第 8 章 中断

目录

本章包括下面一些主要内容：

8.1 简介

PICmicro® 单片机有丰富的中断源。虽然某些外设模块可能产生多个中断（比如 USART 模块），但一般情况下是一个外设模块只有一个中断源。目前的中断包括：

- INT 引脚中断 （外部中断）
- TMR0 溢出中断
- PORTB 电平变化中断 （引脚 RB7:RB4）
- 比较器变化中断
- 并行从动端口中断
- USART 中断
- 接收中断
- 发送中断
- A/D 转换完成中断
- LCD 中断
- 向 EEPROM 写数据完成中断
- Timer1 溢出中断
- Timer2 溢出中断
- CCP 中断
- SSP 中断

中断的控制和中断状态的表示至少需要一个寄存器。该寄存器是：

- INTCON

此外，如果器件有外设中断，则会有允许外设中断的寄存器和保存中断标志位的寄存器。根据器件的具体型号，这些寄存器为：

- PIE1
- PIR1
- PIE2
- PIR2

我们通常称这些寄存器为 PIR 和 PIE。如果将来的器件拥有更多的中断源，则会增加新的寄存器对 （PIR3 和 PIE3）。

中断控制寄存器 INTCON 记录请求内核中断的各个中断标志位、允许位以及全局中断允许位。

全局中断允许位 GIE（INTCON<7>）置位时，允许所有未屏蔽的中断；清零时，禁止所有中断。通过 INTCON 寄存器中的允许位也能禁止各相应的中断。 GIE 位在复位时被清零。

执行 “中断返回”指令 RETFIE 将退出中断服务程序，同时将 GIE 位置 “1”，从而可响应任何暂挂的中断。

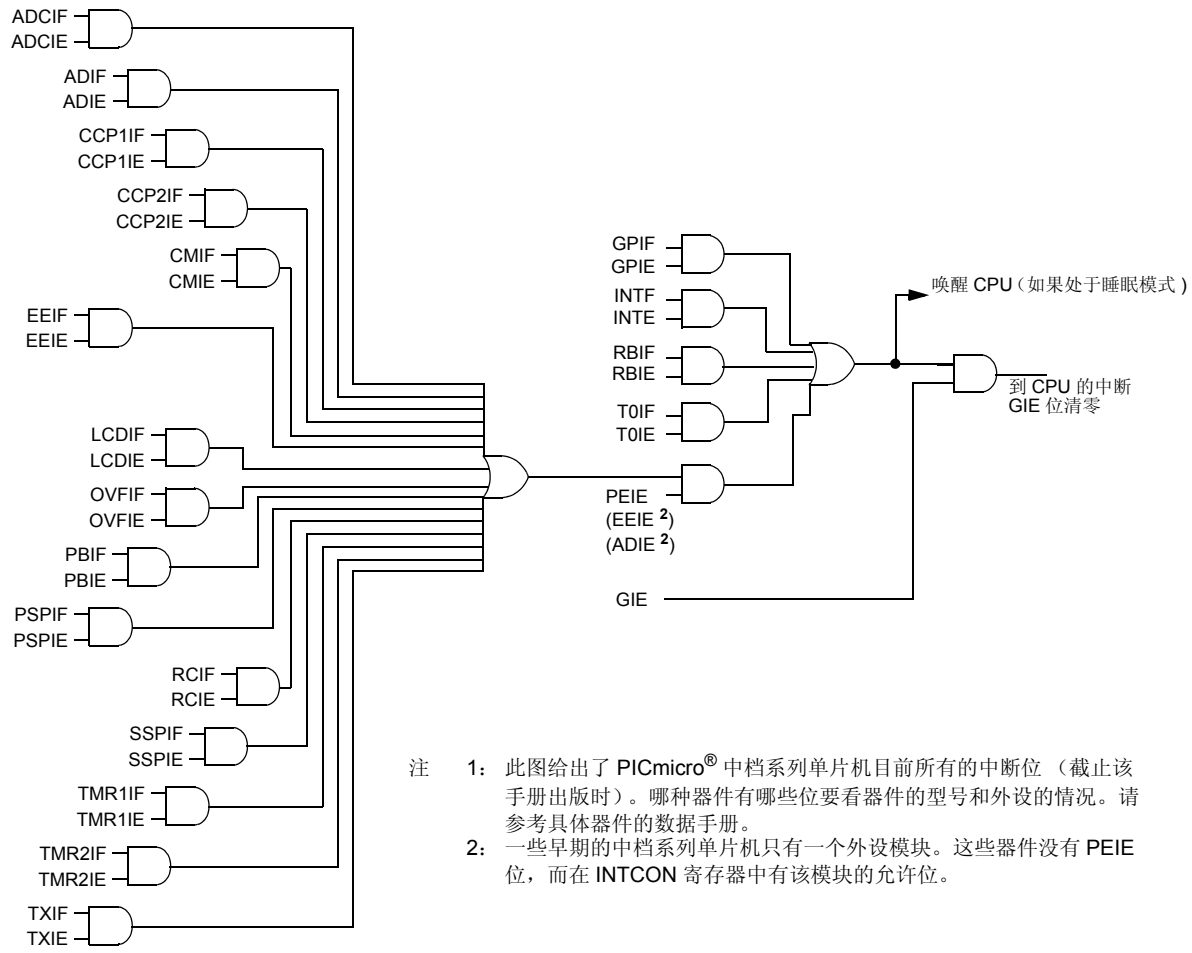
INTCON 寄存器包含以下中断的标志位和允许位：INT 引脚中断、RB 端口电平变化中断和 TMR0 溢出中断。 INTCON 寄存器还包含外设中断允许位 PEIE。当 PEIE 位置 “1”／清零时，将允许／禁止内核响应外设中断请求。

当一个中断被响应时， GIE 位被清零以禁止其它中断，返回地址压入堆栈， PC 中装入 0004H。在中断服务程序中，通过检测中断标志位可判断中断源。通常，中断标志位应在重新允许全局中断允许位 GIE 之前通过软件清零，以避免重复响应该中断。

在中断服务程序中，通过检测中断标志位可以判断中断源。各中断标志位的置位不受相应的中断屏蔽位和 GIE 位的状态影响。

注 1：	各中断标志位的置位不受对应的中断屏蔽位和 GIE 位的状态影响。
注 2：	当一条指令将 GIE 位清零时，会忽略原本将在其紧跟着的一个指令周期内等待响应的中断。CPU 将在紧跟着该指令的指令周期内执行一个空操作。被忽略的中断将继续等待，一直到 GIE 位再次被置位。

图 8-1： 中断逻辑



注 1：此图给出了 PICmicro® 中档系列单片机目前所有的中断位 （截止该手册出版时）。哪种器件有哪些位要看器件的型号和外设的情况。请参考具体器件的数据手册。

2：一些早期的中档系列单片机只有一个外设模块。这些器件没有 PEIE 位，而在 INTCON 寄存器中有该模块的允许位。

8.2 控制寄存器

通常，单片机至少有三个和中断有关的寄存器。 INTCON 寄存器包含全局中断允许位 GIE 和外设中断允许位 PEIE。 PIE / PIR 寄存器对分别用于允许外设中断和显示中断标志状态。

8.2.1 INTCON 寄存器

INTCON 寄存器是一个可读写的寄存器，包含了多个允许位和标志位。

注：	当一个中断条件发生时，不管相应的中断允许位或全局允许位 GIE（INTCON<7>）的状态如何，中断标志位都将置 “1”。故中断标志位可以用于软件查询。

寄存器 8-1： INTCON 寄存器

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

bit 7 **GIE**：全局中断允许位
1 = 允许所有未屏蔽的中断
0 = 禁止所有中断

bit 6 **PEIE**：外设中断允许位
1 = 允许所有未屏蔽的外设中断
0 = 禁止所有的外设中断

bit 5 **T0IE**：TMR0 溢出中断允许位
1 = 允许 TMR0 溢出中断
0 = 禁止 TMR0 溢出中断

bit 4 **INTE**：INT 外部引脚中断允许位
1 = 允许 INT 外部引脚中断
0 = 禁止 INT 外部引脚中断

bit 3 **RBIE (1)**：RB 端口电平变化中断允许位
1 = 允许 RB 端口电平变化中断
0 = 禁止 RB 端口电平变化中断

bit 2 **T0IF**：TMR0 溢出中断标志位
1 = TMR0 寄存器已经溢出 （必须用软件清零）
0 = TMR0 寄存器尚未发生溢出

bit 1 **INTF**：INT 外部引脚中断标志位
1 = 发生 INT 外部中断 （必须用软件清零）
0 = 未发生 INT 外部中断

bit 0 **RBIF (1)**：RB 端口电平变化中断标志位
1 = RB7:RB4 引脚中至少有一位的状态发生了变化 （必须用软件清零）
0 = RB7:RB4 引脚没有发生状态变化

图注		
R = 可读位	W = 可写位	
U = 未用位，读作 “0”		- n = 上电复位时的值

注 1：	某些型号单片机里， RBIE 位也称作 GPIE，而 RBIF 位称为 GPIF。
注 2：	某些型号单片机可能没有该特点。对于那些单片机，该位是保留的。
注 3：	对那些只有一个外设中断的单片机，这个位可能是 EEIE 或 ADIE。

8.2.2 PIE 寄存器

根据外设中断源的数量，可以有多个外设中断允许寄存器 （PIE1， PIE2） 。这些寄存器包含各外设中断的允许位。这些寄存器通常被称为 PIE。如果器件有 PIE 寄存器，那么要允许任何一个外设中断，必须将 PEIE 位置 “1”。

注： 要允许任何一个外设中断，必须将 PEIE （INTCON<6>）位置 “1”。

PIE 寄存器中各位的定义一般是固定的，但不保证未来的新器件也采用相同的定义。如果用户使用 Microchip 在头文件中提供的符号表示这些位，则可以忽略不同型号在位定义上的差别。这使得汇编器 / 编译器通过正确的寄存器和位名称能够自动处理这些位的位置。

寄存器 8-2： PIE 寄存器

R/W-0
（注 1）
bit 7 ... bit 0

bit **TMR1IE**：TMR1 溢出中断允许位
1 = 允许 TMR1 溢出中断
0 = 禁止 TMR1 溢出中断

bit **TMR2IE**：TMR2 对 PR2 匹配中断允许位
1 = 允许 TMR2 对 PR2 匹配中断
0 = 禁止 TMR2 对 PR2 匹配中断

bit **CCP1IE**：CCP1 中断允许位
1 = 允许 CCP1 中断
0 = 禁止 CCP1 中断

bit **CCP2IE**：CCP2 中断允许位
1 = 允许 CCP2 中断
0 = 禁止 CCP2 中断

bit **SSPIE**：同步串行口中断允许位
1 = 允许 SSP 中断
0 = 禁止 SSP 中断

bit **RCIE**：USART 接收中断允许位
1 = 允许 USART 接收中断
0 = 禁止 USART 接收中断

bit **TXIE**：USART 发送中断允许位
1 = 允许 USART 发送中断
0 = 禁止 USART 发送中断

bit **ADIE**：A/D 转换器中断允许位
1 = 允许 A/D 中断
0 = 禁止 A/D 中断

bit **ADCIE**：积分型 A/D 转换器比较器翻转中断允许位
1 = 允许积分型 A/D 转换器中断
0 = 禁止积分型 A/D 转换器中断

bit **OVFIE**：积分型 A/D TMR 溢出中断允许位
1 = 允许积分型 A/D TMR 溢出中断
0 = 禁止积分型 A/D TMR 溢出中断

bit **PSPIE**：并行从动端口的读 / 写中断允许位
1 = 允许 PSP 的读 / 写中断
0 = 禁止 PSP 的读 / 写中断

bit **EEIE**：EEPROM 写操作完成中断允许位
1 = 允许 EEPROM 写操作完成中断
0 = 禁止 EEPROM 写操作完成中断

bit **LCDIE**：LCD 中断允许位
1 = 允许 LCD 中断
0 = 禁止 LCD 中断

bit **CMIE**：比较器中断允许位
1 = 允许比较器中断
0 = 禁止比较器中断

图注	
R = 可读位	W = 可写位
U = 未用位，读作 “0”	- n = 上电复位时的值

注 1： 各允许位的位置依具体型号而不同。请参考相关器件的数据手册，了解具体的各位位置。

8.2.3 PIR 寄存器

根据外设中断源的数量，可以有多个外设中断标志寄存器 （PIR1， PIR2）。这些寄存器包含各外设中断的标志位。这些寄存器通常被称为 PIR。

注 1：	当一个中断条件发生时，不管相应的中断允许位和全局允许位 GIE (INTCON<7>) 的状态如何，中断标志位都将置位。
注 2：	用户软件应在允许一个中断之前，先将该中断标志位清零；同时在响应该中断后，也应将该中断标志位清零。

PIR 寄存器中各位的定义一般是固定的，但不保证未来的新器件也采用相同的定义。如果用户使用 Microchip 在头文件中提供的符号表示这些位，则可以忽略不同型号在位定义上的差别。这使得汇编器 / 编译器能够自动处理指定寄存器内这些位的位置。

寄存器 8-3： PIR 寄存器

R/W-0
（注 1）
bit 7 ... bit 0

bit **TMR1IF**：TMR1 溢出中断标志位
1 = TMR1 寄存器发生溢出 （必须用软件清零）
0 = TMR1 寄存器未发生溢出

bit **TMR2IF**：TMR2 对 PR2 匹配中断标志位
1 = TMR2 对 PR2 匹配 （必须用软件清零）
0 = TMR2 对 PR2 不匹配

bit **CCP1IF**：CCP1 中断标志位

输入捕捉模式
1 = 发生了 TMR1 寄存器捕捉 （必须用软件清零）
0 = 未发生 TMR1 寄存器捕捉

输出比较模式
1 = 发生了 TMR1 寄存器的比较匹配 （必须用软件清零）
0 = 未发生 TRM1 寄存器的比较匹配

脉宽调制模式下
此模式下未定义

bit **CCP2IF**：CCP2 中断标志位

输入捕捉模式
1 = 发生了 TMR1 寄存器捕捉 （必须用软件清零）
0 = 未发生 TMR1 寄存器捕捉

输出比较模式
1 = 发生了 TMR1 寄存器的比较匹配 （必须用软件清零）
0 = 未发生 TRM1 寄存器的比较匹配

脉宽调制模式下
此模式下未定义

bit **SSPIF**：同步串行口中断标志位
1 = 完成发送 / 接收
0 = 等待发送 / 接收完成

bit **RCIF**：USART 接收中断标志位
1 = USART 接收缓冲器 RCREG 满 （当读取 RCREG 时清零）
0 = USART 接收缓冲器为空

bit **TXIF**：USART 发送中断标志位
1 = USART 发送缓冲器 TXREG 为空 （当写入 TXREG 时清零）
0 = USART 发送缓冲器满

bit **ADIF**：A/D 转换器中断标志位
1 = 完成 A/D 转换 （必须用软件清零）
0 = 未完成 A/D 转换

寄存器 8-3： PIR 寄存器 （续）

bit **ADCIF**：积分型 A/D 转换器的比较器翻转中断标志位
1 = 完成 A/D 转换 （必须用软件清零）
0 = 未完成 A/D 转换

bit **OVFIF**：积分型 A/D TMR 溢出中断标志位
1 = 积分型 A/D TMR 发生了溢出 （必须用软件清零）
0 = 积分型 A/D TMR 没有溢出

bit **PSPIF**：并行从动端口读 / 写中断标志位
1 = 发生了读 / 写操作 （必须用软件清零）
0 = 未发生读 / 写操作

bit **EEIF**：EEPROM 写操作完成中断标志位
1 = 数据 EEPROM 写操作已完成 （必须用软件清零）
0 = 数据 EEPROM 写操作未完成

bit **LCDIF**：LCD 中断标志位
1 = 发生 LCD 中断 （必须用软件清零）
0 = 未发生 LCD 中断

bit **CMIF**：比较器中断标志位
1 = 比较器输入发生变化 （必须用软件清零）
0 = 比较器输入未发生变化

图注		
R = 可读位	W = 可写位	
U = 未用位，读作 “0”		- n = 上电复位时的值

注 1： 各标志位的位置依具体型号而不同。请参考相关器件的数据手册，了解具体的各位位置。

8.3 中断响应延时

中断响应延时被定义为从中断事件发生（中断标志位被置位）到地址 0004h 的指令开始执行（该中断是允许时）之间的这段时间。

对同步中断 （一般是器件的内部中断），响应延时为 3 个指令周期。

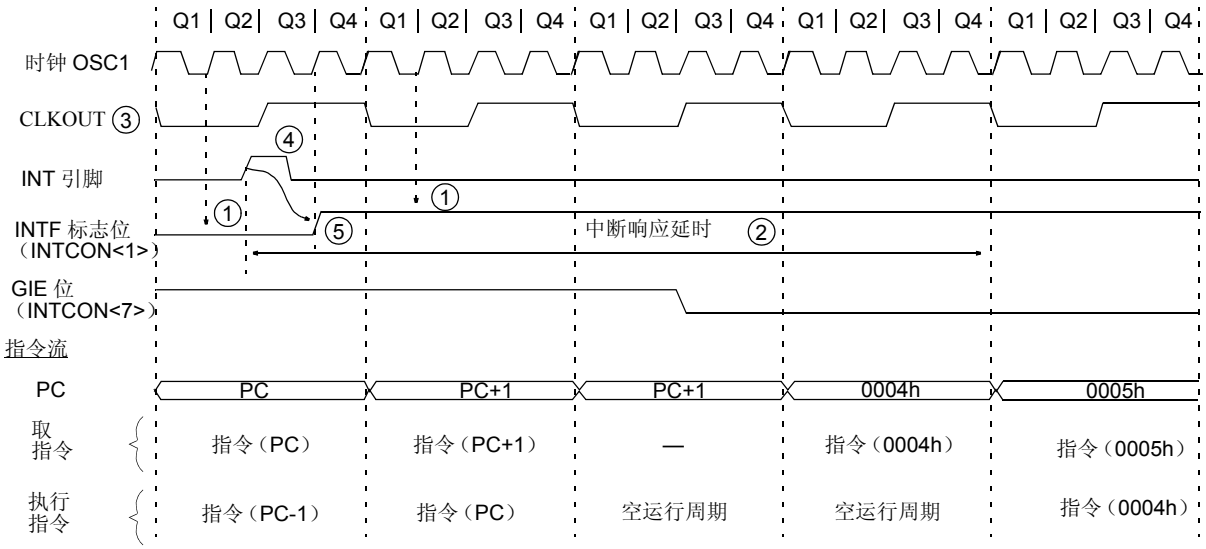
对异步中断 （一般是器件的外部中断，如 INT 引脚中断或 RB 端口的电平变化中断），中断响应延时将是3 - 3.75个指令周期。确切的延时取决于中断事件在指令周期的哪个时刻发生（图 8-2）。

对于单周期或双周期指令，中断响应延时是一样的。

8.4 INT 和外部中断

INT 引脚的外部中断是边沿触发的：如果 INTEDG 位（OPTION<6>）被置位，为上升沿触发；若清零，则为下降沿触发。当 INT 引脚上出现一个有效边沿时， INTF 标志位 （INTCON<1>）被置位。通过对 INTE 允许位 （INTCON<4>）置位 / 清零，该中断可以被允许 / 禁止。在重新允许该中断前，必须在中断服务程序中先用软件将 INTF 位清零。如果 INTE 位在进入休眠状态前被置位，则 INT 中断能把处理器从休眠状态中唤醒。GIE 位的状态取决于处理器在唤醒之后是否进入中断服务程序。请参见 **“ 看门狗定时器与休眠模式 ”** 一 章，了解休眠的详细信息和 INT 中断唤醒处理器的时序。

图 8-2： INT 引脚中断和其它外部中断的时序



注 1: INTF 标志在每个 Q1 时钟节拍采样。
2: 中断响应延时 = 3-4 TCY（这里 TCY = 指令周期）。
不管指令（PC）是单周期还是双周期，响应延时是一样的。
3: CLKOUT 在 RC 振荡器模式下才存在。
4: 关于 INT 脉冲的最小宽度要求，请参考 AC 规范。
5: 在 Q4-Q1 时钟节拍期间，随时可以对 INTF 位置位。

注： 任何外部信号（如定时器，捕捉，端口电平变化）引起的中断都有以上相似的时序。

8.5 中断的现场保护

在中断期间，仅将返回的 PC 地址压入堆栈。而用户通常可能还希望保存中断期间的一些重要寄存器值，如 W 寄存器和 STATUS 寄存器。这要通过软件来实现。

保存信息的操作通常被称作 “PUSHing”（压入），而恢复信息的操作被称作 “POPing”（弹出）。 PUSH 和 POP 不是指令符，而是概念性的操作。该操作通过一串指令序列实现。为了使代码易于移植，这些代码段可被写成宏形式（请参看 MPASM Assembler User’s Guide，了解宏创建的详细信息）。

例 8-1 说明了对于具有公共 RAM 的器件（PIC16C77），如何保存和恢复 STATUS 和 W 寄存器的值。用户定义的寄存器 W_TEMP，必须在各个存储区上都有定义，且应定义在各存储区内相同的偏移地址处（例如，在 Bank0 的 0x70 - 0x7F 范围内定义 W_TEMP）。用户定义的寄存器 STATUS_TEMP 应定义在 Bank0 中，本例即是如此。

例 8-1 的程序流程：

1. 保存 W 寄存器内容，不必考虑当前操作的是第几个存储区。
2. 在 Bank0 中保存 STATUS 寄存器内容。
3. 执行中断服务程序（ISR）代码。
4. 恢复 STATUS 寄存器（和存储区选择位寄存器）。
5. 恢复 W 寄存器。

如果在执行中断服务程序（ISR）代码前，还需保存其它内容，则应在保存了 STATUS 寄存器内容后（第 2 步）保存这些内容，在恢复 STATUS 寄存器内容前（第 4 步）恢复这些内容。

例 8-1： 在 RAM 内保存 STATUS 和 W 寄存器（适用于有公共 RAM 的器件）

```
    MOVWF   W_TEMP          ; Copy W to a Temporary Register
                            ;    regardless of current bank
    SWAPF   STATUS,W        ; Swap STATUS nibbles and place
                            ;    into W register
    MOVWF   STATUS_TEMP     ; Save STATUS to a Temporary register
                            ;    in Bank0
    :
    : (Interrupt Service Routine (ISR) )
    :
    SWAPF   STATUS_TEMP,W   ; Swap original STATUS register value
                            ;    into W (restores original bank)
    MOVWF   STATUS          ; Restore STATUS register from
                            ;    W register
    SWAPF   W_TEMP,F        ; Swap W_Temp nibbles and return
                            ;    value to W_Temp
    SWAPF   W_TEMP,W        ; Swap W_Temp to W to restore original
                            ;    W value without affecting STATUS
```

例 8-2 说明了对于没有公共 RAM 的器件（如 PIC16C74A），如何保存和恢复 STATUS 和 W 寄存器的值。用户定义的寄存器 W_TEMP，必须在各存储区上都有定义，且应定义在各存储区内相同的偏移地址处（例如，在 Bank0 的 0x70 - 0x7F 范围内定义 W_TEMP）。用户定义的寄存器 STATUS_TEMP 应定义在 Bank0 中。

不论 W_TEMP 定义在 Bank0 的 70h - 7Fh 范围内的何处，其它存储区内的相同位置都应作为该 W 寄存器的备份单元。

例 8-2 的程序流程：

1. 保存 W 寄存器内容，不必考虑当前操作的是第几个存储区。
2. 在 Bank0 中保存 STATUS 寄存器内容。
3. 执行中断服务程序（ISR）代码。
4. 恢复 STATUS 寄存器（和存储区选择位寄存器）。
5. 恢复 W 寄存器。

如果在执行中断服务程序（ISR）代码前，还需保存其它内容，则应在保存了 STATUS 寄存器内容后（第 2 步）保存这些内容，在恢复 STATUS 寄存器内容前（第 4 步）恢复这些内容。

例 8-2：　在 RAM 内保存 STATUS 和 W 寄存器（适用于无公共 RAM 的器件）

```
    MOVWF   W_TEMP          ; Copy W to a Temporary Register
                            ;    regardless of current bank
    SWAPF   STATUS,W        ; Swap STATUS nibbles and place
                            ;    into W register
    BCF     STATUS,RP0      ; Change to Bank0 regardless of
                            ;    current bank
    MOVWF   STATUS_TEMP     ; Save STATUS to a Temporary register
                            ;    in Bank0
    :
    : (Interrupt Service Routine (ISR) )
    :
    SWAPF   STATUS_TEMP,W   ; Swap original STATUS register value
                            ;    into W (restores original bank)
    MOVWF   STATUS          ; Restore STATUS register from
                            ;    W register
    SWAPF   W_TEMP,F        ; Swap W_Temp nibbles and return
                            ;    value to W_Temp
    SWAPF   W_TEMP,W        ; Swap W_Temp to W to restore original
                            ;    W value without affecting STATUS
```

例 8-3 说明了对于仅在Bank0内有通用RAM的器件（如PIC16C620），如何保存和恢复STATUS和 W 寄存器的值。在保存任何用户定义的寄存器前，必须检测存储区。用户定义的寄存器W_TEMP，必须在各存储区上都有定义，且应定义在各存储区内相同的偏移地址处。用户寄存器STATUS_TEMP 应定义在 Bank0 中。

例 8-3 的程序流程：

1. 检测当前操作的存储区。
2. 保存 W 寄存器内容，不必考虑当前操作的是第几个存储区。
3. 在 Bank0 中保存 STATUS 寄存器内容。
4. 执行中断服务程序 （ISR）代码。
5. 恢复 STATUS 寄存器 （和存储区选择位寄存器）。
6. 恢复 W 寄存器。

如果在执行中断服务程序 （ISR）代码前，还需保存其它内容，则应在保存了 STATUS 寄存器内容后 （第 2 步）保存这些内容，在恢复 STATUS 寄存器内容前 （第 4 步）恢复这些内容。

例 8-3： 在 RAM 内保存 STATUS 和 W 寄存器
（适用于仅在 Bank0 内有通用 RAM 的器件）

```
Push
      BTFSS   STATUS, RP0         ; In Bank 0?
      GOTO    RP0CLEAR            ; YES,
      BCF     STATUS, RP0         ; NO, Force to Bank 0
      MOVWF   W_TEMP              ; Store W register
      SWAPF   STATUS, W           ; Swap STATUS register and
      MOVWF   STATUS_TEMP         ;   store in STATUS_TEMP
      BSF     STATUS_TEMP, 1      ; Set the bit that corresponds to RP0
      GOTO    ISR_Code            ; Push completed
RP0CLEAR
      MOVWF   W_TEMP              ; Store W register
      SWAPF   STATUS, W           ; Swap STATUS register and
      MOVWF   STATUS_TEMP         ;   store in STATUS_TEMP
;
ISR_Code
      :
      : (Interrupt Service Routine (ISR) )
      :
;
Pop
      SWAPF   STATUS_TEMP, W      ; Restore Status register
      MOVWF   STATUS              ;
      BTFSS   STATUS, RP0         ; In Bank 1?
      GOTO    Restore_WREG        ; NO,
      BCF     STATUS, RP0         ; YES, Force Bank 0
      SWAPF   W_TEMP, F           ; Restore W register
      SWAPF   W_TEMP, W           ;
      BSF     STATUS, RP0         ; Back to Bank 1
      RETFIE                      ; POP completed
Restore_WREG
      SWAPF   W_TEMP, F           ; Restore W register
      SWAPF   W_TEMP, W           ;
      RETFIE                      ; POP completed
```

8.6 初始化

例 8-4 给出了初始化和中断允许的示例，其中 PIE1_MASK1 值就是将要写入中断允许寄存器的值。

例 8-5 给出了如何创建函数的宏定义。宏必须在使用前定义。为了调试方便，可将宏定义写在其它文件中并包含进来，这会使源代码看起来不那么混乱。所有需包含进来的宏定义文件必须在源程序前实现，这样简化了程序的调试，如例 8-6 所示。

例 8-7 给出了一个典型的中断服务程序结构。在执行中断代码前，该 ISR 用宏命令来保存和恢复寄存器。

例 8-4： 初始化和中断允许

```
PIE1_MASK1   EQU B'01101010'   ; This is the Interrupt Enable
      :                        ;    Register mask value
      :
      CLRF   STATUS         ; Bank0
      CLRF   INTCON         ; Disable interrupts and clear some flags
      CLRF   PIR1           ; Clear all flag bits
      BSF    STATUS, RP0    ; Bank1
      MOVLW  PIE1_MASK1     ; This is the initial masking for PIE1
      MOVWF  PIE1           ;
      BCF    STATUS, RP0    ; Bank0
      BSF    INTCON, GIE    ; Enable Interrupts
```

例 8-5： 用宏命令来保存 / 恢复寄存器

```
PUSH_MACRO    MACRO           ; This Macro Saves register contents
     MOVWF    W_TEMP          ; Copy W to a Temporary Register
                              ;    regardless of current bank
     SWAPF    STATUS,W        ; Swap STATUS nibbles and place
                              ;    into W register
     MOVWF    STATUS_TEMP     ; Save STATUS to a Temporary register
                              ;    in Bank0
     ENDM                     ; End this Macro
;
POP_MACRO     MACRO           ; This Macro Restores register contents
     SWAPF    STATUS_TEMP,W   ; Swap original STATUS register value
                              ;    into W (restores original bank)
     MOVWF    STATUS          ; Restore STATUS register from
                              ;    W register
     SWAPF    W_TEMP,F        ; Swap W_Temp nibbles and return
                              ;    value to W_Temp
     SWAPF    W_TEMP,W        ; Swap W_Temp to W to restore original
                              ;    W value without affecting STATUS
     ENDM                     ; End this Macro
```

例 8-6： 源文件模板

```
     LIST   p = p16C77      ; List Directive,
;    Revision History
;
     #INCLUDE    <P16C77.INC>    ; Microchip Device Header File
;
     #INCLUDE    <MY_STD.MAC>    ; Include my standard macros
     #INCLUDE    <APP.MAC>       ; File which includes macros specific
                                 ;   to this application
; Specify Device Configuration Bits
     __CONFIG    _XT_OSC & _PWRTE_ON & _BODEN_OFF & _CP_OFF & _WDT_ON
;
     org   0x00            ; Start of Program Memory
RESET_ADDR   :             ; First instruction to execute after a reset


     end
```

例 8-7： 典型的中断服务程序（ISR）

```
    org  ISR_ADDR           ;
      PUSH_MACRO            ; MACRO that saves required context registers,
                            ;   or in-line code
      CLRF   STATUS         ; Bank0
      BTFSC  PIR1, TMR1IF   ; Timer1 overflow interrupt?
      GOTO   T1_INT         ; YES
      BTFSC  PIR1, ADIF     ; NO, A/D interrupt?
      GOTO   AD_INT         ; YES, do A/D thing
      :                     ; NO, do this for all sources
      :                     ;
      BTFSC  PIR1, LCDIF    ; NO, LCD interrupt
      GOTO   LCD_INT        ; YES, do LCD thing
      BTFSC  INTCON, RBIF   ; NO, Change on PORTB interrupt?
      GOTO   PORTB_INT      ; YES, Do PortB Change thing
INT_ERROR_LP1               ; NO, do error recovery
      GOTO   INT_ERROR_LP1  ; This is the trap if you enter the ISR
                            ;    but there were no expected
                            ;    interrupts
T1_INT                      ; Routine when the Timer1 overflows
      :                     ;
      BCF     PIR1, TMR1IF  ; Clear the Timer1 overflow interrupt flag
      GOTO    END_ISR       ; Ready to leave ISR (for this request)
AD_INT                      ; Routine when the A/D completes
      :                     ;
      BCF     PIR1, ADIF    ; Clear the A/D interrupt flag
      GOTO    END_ISR       ; Ready to leave ISR (for this request)
LCD_INT                     ; Routine when the LCD Frame begins
      :                     ;
      BCF     PIR1, LCDIF   ; Clear the LCD interrupt flag
      GOTO    END_ISR       ; Ready to leave ISR (for this request)
PORTB_INT                   ; Routine when PortB has a change
      :                     ;
END_ISR                     ;
        POP_MACRO           ; MACRO that restores required registers,
                            ;   or in-line code
        RETFIE              ; Return and enable interrupts
```

8.7 设计技巧

问 1： *为什么算法没有给出正确结果？*

答 1：

假设算法是正确的，并且在算法执行期间中断已允许，对于那些既在算法中，又在中断服务程序中使用的寄存器，应确保有正确的中断现场保存和恢复。如果没有正确的中断现场保存和恢复，一些寄存器值可能会因 ISR 的执行而遭破坏。

问 2： *我的系统好象锁死了。*

答 2：

如果使用了中断，要确保在提供了中断服务后，执行 RETFIE 指令前，中断标志位被清零。在 RETFIE 指令执行后，如果中断标志位仍然保持为 1，会被误认为又是一次允许中断，程序执行立即再次进入中断服务。

8.8 相关应用笔记

本部分列出了与本章内容相关的应用笔记。这些应用笔记并非都是专门针对中档单片机系列而写的（即有些针对低档系列，有些针对高档系列），但是其概念是相近的，通过适当修改并受到 一定限制，即可使用。目前与本章内容相关的应用笔记有：

标题	应用笔记 #
Using the PortB Interrupt On Change as an External Interrupt	AN566

8.9 版本历史

版本 A

这是描述中断的初始发行版。