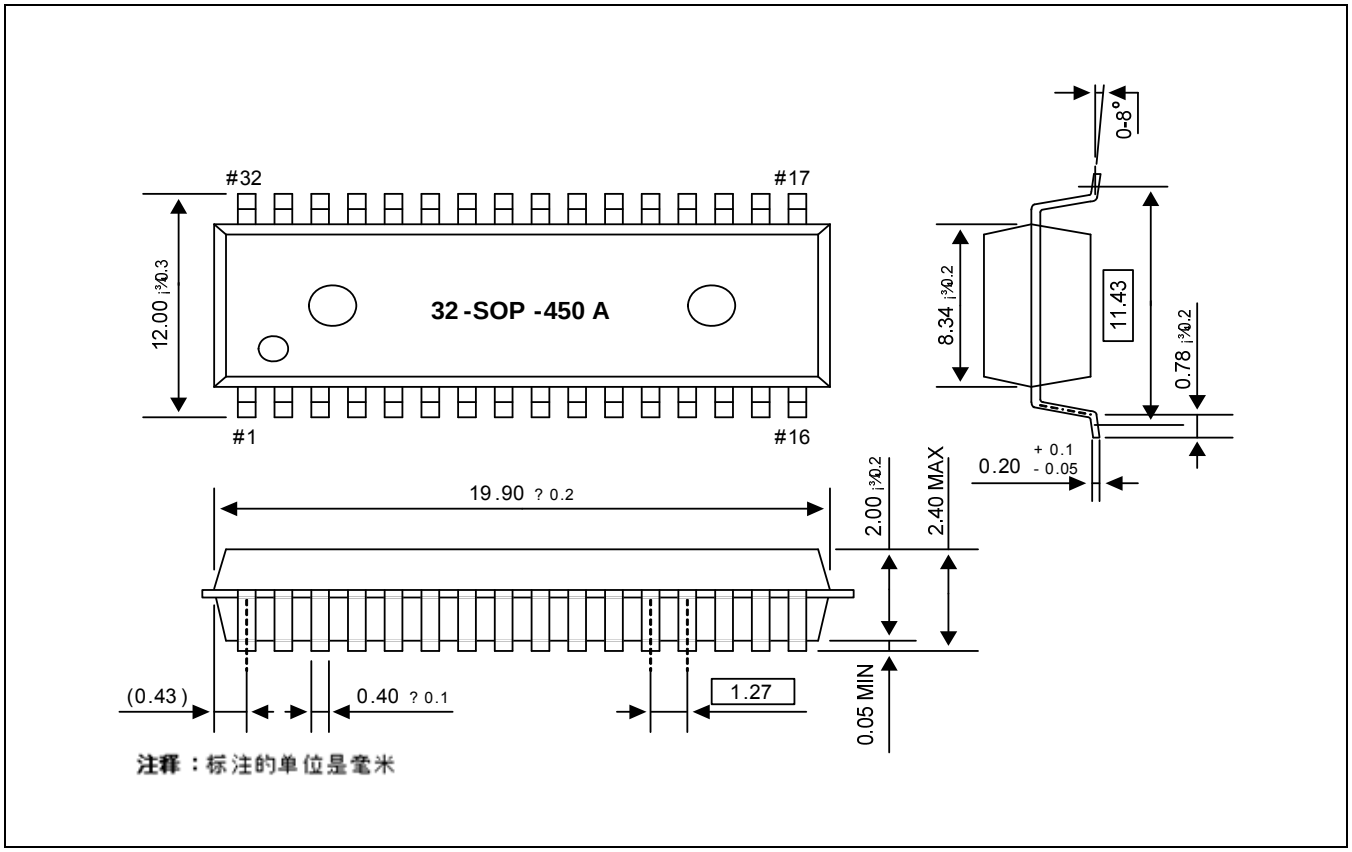


图 21-1 32-SOP-450A 封装尺寸

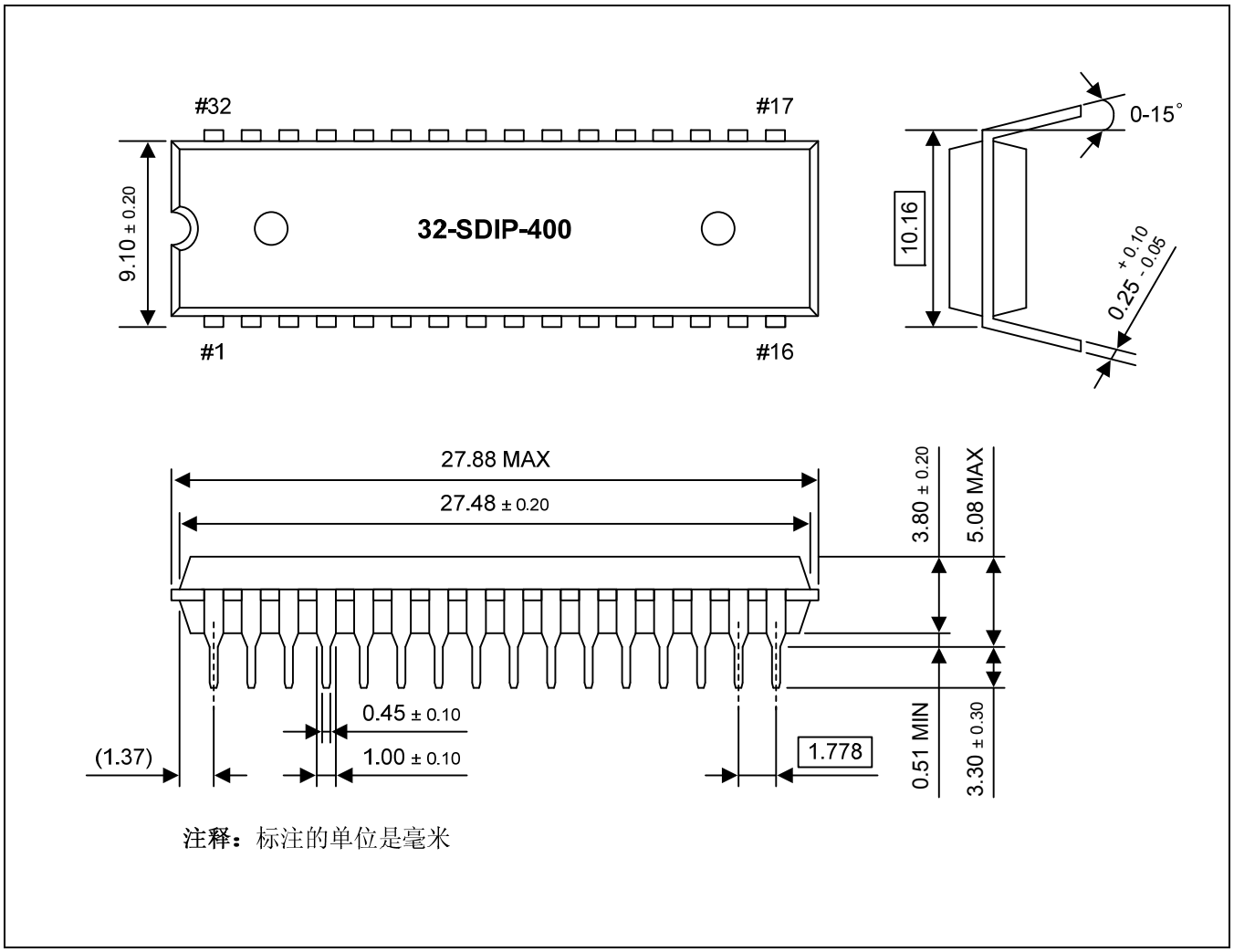


图 21-2 32-SDIP-400 封装尺寸

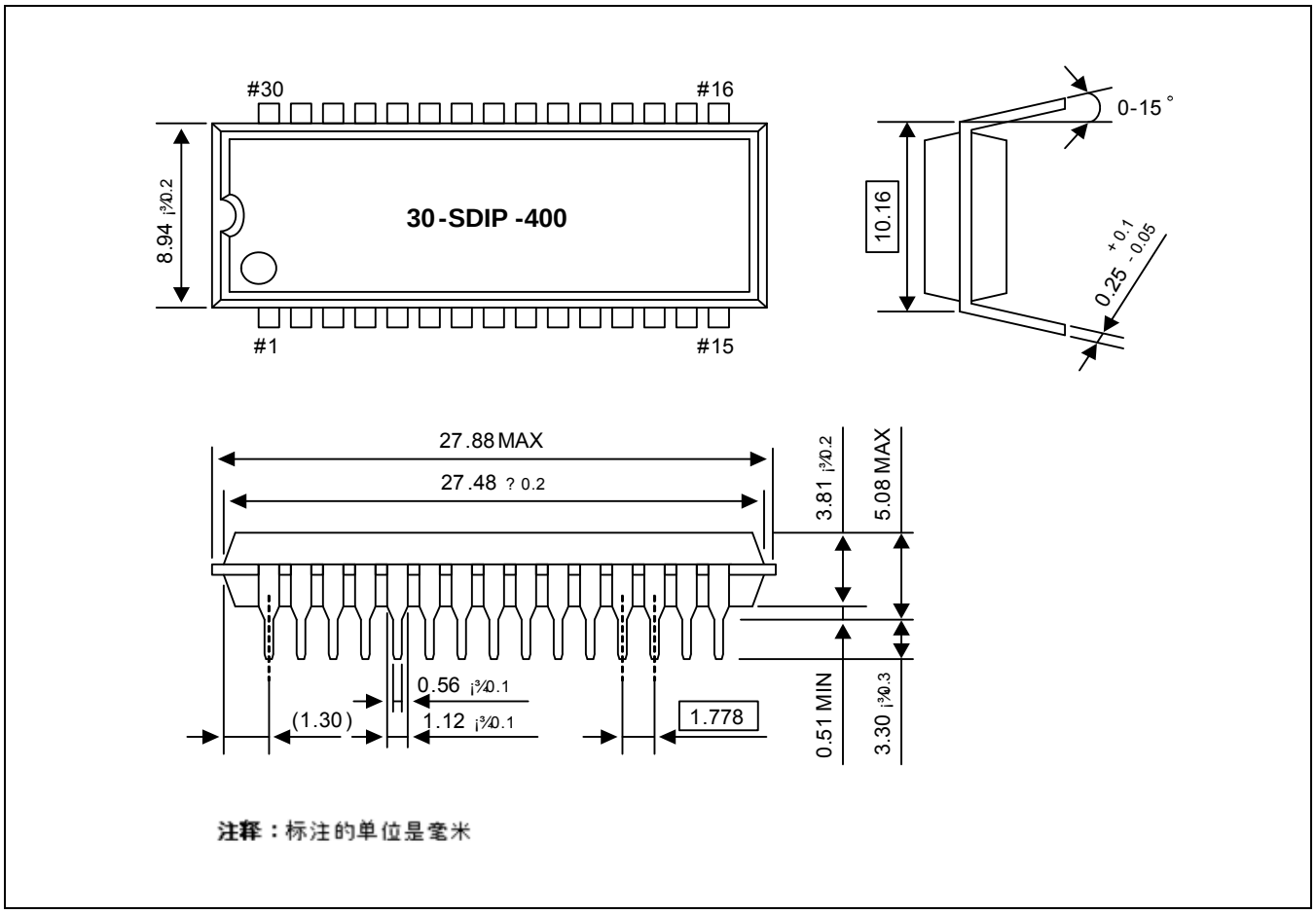


图 21-3 30-SDIP 封装尺寸

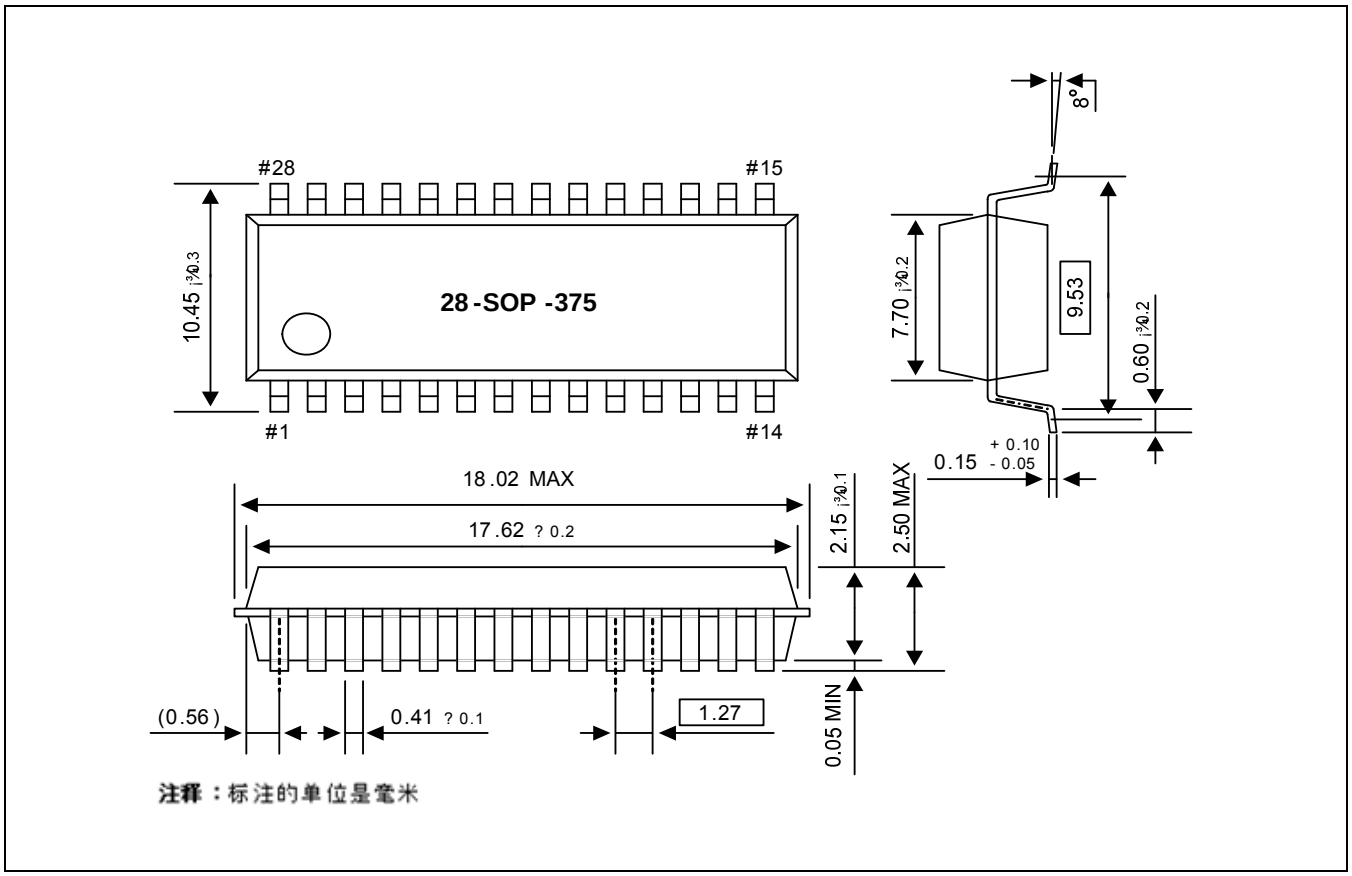


图 21-4 28-SOP-375 封装尺寸

22 开发工具

22.1 概述

三星提供了一套强大易用的开发工具，该开发工具由一个主系统，一套调试工具和相应的支持软件组成。所谓主系统，即任何一台使用 Win95/98/2000/XP 操作系统的标准电脑。成熟的调试工具包含了硬件和软件：一个强大的电路仿真器，OPENice-i500 或SK-1200，支持所有的 S3F7-, S3F9- 和 S3F8- MCU 家族。 三星同时可提供各种支持仿真器的软件，包括调试器、汇编编译器和仿真器安装设置程序。

22.1.1 SHINE

三星在线仿真器的主接口 SHINE 是 SMDS2+ 的多窗口调试器。SHINE 提供下拉和弹出窗口，支持鼠标和热键操作、上下文相关超链接帮助。它有一个多窗口的高级用户接口便于使用。每个窗口可以随意改变大小、移动、滚动、高亮、添加或删除。

22.1.2 SASM

SASM 是适用于三星 S3C8- 系列微控制器的可重载汇编器。 SASM 能够将含有汇编语言的源文件编译成相应的源代码、目标文件和注释。SASM 支持宏定义和条件编译，运行在 MS-DOS 操作系统下。 由于它只生成可重载的目标代码，用户必须把这些目标代码链接起来。目标文件可以跟其它目标文件一起链接，并被分配到存储单元中。SASM 需要源代码文件和一个辅助的设备描述文件（device_name.reg）。

22.1.3 SAMA 汇编编译器

SAMA是一个通用的汇编编译器，标准十六进制格式的目标文件。汇编程序代码包括 ROM 数据需要的目标代码，和 SMDS 需要的配置数据编译程序的时候，SAMA 需要源代码文件和一个辅助的器件描述文件(device_name.def）。

22.1.4 HEX2 ROM

HEX2ROM 可将汇编器生成的 HEX 文件转换为 ROM 代码。Mask ROM 的微控制器需要 ROM 代码来实现架构。 当通过 HEX2ROM 生成ROM 代码(.OBJ 文件)，在 ROM 中未使用的区域就会被赋值 "FF"，直至填满目标设备上的 ROM 空间。

22.1.5 目标板

S3C8/S3F8- 系列 MCU 调试时需要目标板，该板上提供了所有需要的目标系统连线和调试接口。TB84H5 是为开发 S3C84H5/F84H5 专用的目标板。

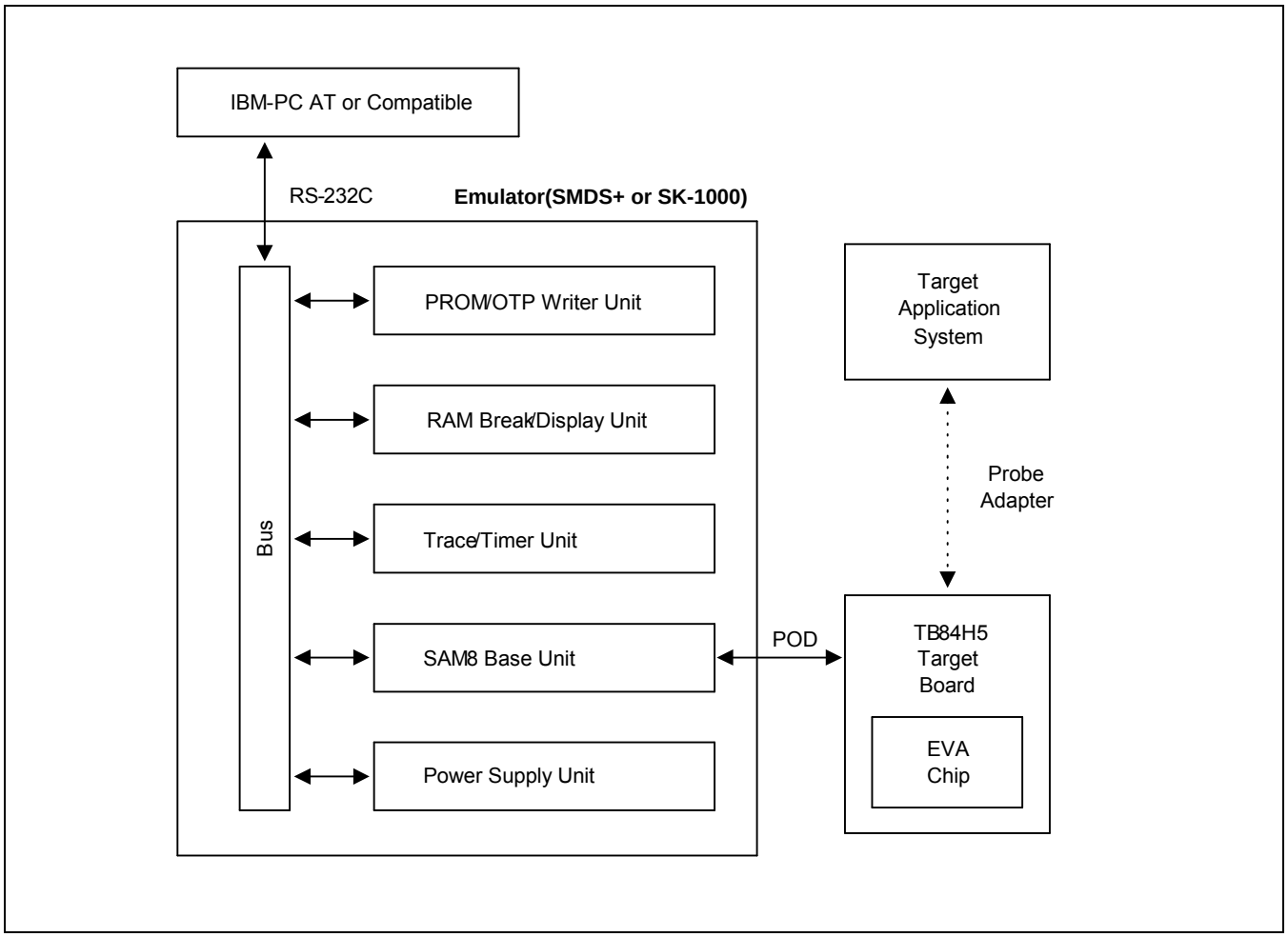


图 22-1 SMDS+ 或者 SK-1000 开发系统配置

22.1.6 TB84H5 目标板

TB84H5是 S3C84H5 和 S3F84H5 MCU 的专用目标板。
它支持SMDS2+ 和 SK-1000 开发系统(在线仿真)。TB84H5 目标板配置如*图 22-2*。

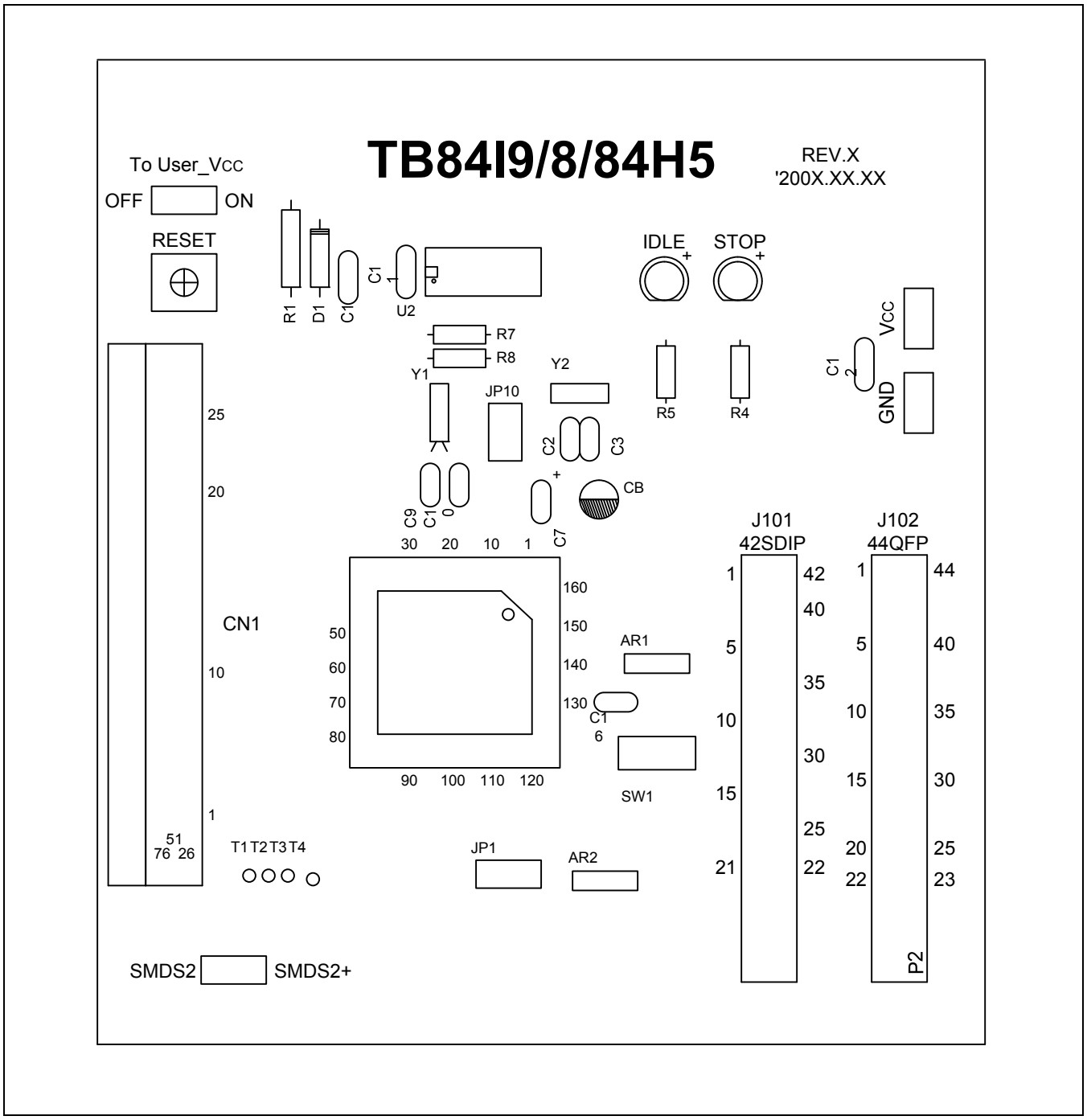


图 22-2 S3F84I9/ S3F84I8/S3F84H5 目标板配置

表 22-1 TB84H5 电源选择设定

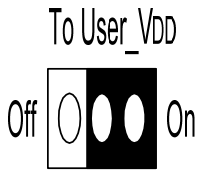
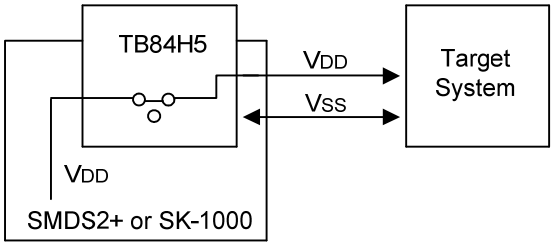
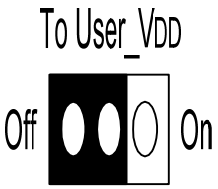
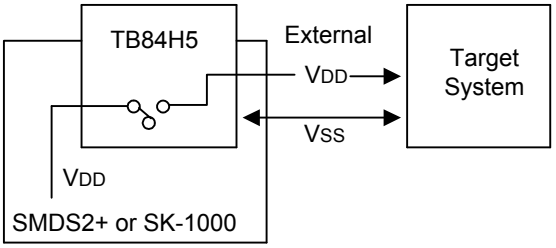
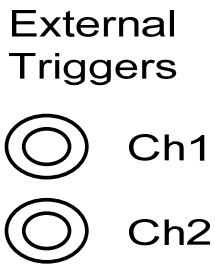
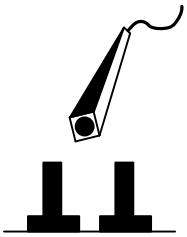
"To User_Vcc" 设定	工作模式	注释
		SMDS2+ 或 SK-1000 提供 V_{CC} 给目标板（评估芯片 evaluation chip）和目标系统。
		SMDS2+ or SK-1000 只提供 V_{CC} 给目标板（评估芯片 evaluation chip）。 目标系统必需有自己的电源。

表 22-2 用测试针作为外部触发源的输入路径

目标板部分	注释
	 可以连接外部触发源到2个外部触发路径中的一路 (CH1 or CH2) 来实现 SMDS2+ 断点和跟踪功能。

22.1.7 IDLE LED

当评估芯片（S3E84H0）在 IDLE 模式时 LED 亮。

22.1.8 STOP LED

当评估芯片（S3E84H0）在 STOP 模式时 LED 亮。

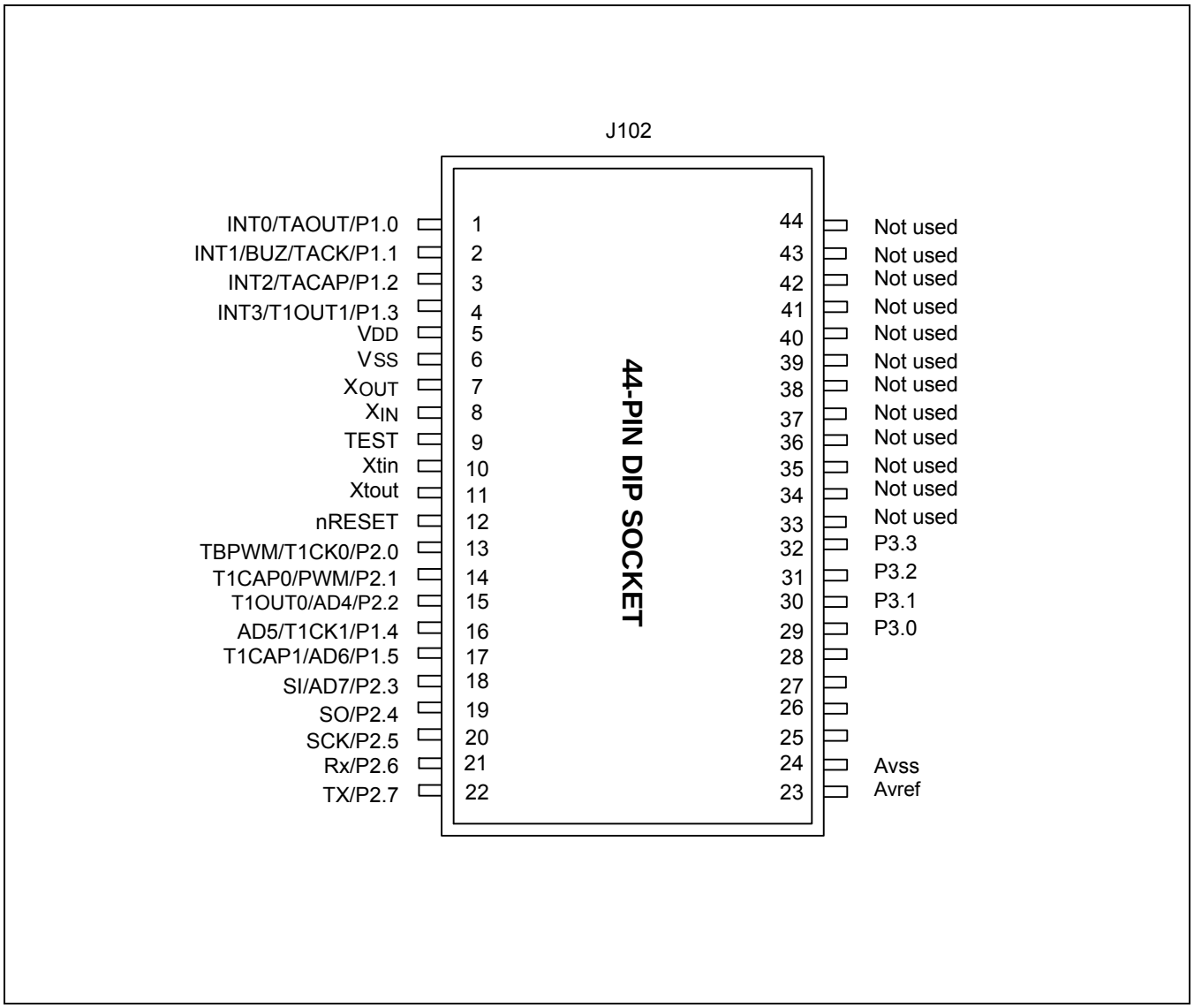


图 22-3 TB84H5的44 脚接口

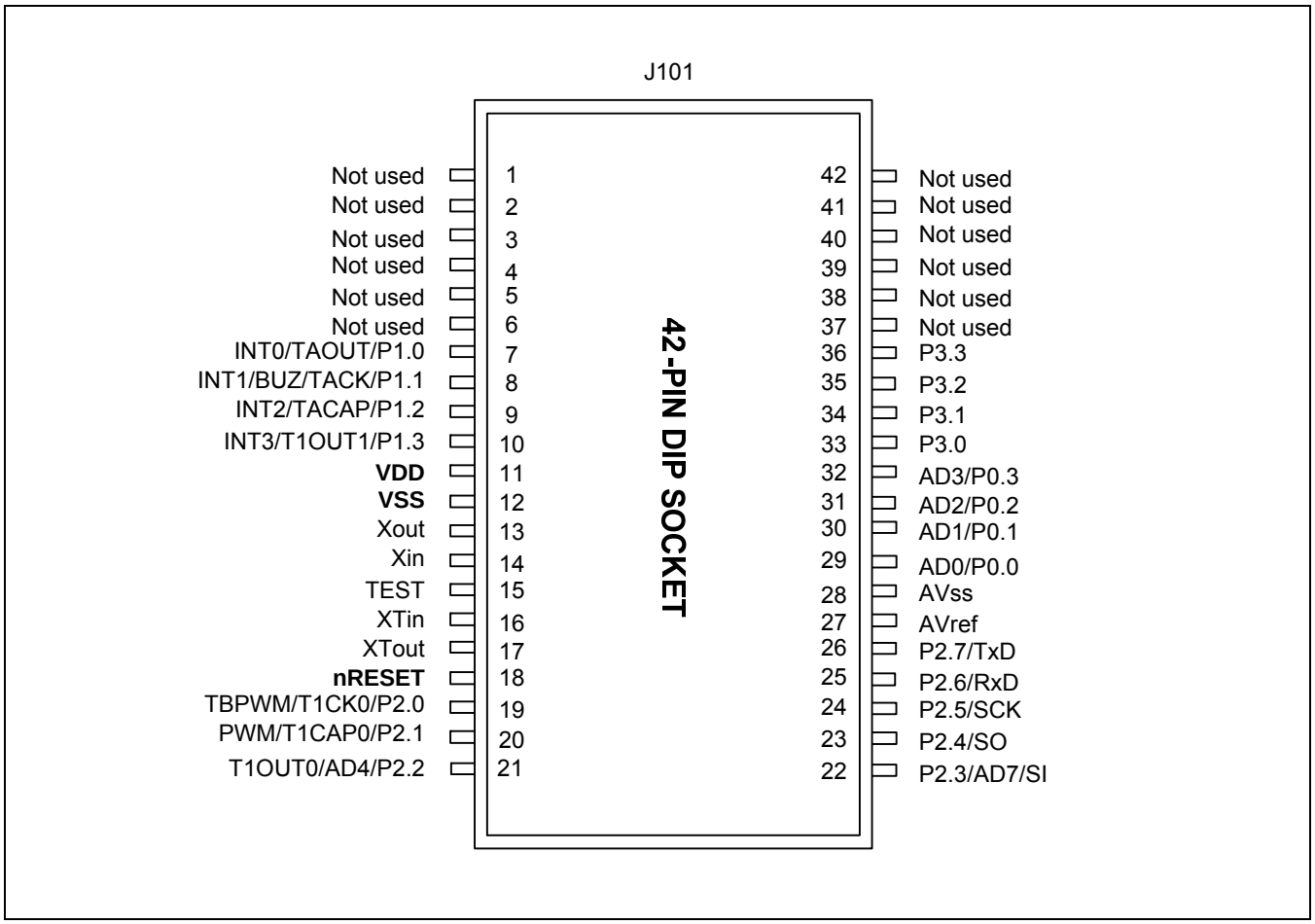


图 22-4 TB84H5的42 脚接口

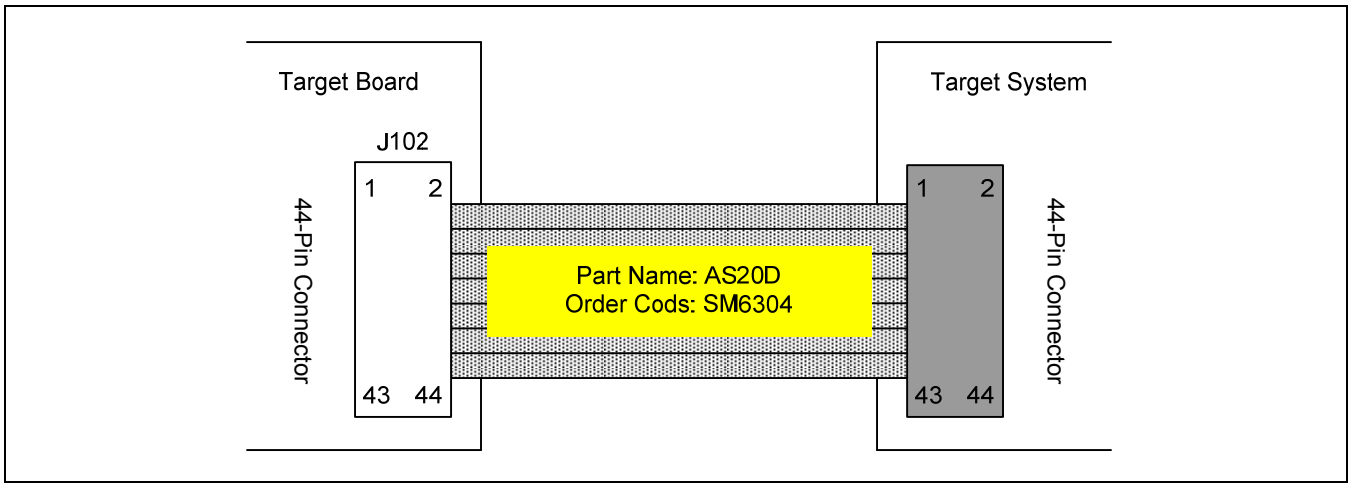


图 22-5 TB84H5 44 脚封装的适配器数据线