第6章 中断与定时技术

微处理器与外设间数据传送方式

查询方式	中断方式	DMA方式
查询程序周期性地读取端口或部件状态，根据其状态，进行数据、命令传送。	外设发中断请求；处理器响应请求，暂停当前操作，转而处理新的操作。	由DMA控制器协助存储器和I/O部件之间、存储器和存储器之间直接进行数据传送
效率低 实时性差	实时性能好 占用处理器资源	占用总线周期少 响应速度快 快速批量传送

6.1 中断概述

- 中断是为快速处理随机事件而设置的一种响应机制
- 中断是CPU在执行程序的过程中，由于计算机系统内、外的某种原因（事件），而必须中止原程序的执行，转去执行相应的事件处理程序，待处理结束之后，再返回继续执行被中止的原程序的过程。
- 中断系统是微处理器系统的核心模块
- 中断三要素：事件、响应机制、处理程序
- 中断基本概念：中断向量、中断优先级、中断屏蔽

中断的作用

- 解决快速的 CPU与慢速的外设之间的同步问题；
  - CPU可以启动多个外设同时工作，每当外设有事件发生，则请求CPU执行中断服务，中断处理完之后，CPU 恢复执行主程序，大大地提高了 CPU 的效率。
- 实现对外部事件的实时处理；
  - 嵌入式系统中，各种参数、信息随时间变化，系统需要对这些外界变化做出及时响应，常通过中断请求实现。
- 系统故障的应急处理；
  - 针对难以预料的情况或故障，如掉电、存储出错、运算溢出等，可通过中断系统由故障源向CPU发出中断请求，由相应的故障处理程序进行处理 。

中断响应的一般过程

- (1)每条指令结束后，系统都自动检测中断请求信号，如果有中断请求，且CPU处于开中断状态下，则响应中断。
- (2)保护现场，在保护现场前，一般要关中断，以防止现场被破坏。保护现场是用堆栈指令将原程序中用到的寄存器推入堆栈。
- (3)中断服务，即为相应的中断源服务。
- (4)恢复现场，用堆栈指令将保护在堆栈中的数据弹出来，在恢复现场前要关中断，以防止现场被破坏。在恢复现场后应及时开中断。
- (5)返回，此时CPU将压入到堆栈的断点地址弹回到程序计数器，从而使CPU继续执行刚才被中断的程序。

6.1.1 中断向量

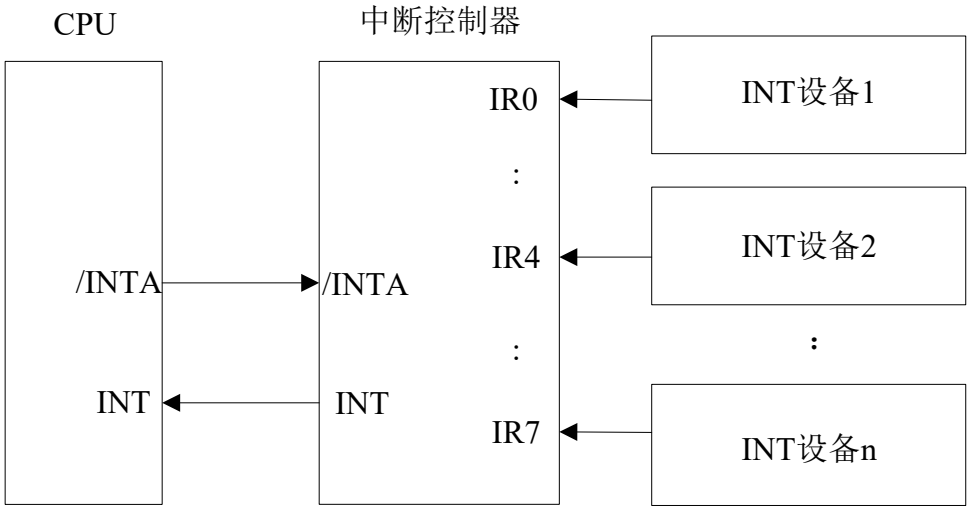
- 中断服务程序是为快速处理事件而编制的
- 不同的中断服务程序对应不同的事件
- 中断向量为不同的中断服务程序提供入口地址
- 中断服务程序的入口地址即为中断向量
- 所有事件的入口地址构成中断向量表
- 微处理器根据中断向量表启动对应的中断服务程序

形成中断服务程序入口地址的机制不同，中断向量可分成固定中断向量和可变中断向量：

- 固定中断向量
  - 各中断服务入口地址是固定不变的，微处理器设计时就已经确定，系统设计者不能改变；
  - 固定中断向量中断响应速度快
- 可变中断向量
  - 中断服务程序的入口地址不是固定不变的，系统设计者可以根据自己的需要对中断控制器进行设置。
  - 优点：设计比较灵活，用户可根据需要设定中断向量表在主存中的位置 。缺点：中断响应速度较慢。

6.1.2 中断优先级

●多数微处理器系统中都有多个中断源，为使系统能及时响应并处理发生的所有中断，系统根据引起中断事件的重要性和紧迫程度，将中断源分为若干个级别，称作中断优先级。



硬件方式中断原理图

中断优先级仲裁

中断优先级仲裁有两层含义：

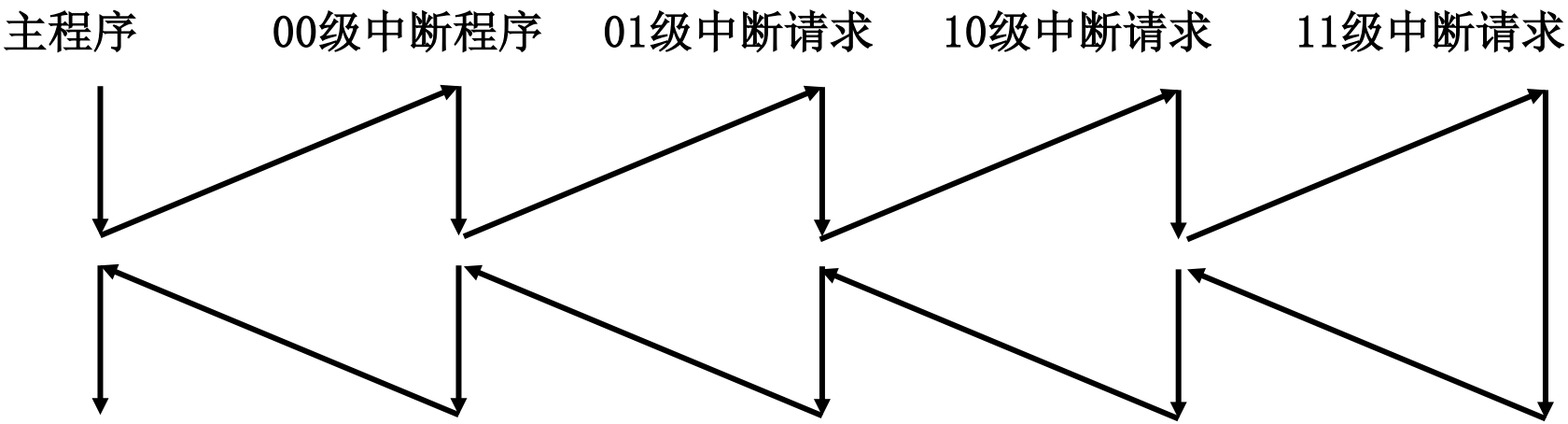
a) 2个及2个以上的中断源同时提出中断请求，微处理器先响应哪个中断源，后响应哪个中断源

b) 1个中断源提出中断请求并得到响应后，又有1个中断源提出中断请求，新的中断源能否中断前一个中断的中断服务程序

即：中断并发和中断嵌套问题

中断优先级及中断嵌套

- 在使用中断嵌套时应特别注意**堆栈深度**，堆栈深度不够时，将导致**中断返回错误**，不能返回到原来的断点。



6.1.3 中断屏蔽

- 为了更灵活地运用中断，采用中断屏蔽技术，可以让部分中断不起作用。
- 一个中断源对应一个中断屏蔽位。
- 中断控制器通常将中断屏蔽位集中在一起，构成中断屏蔽寄存器。
- 约定屏蔽位为“1”时，对应的中断源被屏蔽。
- 部分中断，比如电源掉电，是不可屏蔽的，称为非屏蔽中断（NMI，NonMaskable Interrupt），因此中断可分为：可屏蔽中断和不可屏蔽中断

6.2 S3C2410中断系统

6.2.1 概述

- ARM920T内核支持复位、未定义指令、软中断、指令预取中止、数据访问中止、IRQ和FIQ 7种异常中断
- 每种异常中断对应不同的处理器模式和对应的异常中断向量（中断服务程序入口地址）
- ARM920T内核采用固定中断向量

表6.1 异常中断类型

优先级	异常中断类型	对应的模式	中断向量
1	复位	管理模式(svc)	0x00000000
2	数据访问中止	中止模式(abt)	0x00000010
3	快速中断请求	快速中断模式(FIQ)	0x0000001C
4	外部终端请求	外部中断模式(IRQ)	0x00000018
5	指令预取中止	中止模式(abt)	0x0000000C
6	未定义指令	未定义指令中止模式(und)	0x00000004
7	软件中断(SWI)	管理模式(svc)	0x00000008
未使用	未使用	未使用	0X00000014

6.2.1 概述（续）

- ARM920T内核的异常中断控制逻辑处理复位、数据访问中止、指令预取中止、未定义指令、软件中断等几个异常中断
- ARM920T内核提供IRQ中断和快速中断FIQ，由外围设备使用
- FIQ中断通常用于进行大批量的复制、数据传送
- IRQ中断用于常规中断服务，可以被FIQ中断中断

6.2.1 概述（续）

- S3C2410片内集成了中断控制器，通过IRQ中断和FIQ中断提供56个中断源
- 中断控制器为S3C2410芯片内部所有设备都分配有中断请求信号，例如DMA控制器、UART、IIC等等
- 中断控制器另外提供了24个外部中断输入引脚供用户使用
- 虽然56个中断源可以通过设置处理器的寄存器设置，映射到IRQ中断或者FIQ中断，但为方便操作系统对外部中断的统一处理，一般会把所有中断都映射到IRQ中断上，而保留FIQ备用

表6.2 S3C2410中断控制器支持的中断源

中断源	描述	仲裁器
INT_ADC	ADC结束中断、触摸屏中断(INT_ADC、INT_TC)	ARB5
INT_RTC	RTC闹钟中断	ARB5
INT_SPI1	SPI1中断	ARB5
INT_UART0	UART0中断（ERR、RXD、TXD）	ARB5
INT_IIC	I^2C中断	ARB4
INT_USBH	USB主设备中断	ARB4
INT_USBD	USB从设备中断	ARB4
保留	保留	ARB4
INT_UART1	UART1中断（ ERR、RXD、TXD ）	ARB4
INT_SPI0	SPI0中断	ARB4

表6.2 S3C2410中断控制器支持的中断源

中断源	描述	仲裁器
INT_SDI	SDI中断	ARB3
INT_DMA3	DMA通道3中断	ARB3
INT_DMA2	DMA通道2中断	ARB3
INT_DMA1	DMA通道1中断	ARB3
INT_DMA0	DMA通道0中断	ARB3
INT_LCD	LCD中断 (INT_FrSyn、INT_FiCnt)	ARB3
INT_UART2	UART2中断（ ERR、RXD、TXD ）	ARB2
INT_TIMER4	定时器4中断	ARB2
INT_TIMER3	定时器3中断	ARB2
INT_TIMER2	定时器2中断	ARB2
INT_TIMER1	定时器1中断	ARB2
INT_TIMER0	定时器0中断	ARB2

表6.2 S3C2410中断控制器支持的中断源

中断源	描述	仲裁器
INT_WDT	看门狗定时器中断	ARB1
INT_TICK	RTC定时器时间片中断	ARB1
nBATT_FLT	电池故障中断	ARB1
保留	保留	ARB1
EINT8_23	外部中断8～23	ARB1
EINT4_7	外部中断4～7	ARB1
EINT3	外部中断3	ARB0
EINT2	外部中断2	ARB0
EINTI	外部中断1	ARB0
EINT0	外部中断0	ARB0

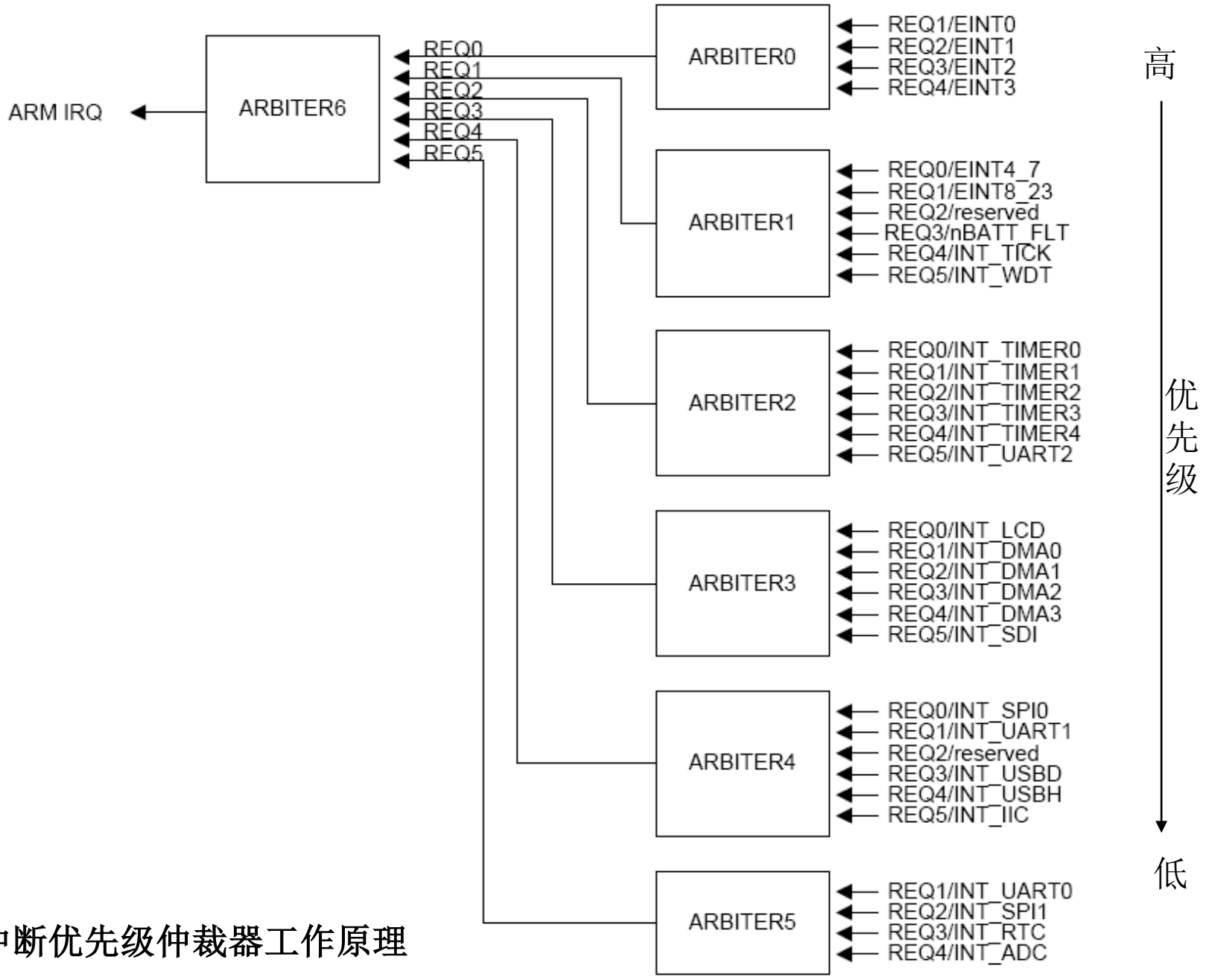
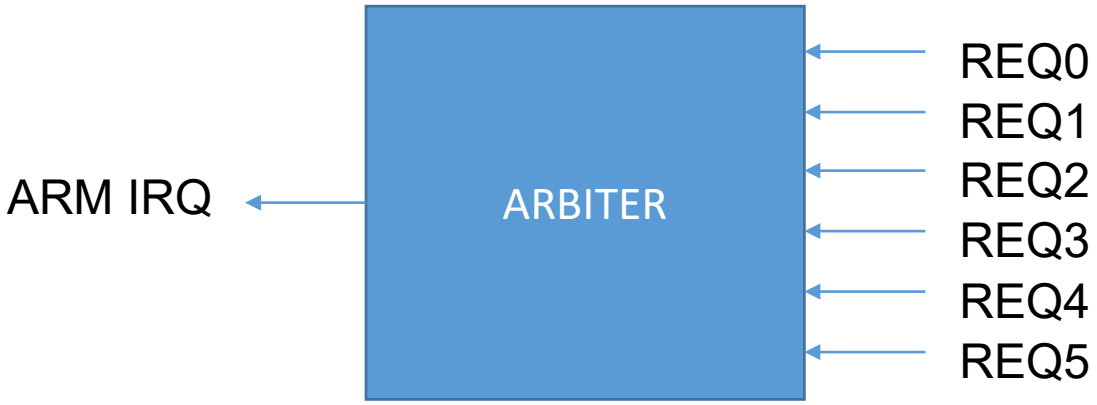


图6.2 中断优先级仲裁器工作原理

中断优先级裁决器



- 中断优先级和工作模式由仲裁器的PRIORITY寄存器设置，优先级ARB_SEL的2位值确定，工作模式由ARB_MODE位确定
- REQ0总是具有最高优先权，REQ5总是具有最低优先权
- REQ1~REQ4的优先级可以通过改变PRIORITY寄存器的ARB_SEL的2位值，循环改变

中断优先级裁决器

ABR_SEL	优先顺序（由高到低）
00	REQ0 REQ1 REQ2 REQ3 REQ4 REQ5
01	REQ0 REQ2 REQ3 REQ4 REQ1 REQ5
10	REQ0 REQ3 REQ4 REQ1 REQ2 REQ5
11	REQ0 REQ4 REQ1 REQ2 REQ3 REQ5

工作模式位(ARB_MODE)设置优先顺序的变化模式：

- 若ARB_MODE位设置为“0”，则由ARB_SEL确定的优先顺序固定不变；
- 若ARB_MODE位设置为“1”，则除REQ0、REQ5的优先级固定不变以外，由ARB_SEL确定的优先顺序，在中断响应处理后，该中断优先级自动排到最后

6.2.2 中断控制寄存器

- ARM9内核程序状态寄存器(CPSR)的F位和I位是FIQ、IRQ中断屏蔽位。
  - CPSR的F位为1，则CPU屏蔽FIQ中断
  - CPSR的I位为1，则CPU屏蔽IRQ中断
- 因此：在使用FIQ、IRQ中断之前需合理设置CPSR中的F位、I位，仅设置S3C2410的中断控制器CPU可能不会响应中断
- S3C2410中断控制器有8个控制寄存器

2410中断控制器专用寄存器

Register	Address	R/W	Description	Reset Value
SRCPND	0x4A000000	R/W	中断请求寄存器	0x00000000
INTMOD	0x4A000004	R/W	中断模式寄存器	0x00000000
INTMSK	0x4A000008	R/W	中断屏蔽寄存器	0xFFFFFFFF
PRIORITY	0x4A00000C	R/W	中断优先级寄存器	0x7F
INTPND	0x4A000010	R/W	中断挂起寄存器	0x00000000
INTOFFSET	0x4A000014	R	中断偏移寄存器	0x00000000
SUBSRCPND	0x4A000018	R/W	子源挂起寄存器	0x00000000
INTSUBMSK	0x4A00001C	R/W	中断子源屏蔽寄存器	0x7FF

注：主要使用前5个寄存器

1. SRCPND---中断请求寄存器

位号	中断源	位号	中断源	位号	中断源	位号	中断源
31	INT_ADC	23	INT_UART1	15	INT_UART2	7	nBATT_FLT
30	INT_RTC	22	INT_SPI0	14	INT_TIM4	6	保留
29	INT_SPI1	21	INT_SDI	13	INT_TIM3	5	EINT8_23
28	INT_UART0	20	INT_DMA3	12	INT_TIM2	4	EINT4_7
27	INT_IIC	19	INT_DMA2	11	INT_TIM1	3	EINT3
26	INT_USBH	18	INT_DMA1	10	INT_TIM0	2	EINT2
25	INT_USBD	17	INT_DMA0	9	INT_WDT	1	EINT1
24	保留	16	INT_LCD	8	INT_TICK	0	EINT0

该寄存器也就是中断标志寄存器，或者叫中断源登记寄存器，

1：对应中断源有中断请求

0：对应中断源无中断请求

注意：必须在中断处理程序中对其标志位清0，清0方法是向其写1。

2. INTMOD---中断模式寄存器

位号	中断源	位号	中断源	位号	中断源	位号	中断源
31	INT_ADC	23	INT_UART1	15	INT_UART2	7	nBATT_FLT
30	INT_RTC	22	INT_SPI0	14	INT_TIM4	6	保留
29	INT_SPI1	21	INT_SDI	13	INT_TIM3	5	EINT8_23
28	INT_UART0	20	INT_DMA3	12	INT_TIM2	4	EINT4_7
27	INT_IIC	19	INT_DMA2	11	INT_TIM1	3	EINT3
26	INT_USBH	18	INT_DMA1	10	INT_TIM0	2	EINT2
25	INT_USBD	17	INT_DMA0	9	INT_WDT	1	EINT1
24	保留	16	INT_LCD	8	INT_TICK	0	EINT0

该寄存器设置各中断源是FIQ中断还是IRQ中断

1：对应中断源设为FIQ中断模式

0：对应中断源设为IRQ中断模式

3. INTMSK---中断屏蔽寄存器

位号	中断源	位号	中断源	位号	中断源	位号	中断源
31	INT_ADC	23	INT_UART1	15	INT_UART2	7	nBATT_FLT
30	INT_RTC	22	INT_SPI0	14	INT_TIM4	6	保留
29	INT_SPI1	21	INT_SDI	13	INT_TIM3	5	EINT8_23
28	INT_UART0	20	INT_DMA3	12	INT_TIM2	4	EINT4_7
27	INT_IIC	19	INT_DMA2	11	INT_TIM1	3	EINT3
26	INT_USBH	18	INT_DMA1	10	INT_TIM0	2	EINT2
25	INT_USBD	17	INT_DMA0	9	INT_WDT	1	EINT1
24	保留	16	INT_LCD	8	INT_TICK	0	EINT0

1：屏蔽对应中断源

0：开放对应中断源

4. PRIORITY---中断优先级寄存器

位号	含　义	位号	含　义	位号	含　义
31:21	保　留	12:11	ARB_SEL2	4	ARB_MODE4
20:19	ARB_SEL6	10:9	ARB_SEL1	3	ARB_MODE3
18:17	ARB_SEL5	8:7	ARB_SEL0	2	ARB_MODE2
16:15	ARB_SEL4	6	ARB_MODE6	1	ARB_MODE1
14:13	ARB_SEL3	5	ARB_MODE5	0	ARB_MODE0

ARB_SELn---设定仲裁器n的优先级顺序控制位

00：REQ0,1,2,3,4,5　　01：REQ0,2,3,4,1,5

10：REQ0,3,4,1,2,5　　11：REQ0,4,1,2,3,5

ARB_MODEn---设定仲裁器n的优先级循环控制位

0：优先顺序固定不变

1：优先顺序循环(轮换)，每响应一次中断，其优先顺序循环改变一次，但REQ0、REQ5位置不变。

5. INTPND---中断服务(挂起)寄存器

位号	中断源	位号	中断源	位号	中断源	位号	中断源
31	INT_ADC	23	INT_UART1	15	INT_UART2	7	nBATT_FLT
30	INT_RTC	22	INT_SPI0	14	INT_TIM4	6	保留
29	INT_SPI1	21	INT_SDI	13	INT_TIM3	5	EINT8_23
28	INT_UART0	20	INT_DMA3	12	INT_TIM2	4	EINT4_7
27	INT_IIC	19	INT_DMA2	11	INT_TIM1	3	EINT3
26	INT_USBH	18	INT_DMA1	10	INT_TIM0	2	EINT2
25	INT_USBD	17	INT_DMA0	9	INT_WDT	1	EINT1
24	保留	16	INT_LCD	8	INT_TICK	0	EINT0

INTPND用于指示中断请求挂起状态(未处理)。

1：对应的中断源被响应，且正在执行中断服务

0：对应中断源未被响应

注意：必须在中断处理程序中对其服务(挂起)标志位清0，方法为对某位写1便清除为0。在清除SRCPND 中相应位后，要清除该寄存器相应位。

6. INTOFFSET---中断偏移寄存器

中断源	偏移值	中断源	偏移值	中断源	偏移值	中断源	偏移值
INT_ADC	31	INT_UART1	23	INT_UART2	15	nBATT_FLT	7
INT_RTC	30	INT_SPI0	22	INT_TIM4	14	保留	6
INT_SPI1	29	INT_SDI	21	INT_TIM3	13	EINT8_23	5
INT_UART0	28	INT_DMA3	20	INT_TIM2	12	EINT4_7	4
INT_IIC	27	INT_DMA2	19	INT_TIM1	11	EINT3	3
INT_USBH	26	INT_DMA1	18	INT_TIM0	10	EINT2	2
INT_USBD	25	INT_DMA0	17	INT_WDT	9	EINT1	1
保留	24	INT_LCD	16	INT_TICK	8	EINT0	0

该寄存器的偏移值指示在INTPND中显示的中断源。

1：对应的中断源，在INTPND中被置位

说明：当在中断服务程序中对SRCPND、INTPND中的标志位清0时，该寄存器的对应位将自动清0。

7. SUBSRCPND---子中断请求寄存器

部分中断源复用一个中断请求(如ADC有ADC EOC、触摸屏中断等)。S3C2410提供了SUBSRCPND和INTSUBMSK，以便更详细地区分这些中断请求。

位号	中断源	位号	中断源	位号	中断源
31:11	保 留	7	INT_TXD2	3	INT_RXD1
10	INT_ADC	6	INT_RXD2	2	INT_ERR0
9	INT_TC	5	INT_ERR1	1	INT_TXD0
8	INT_ERR2	4	INT_TXD1	0	INT_RXD0

对有多个中断源的外设，显示其具体的中断请求。

1：对应的子中断源有请求

0：对应的子中断源无请求

注意：在中断服务程序中，需要对其置1的标志位清0。

8. INTSUBMSK---子中断源屏蔽寄存器

位号	中断源	位号	中断源	位号	中断源
31:11	保　留	7	INT_TXD2	3	INT_RXD1
10	INT_ADC	6	INT_RXD2	2	INT_ERR0
9	INT_TC	5	INT_ERR1	1	INT_TXD0
8	INT_ERR2	4	INT_TXD1	0	INT_RXD0

对有多个中断源的外设，对具体的中断源进行屏蔽

1：屏蔽对应的子中断源

0：开放对应的子中断源

6.2.3 其他与中断有关的控制寄存器

1. 外中断触发方式控制寄存器

（1）EXTINT0---外中断触发方式控制寄存器0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
X	EINT7			X	EINT6			X	EINT5			X	EINT4		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	EINT3			X	EINT2			X	EINT1			X	EINT0		

EINT0~7---中断请求信号触发方式选择

000：低电平触发　　001：高电平触发

01x：下降沿触发　　10x：上升沿触发

11x：双边沿触发

第3、7、11、15、19、23、27、31位---保留

（2）EXTINT1---外中断触发方式控制寄存器1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
X	EINT15			X	EINT14			X	EINT13			X	EINT12		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	EINT11			X	EINT10			X	EINT9			X	EINT8		

EINT8~15---中断请求信号触发方式选择

000：低电平触发　　001：高电平触发

01x：下降沿触发　　10x：上升沿触发

11x：双边沿触发

第3、7、11、15、19、23、27、31位---保留

（3）EXTINT2---外中断控制寄存器2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
F23	EINT23			F22	EINT22			F21	EINT21			F20	EINT20		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F19	EINT19			F18	EINT18			F17	EINT17			F16	EINT16		

EINT16~23---外中断请求信号触发方式选择

000：低电平触发　　001：高电平触发

01x：下降沿触发　　10x：上升沿触发

11x：双边沿触发

第3、7、11、15、19、23、27、31位---为FILTEN

各引脚滤波控制位

0：禁止滤波　　1：使能滤波

2. 外中断滤波控制寄存器

外中断滤波控制寄存器主要设置各个外中断源的滤波器设置，包括EINTFLT0~EINTFLT3

Register	Address	R/W	Description	Reset Value
EINTFLT0	0x56000094	R/W	保留	-
EINTFLT1	0x56000098	R/W	保留	-
EINTFLT2	0x5600009C	R/W	外中断滤波控制寄存器2	0x0
EINTFLT3	0x560000A0	R/W	外中断滤波控制寄存器3	0x0

（1）EINTFLT2---外中断滤波控制寄存器2

31	30	……	24	23	22		16
FLTCLK19	EINTFLT19			FLTCLK18	EINTFLT18		

15	14	……	8	7	6	……	0
FLTCLK17	EINTFLT17			FLTCLK16	EINTFLT16		

FLTCLK16~19---外中断16~19滤波器时钟选择

0：PCLK

1：外部/振荡时钟（由OM[3:2]引脚选择）

EINTFLT16~19---外中断16~19滤波器宽度（频带宽度）

（2）EINTFLT3---外中断滤波控制寄存器3

31	30	……	24	23	22		16
FLTCLK23	EINTFLT23			FLTCLK22	EINTFLT22		

15	14	……	8	7	6	……	0
FLTCLK21	EINTFLT21			FLTCLK20	EINTFLT20		

FLTCLK20~23---外中断20~23滤波器时钟选择

0：PCLK

1：外部/振荡时钟(由OM[3:2]引脚选择)

EINTFLT20~23---外中断20~23滤波器宽度（频带宽度）

3. 外中断屏蔽、标志寄存器

Register	Address	R/W	Description	Reset Value
EINTMASK	0x560000A4	R/W	外中断屏蔽寄存器	0x00FFFFF0
EINTPEND	0x560000A8	R/W	外中断标志寄存器	0x0

（1）外中断屏蔽寄存器(EINTMASK)

位号	含 义	位号	含 义	位号	含 义
23	EINT23	15	EINT15	7	EINT7
22	EINT22	14	EINT14	6	EINT6
21	EINT21	13	EINT13	5	EINT5
20	EINT20	12	EINT12	4	EINT4
19	EINT19	11	EINT11	3	保留
18	EINT18	10	EINT10	2	保留
17	EINT17	9	EINT9	1	保留
16	EINT16	8	EINT8	0	保留

各位：

0：允许中断　　1：禁止中断

注意： EINT0--- EINT3不能在此被屏蔽，在SRCPND中屏蔽。

（2）外中断标志寄存器(EINTPEND)

位号	含 义	位号	含 义	位号	含 义
23	EINT23	15	EINT15	7	EINT7
22	EINT22	14	EINT14	6	EINT6
21	EINT21	13	EINT13	5	EINT5
20	EINT20	12	EINT12	4	EINT4
19	EINT19	11	EINT11	3	保留
18	EINT18	10	EINT10	2	保留
17	EINT17	9	EINT9	1	保留
16	EINT16	8	EINT8	0	保留

各位：

0：无中断请求　　1：有中断请求

注意：对某位写1，则清除相应标志，即清为0.

4. 外中断状态寄存器

Register	Address	R/W	Description	Reset Value
GSTATUS0	0x560000AC	R	外部引脚状态寄存器	不确定
GSTATUS1	0x560000B0	R	芯片ID(标识)寄存器	0x32410000
GSTATUS2	0x560000B4	R/W	复位状态寄存器	0x1
GSTATUS3	0x560000B8	R/W	信息保存寄存器	0x0
GSTATUS4	0x560000C0	R/W	信息保存寄存器	0x0

GSTATUS3、4：

复位时被清0，其它情况下其数据不变。

用户可以用于保存数据。

（1）GSTATUS0---外部引脚状态寄存器

31	……	4	3	2	1	0
保 留			nWAIT	NCON	RnB	nBATT_FLT

nWAIT---引脚nWAIT状态

NCON---引脚NCON状态

(读出:0- 3 step addressing, 1-4 step addressing)

RnB---引脚R/nB状态

nBATT_FLT---引脚nBATT_FLT状态

注意：各位的数值0、1，随着对应引脚变化。

（2）GSTATUS2---复位状态寄存器

31	……	3	2	1	0
保 留			WDTRST	OFFRST	PWRST

WDTRST---上电复位控制状态

1：出现了上电复位

对该位写1，则将该位清0

OFFRST---掉电模式复位状态。

1：系统出现了从掉电模式唤醒复位

对该位写1，则将该位清0

PWRST---看门狗复位状态

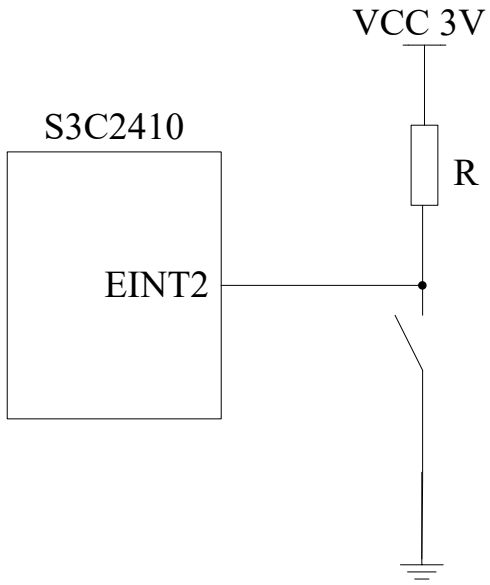
1：系统出现了看门狗定时器复位

对该位写1，则将该位清0

6.2.4 中断举例

目标：使用外部中断EINT2响应按键事件

原理分析：电路图如左图所示，在按键没有被按下时，EINT2管脚一直是高电平（逻辑“1”），当按键被按下时，EINT2管脚电平被拉低（逻辑“0”），产生中断。



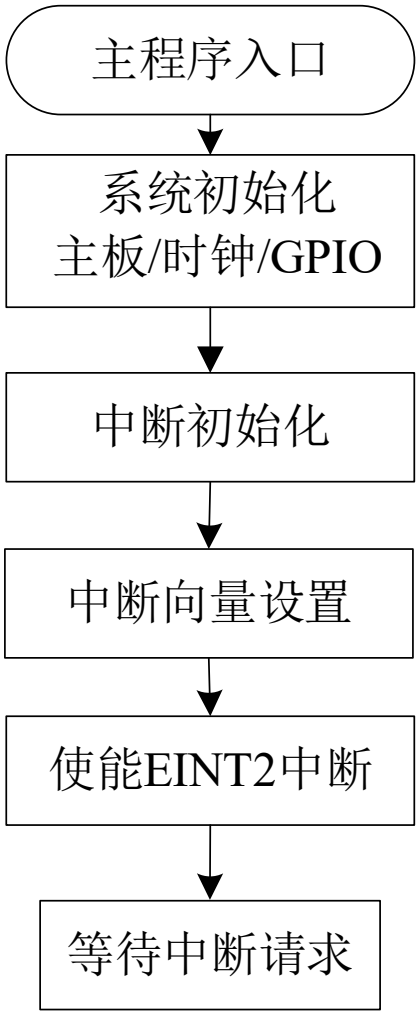
程序设计：

1. 系统初始化（实验箱、时钟、端口、中断等）；
2. 中断向量设置（跳转到中断服务程序ISR入口）；
3. 开相应中断；
4. 等待按键事件发生（或其他处理程序）

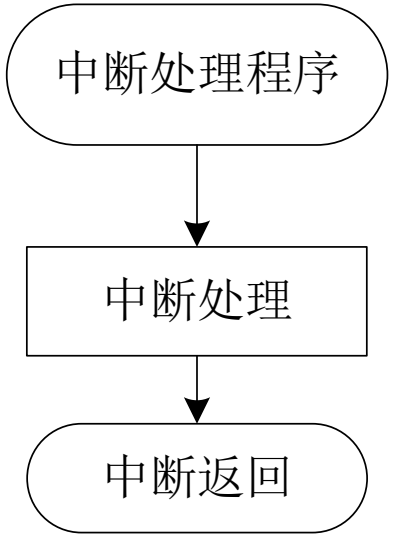
编程实施：采用C语言，模块化设计

1. 系统初始化函数 SystemInit()
2. 中断初始化函数 Isr_Init()
3. 中断向量注册函数 Irq_Request(irq_no, void *irq_routine)
4. 开中断函数 Irq_Enable(int irq_no)
5. 清断中断标志函数 Irq_Clear(int irq_no)
6. 中断处理函数 Eint2_isr()
7. 主程序 Main()

程序流程图



源程序见教材



6.2.5 IRQ和FIQ异常中断处理（扩展阅读）

1. IRQ/FIQ中断处理机制

- 优先级仲裁：FIQ比IRQ中断优先级高，表现在：

① 当FIQ和IRQ异常中断同时发生时，CPU优先处理FIQ异常中断；

② 在FIQ异常中断处理程序中IRQ异常中断被禁止。

- ARM体系结构在设计上对FIQ有特别安排，以尽量减少FIQ异常中断的响应时间：

6.2.5 IRQ和FIQ异常中断处理（扩展阅读）

1. IRQ/FIQ中断处理机制

① FIQ的中断向量0x1C位于中断向量表的最后。这样FIQ异常中断处理程序可以直接放在地址0x1C开始的存储单元，省掉了向量表中的跳转指令，节省了中断响应的时间。

② 若系统有Cache，可以把FIQ异常中断向量以及处理程序一起锁定在Cache中，提高FIQ异常中断的响应时间。

③ FIQ异常模式还有额外的5个物理寄存器(R8_fiq~R12_fiq)，在进入FIQ中断处理程序时可以不用保存这5个寄存器，从而也提高了FIQ异常中断的处理速度。

2. IRQ和FIQ异常中断处理程序的返回

- ARM920T采用五级流水线架构，当异常中断产生时，程序计数器PC已经指向当前执行指令后面的第3条指令；
- ARM920T核将(PC-4)的值保存到异常模式下的寄存器LR/R14_<Exception_Mode>中。这时(PC-4)即指向当前指令后的第2条指令（中断返回指令的下一条指令位置）。

www		PC-12	←异常发生
xxx	★	PC-8	
yyy		PC-4	←LR中的值
zzz		PC	

★ 表示异常返回后将执行的那条指令

图 IRQ/FRQ的返回地址

2. IRQ和FIQ异常中断处理程序的返回

- 因此，返回操作可通过指令:

SUBS PC, LR, #4(R14_irq/R14_fiq)

来实现。同时，SPSR_<Exception_Mode>的内容被复制到当前程序状态寄存器中。

3. ARM编译器关键词及其支持的数据类型

ARM C编译器支持一些对ANSI C进行扩展的关键词。这些关键词用于声明变量和函数，会对特定的数据类型进行一定的限制。

(1)用于声明函数的关键词

某些关键词告诉编译器用于对被声明的函数给予特别的处理，这是对ANSI C的ARM功能扩展。

_irq

关键词_irq 声明的函数可用作IRQ或FIQ异常中断的中断处理函数。可以保存除浮点寄存器外被该函数破坏的寄存器，包括ATPCS标准要求的寄存器。

3. ARM编译器关键词及其支持的数据类型

_irq声明的函数通过将LR-4的值赋给PC，将SPSR的值赋给CPSR实现函数的返回。(注：该关键词不适用于TCC或者TCPP编译器，所声明的函数不能返回参数或数值)

举例：
```
static void _irq Eint1Int(void)
{
 ClearPending(BIT_EINT1);
 Uart_Printf( “EINT1 interrupt is occurred.\n” );
 }
```

上面中断处理程序由系统外部中断EINT1触发，在中断处理程序中清除该中断位，并给出中断发生的提示信息。

_swi

关键词_swi声明的函数最多可以接收4个整型变量，并最多可以利用value_in_regs返回4个结果。

当函数不返回参数时，可以使用下面的格式：

```
void _swi(swi_num) swi_name(int arg1, …, int argn);
```

当函数返回1个参数时，可以使用下面的格式：

```
int _swi(swi_num) swi_name(int arg1, …, int argn);
```

当函数返回参数多于1时，可以使用下面的格式：

```
typedef struct res_type
{int res1, …, int resn;} res_type_value_in_regs;
res_type_value_in_regs _swi(swi_num) swi_name(int arg1, …, int argn);
```

(2) 用于声明变量的关键词

register

关键词register所声明的变量，在编译器处理时要尽量保存到寄存器中。但是这种声明仅起建议作用，编译器会根据具体情况处理各变量。实际上不提倡声明多于4个整型register变量和2个浮点register变量。

所有的整数类型、整数的结构型数据类型、指针型变量和浮点变量都可以声明成register。

(2) 用于声明变量的关键词

volatile

关键词**volatile**所声明的变量用于告知编译器该变量可能在程序之外修改。编译器在编译时不优化对volatile变量的操作。

举例：使用volatile类型的结构访问系统中的I/O寄存器。

```
/*将I/O端口寄存器映射到存储器*/
volatile unsigned *port=(unsigned int *) 0x400000000;
/*访问寄存器端口*/
*port=value;   /*写寄存器端口*/
value=*port;  /*读寄存器端口*/
```

（3） ARM编译器支持的基本数据类型

表　基本数据类型长度和对齐方式

数据类型	长度	对齐方式	数据类型	长度	对齐方式
char	8	1(字节对齐)	float	32	4(字对齐)
short	16	2(半字对齐)	double	32	4(字对齐)
int	32	4(字对齐)	long double	32	4(字对齐)
long	32	4(字对齐)	all pointers	32	4(字对齐)
long long	64	4(字对齐)	bool(仅限C++)	32	4(字对齐)

4. IRQ/FIQ异常中断处理程序

在有些IRQ/FIQ异常中断处理程序中，允许新的IRQ/FIQ异常中断，这时将需要一些特别的操作保证“老的”异常中断的寄存器不会被“新的”异常中断破坏，这种IRQ/FIQ异常中断处理程序称为**可重入的异常中断处理程序**(reentrant interrupt handler)。

(1)不可重入的IRQ/FIQ异常中断处理程序

对于C语言，不可重入的IRQ/FIQ异常中断处理程序可以使用关键词_iqr来说明。关键词_irq可以实现如下操作：

- 保存ATPCS规定的被破坏的寄存器；
- 保存其他中断处理程序中用到的寄存器；
- 同时将LR-4赋与程序计数器PC实现中断处理程序的返回，并且恢复CPSR寄存器的内容。

下例程序说明了关键词_irq的作用，其中列出了C语言程序及其对应的汇编程序。2个C语言程序，第1个使用关键词_irq，第2个没有使用关键词_irq声明。

例：关键词_irq的作用。

程序①：

```
_irq void IRQHandler1(void)
    {
      volatile unsigned int *base=(unsigned int*)0x80000000;
      if(*base==1)
      {
       C_int_handler();      //调用相应的C语言IRQ中断处理程序
      }
     *(base+1)=0;
    }
```

程序①对应的汇编语言程序如下：

```
AREA IRQHandler_1, CODE, READONLY
IMPORT C_int_handler      ;引入C语言的IRQ中断处理程序
EXPORT  IRQHandler1

IRQHandler1 PROC
STMFD SP!, {R0-R4, R12, LR}
MOV R4, #0x80000000
LDR R0, [R4, #0]
SUB SP, SP, #4
CMP R0, #1
BLEQ C_int_handler
MOV R0, #0
STR R0, [R4, #4]
ADD SP, SP, #4
LDMFD SP!, {R0-R4, R12, LR}
SUBS PC, LR, #4          ; IRQ中断返回
ENDP
END
```

程序②：

```
void IRQHandler2(void)
{
  volatile unsigned int *base=(unsigned int*)0x80000000;
  if(*base==1)
  {
   C_int_handler();      //调用相应的C语言IRQ中断处理程序
  }
  *(base+1)=0;
}
```

程序②对应的汇编语言程序如下：

```
AREA IRQHandler_2,CODE,READONLY
IMPORT C_int_handler  ;引入C语言的IRQ中断处理程序
EXPORT IRQHandler2

IRQHandler2 PROC
STMFD SP!, {R0-R4, R12, LR}
MOV R4, #0x80000000
LDR R0, [R4, #0]
CMP R0, #1
BLEQ C_int_handler
MOV R0, #0
STR R0, [R4, #4]
LDMFD SP!, {R0-R4, R12, LR}
ENDP
END
```

比较程序①和程序②，使用了关键词_irq声明的C语言程序在汇编后生成了对堆栈指针处理的指令、返回时对断点处理的指令。没有使用关键词_irq声明，则对堆栈指针的处理依赖于系统。

(2) 可重入的IRQ/FIQ异常中断处理程序

如果在可重入的IRQ/FIQ异常中断处理程序中调用了子程序，子程序的返回地址将被保存到寄存器LR_irq中，这时若发生了IRQ/FIQ异常中断，该LR_irq寄存器的值将被破坏，那么调用的子程序将不能正确返回。因此，对于可重入的IRQ/FIQ异常中断处理程序需要一些特别的操作。

这时，第1级中断处理程序(对应于IRQ/FIQ异常中断的程序)不能使用C语言，因为其中一些操作不能通过C语言实现。

可重入的IRQ/FIQ异常中断处理程序操作：

1. 将返回地址保存到IRQ的数据栈中；
2. 保存工作寄存器和SPSR_irq；
3. 清除中断标志；
4. 将处理器切换到系统模式，重新使能中断(IRQ/FIQ)；
5. 保存用户模式的LR寄存器和被调用者不保存的寄存器；
6. 调用C语言程序的IRQ/FIQ异常中断处理程序；
7. 当C语言程序的IRQ/FIQ异常中断处理程序返回后，恢复用户模式寄存器，并禁止中断(IRQ/FIQ)；
8. 切换到IRQ模式，禁止中断；
9. 恢复工作组寄存器和寄存器LR_irq；
10. 从IRQ异常中断处理程序中返回。

5. S3C2410中断编程模式

S3C2410芯片的I/O端口或部件若采用中断方式控制操作时，其编程的内容实际上涉及4部分，即：

（1）建立系统中断向量表，并且设置ARM920T核的程序状态寄存器CPSR中的F位和I位。一般情况下中断均需使用数据栈。因此，还需建立用户数据栈。这一部分内容对应的程序指令，通常编写在系统引导程序中。

（2）设置S3C2410芯片中56个中断源的中断向量。通常需要利用挂起寄存器或地址偏移寄存器来计算，若中断号还对应有子中断（如中断号为5时，对应EINT8_23），需求出子中断的地址偏移。

5. S3C2410中断编程模式

(3)中断控制初始化。主要是初始化S3C2410芯片内部的中断控制的寄存器。针对某个具体的中断源，设置其中断控制模式、中断是否屏蔽、中断优先级等。

(4) 完成I/O端口或部件具体操作功能的中断服务程序。中断服务程序中，在返回之前必须对中断挂起寄存器（INTPND）的相应挂起标志位进行清除操作。

上述4部分的程序，第1部分应属于系统引导程序完成的功能。若用户在开发嵌入式系统时使用的是现成硬件平台，则用户对第1部分的程序通常不需要进行编写，硬件平台已带有系统引导程序，用户主要需编写的是后3部分的程序。

5.2.4 中断应用举例

【例1】使用定时器1控制一只LED发光二极管每1秒钟改变一次状态(设LED连接到GPC0)。

(1)对定时器1初始化，并设定定时器的中断时间为1s，具体代码如下：

```
void Timer1_init（void）
{
  rGPCCON = (rGPCCON | 0x00000001) &0xfffffffd;
  //配置GPC0口线为输出
  //rGPCCON: Port C control
  rGPCDAT = rGPCDAT | 0x00000001; //rGPCDAT: Port C data

  rTCFG0 = 255;                  //rTCG0:Timer 0/1 configuration
  rTCFG1 = 0x01<<4;              //rTCG1:Timer 1 configuration
  rTCNTB1 = 48828;               //rTCNTB1:Timer count buffer1
                                 //在pclk = 50MHz下, 1s的记数值
                                 //rTCNTB1 = 50000000/4/256=48828
  rTCMPB1 ＝ 0x00;               //rTCMPB1:Timer compare buffer 1
  rTCON ＝(1+11) | (1<<9) | (0<<8);      //禁用定时器1, 手动加载
  rTCON ＝(1+11) | (0<<9) | (1<<8);      //启动定时器1, 自动加载
}
```

(2) 为了使CPU响应中断，在中断服务子程序执行之前，必须打开S3C2410的CPSR中的I位，以及相应的中断屏蔽寄存器中的位。打开相应的中断屏蔽寄存器中的位是在Timer1INT Init()函数中实现的，代码如下：

```
   void Timer1INT_Init(void)
  {//定时器接口使能
     if (rINTPND&BIT_TIMER1)                    //若有INT_TIMER1中断请求
         rSRCPND | =  BIT_TIMER1;
     pISR_TIMER1=(int) Timer1_ISR;              //写入定时器1中断服务子程序

//的入口地址
     rINTMSK &= ~ BIT_TIMER1;                   //开中断;
    }
```

注: #define BIT_TIMER1 (0x1<<11) //Pending bit(见2410addr.h文件)

说明：上面初始化函数中，pISR_TIMER1在三星公司test2410_r11 软件包的2410addr.h头文件中有定义:

#define pISR_TIMER1 (*unsigned *)(_ISR_STARTADDRESS+0x4C))

其中, 从_ISR_STARTADDRESS+0x20单元开始是在RAM中定义的外部(二级)中断向量表。对于二级中断向量表， 各外部中断向量指针定义（ 摘自2410addr.h文档 ）如下：

```
#define pISR_EINT0        (*(unsigned *)(_ISR_STARTADDRESS+0x20))
#define pISR_EINT1        (*(unsigned *)(_ISR_STARTADDRESS+0x24))
#define pISR_EINT2        (*(unsigned *)(_ISR_STARTADDRESS+0x28))
#define pISR_EINT3        (*(unsigned *)(_ISR_STARTADDRESS+0x2c))
#define pISR_EINT4_7      (*(unsigned *)(_ISR_STARTADDRESS+0x30))
#define pISR_EINT8_23     (*(unsigned *)(_ISR_STARTADDRESS+0x34))
#define pISR_NOTUSED6     (*(unsigned *)(_ISR_STARTADDRESS+0x38))
#define pISR_BAT_FLT      (*(unsigned *)(_ISR_STARTADDRESS+0x3c))
#define pISR_TICK         (*(unsigned *)(_ISR_STARTADDRESS+0x40))
#define pISR_WDT          (*(unsigned *)(_ISR_STARTADDRESS+0x44))
#define pISR_TIMER0       (*(unsigned *)(_ISR_STARTADDRESS+0x48))
#define pISR_TIMER1       (*(unsigned *)(_ISR_STARTADDRESS+0x4c))
#define pISR_TIMER2       (*(unsigned *)(_ISR_STARTADDRESS+0x50))
```

```
#define pISR_TIMER3       (*(unsigned *)(_ISR_STARTADDRESS+0x54))
#define pISR_TIMER4       (*(unsigned *)(_ISR_STARTADDRESS+0x58))
#define pISR_UART2        (*(unsigned *)(_ISR_STARTADDRESS+0x5c))
#define pISR_LCD          (*(unsigned *)(_ISR_STARTADDRESS+0x60))
#define pISR_DMA0         (*(unsigned *)(_ISR_STARTADDRESS+0x64))
#define pISR_DMA1         (*(unsigned *)(_ISR_STARTADDRESS+0x68))
#define pISR_DMA2         (*(unsigned *)(_ISR_STARTADDRESS+0x6c))
#define pISR_DMA3         (*(unsigned *)(_ISR_STARTADDRESS+0x70))
#define pISR_SDI          (*(unsigned *)(_ISR_STARTADDRESS+0x74))
#define pISR_SPI0         (*(unsigned *)(_ISR_STARTADDRESS+0x78))
#define pISR_UART1        (*(unsigned *)(_ISR_STARTADDRESS+0x7c))
#define pISR_NOTUSED24    (*(unsigned *)(_ISR_STARTADDRESS+0x80))
#define pISR_USBD         (*(unsigned *)(_ISR_STARTADDRESS+0x84))
#define pISR_USBH         (*(unsigned *)(_ISR_STARTADDRESS+0x88))
#define pISR_IIC          (*(unsigned *)(_ISR_STARTADDRESS+0x8c))
#define pISR_UART0        (*(unsigned *)(_ISR_STARTADDRESS+0x90))
#define pISR_SPI1         (*(unsigned *)(_ISR_STARTADDRESS+0x94))
#define pISR_RTC          (*(unsigned *)(_ISR_STARTADDRESS+0x98))
#define pISR_ADC          (*(unsigned *)(_ISR_STARTADDRESS+0x9c))
```

(3) 等待定时器中断，通过一个死循环，如“while(1);”实现等待过程。

(4) 在S3C2410 体系中, 中断的调用可以看成是经历了两次“中断向量表”的查询。2410init.s 中的以下代码所完成功能就是查询中断偏移寄存器INTOFFSET, 得到当前中断的中断号, 并根据中断号再调用相应的中断服务程序。

```
IsrIRQ
sub     sp,sp,#4;               //为保存PC预留堆栈空间
stmfd   sp!,{r8-r9}
ldr     r9,＝INTOFFSET
ldr     r9,[r9];     //加载INTOFFSET寄存器值到r9(Timer 1中断偏移值即中断号为0xB)
ldr     r8,＝HandleEINT0;//加载二级中断向量表的基地址到r8
add     r8,r8,r9,lsl #2;        //获得中断向量
ldr     r8, [r8];               //加载中断服务程序的入口地址到r8
str     r8,[sp,#8];             //保存sp，将其作为新的pc值
ldmfd   sp!,{r8-r9,pc}; //跳转到新的pc处执行，即跳转到中断服务子程序执行
```

注：HandleEINT0在2410init.s中定义，它为外部(二级)中断向量表的基地址。

HandleEINT0可理解为用户自己开辟的一块存储空间的起始地址，后面按次序存放异常中断处理程序的入口地址，也可以理解为二级中断向量表。IsrIRQ 从中断控制器处获取中断源(中断号)信息，然后再从二级中断向量表中的对应地址单元得到异常中断处理程序的入口地址，完成异常响应的跳转。二级中断向量表位于HandleFIQ 的后面，也就是以_ISR_STARTADDRESS+ 0x20 为起始地址，这里定义了S3C2410 处理器所有中断源的相关中断处理程序入口。这种方法的好处是用户程序在运行过程中能够动态改变异常向量。

(5) 为了方便C 程序使用中断, 将IsrIRQ 设为IRQ 的中断服务程序，代码如下：

```
ldr     r0,= HandleIRQ
ldr     r1,= IsrIRQ
str     r1,[r0];
```

(6) 执行中断服务子程序，该子程序实现LED灯每一秒钟改变一次状态。看到LEDI灯闪烁一次，则说明定时器发生了一次中断。具体实现见下面Timer1_ISR()函数：

```
int f ;
void _ _irq Timer1_ISR(void)
{
  if (f= = 0)
  { rGPCDAT =  rGPCDAT | 0x00000001;
    f=1;
  }
  if (f= = 1)
  { rGPCDAT= rGPCDAT & 0xfffffffe;
    f=0;
  }
 rSRCPND | =  BIT_TIMER1;
 rINTPND | = BIT_TIMER1;
}
```

中断服务程序中是先清除SRCPND的相应标志位,再清除INTPND的相应标志位,清除方法是将对应位写1。

(7) 从中断返回，恢复现场，跳转到被中断的主程序继续执行，等待下一次中断的到来。

【例2】外中断编程。

```
static void __irq Eint0Int(void)
{
   ClearPending(BIT_EINT0);
   Uart_Printf("EINT0 interrupt is occurred.\n");
}
static void __irq Eint1Int(void)
{
   ClearPending(BIT_EINT1);
   Uart_Printf("EINT1 interrupt is occurred.\n");
}
```

注：#define ClearPending(pending_bit) { rSRCPND = bit; rINTPND = bit; }

```
void Test_Eint(void)
{
    int i;
    int extintMode;                    //选择外中断触发方式变量

    Uart_Printf("[External Interrupt Test]\n");

    Uart_Printf("1.L-LEVEL  2.H-LEVEL  3.F-EDGE  4.R-EDGE  5.B-EDGE\n");
    Uart_Printf("Select the external interrupt type.\n");
    extintMode=Uart_Getch();

                                       //extintMode='3'
    rGPFCON = (rGPFCON & 0xfffa)|(1<<3)|(1<<1);
    // 设置引脚配置，GPF0配置为EINT0, GPF1配置为EINT1
```

```
switch(extintMode)
{
  case '1':
      rEXTINT0 = (rEXTINT0 & ~((7<<4) | (0x7<<0))) | 0x0<<4 | 0x0<<0;
      //EINT0/1=low level triggered
      break;

  case '2':
      rEXTINT0 = (rEXTINT0 & ~((7<<4) | (0x7<<0))) | 0x1<<4 | 0x1<<0;
      //EINT0/1=high level triggered
      break;

  case '3':
      rEXTINT0 = (rEXTINT0 & ~((7<<4) | (0x7<<0))) | 0x2<<4 | 0x2<<0;
      //EINT0/1=falling edge triggered
 break;
```

```
    case '4':
        rEXTINT0 = (rEXTINT0 & ~((7<<4) | (0x7<<0))) | 0x4<<4 | 0x4<<0;
        //EINT0/1=rising edge triggered
     break;

    case '5':
        rEXTINT0 = (rEXTINT0 & ~((7<<4) | (0x7<<0))) | 0x6<<4 | 0x6<<0;
        //EINT0/1=both edge triggered
        break;

    default:    break;
}
```

注：EXTINT0: 外中断触发方式控制寄存器0，EINT0/1---EINT0/1中断请求信号触发方式选择, EINT0=EXTINT0[2:0], EINT1=EXTINT1[6:4]

000：低电平触发　　001：高电平触发
01x：下降沿触发　　10x：上升沿触发
11x：双边沿触发

```
Uart_Printf("Press the EINT0/1 buttons or Press any key to exit.\n");
                        //设置中断向量
pISR_EINT0=(U32)Eint0Int;                  //将中断处理程序的开始
pISR_EINT1=(U32)Eint1Int;                  //地址送到中断向量表
rEINTPEND = 0xffffff;        //清除EINTPND需要向其中写入1，因此这句代码
  的含义是清除EINTPND
rSRCPND = BIT_EINT0|BIT_EINT1;
rINTPND = BIT_EINT0|BIT_EINT1; //to clear the previous pending states
rINTMSK=~(BIT_EINT0|BIT_EINT1);
Uart_Getch();
rEINTMASK=0xffffff;
rINTMSK=BIT_ALLMSK;
}       //{void Test_Eint(void)函数结束}
```

注： #define BIT_EINT0 (0x1)

#define BIT_EINT1 (0x1<<1)

#define BIT_ALLMSK (0xffffffff)

S3C2410 中断处理小结

- **S3C2410**的中断异常处理模块由下面寄存器构成

SRCPND (SOURCE PENDING REGISTER)
INTMOD (INTERRUPT MODE REGISTER)
INTMSK (INTERRUPT MASK REGISTER)
PRIORITY (PRIORITY REGISTER)
INTPND (INTERRUPT PENDING REGISTER)
INTOFFSET (INTERRUPT OFFSET REGISTER)
SUBSRCPND (INTERRUPT SUB SOURCE PENDING)
INTSUBMSK (INTERRUPT SUB MASK REGISTER)

● 各寄存器在中断处理流程中所扮演的角色

SRCPND/SUBSRCPND：

这两个寄存器在功能上是相同的，它们是中断源引脚寄存器，在一个中断异常处理流程中，中断信号传进中断异常处理模块后首先遇到的就是SRCPND/SUBSRCPND, 这两个寄存器的作用是用于标识哪个中断请求被触发。SRCPND的有效位为32，SUBSRCPND 的有效位为11，它们中的每一位分别代表一个中断源。SRCPND为主中断源引脚寄存器，SUBSRCPND为子中断源引脚寄存器，每个位的初始值皆为0。

假设现在系统触发了TIMER0中断，则SRCPND第10bit将被置1，代表TIMER0中断被触发，该中断请求即将被处理（若该中断没有被屏蔽的话）。SUBSRCPND情况与SRCPND相同。

INTMOD：

INTMOD寄存器有效位为32位，每一位与**SRCPND**中各位相对应，它的作用是指定该位相应的中断源处理模式（IRQ还是FIQ）。若某位为0，则该位相对应的中断按IRQ模式处理，为1则以FIQ模式进行处理，该寄存器初始化值为0x00000000,即所有中断皆以IRQ模式进行处理 。

INTMSK/ INTSUBMSK：

INTMSK为主中断屏蔽寄存器，**INTSUBMSK**为中断子源屏蔽寄存器。**INTMSK**有效位为32，**INTSUBMSK**有效位为11，这两个寄存器各位与**SRCPND**和**SUBSRCPND**分别对应。它们的作用是决定该位相应的中断请求是否被处理。若某位被设置为1，则该位相对应的中断产生后将被忽略（CPU不处理该中断请求），设置为0则对其进行处理。这两个寄存器初始化后的值是0xFFFFFFFF和0x7FF，既默认情况下所有的中断都是被屏蔽的。

INTPND：

INTPND寄存器可能是整个中断处理过程中要特别注意的一个寄存器，他的操作比较特别。如果说**SRCPND**是中断信号进入中断处理模块后所经过的第一个场所的话，那么**INTPND** 则是中断信号在中断处理模块里经历的最后一个寄存器。它的每个位对应一个中断请求，若该位被置1，则表示相应的中断请求被触发，描述到这里你可能会发现它不仅和**SRCPND**长得一模一样，就连功能都一样，其实不然，他们在功能上有着重大的区别。

SRCPND是中断源引脚寄存器，某个位被置1表示相应的中断被触发，但我们知道在同一时刻内系统可以触发若干个中断，只要中断被触发了，**SRCPND**的相应位便被置1，也就是说**SRCPND** 在同一时刻可以有若干位同时被置1，然而**INTPND**则不同，他在某一时刻只能有1个位被置1，INTPND 某个位被置1（该位对应的中断在所有已触发的中断里具有最高优先级且该中断没有被屏蔽），则表示CPU即将或已经在对该位相应的中断进行处理。于是我们可以有一个总结：SRCPND说明了有什么中断被触发了，INTPND说明了CPU即将或已经在对某一个中断进行处理。

特别注意：每当某一个中断被处理完之后，我们必须人为地把**SRCPND/SUBSRCPND、INTPND**三个寄存器中与该中断相应的位由1设置为0，刚才说到**INTPND**的操作很特别，它的特别之处就在于对当我们要把该寄存器中某个值为1的位设置为0时，我们不是往该位置0，而是往该位置1。假设**SRCPND=0x00000003，INTPND=0x00000001**,该值说明当前0号中断和1号中断被触发，但当前正在被处理的是0号中断，处理完毕后我们应该这样设置**INTPND**和**SRCPND**:

SRCPND=0x00000002;　　//位0被置为0

INTPND =0x00000001;　　//位0被置为0（方法是往该位写入1）

INTOFFSET：

INTOFFSET寄存器的功能则很简单，它的作用只是用于表明哪个中断正在被处理(**INTOFFSET**中的偏移值指示在**INTPND**中显示的中断源)。

课堂思考

用INT_TIMER0、INT_TIMER2和INT_UART0三个中断完整地描述一次中断异常处理。

几个假设:

假设1：这三个中断的屏蔽被取消。

假设2：**PRIORITY**寄存器中ARB_MODE2，ARB_MODE5皆为0，既不进行优先级的自动循环排序，任何时候 **ARBITER2、ARBITER5**控制的中断组优先级次序分别为0-1-2-3-4-5和1-2-3-4。

假设3：这三个中断皆为IRQ类型。

假设4：这三个中断同时被触发。

Ans:

INT_TIMER0、INT_TIMER2和INT_UART0三个中断被同时触发，此时3个中断信号流向**SRCPND**寄存器，使该寄存器中的第10位，12位，28位被置为1，中断信号继续向前流经**INTMODE**，这三个中断皆为IRQ类型，于是信号进一步流经**INTMASK** 寄存器，这三个中断都没有被屏蔽，故中断信号继续向前流向**PRIORITY**寄存器，经过优先级判断，INT_TIMER0中断信号使**INTPND** 寄存器的第10位置1(INT_TIMER0优先级最高)，此时**INTOFFSET** 寄存器的值为10，CPU转向相应的中断服务例程进行处理。处理完毕后，程序将**INTPND**和**SRCPND**的第10置为0，至此INT_TIMER0中断处理完毕。此时**SRCPND** 的第12位，28位仍为1（这两个中断请求未被处理），故它们会继续被CPU按刚才所描述的过程进行处理。

(注：参考本课件文件Slide 19 S3C2410中断处理流程图就不难理解本问题的回答)