PIC16F87/88 Interrupts

An Interrupt is a specific event that can interrupt the main program allowing you to immediately go off and perform some service routine before continuing on with the main program. The PIC16F87/88 has 12 sources of interrupt events, most of them associated with the micro's peripheral devices such as the Timers, Serial Transmitter/Receivers, Analog to Digital Converter.

Each of the 12 interrupt sources has an Enable bit which, when set high, will enable the main program to be interrupted when that interrupt event occurs. When the Enable bit is cleared to low, the event can still happen but it will not trigger the Interrupt routine to start up, interrupting the main program.

Each of the 12 interrupt sources also has a Flag bit. A high Interrupt Flag indicates that the Interrupt Event has occurred and needs to be serviced. A low Interrupt Flag indicates that the event has not ocurred yet. When several interrupt sources are enabled, the programmer would start off the interrupt service code by polling these interrupt flags to see which event happened. Flags are usually cleared by the programmer in software before exiting the interrupt routine.

There is also a "Global Interrupt Enable",GIE (INTCON bit 7) bit. This bit is used to enable or disable all 12 interrupts at once. Therefore, to enable any particular interrupt, the GIE bit must be set high as well as the individual interrupt enable bit. At the start of the interrupt service routine, the GIE bit is automatically disabled (cleared to low) to prevent any other interrupts from happening while servicing the interrupt. Then, at the end of the interrupt service routine the GIE bit is automatically enabled again (set high).

All the Peripheral device interrupts also have a global type enable bit called the PEIE (INTCON bit 6) which must be set high to enable any pheripheral device interrupt.

On Reset or Power-up the PIC16F87/88 will jump to memory location 0x0000 in the program memory space and start executing the program that lives there. Similarly, when an interrupt event occurs and any of the Interrupts are enabled along with the GIE bit, the processor will immediately stop executing the current program, jump to memory location 0x0004 and start executing the program that lives there. This vectored address behavior requires your assembly code to have something like the following at the very beginning of the code:

```
org     0x00
goto    main_program
org     0x04
goto    interrupt_routine
```

The first line directs the assembler to move to the top of the program memory space (0x00). The second line is the code that will be loaded into this program memory space. It is usually a goto command followed by the memory location labeled "main_program" (or any other label you choose to give it), a pointer to where the main startup program begins. There is little room for any code more than a goto here since we immediately bump up against the interrupt vector address of 0x04 which must be filled with directives for the interrupt routine. Here, the location 0x04 is filled with another goto command followed by the location of the interrupt routine labeled "interrupt_routine" (or any other label you choose to give it).

Whenever an enabled interrupt event occurs the following happens. The current program is halted and the program counter is saved so that the current program can later be continued from where it stopped. The GIE bit is cleared, to prevent any new interrupt events from interrupting the interrupt routine. The processor then jumps to the interrupt handler vector, the program starting at address 0x04. In the code example above there is an immediate goto command to jump to an interrupt handling routine.

For more involved programs, the programmer might want to start the interrupt routine with a save of important registers, like the w, STATUS, PCLATH - any registers that are used by both the interrupt handler and the main program. Do not worry about the program counter; it is automatically saved. An example of code that does this is shown below. Remember that if the interrupt handler and the main program do not both use these registers, this code is unecessary.

```
movwf   _w
movf    STATUS, w
bcf     STATUS, RP0
movwf   _status
movf    PCLATH, w
movwf   _pclath
clrf    PCLATH
```

If more than one interrupt event is enabled, the programmer will then poll the enabled interrupt flags to see which one took place, and then direct the program to the desired handler code. When the interrupt handler code is finished, the programmer must then return any register values saved at the start of the handler. The action of the code shown above is reversed with the following code.

```
movf    _pclath, w
movwf   PCLATH
movf    _status, w
movwf   STATUS
swapf   _w, f
swapf   _w,w
```

Finally, the interrupt routine is ended with the "return from interrupt" command – retfie. With this command the processor automatically re-enables the GIE bit, reloads the saved Program Counter value and returns control to the interrupted main program from where it left off.

Interrupt Event	Enable Bit	Flag Bit	Other Settings
RB Port Change On any change RB inputs 4 thru 7	RBIE INTCON (bit 3)	RBIF INTCON (bit 0)	Can awaken from sleep.
RB0 External On rising or falling edge of input RB0.	INT0IE INTCON (bit 4)	INT0IF INTCON (bit 1)	INTEDG OPTION_REG (bit 6) Rising edge (1) or falling edge (0)
Timer 0 Overflow Timer overflow from FFh to 00h	TMR0IE INTCON (bit 5)	TMR0IF INTCON (bit 2)	Cannot awaked from sleep. OPTION_REG
A/D Converter On Conversion Completed	ADIE PIE1 (bit 6)	ADIF PIR1 (bit 6)	ANSEL, ADCON0, ADCON1 reg. Can awaken from sleep.
AUSART Receive On receive buffer full.	RCIE PIE1 (bit 5)	RCIF PIR1 (bit 5)	RCSTA, TCREG, SPBRG
AUSART Transmit On transmit buffer empty.	TXIE PIE1 (bit 4)	TXIF PIR1 (bit 4)	TXSTA, TXREG, SPBRG,
Synchronous Serial Port On transmission/ reception complete.	SSPIE PIE1 (bit 3)	SSPIF PIR1 (bit 3)	SSPSTAT, SSPCON, SSPBUF
CCP1 On Timer 1 register capture in capture mode or on Timer 1 register comparison match in compare mode.	CCP1IE PIE1 (bit 2)	CCP1IF PIR1 (bit 2)	CCO1M0 thru 3 CCP1CON (bits3-0) CCPR1H:CCPR1L Register pair
Timer 2 to PR2 Match On a timer 2 to PR2 match occurring.	TMR2IE PIE1 (bit 1)	TMR2IF PIR1 (bit 1)	TMR2, T2CON, PR2
Timer 1 Overflow Timer overflow from FFh to 00h	TMR1IE PIE1 (bit 0)	TMR1IF PIR1 (bit 0)	T1CON reg. Can awaken from sleep.
Oscillator Fail On system oscillator failed; clock input changed to INTRC.	OSFIE PIE2 (bit 7)	OSFIF PIR2 (bit 7)	
Comparator Interrupt On change of comparator input.	CMIE PIE2 (bit 6)	CMIF PIR2 (bit 6)	CMCON register Can awaken from sleep.
EEPROM Write Operation On write operation completed.	EEIE PIE2 (bit 4)	EEIF PIR2 (bit 4)	