第5章 中断系统

5.1 中断的概述

中断是为处理器对外界异步事件具有处理能力而设置的，中断技术的引入把计算机的发展和应用大大地推进一步。因此中断功能的强弱已成为衡量一台计算机性能的重要指标。

1．中断

中断是指计算机在执行某一程序的过程中，由于计算机系统内、外的某种原因，而必须终止原程序的执行，转去执行相应的处理程序，待处理结束之后，再回来继续执行被终止的原程序过程。

中断技术能实现 CPU 与外部设备的并行工作，提高 CPU 的利用率以及数据的输入/输出效率；中断技术也能对计算机运行过程中突然发生的故障做到及时发现并进行自动处理如：硬件故障、运算错误及程序故障等；中断技术还能使我们通过键盘向计算机发出请求，随时对运行中的计算机进行干预，而不用先停机，然后再重新开机等等。

2．中断源

中断源是指在计算机系统中向 CPU 发出中断请求的来源，中断源可以人为设定

也可以是为响应突发性随机事件而设置。如定时器中断，它的中断源即是定时器。

3．中断优先级

由于在实际的系统中，往往有多个中断源，且中断申请是随机的，有时可能会有多个中断源同时提出中断申请，但 CPU 一次只能响应一个中断源发出的中断请求，这时 CPU 应响应那个中断请求？这就需要用软件或硬件按中断源工作性质的轻重缓急，给它们安排一个优先顺序，即所谓的优先级排队。中断优先级越高，则响应优先权就越高。当 CPU 正执行中断服务程序时，又有中断优先级更高的中断申请产生，如果 CPU 能够暂停对原来的中断处理程序，转而去处理优先级更高的中断请求，处理完毕后，再回到原低级中断处理程序，这一过程称为中断嵌套。具有这种功能的中断系统称为多级中断系统；没有中断嵌套功能的则称为单级中断系统。具有二级中断服务程序嵌套的中断过程如图 5.1所示。

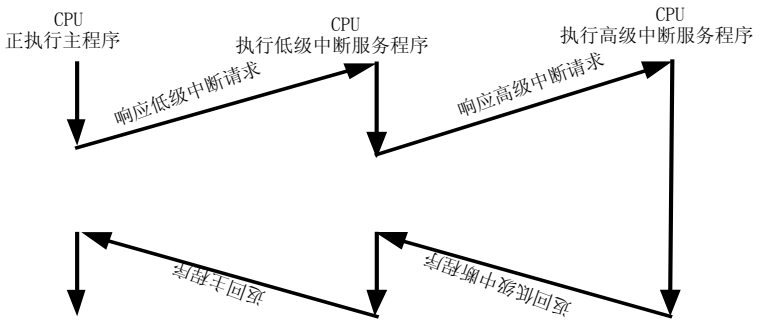

图5.1 中断嵌套示意图

4．中断响应的过程

（1） 在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，相应的中断允许位为真（允许中断），相应的总中断允许位为真（允许中断），则响应中断。

（2） 保护现场，CPU 一旦响应中断，中断系统会自动的保存当前的 PC 和 SR 寄存器（入栈）进入中断服务程序地址入口,中断服务程序中可以通过入栈保护原程序中用到的数据，保护现场前，一般要关中断以防止现场被破坏。保护现场一般是将堆栈指令将原程序中用到的寄存器推入堆栈，在保护现场之后要开中断，以响应更高优先级的中断申请。

（3） 中断服务，即为相应的中断源服务。

（4） 清相应的中断请求标志位，以免 CPU 总是执行该中断。

（5） 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。

（6） 返回，此时 CPU 将 PC 指针和 SR 内容出栈恢复断点，从而使 CPU 继续执行刚才被中断的程序。

在单片机中，中断技术主要用于实时控制。所谓实时控制，就是要求单片机能及时地响应被控对象提出的分析、计算和控制等请求，使被控对象保持在最佳工作状态，以达到预定的控制效果。由于这些控制参量的请求都是随机发出的，而且要求单片机必须作出快速响应并及时处理，对此，只有靠中断技术才能实现。

5.2 SPCE061A 中断系统

SPCE061A 系列单片机中断系统，是凌阳 16 位单片机中中断功能较强的一种，它可以提供 14 个中断源，具有两个中断优先级，可实现两级中断嵌套功能。用户可以用关中断指令（或复位）屏蔽所有的中断请求，也可以用开中断指令使 CPU 接受中断申请。每一个中断源可以用软件独立控制为开或关中断状态；但中断级别不可用软件设置。

- SPCE061A 的中断类型

SPCE061A 的结构给出了三种类型的中断：软件中断、异常中断和事件中断。

1)　软件中断

软件中断是由软件指令 break 产生的中断。软件中断的向量地址为 FFF5H

2）异常中断

异常中断表示为非常重要的事件，一旦发生，CPU 必须立即进行处理。目前 SPCE061A 定义的异常中断只有‘复位’一种。通常，SPCE061A 系统复位可以由以下三种情况引起：上电、看门狗计数器溢出以及系统电源低于电压低限。不论什么情况引起复位，都会使复位引脚的电位变低，进而使程序指针 PC 指向由一个复位向量（FFF7H）所指的系统复位程序入口地址。

3）事件中断

事件中断（可简称“中断”，以下提到的“中断”均为事件中断）一般产生于片内设部件或由外设中断输入引脚引入的某个事件。这种中断的开通/禁止，由相应独立使能和相应的 IRQ 或 FIQ 总使能控制。

SPCE061A 的事件中断可采用两种方式：快速中断请求即 FIQ 中断和中断请求即 IRQ 中断。这两种中断都有相应的总使能。

中断向量和中断源：

共有 9 个中断向量即 FIQ、IRQ0～IRQ6 及 UART IRQ。这 9 个中断向量共可安置 14 个中断源供用户使用，其中有 3 个中断源可安置在 FIQ 或 IRQ0～IRQ2 中，另有 10 个中断源则可安置在 IRQ3～IRQ6 中。还有一个专门用于通用异步串行口 UART 的中断源，须安置在 UART IRQ 向量中。

5.2.1 中断源

SPCE061A 单片机的中断系统有 14 个中断源分为两个定时器溢出中断、两个外部中断、一个串行口中断、一个触键唤醒中断、7 个时基信号中断、PWM 音频输出中断。如下表 5.1。

表5.1 中断源列表

中断源	中断优先级	中断向量	保留字
Fosc/1024 溢出信号 PWM INT	FIQ/IRQ0	FFF8H/FFF6H	_FIQ/_IRQ0
TimerA 溢出信号	FIQ /IRQ1	FFF9H/FFF6H	_FIQ/_IRQ1
TimerB 溢出信号	FIQ /IRQ2	FFFAH/FFF6H	_FIQ/_IRQ2
外部时钟源输入 信号 EXT2	IRQ3	FFFBH	_IRQ3
外部时钟源输入 信号 EXT1			
触键唤醒信号			
4096Hz 时基信号	IRQ4	FFFCH	_IRQ4
2048Hz 时基信号			
1024Hz 时基信号			
4Hz 时基信号	IRQ5	FFFDH	_IRQ5
2Hz 时基信号			
频选信号 TMB1	IRQ6	FFFEH	_IRQ6
频选信号 TMB2			
UART 传输中断	IRQ7	FFFFH	_IRQ7
BREAK	软中断		

从表中可以看到每个中断入口地址对应多个中断源，因此在中断服务程序中需通过查询中断请求位来判断是那个中断源请求的中断。

5.2.1.2 定时器溢出中断源

定时器溢出中断由 SPCE061A 内部定时器中断源产生，故它们属于内部中断；在 SPCE061A 内部有两个 16 位定时器/计数器，定时器 TimerA/TimerB 在定时脉冲作用下从预置数单元开始加计数,当计数为“0xFFFF”时可以自动向 CPU 提出溢出中断请求，以表明定时器 TimerA 或 TimerB 的定时时间已到。定时器 TimerA/TimerB 的定时时间可由用户通过程序设定，以便 CPU 在定时器溢出中断服务程序内进行计时。另外，SPCE061A 单片机的定时器时钟源很丰富，从高频到低频都有，因此，根据定时时间长短可以选择不同的时钟源，定时器 A 的时钟源比定时器 B 多，定时器 B 无低频时钟源。在中断一览表中也可以看出，定时器 A 的中断，IRQ 和 FIQ 中都有，方便开发人员的使用。详见第二章**定时器/计数器**内容。定时器溢出中断通常用于需要进行定时控制的场合。

5.2.1.3 外部中断源

SPCE061A 单片机有两个外部中断，分别为 EXT1 和 EXT2，两个外部输入脚分别为 B 口的 IOB2 和 IOB3 的复用脚。EXT1（IOB2）和 EXT2（IOB3）两条外部中断请求输入线，用于输入两个外部中断源的中断请求信号，并允许外部中断以负跳沿触发方式来输入中断请求信号。如图 5.2。

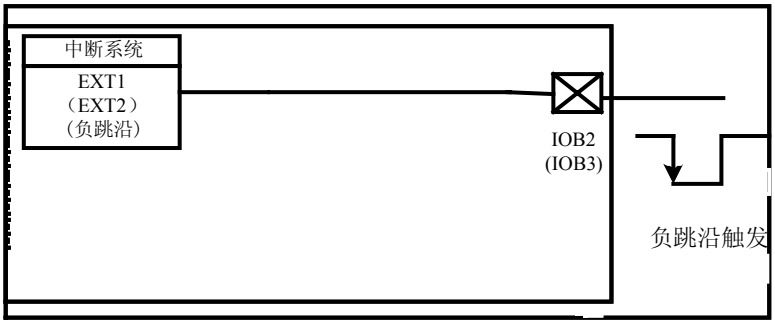

图5.2 外部中断结构

另外，SPCE061A 单片机在 IOB2 和 IOB4 之间以及 IOB3 和 IOB5 之间分别加入两个反馈电路，可以外接 RC 振荡器，做外部定时中断使用。如图 5.3，外部中断的反馈电路使用四个管脚为 B 口的 IOB2，IOB4，IOB3，和 IOB5 的复用脚，其中 IOB4 和 IOB5 主要用来组成 RC 反馈电路之结构。通过 IOB2 和 IOB4 之间或者 IOB3 和 IOB5 之间增加一个 RC 振荡电路，便可在 EXT1 或 EXT2 端得到振荡信号。为使反馈电路正常工作，必须将 IOB2 或 IOB3 设置成反相输出口，且 IOB4 或 IOB5 设成输入口。

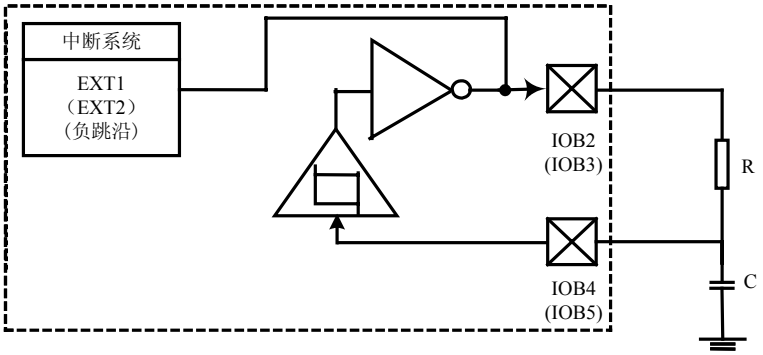

图5.3 IOB2,IOB4,或 IOB3，IOB5 之间的反馈结构

5.2.1.4 串行口中断源

串行口中断由 SPCE061A 内部串行口中断源产生，故也是一种内部中断。串行口中断分为串行口发送中断和串行口接收中断两种，但其中断向量是一个，因此，进入串行中断服务程序时，也需要判断是接收中断还是发送中断。在串行口进行发送/接收完一组串行数据时，串行口电路自动使串行口控制寄存器 P_UART_Command2 中的 TXReady 和 RXReady 中断标志位置位。并自动向 CPU 发出串行口中断请求，CPU 响应串行口中断后便立即转入串行口中断服务程序执行。因此，只要在串行中断服务程序中安排一段对 P_UART_Command2 对 TXReady 和 RXReady 中断标志位状态的判断程序，便可区分串行口发生了接收中断请求还是发送中断。当然，串行传输中，既可以使用中断方式收发数据也可使用查询方式来收发数据。主要根据程序的设计。在 SPCE061A 中串行口为 B 口的 IOB7（RXD）和 IOB10（TXD）两个复用脚。如图 5.4。

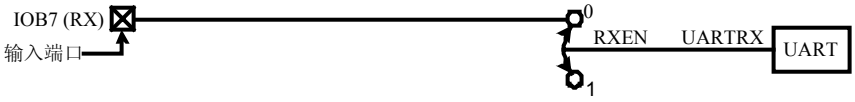

图5.4 UART 接收数据的方式

5.2.1.5 触键唤醒中断源

当系统给出睡眠命令时，CPU 便关闭 PLL 倍频电路，停止 CPU 时钟工作而使系统进入睡眠状态，在睡眠过程中，通过 IOA 口低 8 位接的键盘就可以给出唤醒信号使系统接通 PLL 倍频电路，启动 CPU 时钟工作，将系统从睡眠状态转到工作状态。与此同时，产生一个 IRQ3 中断请求。进入键唤醒中断，CPU 继续执行下一个程序指令。一般来讲，中断系统提供的中断源 FIQ(TMA)、IRQ1～IRQ7 均可作为系统的唤醒源来用，做定时唤醒系统。

若以触键作为唤醒源，其功能通过并行 A 口的 IOA0～IOA7 及中断源 IRQ3_KEY 的设置来实现。

5.2.1.6 时基信号中断源

时基信号发生器的输入信号来自实时时钟 32768Hz；输出有通过选频逻辑的 TMB1、TMB2 信号和直接从时基计数器溢出而来的各种实时时基信号。当开启时基信号中断后，有时基信号到来，发出时基信号中断申请，CPU 查询到有中断请求后，允许中断并置位 P_INT_Ctrl 中相应的中断请求位，在中断服务程序中通过测试 P_INT_Ctrl 来确定是那个频率时基信号产生的中断，可以通过在计数不同频率的时基信号来做长时间或短时间的定时控制。

5.2.2 中断控制

SPCE061A 单片机有多个中断源，为了使每个中断源都能独立地被开放和屏蔽，以便用户能灵活使用，它在每个中断信号的通道中设置了一个中断屏蔽触发器，只有该触发器无效，它所对应的中断请求信号才能进入 CPU，即此类型中断开放。否则即使其对应的中断请求标志位置“1”，CPU 也不会响应中断，即此类型的中断被屏蔽。同时 CPU 内还设置了一个中断允许触发器，它控制 CPU 能否响应中断。

5.2.2.1 中断控制寄存器

SPCE061A 对中断源的开放和屏蔽，以及每个中断源是否被允许中断，都受中断允许寄存器 P_INT_Ctrl 和 P_INT_Clear 及 P_INT_Ctrl_New 控制和一些中断控制指令。

1.中断控制单元 P_INT_Ctrl（读/写）（7010H）

P_INT_Ctrl 控制单元具有可读和可写的属性，其读写时的意义是不同的。

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_Fosc/1024	IRQ0_Fosc/1024	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

当写中断控制单元中的某位为“1”时，即允许该位所代表的中断被开放，并关闭屏蔽中断触发器，此时当有该中断申请时，CPU 会响应。否则如果该位被置 0 则禁止该位所代表的中断。即使有中断申请，CPU 也不会响应。

当读取中断控制单元时，其主要作为中断标志，因为其每一位均代表一个中断，当 CPU 响应某中断时，便将该中断标志置“1”，即将 P_INT_Ctrl 中的某位置“1”，可以通过读取该寄存器来确定 CPU 响应的中断。

2．清除中断标志控制单元 P_INT_Clear（写）（7011H）

清除中断标志控制单元主要用于清除中断控制标志位，当 CPU 响应中断后，会将中断标志置位为“1”，当进入中断服务程序后，要将其控制标志清零，否则 CPU 总是执行该中断。

b7	B6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_Fosc/1024	IRQ0_Fosc/1024	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

因为 P_INT_Clear 寄存器的每一位均对应一个中断，所以如果想清除某个中断状态标志，只要将该寄存器中对应的中断位置 1 即可清除该中断状态标志位。该寄存器只有写的属性，读该寄存器是无任何意义的。

3.激活和屏蔽中断控制单元 P_INT_Ctrl_New(读/写)($702DH)

该单元用于激活和屏蔽中断。

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

b8	b9	b10	b11	b12	b13	b14	b15
IRQ3	IRQ3	IRQ2	FIQ	IRQ1	FIQ	IRQ0	FIQ

当写该控制单元时，与 P_INT_Ctrl 功能相似。

读该控制单元时，只作为了解激活那一中断的功能使用。与其写入值是一致的。

5.2.2.2 中断控制配置端口

P_INT_Ctrl_New(写)	P_INT_Ctrl (读)	P_INT_Clear(写)	功能
1	—	—	允许中断/唤醒功能
0	—	—	屏蔽中断/唤醒功能，但不清除 P_INT_Ctrl (读)单元相应的中断标志位
—	1	—	有中断事件发生
—	0	—	没有中断事件发生
—	—	1	清除中断事件
—	—	0	不改变中断源的状态

5.2.2.3 中断控制指令

1．FIQ ON

功能:用来开通FIQ中断(FIQ的总中断允许开),其控制指令不能代替P_INT_Ctrl,也就是说即使在程序中，写了该代码但是没有在P_INT_Ctrl寄存器中FIQ处置位1，CPU无法响应该中断。FIQ ON与FIQ OFF配对使用的。

例：

错误例子：

```
.INCLUDE  hardware.inc
.INCLUDE  A2000.inc
.CODE
.PUBLIC  _main
_main:
 FIQ  ON      //只使用这个控制指令无法打开 FIQ 中断
loop:goto  loop;
```

正确例子：

```
.INCLUDE  hardware.inc
.INCLUDE  A2000.inc
.CODE
.PUBLIC  _main
_main:
FIQ  OFF
R1  =  0x8000;
[P_INT_Ctrl]  =  R1;        //只有在 P_INT_Ctrl 开放 FIQ 中断才可以响应 FIQ 中断
FIQ  ON
loop:goto  loop;:
```

2．FIQ OFF

功能：这个指令用来屏蔽FIQ中断。该指令可以屏蔽P_INT_Ctrl控制寄存器打开的FIQ中断；

例：

```
.INCLUDE  hardware.inc
.INCLUDE  A2000.inc
.CODE
.PUBLIC  _main
_main:
 FIQ  ON
 R1  =  0x8000;
 [P_INT_Ctrl]  =  R1;      //开放 FIQ 中断
 FIQ  OFF                  //闭 FIQ 中断
 FIQ  ON                   //开放 FIQ 中断
 loop:goto  loop;
```

3．IRQ ON（IRQ的总中断允许开）

功能：这个指令用来开放IRQ中断，该控制指令不能代替P_INT_Ctrl，与FIQ ON

相同。必须通过 P_INT_Ctrl 来开通中断，其与 IRQ OFF 是对应使用的。

4．IRQ OFF

功能：这个指令是用来屏蔽 IRQ 中断。与 FIQ OFF 相同，可以屏蔽 P_INT_Ctrl 开放的中断，并通过 IRQ ON 来打开。

5．INT

功能：这个指令用来设置允许/禁止 FIQ 和 IRQ 中断。该控制指令与前面的指令相同，只有先通过 P_INT_Ctrl 寄存器来打开中断通道。INT 控制指令还可以以细分为：

i. INT FIQ 功能：允许 FIQ 中断，关闭 IRQ 中断。
ii. INT IRQ 功能：允许 IRQ 中断，关闭 FIQ 中断。
iii. INT FIQ，IRQ 功能：允许 FIQ 中断，允许 IRQ 中断。
iv. INT OFF 功能：关闭 FIQ 中断，关闭 IRQ 中断。

例如：

```
.INCLUDE  hardware.inc
.INCLUDE  A2000.inc
.CODE
.PUBLIC  _main
_main:
 R1=0x8004                  //开中断 IRQ5_2Hz 和 FIQ_PWM
 [P_INT_Ctrl]=R1
INT  FIQ                    //屏蔽 IRQ5 中断
 INT  IRQ                   //允许 IRQ
 INT  FIQ，IRQ              //允许 IRQ ，FIQ
 INT  OFF                   //屏蔽 IRQ 和 FIQ
  loop:goto  loop;
```

表5.2 中断控制指令一览表

格式	指令长度	运行周期	说明
FIQ ON FIQ OFF	1	3	这两条指令用来控制 FIQ 中断的开通和禁止
IRQ ON IRQ OFF	1	3	这两条指令用来控制 IRQ 中断的开通和禁止
INT FIQ	1	3	允许 FIQ 中断关 IRQ 中断
INT IRQ			允许 IRQ 中断关 FIQ 中断
INT FIQ,IRQ			允许 FIQ,IRQ 中断
INT OFF			禁止 FIQ,IRQ 中断

5.2.2.4 中断优先级

SPCE061A 单片机中，快速中断的优先级高于普通中断的优先级，在 IRQ 中断中 IRQ1 的中断优先级高于 IRQ2，IRQ2 的中断优先级高于 IRQ3，按照 IRQ 的序号，序号越高则中断优先级越低，UART 的中断优先级最低。在 IRQ 中断中，只是中断查询有先后，不能进行中断嵌套。同中断向量内的中断源中断优先级相同。

中断优先级关系如下：

FIQ>（IRQ0>IRQ1>IRQ2>IRQ3>IRQ4>IRQ5>IRQ6 同级）>UART IRQ

如表 5.3

表5.3　SPCE061A 中断的优先级别

中断向量[注]	中断优先级别
FFF7H（复位向量）	RESET
FFF6H	FIQ
FFF8H	IRQ0
FFF9H	IRQ1
FFFAH	IRQ2
FFFBH	IRQ3
FFFCH	IRQ4
FFFDH	IRQ5
FFFEH	IRQ6
FFFFH	UART IRQ

注：所谓中断向量即指向中断服务子程序入口地址的指针。

5.2.3 中断响应

5.2.3.1 中断响应过程

从中断请求发生到被响应,从中断响应到转向执行中断服务程序,完成中断所要求的操作任务,是一个复杂的过程。整个过程都是在 CPU 的控制下有序进行的,下面按顺序叙述 SPCE061A 单片机中断响应过程。

1．中断查询

SPCE061A 的设计思想是把所有的中断请求都汇集到 P_INT_Ctrl 和 P_UART_Command2（该寄存器用于检测串行传输中断标志位）寄存器中。其中外中断是使用采样的方法将中断请求锁定在 P_INT_Ctrl 寄存器的相应标志位中，而音频输出中断、触键唤醒、定时中断、时基中断、串行异步中断的中断请求由于都发生在芯片的内部，可以直接去置位 P_INT_Ctrl 和 P_UART_Command2 中各自的中断请求标志，不存在采样的问题，所谓查询就是由 CPU 测试 P_INT_Ctrl 和 P_UART_Command2 中各标志位的状态，已确定有没有中断请求发生以及是哪一个中断请求，中断请求汇集使中断查询变得简单，因为只需对两寄存器查询即可。

SPCE061A 中断查询发生在每一个指令周期结束后，按中断优先级顺序对中断请求进行查询，即先查询高级中断后，再查询低级中断，即先查询 FIQ 再查询 IRQ，同级中断按 IRQ0→IRQ1→IRQ2→IRQ3→IRQ4→IRQ5→IRQ6→UART 的顺序查询。如果查询到有标志位为“1”，则表明有中断请求发生。因为中断请求是随机的发生的，CPU 无法预先得知，因此在程序执行过程中，中断查询要在每个指令结束后不停的进行。

2．中断响应

中断响应就是 CPU 对中断源提出的中断请求的接受，是在中断查询后进行的，当查询到有效的中断请求时，紧接着就进行中断响应。中断响应的主要内容可以理解为是硬件自动生成一条调用指令，其格式为 CALL addR16，这里的 addR16 就是存储器中断区中相应中断入口地址。在 SPCE061A 单片机中，这些入口地址已经由系统设定，例

如，对于时基信号 2Hz 中断的响应，产生的调用指令为：

```
CALL     0xFFFD
```

生成 CALL 指令后，紧接着就由 CPU 执行，首先将程序计数器 PC 的内容压入堆栈，然后，再将 SR 压入堆栈，以保护断点，再将中断入口地址装入 PC，使程序执行转向相应的中断区入口地址，调用中断服务程序。

中断响应是有条件的，并不是查询到所有中断请求都能被立即响应，当存在下列情况时，中断响应被封锁：

• CPU 正处在为一个同级或高级的中断服务中。因为当一个中断被响应时，要求把对应的优先级触发器置位，封锁低级和同级中断。

5.2.3.2 中断响应时间

中断响应的时间应首先从中断信号出现到 CPU 响应的时间与 CPU 响应中断信号到进入中断服务程序的时间之和。首先中断信号出现，CPU 查询到后，再执行下一条指令结束后去响应中断，这个时间可以根据指令周期长短来确定;一般指令周期最长为 182 个时钟周期，原因是累乘加指令需要的时间最长为 182 个时钟周期；其次 CPU 响应中断后，到 CPU 执行中断服务程序又需要 8 个时钟，原因是需要堆栈 PC 指针和 SR 寄存器及将中断向量赋值给 PC 及跳转到中断服务程序,这些操作共需要 8 个时钟周期。

因此,SPCE061A 从中断信号出现到进入中断服务最长需要 190 个时钟周期。当然，如果出现有同级或高级中断正在响应或服务中须等待的时候，那么响应时间是无法计算的。

在一般应用情况下，中断响应时间的长短通常无需考虑。只有在精确定时应用场合，才需要知道中断响应时间，以保证定时的精确控制。

5.2.3.3 中断请求的撤销

中断响应后，P_INT_Ctrl 和 P_UART_Command2 中的中断请求标志应及时清除。否则就意味着中断请求仍然存在，弄不好就会造成中断的重复查询和响应，因此就存在一个中断请求的撤销问题。在 SPCE061A 中断中，中断撤销只是标志位的置“0”问题。SPCE061A 中断除 UART 中断外，所有的中断均需软件清除标志位，即将 P_INT_Ctrl 中相应的中断位清零。即可将中断请求撤销。而 UART 中断，则是硬件自动清零，不需要软件操作。如当接收到数据后，P_UART_Command2 中的接收标志位自动置“1”，进入 UART 中断，在 UART 中断中读出数据，P_UART_Command2 相应的中断标志位自动清零。

5.2.4 中断服务流程

SPCE061A 单片机的中断服务流程图 5.5所示

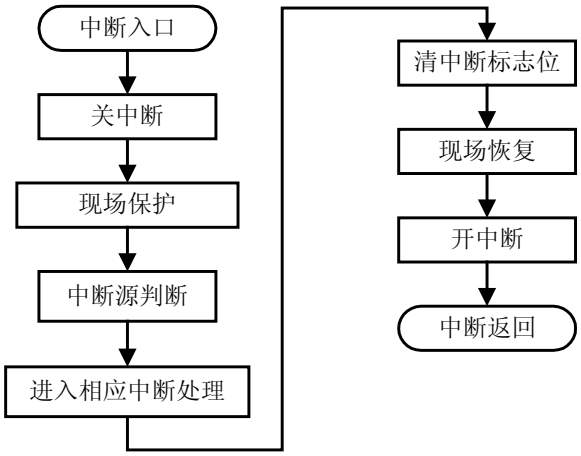

图5.5 中断服务流程图

1.中断入口

所谓中断的入口即中断的入口地址，每个中断源都有自己的入口地址，如表 5.4。

表5.4 中断入口地址表

中断向量[注]	中断优先级别
FFF7H（复位向量）	RESET
FFF6H	FIQ
FFF8H	IRQ0
FFF9H	IRQ1
FFFAH	IRQ2
FFFBH	IRQ3
FFFCH	IRQ4
FFFDH	IRQ5
FFFEH	IRQ6
FFFFH	UART IRQ

当 CPU 响应中断后，就是通过中断入口地址进入中断服务程序。

2.关中断和开中断

在一个中断执行过程中又可能有新的中断请求，但对于重要的中断必须执行到底，不允许被其它的中断所嵌套。如 FIQ 中断，对此，可以采用关闭中断的方法来解决，如在 IRQ 中断中不允许 FIQ 中断嵌套，就可以在 IRQ 中断中关闭中断，当中断服务程序执行结束后，再打开中断，去响应 FIQ 中断。即在现场保护之前先关闭中断系统，彻底屏蔽其它中断请求，待中断处理完成后再打开中断系统。

还有一种情况是中断处理可以被打扰，但现场的保护和恢复不允许打扰，以免现场被破坏，为此应在现场保护和现场恢复的前后进行开关中断。这样做的结果是除现场保护和现场恢复的片刻外，仍然保持着系统中断嵌套功能，对于 SPCE061A 单片机中断的开和关可通过中断控制指令来控制 IRQ 和 FIQ 中断，如果想实现单个中断源的控制，可以通过 P_INT_Ctrl 控制寄存器来置位和清位来打开或关闭某个中断。

3．现场保护和现场恢复

所谓现场是指中断时刻单片机中存储单元中的数据状态。为了使中断服务的执行不破坏这些数据或状态，以免在中断返回后影响主程序的运行，因此要把它们送入堆栈中保存起来，这就是现场保护。现场保护一定要位于中断处理程序的前面。中断服务结束后，在返回主程序前，则需要把保存的现场内容从堆栈中弹出，以恢复那些存储单元的原有内容，这就是现场恢复。现场恢复一定要位于中断处理程序的后面。

4．中断源判断

因为 SPCE061A 中断源多于中断入口地址，所以当 CPU 响应中断后，经中断入口地址进入中断服务程序，通过读 P_INT_Ctrl 可判断产生中断请求的中断源。

5.中断处理

中断处理是中断服务程序的核心内容，是中断的具体目的。

6.清中断标志位

因为 CPU 是根据中断标志位来判断并进行响应中断的，除串口中断外，所有的中断标志位不是靠硬件清除，而是软件清除的，所以在中断服务程序中，必须将中断标志清除，否则 CPU 总是会响应该中断的。清除标志位只要在中断服务程序中即可，位置不是固定，如也可以在中断处理程序前清除中断标志。

7.中断返回

中断服务程序最后一条指令必须是中断返回指令 RETI，当 CPU 执行这条指令时，从堆栈中弹出断点 PC、及 SR，恢复断点重新执行被中断的程序。

5.3 中断系统的应用

5.3.1 单中断源的应用

5.3.1.1 定时器中断

定时器中断包括定时器 A 中断和定时器 B 中断，而且定时器 A，B 中断源不仅在 FIQ 中断方式中有，在 IRQ1（TimerA），IRQ2（TimerB）中也有，当然这可以根据具体程序需要，如果需要定时器的中断优先级高，即可以打开 FIQ 方式的定时器中断。如果在不要求较高的中断优先级，则可以将定时器中断放在 IRQ 中断方式中。

定时中断所使用的寄存器：

配置单元	读写属性	存储地址	配置单元功能说明
P_TimerA_Data	读/写	700AH	16 位定时器/计数器 TimerA 的预置计数初值存储单元
P_TimerA_Ctrl	读/写	700BH	TimerA 的控制单元，可进行时钟源 ClkA、ClkB 输入选择和脉宽调制占空比输出 APWMO 的控制
P_TimerB_Data	读/写	700CH	16 位定时器/计数器 TimerB 的预置计数初值存储单元
P_TimerB_Ctrl	读/写	700DH	TimerB 的控制单元，可进行时钟源 ClkA 输入选择和脉宽调制占空比输出 BPWMO 的控制

P_INT_Ctrl	读/写	7010H	写可控制各中断源允许或禁止中断，读可判断产生中断请求的中断源
P_INT_Clear	写	7011H	用来清除中断源的中断请求

举例：利用定时器 A 定时 10ms，在 A 口的 IOA0 脚输出周期 20ms 的方波。

举例分析：采用定时器 A 定时，首先解决使用那种中断方式，是 FIQ 中断方式，还是 IRQ 中断方式。当然在这个例子中，采用哪一种中断都可以，这里我们采用 IRQ 中断，即开中断时需打开 IRQ1 中断即定时器 A 中断；即将 P_INT_Ctrl 的 IRQ1_TMA 置位。其次考虑定时 10ms 的问题；确定定时 10ms，首先考虑采用的时钟源（P_TimerA_Ctrl），这里采用 8kHz 的时钟源 A，时钟源 B 为 1。当然选用其他频率也可以定时 10ms；接下来确定定时器 A 的预置数(P_TimerA_Data)是多少?

计算方法：0xFFFF-时钟源频率/定时器溢出频率 = 0xFFFF-8k/0.1k = 0xFFFF-80 = 0xFFAF

主程序流程图

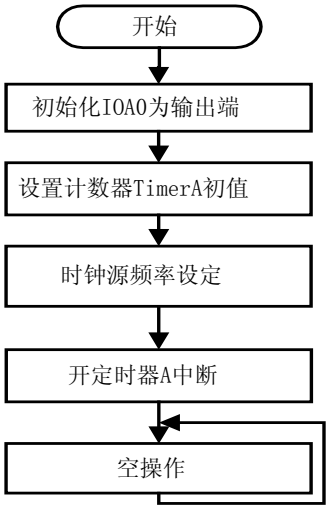

图5.6 定时器中断主程序流程图

中断服务程序

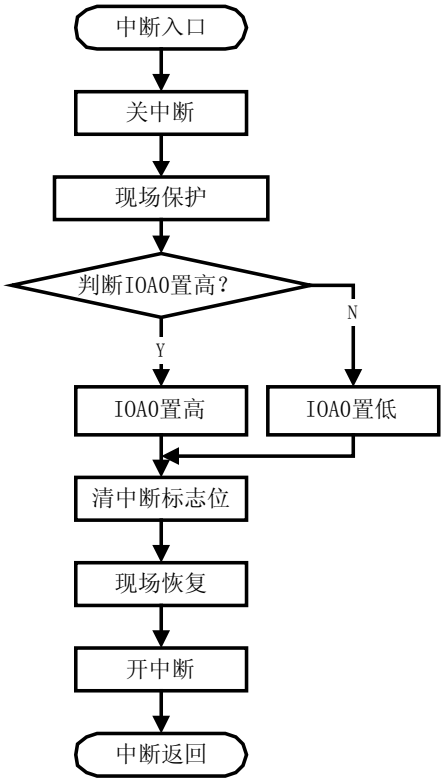

图5.7 定时器中断服务流程图

程序5-1定时中断源程序代码：

```
.INCLUDE    hardware.inc
.DEFINE     TIMERA_CLKA_8K      0x0003; //时钟源 A 选择 8096Hz
.DEFINE     TIMERA_CLKB_1       0x0030; //时钟源 B 选择 1;
.DEFINE     RUN_TIMERA          0x2000  //定义启动定时器
.DEFINE     TIMER_DATA_FOR_8KHZ (0xffff -0x1fff)
        //公式:0xFFFF-时钟源频率/定时器溢出频率 = 0xFFFF-8k/0.1k = 0xFFAF
.RAM
.VAR    C_Flag                          //方波标识,1,IOA0 为高,0,IOA0 为低
.CODE
//=========================================================================
// 函数： main()
//=========================================================================
.PUBLIC _main
_main:
    R1 = 0x0001;                        //定义 IOA0 为高电平输出端
    [P_IOA_Dir] = R1;
```

```
    [P_IOA_Attrib] = R1;
    [P_IOA_Data] = R1;
    [C_Flag] = R1;                          //初始化 10ms 方波输出标识位
    R1 = TIMER_DATA_FOR_8KHZ                //定义定时器 A 预置数
    [P_TimerA_Data]=R1
    R1 = TIMERA_CLKA_8K + TIMERA_CLKB_1//时钟源频率时钟源 A 为 8096Hz    [P_
TimerA_Ctrl]=R1                            //开放定时器 A 的中断
    R1 = RUN_TIMERA
    R1 |= 0x0010
    [P_INT_Ctrl]=R1
    INT IRQ,FIQ                             //开 IRQ 中断
L_Loop:                                     //空操作
    NOP;
    GOTO L_Loop;
//=========================================================================
//函数: FIQ()
//语法：void FIQ（void）
//描述：定时器 A 中断程序
//参数：无
//返回：无
//=========================================================================
.TEXT
.PUBLIC _FIQ;
_FIQ:
   PUSH R1,R5 TO [SP]                       //现场保护
   R1 = [C_Flag]
   [P_IOA_Data]=R1                          //波形输出端
   R1 ^= 0xffff;                            //方波标识位取反
   [C_Flag] = R1;
   R1=0x1000                                //清中断
   [P_INT_Clear]=R1
   POP R1,R5 FROM [SP]                      //恢复现场
   RETI                                     //返回
.PUBLIC _IRQ4;
_IRQ4:
   NOP
   NOP
   NOP
L_L1:
   NOP
   JMP L_L1;
   R4 = 100;
L_Loop1:
   R4 -= 1;
```

```
JNZ L_Loop1;
R2 = 0x0;
[P_IOB_Data] = R2;
R4 = 0x0040;
[P_INT_Clear] = R4;
RETI;
```

5.3.1.2 时基中断

SPCE061A 单片机具有时基中断，减少了软硬件关于实时时钟处理过程。时基信号频率丰富有 2Hz、4Hz、8Hz、16Hz、32Hz、64Hz、128Hz、256Hz、512Hz、1024H、2048Hz、4096Hz 等多种频率，为用户在实时处理上提供了各种时钟选择。

时基中断源 7 个，占有三个中断入口地址。

举例：定时 0.5s，使 A 口的 8 个二极管闪烁。

举例分析：首先考虑定时 0.5s 采用哪个时基信号比较方便，我们可以很明显的看出 2Hz 时基信号中断是最方便的。只要触发 2Hz 的时基信号中断即为 0.5s 的定时时间。

时基中断使用的控制寄存器

配置单元	读写属性	存储地址	配置单元功能说明
P_TimeBase_Setup	写	700EH	时基信号发生器输出的 *TMB1*，*TMB2* 频率设定
P_TimeBase_Clear	写	700FH	时基信号发生器复位并进行时间的精确校准
P_SystemClock	写	7013H	系统时钟的选择控制单元
P_INT_Ctrl	读/写	7010H	写可控制各中断源允许或禁止中断，读可判断产生中断请求的中断源
P_INT_Clear	写	7011H	用来清除中断源的中断请求

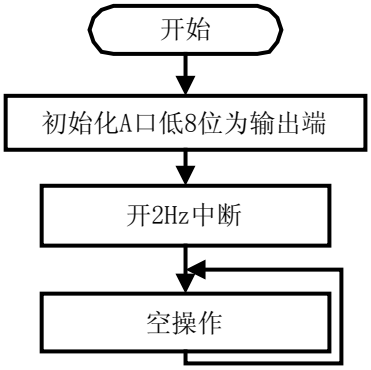

图5.8 主程序流程图

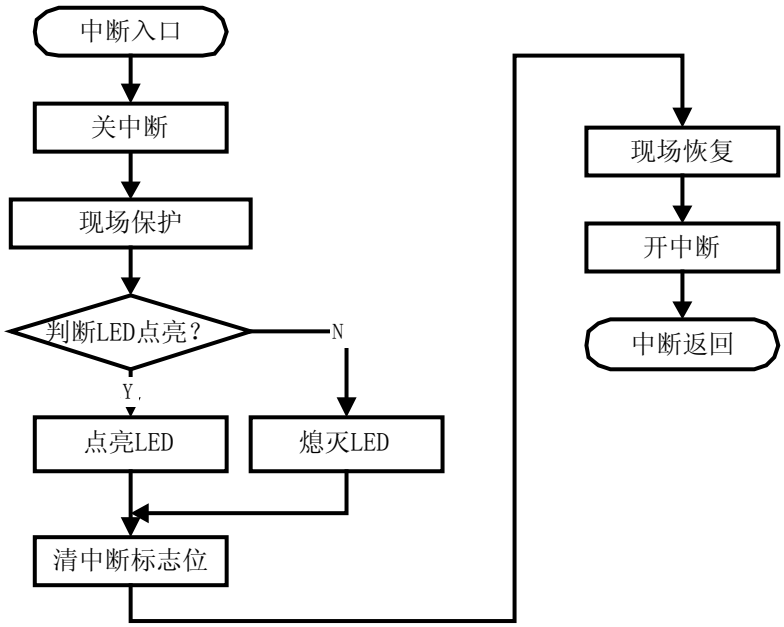

图5.9 中断服务程序流程图

程序5-2时基信号中断程序代码：

```
//*************************************************************************************/
// 名称：  TimeBase_2Hz
// 描述：  时基信号中断,使 A 口低 8 位发光二极管 0.5 秒闪烁,IOA0-IOA7 分别接 8 个发光二极管
// 日期：  2002/12/10
//*************************************************************************************/
.INCLUDE  hardware.inc
.DEFINE   RUN_2HZ_TimeBase_INT   0x0004
.RAM
.VAR  C_Flag                              //发光二极管的标识 1,点亮 2.熄灭

//=========================================================================
// 函数：  main()
// 描述：主函数
//=========================================================================
.CODE
.PUBLIC  _main
_main:
      R1=0x00ff;                          //初始化 A 口低 8 位为低电平输出端
      [P_IOA_Attrib]=R1 ;
      [P_IOA_Dir]=R1 ;
      R1=0x0000 ;
      [P_IOA_Data]=R1 ;
```

```
    [C_Flag] = R1;                          //初始化发光二极管标识
    R1 =  RUN_2HZ_TimeBase_INT;             //开放 2Hz 中断
    [P_INT_Ctrl]=R1 ;
    INT IRQ  ;
L_Loop:
    NOP ;
    NOP;
    NOP;
    NOP;
    GOTO L_Loop

//=========================================================================
//函数: IRQ5()
//语法：void IRQ5（void）
//描述：2Hz 时基中断示例，使发光二极管 0.5s 点亮一次
//参数：无
//返回：无
//=========================================================================
.TEXT
.PUBLIC _IRQ5
_IRQ5:
    PUSH R1,R5 TO [SP];                     //保护现场
    R1 = 0x0004;
    TEST R1,[P_INT_Ctrl];                   //比较是否为 2Hz 的中断源
    JNZ  L_IRQ5_2Hz ;                       //是，则转至对应程序段
L_IRQ5_4Hz:                                 //4Hz 中断的处理
    R1=0x0008 ;
    GOTO L_Exit_INT;
 L_IRQ5_2Hz:                                //2Hz 中断的处理
    R1 = [C_Flag];                          //发光二极管标识
    [P_IOA_Data] = R1 ;
    R1 ^= 0xffff;
    [C_Flag] = R1;
    R1 = 0x0004;
L_Exit_INT:                                 //退出中断
    [P_INT_Clear]=R1 ;
    POP R1,R5 FROM [SP] ;                   //恢复现场
    RETI;
```

5.3.1.3 触键唤醒中断

触键唤醒中断源主要是在系统进入睡眠状态后，通过 A 口低八位的按键来唤醒系统时钟。同时进入触键唤醒中断。恢复睡眠时的 PC 指针。

举例：使系统进入睡眠状态，通过触键唤醒。

举例分析：首先考虑如何使系统进入睡眠状态，可以通过设置时钟系统控制寄存器（P_SystemClock）的 B4 位使系统进入睡眠状态。其次因为只有 A 口的低 8 位具有唤醒功能，所以按键必须接 A 口低 8 位。

触键唤醒中断使用的控制寄存器

配置单元	读写属性	存储地址	配置单元功能说明
P_IOA_RL	读	7004H	从其读数可激活 A 口的唤醒功能，并锁存 IOA0~IOA7 上的键状态
P_SystemClock	写	7013H	系统时钟的选择控制单元
P_INT_Ctrl	读/写	7010H	写可控制各中断源允许或禁止中断，读可判断产生中断请求的中断源
P_INT_Clear	写	7011H	用来清除中断源的中断请求

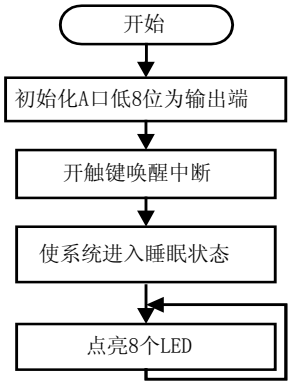

图5.10 触键唤醒主程序流程图

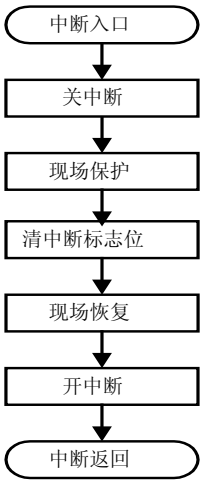

图5.11 触键唤醒中断服务流程图

程序5-3触键唤醒程序代码：

```
//*************************************//
//描述：检测触键唤醒的功能
//硬件连接:A 口低 8 位接键盘;高 8 位接 8 个 LED
//*************************************//
 .INCLUDE  hardware.inc
.DEFINE  P_IOA_RL    0x7004
.DEFINE  RUN_KEYWAKEUP_INT  0x0080
.CODE

//=============================================================================
// 函数：  main()
// 描述：主函数
//=============================================================================
.PUBLIC  _main
_main:
     R1=0xff00                    //初始化 A 口低 8 位为带下拉电阻的输入
     [P_IOA_Dir]=R1               //高 8 位为低电平输出
     [P_IOA_Attrib]=R1
      R1  =  0x0000
      [P_IOA_Data]=R1
     R1  =  RUN_KEYWAKEUP_INT     //开放触键唤醒中断
     [P_INT_Ctrl]  =  R1
     R1=[P_IOA_RL]                //激活 A 口唤醒功能
     INT  IRQ                     //开中断
     R1=0x0007
     [P_SystemClock]=R1           //进入睡眠状态
L_Loop:                           //当有键唤醒时继续执行程序
     R1  =  [P_IOA_Data]
     R1  |=  0xff00;
     [P_IOA_Data]  =  R1;         //点亮 8 个发光二极管
     GOTO  L_Loop;


//=============================================================================
//函数: IRQ3()
//语法：void  IRQ3（void）
//描述：键唤醒中断
//参数：无
//返回：无
//=============================================================================
.TEXT
.PUBLIC  _IRQ3
_IRQ3:
```

```
    INT OFF
    PUSH R1,R4 TO [SP]          //现场保护
    R1 = 0x0080
    TEST R1,[P_INT_Ctrl]        //是否为键唤醒中断
    JZ L_notKeyArouse           //否，外部中断
L_KeyArouse:                    //触键唤醒中断的处理
    R1 = 0x0080
    GOTO L_Exit_INT;
L_notKeyArouse:                 //外部中断的处理
    R1 = 0x0100
    TEST R1,[P_INT_Ctrl]
    JNZ L_EXT1;                 //判断是否外部中断 1
    R1 = 0x0200                 //外部中断 2
L_EXT1:
L_Exit_INT:
    [P_INT_Clear]=R1            //清中断
    POP R1,R4 FROM [SP]         //恢复现场
    INT IRQ
    RETI
.END

//*****************************************************************************
// main.c 结束
//*****************************************************************************
```

5.3.1.4 外部中断

SPCE061A 有两个外部中断，为负跳沿触发。可以使用形成反馈电路定时来触发外部中断，也可以不使用反馈电路，通过给 IOB2 或 IOB3 外部中断触发信号，进入外部中断

举例：在外部中断中点亮 8 个 LED

举例分析：首先考虑使用外部中断 1 还是外部中断 2，此处两个外部中断都可以。只是初始化时略有不同，选择外部中断 1，初始化 IOB2 为带上拉电阻的输入端口，选择外部中断 2，初始化 IOB3 为带上拉电阻的输入端口位高阻输入。此例选择外部中断 1。

反馈电路外部中断使用的控制寄存器

配置单元	读写属性	存储地址	配置单元功能说明
P_INT_Ctrl	读/写	7010H	写可控制各中断源允许或禁止中断，读可判断产生中断请求的中断源
P_INT_Clear	写	7011H	用来清除中断源的中断请求
P_FeedBack	写	7009H	B 口的应用方式控制向量[2]

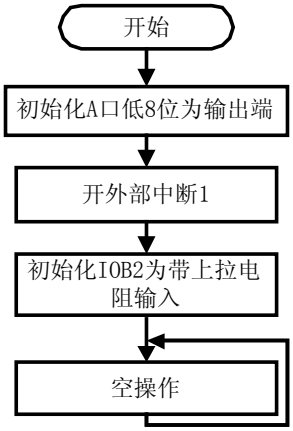

图5.12 外部中断主程序流程图

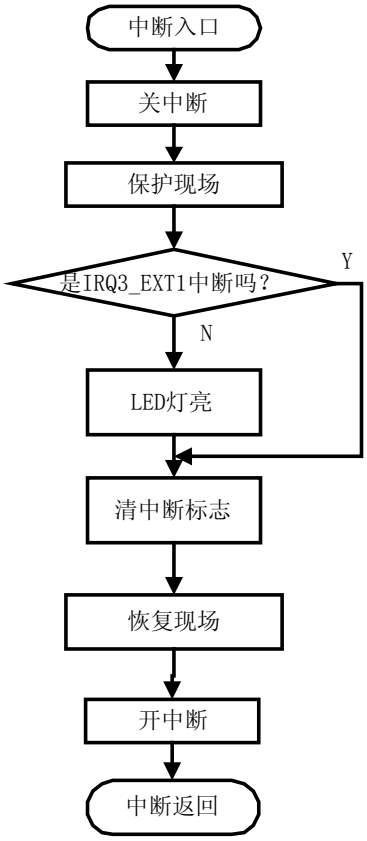

图5.13 外部中断中断服务程序流程图

程序5-4外部中断源程序代码：

```
////////////////////////////////////////
//描述：通过外部中断点亮 A 口低 8 位的 8 个 LED
//硬件连接:A 口低 8 位连接 8 个 LED
////////////////////////////////////////
    .INCLUDE  hardware.inc
    .CODE
    //=========================================================================
    // 函数：  main()
    // 描述：主函数
    //=========================================================================
    .PUBLIC  _main
    _main:
     R1=0x00ff                    //设置 A 口低 8 位为同相高电平输出口；
     [P_IOA_Attrib]=R1
     [P_IOA_Dir]=R1
         R1  =  0x0000
     [P_IOA_Data]=R1
         R1=0x0000;                    //设置 IOB2 带上拉电阻的输入端口
         [P_IOB_Dir]=R1;
     [P_IOB_Attrib]=R1;
     R1=0x0004  ;
         [P_IOB_Data]=R1;
          R1=0x0100;                   //开中断 IRQ3_EXT1
     [P_INT_Ctrl]=R1;
     INT  IRQ;
    L_Loop:
     NOP
     NOP
     NOP
     JMP  L_Loop

    //=========================================================================
    //函数: IRQ3()
    //语法：void  IRQ3（void）
    //描述：IRQ3 的应用
    //参数：无
    //返回：无
    //=========================================================================
    .TEXT
    .PUBLIC  _IRQ3
    _IRQ3:
        INT  OFF
```

```
    PUSH R1,R5 TO [SP]          //现场保护
    R1=0x0100
    TEST R1,[P_INT_Ctrl]        //比较是否为 IRQ3_EXT1
    JNZ  L_Irq3_Ext1            //是，则转至对应程序段；
    R1=0x0200
    TEST R1,[P_INT_Ctrl]        //否，则比较是否为 IRQ3_EXT2
    JNZ  L_Irq3_Ext2            //是，则转至对应程序段；
L_Irq3_Key:                     //否，则进入键唤醒中断
 GOTO L_Exit_INT;
L_Irq3_Ext2:                    //进入外部中断 2
    GOTO L_Exit_INT;
L_Irq3_Ext1:                    //外部中断 1
    R1=0xffff
    [P_IOA_Data]=R1             //点亮 LED
    R1 = 0x0100
L_Exit_INT:
    [P_INT_Clear]=R1
    pop R1,r5 from [sp]         //现场恢复
    INT IRQ,FIQ
    RETI
```

5.3.1.5 串行异步中断

串行异步中断用于串行通讯过程中数据的收发，

此外，UART 还可以缓冲地接收数据。也就是说，它可以在读取缓存器内当前数据之前接收新的数据。但是，如果新的数据被接收到缓存器之前一直未从中读取先前的数据，会发生数据丢失。P_UART_Data (读/写) ($7023H)单元可以用于接收和发送数据的缓存。向该单元写入数据，可以将发送的数据送入缓存器；从该单元读数据，可以从缓存器读出单个的数据字节。UART 模块的接收管脚 Rx 和发送管脚 Tx 分别可与 IOB7 和 IOB10 共用。

举例：准备一组数据自发自收

举例分析：允许串口中断，并设置 P_UART_Command1 允许 UART 收发中断。注意，因为 UART 串行传输方式为 8 位数据位传输，所以传输一个字时需要发送两次，接收也如此。

串行异步中断使用的控制寄存器

配置单元	读写属性	存储地址	配置单元功能说明
P_UART_Command1	写	7021H	UART 功能的控制单元
P_UART_Command2	读/写	7022H	读出时为 UART 工作的状态口。写入时用来控制 UART RX、TX 端口数据传输的开通或关断
P_UART_Data	读/写	7023H	需要接收或发送的数据
P_UART_BaudScalarLow	读/写	7024H	用于选择数据传输波特率控制字的低位
P_UART_BaudScalarHigh	读/写	7025H	用于选择数据传输波特率控制字的高位
P_INT_Ctrl	读/写	7010H	写可控制各中断源允许或禁止中断，读可判断产

			生中断请求的中断源
P_INT_Clear	写	7011H	用来清除中断源的中断请求

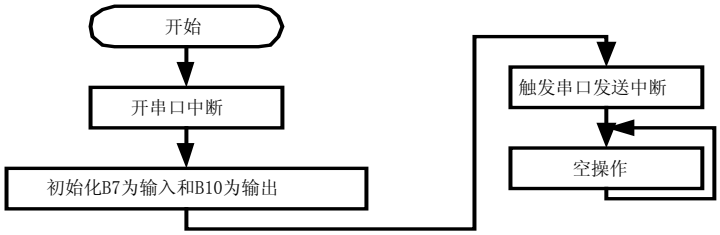

图5.14 串行异步通讯主程序流程图

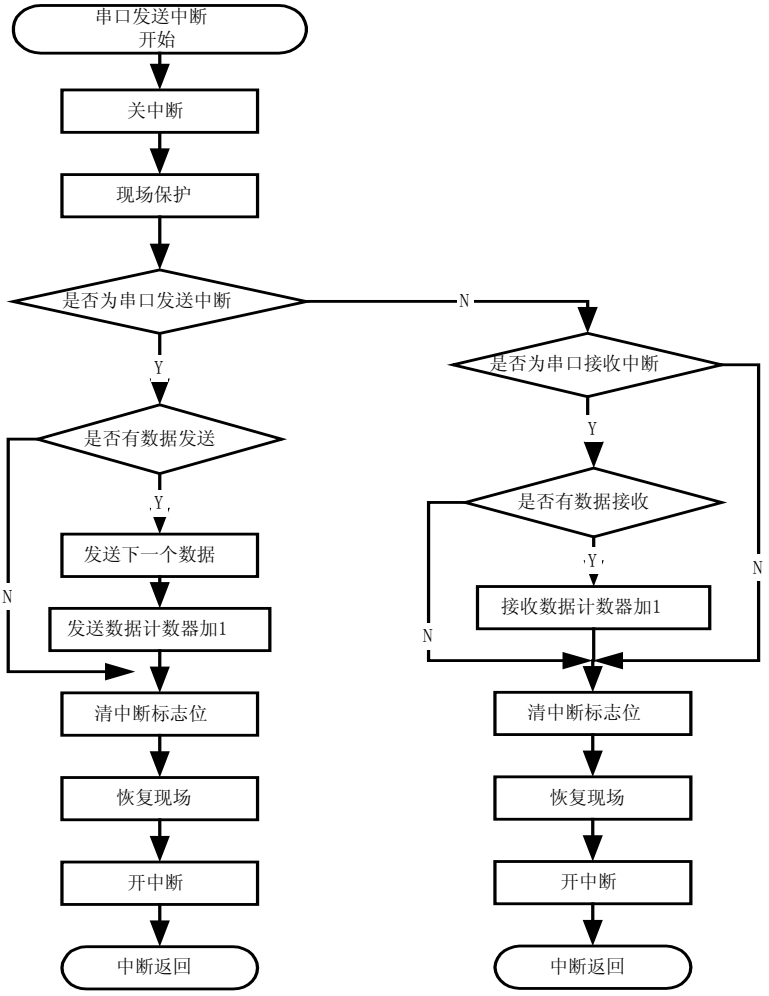

图5.15 串口异步通讯中断服务程序流程图

程序5-5串行异步通讯程序代码：

```
//=========================================//
//程序描述：异步通讯自发自收程序。
//准备 5 个数据分别是'0x55aa,0xff55,0x1010,0x3344,0x66aa'b
//=========================================//
.INCLUDE  hardware.inc
.DEFINE  UART_DATA_SIZE   0x0005;
.DATA                                   //发送的数据
C_SendData:.dw  0x55aa,0xff55,0x1010,0x3344,0x66aa
.ISRAM
.PUBLIC   C_RecData                     //接收数据缓冲区
C_RecData: .DW   5  DUP(0);
.VAR  C_RecNum                          //接收数据个数
.VAR  C_SendNum                         //发送数据个数
.VAR  C_SendFlag;                       //1,发送高 8 位，0，发送低 8 位
.VAR  C_RecFlag                         //1,接收高 8 位，0，接收低 8 位
.CODE

//=============================================================================
// 函数：  main()
// 描述：主函数
//=============================================================================
.PUBLIC  _main;
_main:
     R2  =  C_RecData;
_UART_INIT:
F_UART_INIT:
      R1  =  0x0480;                    //设置 IOB7 为输入 IOB10 为输出
      [P_IOB_Attrib]  =  R1;
      R1  =  0x0400;
      [P_IOB_Dir]  =  R1;
      R1  =  0x0000;
      [P_IOB_Data]  =  R1;
      R1  =  0x006b;                    //设置波特率  114.84KHz(~=115.2KHz)
      [P_UART_BaudScalarLow]  =  R1;
      R1  =  0x0000;
      [P_UART_BaudScalarHigh]  =  R1;
      R1  =  0x00C0;
      [P_UART_Command1]  =  R1;         //开接收发送中断
      [P_UART_Command2]  =  R1;         //使能 RX 和 TX
      R1  =  0x0000;
      [C_SendNum]  =  R1;               //初始化发送数据个数
     [C_RecNum]  =  R1;                 //初始化接收数据个数
```

```
    R1 = 0x0001 ;
    [C_SendFlag] = R1;                    //初始化发送位标识
    [C_RecFlag] = R1;                     //初始化接收位标识
    INT IRQ;                              //开中断
L_Loop:
    NOP;
    GOTO L_Loop;
//=========================================================================
//函数: IRQ7()
//语法：void IRQ7（void）
//描述：UART 服务程序，在发送中断中发送数据，接收中断中接收数据串口异步通讯每次只能收发
//一个字节的数据，所以无论是接收数据和发送数据需进行移位处理
//参数：无
//返回：无
//=========================================================================
.TEXT
.PUBLIC _IRQ7
UART_C_RecC_IRQ:.pRoc
_IRQ7:
    INT OFF
    PUSH R1,R5 TO [SP]
    R1 = 0x0080                           //判断是接收还是发送数据
    TEST R1,[P_UART_Command2]
    JNZ L_UART_C_RecV_IRQ;
L_UART_C_Send_IRQ:                        //发送数据处理
    R2 = [C_SendFlag];
    R2 ^= 0x0001
    [C_SendFlag] = R2;                    //发送位标识取反
    R1 = C_SendData;
    R4 = [C_SendNum]
    R3 = UART_DATA_SIZE
    CMP R4,R3;                            //数据是否发送结束
    JE L_Exit_INT;                        //结束，退出发送
    R1 = R1+R4                            //继续发送处理
    R1 = [R1]                             //取发送数据
    R2 = [C_SendFlag]
    JZ  L_Send_Data;                      //发送一个字的高八位
    R1 = R1 LSR 4;
    R1 = R1 LSR 4;                        //发送高 8 位
    R4 += 1;                              //发送的数据加 1
    [C_SendNum] =  R4;
L_Send_Data:
    [P_UART_Data] = R1 ;                  //发送数据
    GOTO L_Exit_INT;
```

```
L_UART_C_RecV_IRQ:
    R2 = [C_RecFlag];
    R2 ^= 0x0001                    //接收标识取反
    [C_RecFlag] = R2;
    R4 = [C_RecNum];
    R3 = UART_DATA_SIZE
    CMP R3,R2;                      //数据是否接收结束
    JE L_Exit_INT;                  //接收结束，退出接收,否则，继续接收
    R1 = [P_UART_Data] ;            //接收数据
    R2 = [C_RecFlag]                //接收低 8 位数据
    JNZ  L_Shift_Data;
    R3 = R4 + C_RecData;            //保存数据低 8 位
    [R3] = R1;
    GOTO L_Exit_INT;
L_Shift_Data:                       //接收高 8 位数据
    R1 = R1 lsl 4
    R1 = R1 lsl 4;
    R2 = [C_RecNum];
    R3 = R2 + C_RecData;            //保存数据高 8 位
    R4 = [R3]
    R4 |= R1;
    [R3] = R4;
    R2 += 1;
    [C_RecNum] = R2;                //接收数据数量加 1
L_Exit_INT:
    POP R1,R5 FROM [SP];
    INT IRQ,FIQ
    RETI;
.ENDP;
```

5.3.2 多中断源应用

在单片机软件开发中，使用多个中断源的机会有很多。在 SPCE061A 单片机中多中断源的使用有两种方式，一种是同个中断入口中断源的使用；另一种是不同中断入口的多个中断源的使用。

1．同中断向量的多个中断源使用

举例：IRQ6 中断有两个中断源 IRQ6_TMB1 和 IRQ6_TMB2，此处利用两个中断源分别控制 8 个发光二极管，分别为 1s 和 0.5s 闪烁。

举例分析：IRQ6_TMB1 有多种选择 8，16，32，64Hz 选择，可以选择其中任何频率均可做 0.5s 定时。此处我们选择 64Hz；同样 IRQ6_TMB2 有 128，256，512，1024Hz 选择，每种频率都可以达到定时 1s，此处选择 128Hz

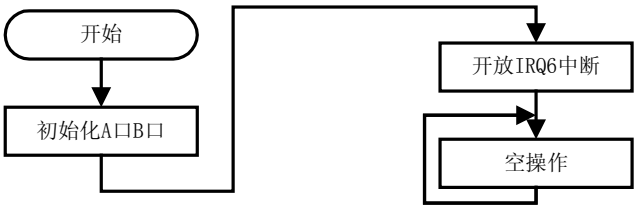

图5.16 同中断向量的多个中断源主程序流程图

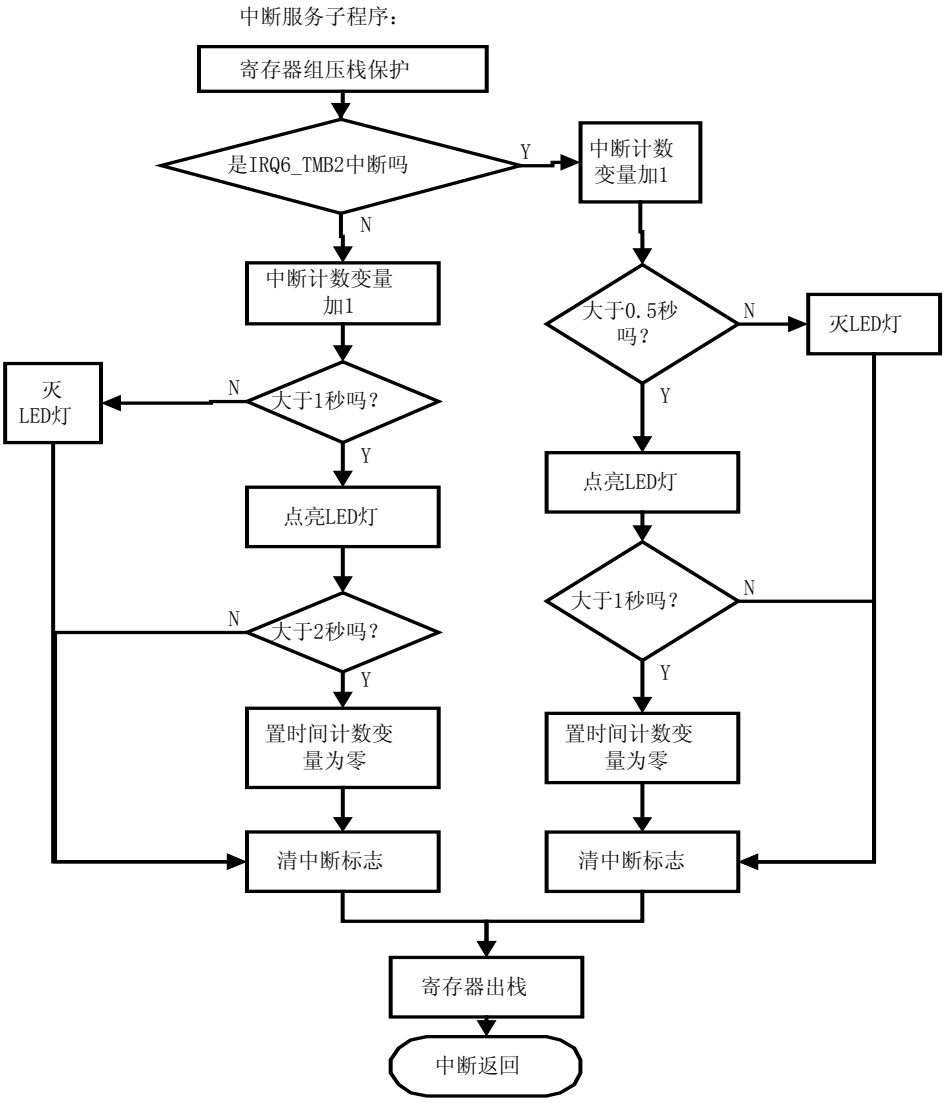

图5.17 同中断向量的多个中断源中断服务子程序

程序5-6同中断向量的多个中断源程序代码

```
//*********************************************************************************************/
// 描述： FIQ 有 FIQ_PWM、FIQ_TMA 和 FIQ_TMB 三个中断源,当定时器 A 或 B 计满溢出时产生
//中断请求信号 TA_TIMEOUT_INT 或 TA_TIMEOUT_INT,CPU 响应后进入中断执行相应的子程序
//控制二极管发光。
// 日期： 2002/12/11
//*********************************************************************************************/
.DEFINE  P_IOA_DATA        0x7000;
.DEFINE  P_IOA_DIR         0x7002;
.DEFINE  P_IOA_ATTRI       0x7003;
.DEFINE  P_IOB_DATA        0x7005;
.DEFINE  P_IOB_DIR         0x7007;
.DEFINE  P_IOB_ATTRI       0x7008;
.DEFINE  P_INT_CTRL        0x7010;
.DEFINE  P_INT_CLEAR       0x7011;
.DEFINE  P_TimerA_Data     0x700A;
.DEFINE  P_TimerA_Ctrl 0x700B;
.DEFINE  P_TimerB_Data     0x700C;
.DEFINE  P_TimerB_Ctrl 0x700D;
.DEFINE  timea_clk       0x020d;  //1024hz
.DEFINE  timeb_clk       0x0004;  //4096hz
.RAM
.VAR  TA_Flag
.VAR  TB_Flag

//=============================================================================
// 函数： main()
// 描述：主函数
//=============================================================================
.CODE
.PUBLIC  _main
_main:
      INT  OFF
      R1=0xffff                    //IOA 口为输出口；
      [P_IOA_ATTRI]=R1
      [P_IOA_DIR]=R1
      R1=0x0000
      [P_IOA_DATA]=R1
      R1=0xffff                    //B 口的低 8 位设置为输出
     [P_IOB_DIR]=R1
      [P_IOB_ATTRI]=R1
      R1=0x0000
      [P_IOB_DATA]=R1
```

```
    R1=0xff9f;
    [P_TimerA_Data]=R1;
    [P_TimerB_Data]=R1;
    R1  =timea_clk;
    [P_TimerA_Ctrl]=R1;
    R2=0x0004                    //开中断 IRQ0_TMA、IRQ1_TMA、IRQ1_TMB
    R1=0x2000                    //开中断 FIQ_PWM、FIQ_TMA、FIQ_TMB
    R1 |= R2;
    [P_INT_CTRL]=R1
    INT IRQ,FIQ;
L_Loop:
    NOP
    NOP
    NOP
    JMP L_Loop

//============================================================================
//函数: FIQ()
//语法：void FIQ（void）
//描述：FIQ 中断服务程序
//参数：无
//返回：无
//============================================================================
.TEXT
.PUBLIC _FIQ
_FIQ:
   PUSH R1,R5 TO [SP]          //压栈保护
   R1=0x0800
   R2 = [P_INT_CTRL]
   TEST R1,[P_INT_CTRL]        //比较是否为 FIQ_TMB
   JNZ  L_FIQ_TMB              //是，则转至对应程序段
   R1=0x2000
   TEST R1,[P_INT_CTRL]        //否，则比较是否为 FIQ_TMA
   JNZ  L_FIQ_TMA              //是，则转至对应程序段
L_FIQ_PWM:                     //否，则进入 FIQ_PWM 中断
    R1=0x8000
    [P_INT_CLEAR]=R1
    POP R1,R5 FROM [SP]
    RETI
L_FIQ_TMA:
   R1 = [TA_Flag]
   R1 ^= 0xffff
   [P_IOA_DATA]=R1
   [TA_Flag] = R1
```

```
    R1=0x2000
    [P_INT_CLEAR]=R1
    POP R1,R5 FROM [SP]
    RETI
L_FIQ_TMB:
    R1=0x0800
    [P_INT_CLEAR]=R1
    POP R1,R5 FROM [SP]
    RETI

//=============================================================================
//函数: IRQ5()
//语法：void IRQ5（void）
//描述：IRQ5 中断服务程序
//参数：无
//返回：无
//=============================================================================
.TEXT
.PUBLIC _IRQ5
_IRQ5:
     PUSH R1,R5 TO [SP]        //压栈保护
    R1=0x0008
     TEST R1,[P_INT_CTRL]      //比较是否为 4Hz 的中断源
    JNZ L_Irq5_4               //是，则转至对应程序段
L_Irq5_2:
     NOP;
     NOP;
     NOP;
    //R1 = 0xffff              //否，则进入 2Hz 程序段
     //[P_IOA_DATA]=R1
    JMP L_Irq5_2;
L_LED2Hz_RET:
    //R1=0x0004
     //[P_INT_CLEAR]=R1
    POP R1,R5 FROM [SP]
    RETI
L_Irq5_4:
     //[P_INT_CLEAR]=R1
    POP R1,R5 FROM [SP]
    RETI

//*****************************************************************************************/
// main.c 结束
//*****************************************************************************************/
```

5.3.3.1 不同中断入口的中断源使用

如上例，用不同中断入口的中断源举例；

举例分析：此例利用的是 0.5s 定时使用的是 IRQ2 中的定时器 B；1s 定时利用的是 IRQ4 中的 IRQ_1kHz 中断。

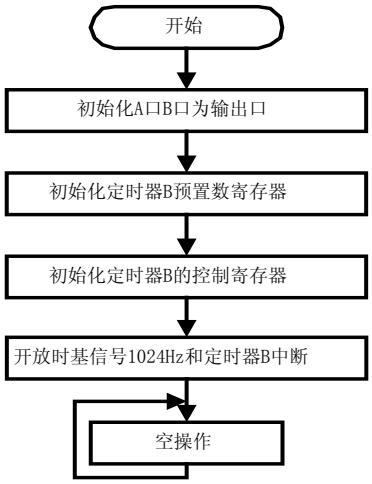

图5.18 不同中断入口的中断源主程序流程图

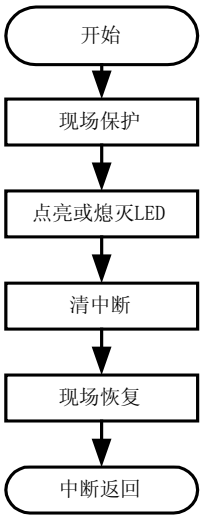

图5.19 不同中断入口的中断源定时器 B 中断服务程序流程图

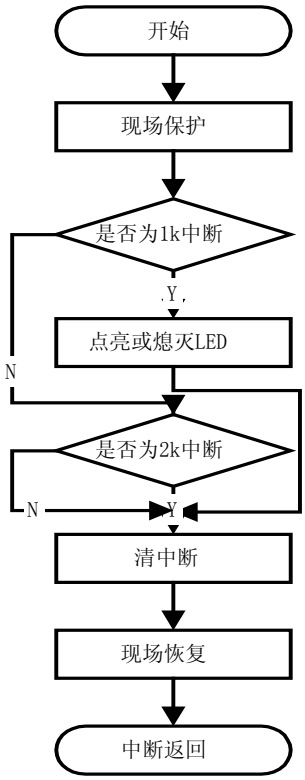

图5.20 不同中断入口的中断源时基信号 1024Hz 中断服务程序流程图

程序5-7不同中断入口的中断源程序代码

```
//*****************************************************************************************/
// 描述：  利用定时器 B  定时 0.5s  使 A 口低四位的 LED 闪烁利用时基信号 1024Hz  中断定时 1s，
// 使 IOA4--IOA7 连接的 LED 闪烁,硬件连接：IOA0--IOA4 接 4 个 LED  IOA4--IOA7 接 4 个 LED
// 日期：  2002/12/11
//*****************************************************************************************/
.INCLUDE      hardware.inc
.DEFINE       RUN_TIMERB              0x0400              //定义启动定时器
.DEFINE       TIMER_DATA_FOR_4KHZ  (0xffff  -  2048)    //定时 0.5 秒 F
.DEFINE       RUN_TIMEBASE_1024   0x0010              //时基信号 1024hz 中断位
.DEFINE       TIMER_CLKA_4096      0x0004;             //时钟源 A  选择 4096Hz
.RAM
.VAR          C_Ioa_Led,C_Iob_Led                     //C_Ioa_Led 为定时器 B  LED 亮灭数据，
                                                      // C_Iob_Led 为时基信号 LED 亮灭的数据

.VAR          C_Clock_Cnt;                            //时基信号的计数器
.CODE
```

```
//=============================================================================
// 函数：  main()
// 描述：主函数
//=============================================================================
.PUBLIC  _main
_main:
      INT  OFF
      R1=0xffff                                          //IOA 口为输出口
      [P_IOA_Attrib]=R1
      [P_IOA_Dir]=R1
      R1=0x00ff
      [P_IOA_Data]=R1
      R1=0xffff                                          //IOA 口为输出口
      [P_IOB_Attrib]=R1
      [P_IOB_Dir]=R1
      R1=0x00ff
      [P_IOB_Data]=R1
     [C_Ioa_Led]  =  R1;
     [C_Iob_Led]  =  R1;
     R1  =  TIMER_DATA_FOR_4KHZ                          //定时器 B 的预置数
     [P_TimerB_Data]=R1
     R1  =  TIMER_CLKA_4096              //定义使用的时钟源频率  时钟源 A 为 4096Hz 时
     [P_TimerB_Ctrl]=R1
     R1  =RUN_TIMEBASE_1024+RUN_TIMERB       //开放定时器 B  中断和时基信号 1024hz 中
断
     [P_INT_Ctrl]=R1
     INT  IRQ                                           //开 IRQ 中断
L_Loop:                                                 //空操作
     NOP;
     GOTO  L_Loop;

//=============================================================================
//函数: IRQ2()
//语法：void  IRQ2（void）
//描述：IRQ2 中断服务程序，使 IOA0--IOA3 位接的 4 个 LED0.5s 闪烁
//参数：无
//返回：无
//=============================================================================
.TEXT
.PUBLIC  _IRQ2;
_IRQ2:
    PUSH  R1,R5  TO  [SP]                               //现场保护
     R1  =  [C_Ioa_Led]                                 //LED  赋值
```

```
    ////////////////////
    // R1 &= 0x000f;
    // R2 = [P_IOA_Data]
    // R2 &= 0x00f0;
    // R1|=R2
    ////////////////////
    [P_IOB_Data]=R1
    R1 ^= 0xffff;
    [C_Ioa_Led] = R1;
    R1=0x0400                                   //清中断
    [P_INT_Clear]=R1
    POP R1,R5 FROM [SP]                         //恢复现场
    RETI                                        //返回

//=============================================================================
//函数: IRQ4()
//语法：void IRQ4（void）
//描述：IRQ4 中断服务程序，1024Hz 时基信号中断，使 IOA4--IOA7 接的 4 个 LED 1 秒闪烁
//参数：无
//返回：无
//=============================================================================
.TEXT
.PUBLIC _IRQ4
_IRQ4:
      PUSH R1,R5 TO [SP]                        //压栈保护；
      R1=0x0010;
      TEST R1,[P_INT_Ctrl];                     //比较是否为 1KHz 的中断源；
      JNZ L_Irq4_1k;                            //是，则转至对应程序段；
      R1=0x0020;
      TEST R1,[P_INT_Ctrl]                      //否，则比较是否为 2KHz 的中断源；
      JNZ L_Irq4_2k;                            //是，则转至对应程序段；
L_Irq4_4k:                                      //否，则进入 4KHz 程序段；
    R1 = 0x0040;
    GOTO L_Exit_Int;
L_Irq4_2k:
    GOTO L_Exit_Int;
L_Irq4_1k:
    R1 = [C_Clock_Cnt]
    CMP R1,1024;
    JE L_Led_Pro;
    R1 += 1;
    [C_Clock_Cnt] = R1;
    R1 = 0x0010
    GOTO L_Exit_Int;
```

```
L_Led_Pro:                                          //LED 赋值
    R1 = [C_Iob_Led]
    [P_IOA_Data] = R1
    R1 ^= 0xffff;
    [C_Iob_Led] = R1;
    R1 = 0x0000
    [C_Clock_Cnt] = R1;                             //清时基计数器
    R1 = 0x0010
L_Exit_Int:
    [P_INT_Clear] = R1;
    POP R1,R5 FROM [SP]
    RETI;

//*************************************************************************************************/
// main.c 结束
//*************************************************************************************************/
```