Section 8. Interrupts

HIGHLIGHTS

This section of the manual contains the following major topics:

8.1 Introduction

PICmicro MCUs can have many sources of interrupt. These sources generally include one interrupt source for each peripheral module, though some modules may generate multiple interrupts (such as the USART module). The current interrupts are:

- INT Pin Interrupt (external interrupt)
- TMR0 Overflow Interrupt
- PORTB Change Interrupt (pins RB7:RB4)
- Comparator Change Interrupt
- Parallel Slave Port Interrupt
- USART Interrupts
- Receive Interrupt
- Transmit Interrupt
- A/D Conversion Complete Interrupt
- LCD Interrupt.
- Data EEPROM Write Complete Interrupt
- Timer1 Overflow Interrupt
- Timer2 Overflow Interrupt
- CCP Interrupt
- SSP Interrupt

There is a minimum of one register used in the control and status of the interrupts. This register is:

- INTCON

Additionally, if the device has peripheral interrupts, then it will have registers to enable the peripheral interrupts and registers to hold the interrupt flag bits. Depending on the device, the registers are:

- PIE1
- PIR1
- PIE2
- PIR2

We will generically refer to these registers as PIR and PIE. If future devices provide more interrupt sources, they will be supported by additional register pairs, such as PIR3 and PIE3.

The Interrupt Control Register, INTCON, records individual flag bits for core interrupt requests. It also has various individual enable bits and the global interrupt enable bit.

The Global Interrupt Enable bit, GIE (INTCON<7>), enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in the INTCON register. The GIE bit is cleared on reset.

The "return from interrupt" instruction, RETFIE, exits the interrupt routine as well as sets the GIE bit, which allows any pending interrupt to execute.

The INTCON register contains these interrupts: INT Pin Interrupt, the RB Port Change Interrupt, and the TMR0 Overflow Interrupt. The INTCON register also contains the Peripheral Interrupt Enable bit, PEIE. The PEIE bit will enable/disable the peripheral interrupts from vectoring when the PEIE bit is set/cleared.

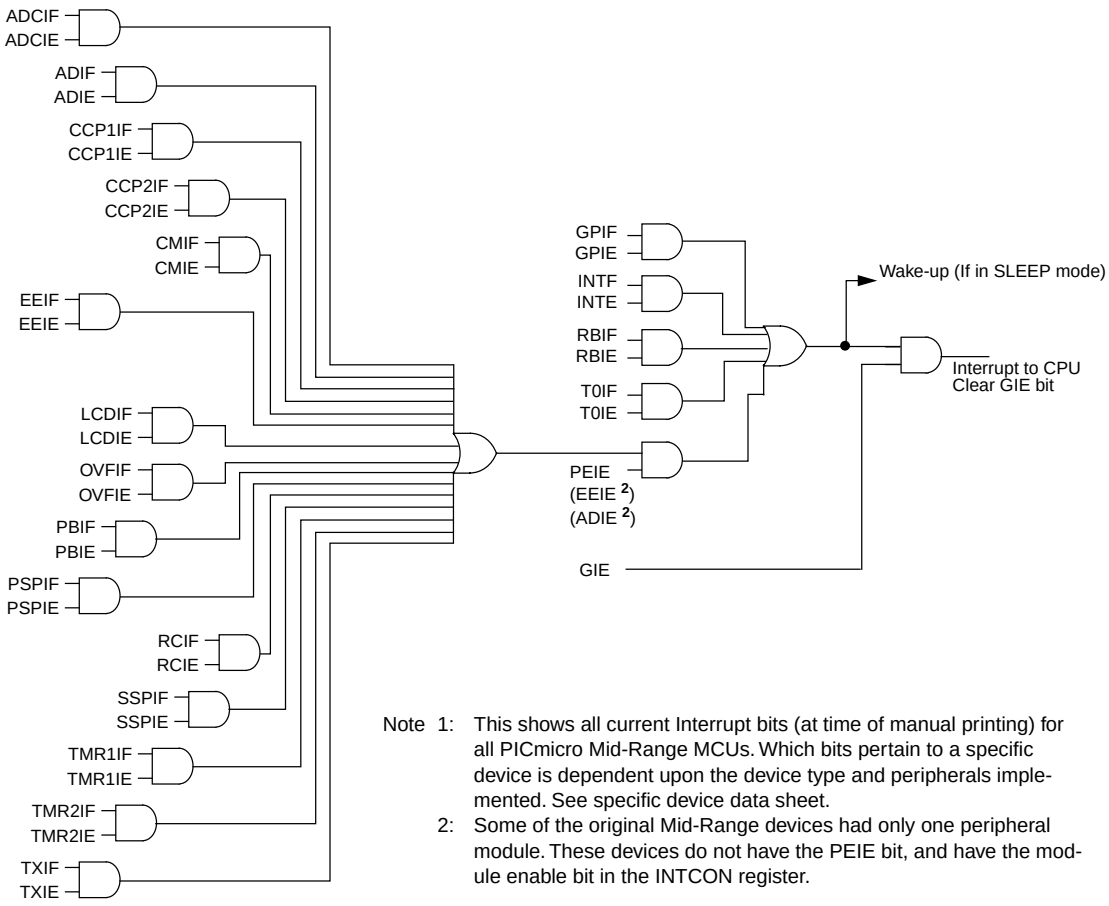
When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed into the stack and the PC is loaded with 0004h. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. Generally the interrupt flag bit(s) must be cleared in software before re-enabling the global interrupt to avoid recursive interrupts.

Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

Note 1: Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

Note 2: When an instruction that clears the GIE bit is executed, any interrupts that were pending for execution in the next cycle are ignored. The CPU will execute a NOP in the cycle immediately following the instruction which clears the GIE bit. The interrupts which were ignored are still pending to be serviced when the GIE bit is set again.

Figure 8-1: Interrupt Logic



Note 1: This shows all current Interrupt bits (at time of manual printing) for all PICmicro Mid-Range MCUs. Which bits pertain to a specific device is dependent upon the device type and peripherals implemented. See specific device data sheet.

2: Some of the original Mid-Range devices had only one peripheral module. These devices do not have the PEIE bit, and have the module enable bit in the INTCON register.

8.2 Control Registers

Generally devices have a minimum of three registers associated with interrupts. The INTCON register which contains Global Interrupt Enable bit, GIE, as well as the Peripheral Interrupt Enable bit, PEIE, and the PIE / PIR register pair which enable the peripheral interrupts and display the interrupt flag status.

8.2.1 INTCON Register

The INTCON Register is a readable and writable register which contains various enable and flag bits.

Note:	Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).This feature allows for software polling.

Register 8-1: INTCON Register

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:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

bit 3 **RBIE (1):** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

bit 0 **RBIF (1):** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend		
R = Readable bit	W = Writable bit	
U = Unimplemented bit, read as '0'		- n = Value at POR reset

Note 1:	In some devices, the RBIE bit may also be known as GPIE and the RBIF bit may be know as GPIF.
Note 2:	Some devices may not have this feature. For those devices this bit is reserved.
Note 3:	In devices with only one peripheral interrupt, this bit may be EEIE or ADIE.

8.2.2 PIE Register(s)

Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Enable registers (PIE1, PIE2). These registers contain the individual enable bits for the Peripheral interrupts. These registers will be generically referred to as PIE. If the device has a PIE register, The PEIE bit must be set to enable any of these peripheral interrupts.

Note:	Bit PEIE (INTCON<6>) must be set to enable any of the peripheral interrupts.

Although, the PIE register bits have a general bit location with each register, future devices may not have consistent placement. Bit location inconsistencies will not be a problem if you use the supplied Microchip Include files for the symbolic use of these bits. This will allow the Assembler/Compiler to automatically take care of the placement of these bits by specifying the correct register and bit name.

Register 8-2: PIE Register

R/W-0
(Note 1)
bit 7 ... bit 0

bit **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

bit **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

bit **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit **ADCIE**: Slope A/D Converter comparator Trip Interrupt Enable bit
1 = Enables the Slope A/D interrupt
0 = Disables the Slope A/D interrupt

bit **OVFIE**: Slope A/D TMR Overflow Interrupt Enable bit
1 = Enables the Slope A/D TMR overflow interrupt
0 = Disables the Slope A/D TMR overflow interrupt

bit **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit **LCDIE**: LCD Interrupt Enable bit
1 = Enables the LCD interrupt
0 = Disables the LCD interrupt

bit **CMIE**: Comparator Interrupt Enable bit
1 = Enables the Comparator interrupt
0 = Disables the Comparator interrupt

Legend		
R = Readable bit	W = Writable bit	
U = Unimplemented bit, read as '0'		- n = Value at POR reset

Note 1: The bit position of the enable bits is device dependent. Please refer to the device data sheet for bit placement.

8.2.3 PIR Register(s)

Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Flag registers (PIR1, PIR2). These registers contain the individual flag bits for the peripheral interrupts. These registers will be generically referred to as PIR.

Note 1:	Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).
Note 2:	User software should ensure the appropriate interrupt flag bits are cleared (by software) prior to enabling an interrupt, and after servicing that interrupt.

Although, the PIR bits have a general bit location within each register, future devices may not be able to be consistent with that. It is recommended that you use the supplied Microchip Include files for the symbolic use of these bits. This will allow the Assembler/Compiler to automatically take care of the placement of these bits within the specified register.

Register 8-3: PIR Register

R/W-0
(Note 1)
bit 7 ... bit 0

bit **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

bit **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit **CCP1IF**: CCP1 Interrupt Flag bit

Capture Mode
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred

Compare Mode
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred

PWM Mode
Unused in this mode

bit **CCP2IF**: CCP2 Interrupt Flag bit

Capture Mode
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred

Compare Mode
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred

PWM Mode
Unused in this mode

bit **SSPIF**: Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete
0 = Waiting to transmit/receive

bit **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer, RCREG, is full (cleared when RCREG is read)
0 = The USART receive buffer is empty

bit **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer, TXREG, is empty (cleared when TXREG is written)
0 = The USART transmit buffer is full

bit **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

Register 8-3: PIR Register (Cont'd)

bit **ADCIF**: Slope A/D Converter Comparator Trip Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit **OVFIF**: Slope A/D TMR Overflow Interrupt Flag bit
1 = Slope A/D TMR overflowed (must be cleared in software)
0 = Slope A/D TMR did not overflow

bit **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit **EEIF**: EE Write Complete Interrupt Flag bit
1 = The data EEPROM write operation is complete (must be cleared in software)
0 = The data EEPROM write operation is not complete

bit **LCDIF**: LCD Interrupt Flag bit
1 = LCD interrupt has occurred (must be cleared in software)
0 = LCD interrupt has not occurred

bit **CMIF**: Comparator Interrupt Flag bit
1 = Comparator input has changed (must be cleared in software)
0 = Comparator input has not changed

Legend		
R = Readable bit	W = Writable bit	
U = Unimplemented bit, read as '0'		- n = Value at POR reset

Note 1: The bit position of the flag bits is device dependent. Please refer to the device data sheet for bit placement.

8.3 Interrupt Latency

Interrupt latency is defined as the time from the interrupt event (the interrupt flag bit gets set) to the time that the instruction at address 0004h starts execution (when that interrupt is enabled).

For synchronous interrupts (typically internal), the latency is 3TCY.

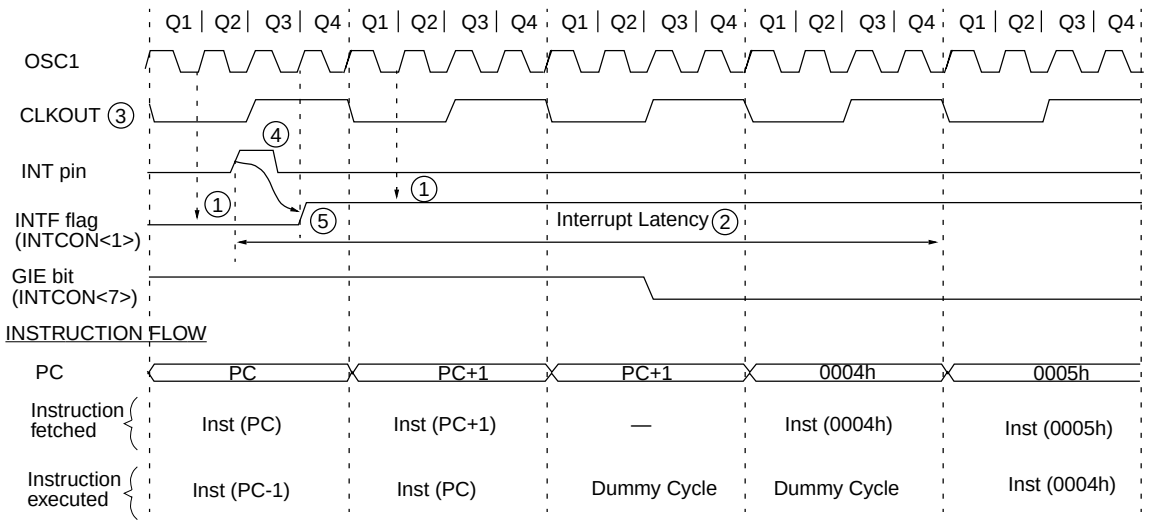
For asynchronous interrupts (typically external), such as the INT or Port RB Change Interrupt, the interrupt latency will be 3 - 3.75TCY (instruction cycles). The exact latency depends upon when the interrupt event occurs (Figure 8-2) in relation to the instruction cycle.

The latency is the same for both one and two cycle instructions.

8.4 INT and External Interrupts

The external interrupt on the INT pin is edge triggered: either rising if the INTEDG bit (OPTION<6>) is set, or falling, if the INTEDG bit is clear. When a valid edge appears on the INT pin, the INTF flag bit (INTCON<1>) is set. This interrupt can be enabled/disabled by setting/clearing the INTE enable bit (INTCON<4>). The INTF bit must be cleared in software in the interrupt service routine before re-enabling this interrupt. The INT interrupt can wake-up the processor from SLEEP, if the INTE bit was set prior to going into SLEEP. The status of the GIE bit decides whether or not the processor branches to the interrupt vector following wake-up. See the **"Watchdog Timer and Sleep Mode"** section for details on SLEEP and for timing of wake-up from SLEEP through INT interrupt.

Figure 8-2: INT Pin and Other External Interrupt Timing



Note 1: INTF flag is sampled here (every Q1).
2: Interrupt latency = 3-4 TCY where TCY = instruction cycle time.
Latency is the same whether Instruction (PC) is a single cycle or a 2-cycle instruction.
3: CLKOUT is available only in RC oscillator mode.
4: For minimum width of INT pulse, refer to AC specs.
5: INTF is enabled to be set anytime during the Q4-Q1 cycles.

Note: Any interrupts caused by external signals (such as timers, capture, change on port) will have similar timing.

8.5 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt e.g. W register and STATUS register. This has to be implemented in software.

The action of saving information is commonly referred to as “PUSHing,” while the action of restoring the information before the return is commonly referred to as “POPing.” These (PUSH, POP) are not instruction mnemonics, but are conceptual actions. This action can be implemented by a sequence of instructions. For ease of code transportability, these code segments can be made into MACROs (see MPASM Assembler User’s Guide for details on creating macros).

Example 8-1 stores and restores the STATUS and W registers for devices with common RAM (such as the PIC16C77). The user register, W_TEMP, must be defined across all banks and must be defined at the same offset from the bank base address (i.e., W_TEMP is defined at 0x70 - 0x7F in Bank0). The user register, STATUS_TEMP, must be defined in Bank0, in this example STATUS_TEMP is also in Bank0.

The steps of Example 8-1:

1. Stores the W register regardless of current bank.
2. Stores the STATUS register in Bank0.
3. Executes the Interrupt Service Routine (ISR) code.
4. Restores the STATUS (and bank select bit register).
5. Restores the W register.

If additional locations need to be saved before executing the Interrupt Service Routine (ISR) code, they should be saved after the STATUS register is saved (step 2), and restored before the STATUS register is restored (step 4).

Example 8-1: Saving the STATUS and W Registers in RAM (for Devices with Common RAM)

```
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
```

Example 8-2 stores and restores the STATUS and W registers for devices without common RAM (such as the PIC16C74A). The user register, W_TEMP, must be defined across all banks and must be defined at the same offset from the bank base address (i.e., W_TEMP is defined at 0x70 - 0x7F in Bank0). The user register, STATUS_TEMP, must be defined in Bank0.

Within the 70h - 7Fh range (Bank0), wherever W_TEMP is expected the corresponding locations in the other banks should be dedicated for the possible saving of the W register.

The steps of Example 8-2:

1. Stores the W register regardless of current bank.
2. Stores the STATUS register in Bank0.
3. Executes the Interrupt Service Routine (ISR) code.
4. Restores the STATUS (and bank select bit register).
5. Restores the W register.

If additional locations need to be saved before executing the Interrupt Service Routine (ISR) code, they should be saved after the STATUS register is saved (step 2), and restored before the STATUS register is restored (step 4).

Example 8-2: Saving the STATUS and W Registers in RAM (for Devices without Common RAM)

```
    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
```

Example 8-3 stores and restores the STATUS and W registers for devices with general purpose RAM only in Bank0 (such as the PIC16C620). The Bank must be tested before saving any of the user registers. , W_TEMP, must be defined across all banks and must be defined at the same offset from the bank base address. The user register, STATUS_TEMP, must be defined in Bank0.

The steps of Example 8-3:

1. Test current bank.
2. Stores the W register regardless of current bank.
3. Stores the STATUS register in Bank0.
4. Executes the Interrupt Service Routine (ISR) code.
5. Restores the STATUS (and bank select bit register).
6. Restores the W register.

If additional locations need to be saved before executing the Interrupt Service Routine (ISR) code, they should be saved after the STATUS register is saved (step 2), and restored before the STATUS register is restored (step 4).

Example 8-3: Saving the STATUS and W Registers in RAM (for Devices with General Purpose RAM Only in Bank0)

```
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 Initialization

Example 8-4 shows the initialization and enabling of device interrupts, where PIE1_MASK1 value is the value to write into the interrupt enable register.

Example 8-5 shows how to create macro definitions for functions. Macros **must** be defined before they are used. For debugging ease, it may help if macros are placed in other files that are included at assembly time. This allows the source to be viewed without all the clutter of the required macros. These files must be included before the macro is used, but it simplifies debugging, if all include files are done at the top of the source file. Example 8-6 shows this structure.

Example 8-7 shows a typical Interrupt Service Routine structure. This ISR uses macros for the saving and restoring of registers before the execution of the interrupt code.

Example 8-4: Initialization and Enabling of Interrupts

```
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
```

Example 8-5: Register Saving / Restoring as Macros

```
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
```

Example 8-6: Source File Template

```
     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
```

Example 8-7: Typical Interrupt Service Routine (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 Design Tips

Question 1: ***An algorithm does not give the correct results.***

Answer 1:

Assuming that the algorithm is correct and that interrupts are enabled during the algorithm, ensure that are registers that are used by the algorithm and by the interrupt service routine are saved and restored. If not some registers may be corrupted by the execution of the ISR.

Question 2: ***My system seems to lock up.***

Answer 2:

If interrupts are being used, ensure that the interrupt flag is cleared after servicing that interrupt (but before executing the `RETFIE` instruction). If the interrupt flag remains set when the `RETFIE` instruction is executed, program execution immediately returns to the interrupt vector, since there is an outstanding enabled interrupt.

8.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Mid-Range MCU family (that is they may be written for the Base-Line, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to this section are:

Title	Application Note #
Using the PortB Interrupt On Change as an External Interrupt	AN566

8.9 Revision History

Revision A

This is the initial released revision of the interrupt description.