Interrupciones de rango medio mejoradas

Resumen

Las interrupciones son eventos detectados por la MCU que hacen que se anule el flujo normal del programa. Las interrupciones pausan el programa actual y transfieren el control a una rutina de firmware específica escrita por el usuario llamada Rutina de servicio de interrupción (ISR). El ISR procesa el evento de interrupción y luego reanuda el flujo normal del programa. Este artículo muestra cómo habilitar y procesar interrupciones en la familia PIC16F1xxx de PIC® de rango medio [mejorado] .

Descripción general del proceso de interrupción

1 **Programa MCU para reaccionar a las interrupciones**

La MCU debe programarse para permitir que ocurran interrupciones. La configuración de Global Interrupt Enable (GIE) y, en muchos casos, Peripheral Interrupt Enable (PEIE), permite que la MCU reciba interrupciones. GIE y PEIE se encuentran en el registro de funciones especiales de control de interrupciones (INTCON).

2 **Habilitar interrupciones de periféricos seleccionados**

Cada periférico en la MCU tiene un bit de habilitación individual. Se debe establecer el bit de activación de interrupción individual de un periférico, además de GIE/PEIE, antes de que el periférico pueda generar una interrupción. Los bits de habilitación de interrupción individuales se encuentran en INTCON, PIE1, PIE2 y PIE3.

3 Periférico afirma una solicitud de interrupción

Cuando un periférico alcanza un estado en el que se necesita la intervención del programa, el periférico establece un indicador de solicitud de interrupción (xxIF). Estos indicadores de interrupción se establecen independientemente del estado de GIE, PEIE y los bits de activación de interrupción individuales. Los indicadores de interrupción se encuentran en INTCON, PIR1, PIR2 y PIR3.

Los indicadores de solicitud de interrupción se bloquean en alto cuando se establecen y deben ser borrados por el ISR escrito por el usuario.

4 Ocurre una interrupción

Cuando se establece un indicador de solicitud de interrupción y la interrupción está habilitada correctamente, comienza el proceso de interrupción:

- Las interrupciones globales se desactivan borrando GIE a 0.
- El contexto del programa actual se guarda en los registros de sombra.
- El valor del contador de programa se almacena en la pila de retorno.
- El control del programa se transfiere al vector de interrupción en la dirección 04h.

5 Ejecuciones de ISR

The ISR is a function written by the user and placed at address 04h. The ISR does the following:

1. Checks the interrupt-enabled peripherals for the source of the interrupt request.
2. Performs the necessary peripheral tasks.
3. Clears the appropriate interrupt request flag.
4. Executes the Return From Interrupt instruction (RETFIE) as the final ISR instruction.

6 Control is returned to the Main program

When RETFIE is executed:

1. Global Interrupts are enabled (GIE=1).
2. The program context is restored from the Shadow registers.
3. The return address from the stack is loaded into the Program Counter.
4. Execution resumes from point at which it was interrupted.

Registers Used to Process Interrupts

Interrupt Control Register

INTCON register

GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7							bit 0

(/local--files/8bit:emr-interrupts/intcon-reg.png)

`GIE -` Global Interrupt Enable

`PEIE -` Peripheral Interrupt Enable

`TMR0IE -` Timer0 Interrupt Enable

`INTE -` External Interrupt Enable

`IOCIE -` Interrupt on Change Enable

`TMR0IF -` Timer0 Interrupt flag

`INTF -` External Interrupt flag

`IOCIF -` Interrupt on Change flag

`INTCON` contains global and peripheral interrupt enable flags as well as the individual interrupt request flags and interrupt enable flags for three of the PIC16F1xxxx interrupts.

Interrupt Enable Registers

PIE1 register

TMR1GIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMRI1E
bit 7							bit 0

(/local--files/8bit:emr-interrupts/pie1-reg.png)

`TMR1GIE -` Timer1 Gate Interrupt Enable

`ADIE -` Analog-to-Digital Converter (ADC) Interrupt Enable

`RCIE -` Universal Synchronous Asynchronous Receiver Tranmsitter (USART) Receive Interrupt Enable

`TXIE -` USART Transmit Interrupt Enable

`SSPIE -` Synchronous Serial Port (MSSP) Interrupt Enable

`CCP1IE -` CCP1 Interrupt Enable

`TMR2IE -` Timer2 Interrupt Enable

`TMR1IE -` Timer1 Interrupt Enable

PIE2 register

OSFIE	C2IE	C1IE	EEIE	BCLIE	LCDIE	----	CCP2IE
bit 7							bit 0

(/local--files/8bit:emr-interrupts/pie2-reg.png)

`OSFIE` - Oscillator Fail Interrupt Enable
`C2IE` - Comparator C2 Interrupt Enable
`C1IE` - Comparator C1 Interrupt Enable
`EEIE` - EEPROM Write Completion Interrupt Enable
`BCLIE` - MSSP Bus Collision Interrupt Enable
`LCDIE` - LCD Module Interrupt Enable
`---` - Unimplemented, read as 0
`CCP2IE` - CCP2 Interrupt Enable

PIE3 register

----	CCP5IE	CCP4IE	CCP3IE	TMR6IE	----	TMR4IE	----
bit 7							bit 0

(/local--files/8bit:emr-interrupts/pie3-reg.png)

`---` - Unimplemented read as 0
`CCP5IE` - CCP5 Interrupt Enable
`CCP4IE` - CCP4 Interrupt Enable
`CCP3IE` - CCP3 Interrupt Enable
`TMR6IE` - Timer6 Interrupt Enable
`---` - Unimplemented, read as 0
`TMR4IE` - Timer4 Interrupt Enable
`---` - Unimplemented, read as 0

`PIE1`, `PIE2`, and `PIE3` contain the individual interrupt enable flags for the MCU's peripherals.

Interrupt Request Registers

PIR1 register

TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMRI1F
bit 7							bit 0

(/local--files/8bit:emr-interrupts/pir1-reg.png)

`TMR1GIF` - Timer1 Gate Interrupt Flag
`ADIF` - ADC Interrupt Flag
`RCIF` - USART Receive Interrupt Flag
`TXIF` - USART Transmit Interrupt Flag
`SSPIF` - MSSP Interrupt Flag
`CCP1IF` - CCP1 Interrupt Flag
`TMR2IF` - Timer2 Interrupt Flag
`TMR1IF` - Timer1 Interrupt Flag

PIR2 register

OSFIF	C2IF	C1IF	EEIF	BCLIF	LCDIF	----	CCP2IF
bit 7							bit 0

(/local--files/8bit:emr-interrupts/pir2-reg.png)

`OSFIF` - Oscillator Fail Interrupt Flag
`C2IF` - Comparator C2 Interrupt Flag
`C1IF` - Comparator C1 Interrupt Flag
`EEIF` - EEPROM Write Completion Interrupt Flag
`BCLIF` - MSSP Bus Collision Interrupt Flag
`LCDIF` - LCD Module Interrupt Flag
`---` - Unimplemented, read as 0
`CCP2IF` - CCP2 Interrupt Flag

PIR3 register

----	CCP5IF	CCP4IF	CCP3IF	TMR6IF	----	TMR4IF	----
bit 7							bit 0

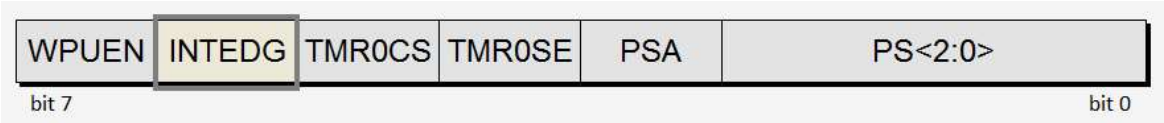
(/local--files/8bit:emr-interrupts/pir3-reg.png)

`---` - Unimplemented, read as 0
`CCP5IF` - CCP5 Interrupt Flag
`CCP4IF` - CCP4 Interrupt Flag
`CCP3IF` - CCP3 Interrupt Flag
`TMR6IF` - Timer6 Interrupt Flag
`---` - Unimplemented, read as 0
`TMR4IF` - Timer4 Interrupt Flag
`---` - Unimplemented, read as 0

`PIR1`, `PIR2`, and `PIR3` contain the individual interrupt request flags for the MCU's peripherals.

OPTION_REG

OPTION_REG



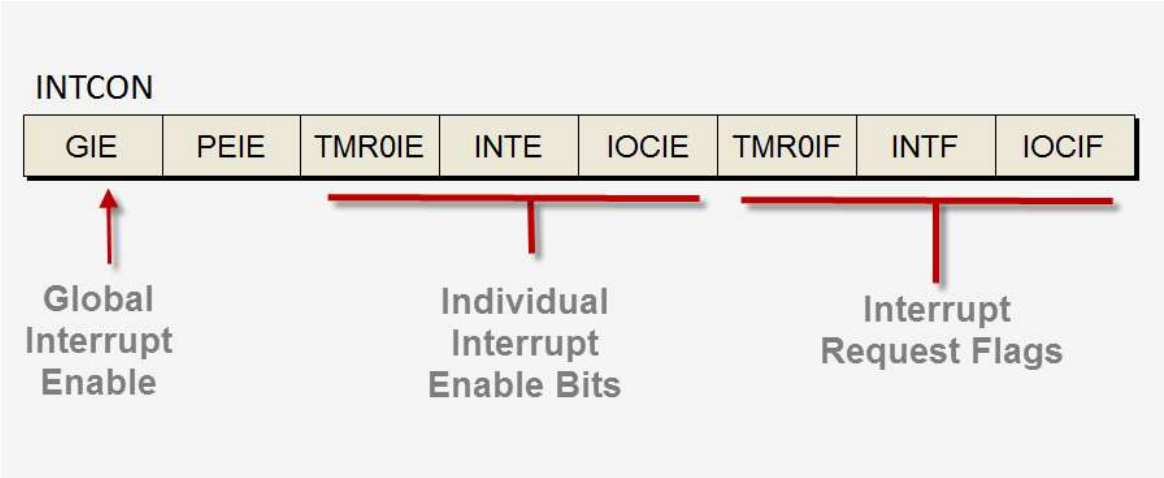
(/local--files/8bit:emr-interrupts/option-reg.png)

The INTEDG flag in OPTION_REG is used to set a rising or failing edge on the INT pin as the trigger for an INTE interrupt.

Enabling Interrupts

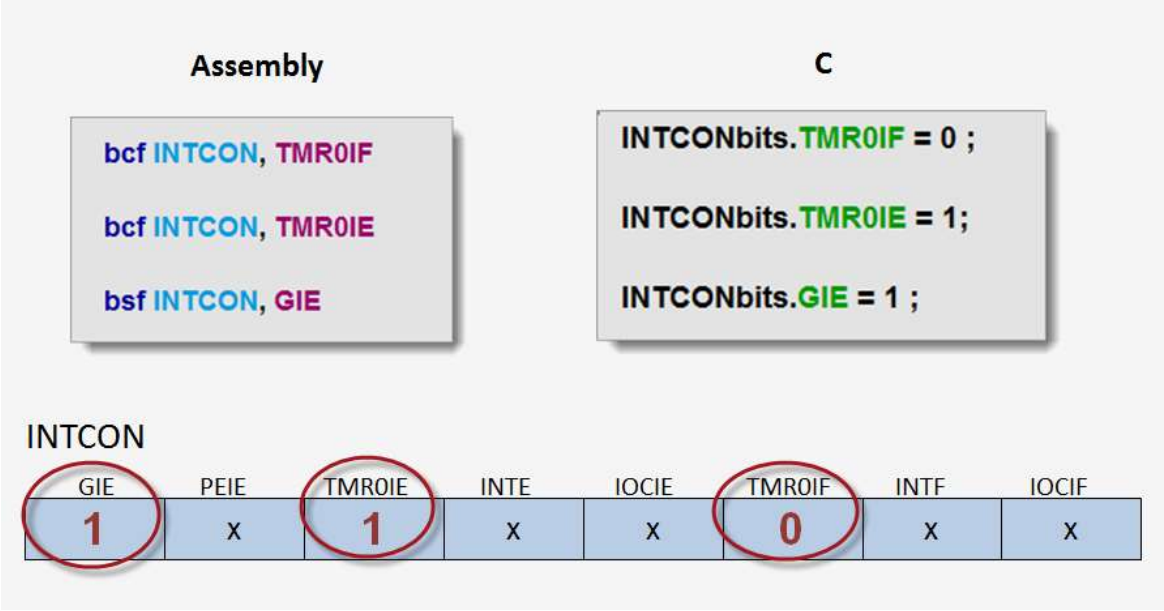
Core Interrupts

Three interrupt sources (Timer0, External Interrupt, and Interrupt on Change) have interrupt enable bits located in INTCON. These interrupts are referred to as core interrupts.



(/local--files/8bit:emr-interrupts/core-enable-reg.png)

To enable one of the core interrupts, only the individual Interrupt Enable bit and GIE need to be set.

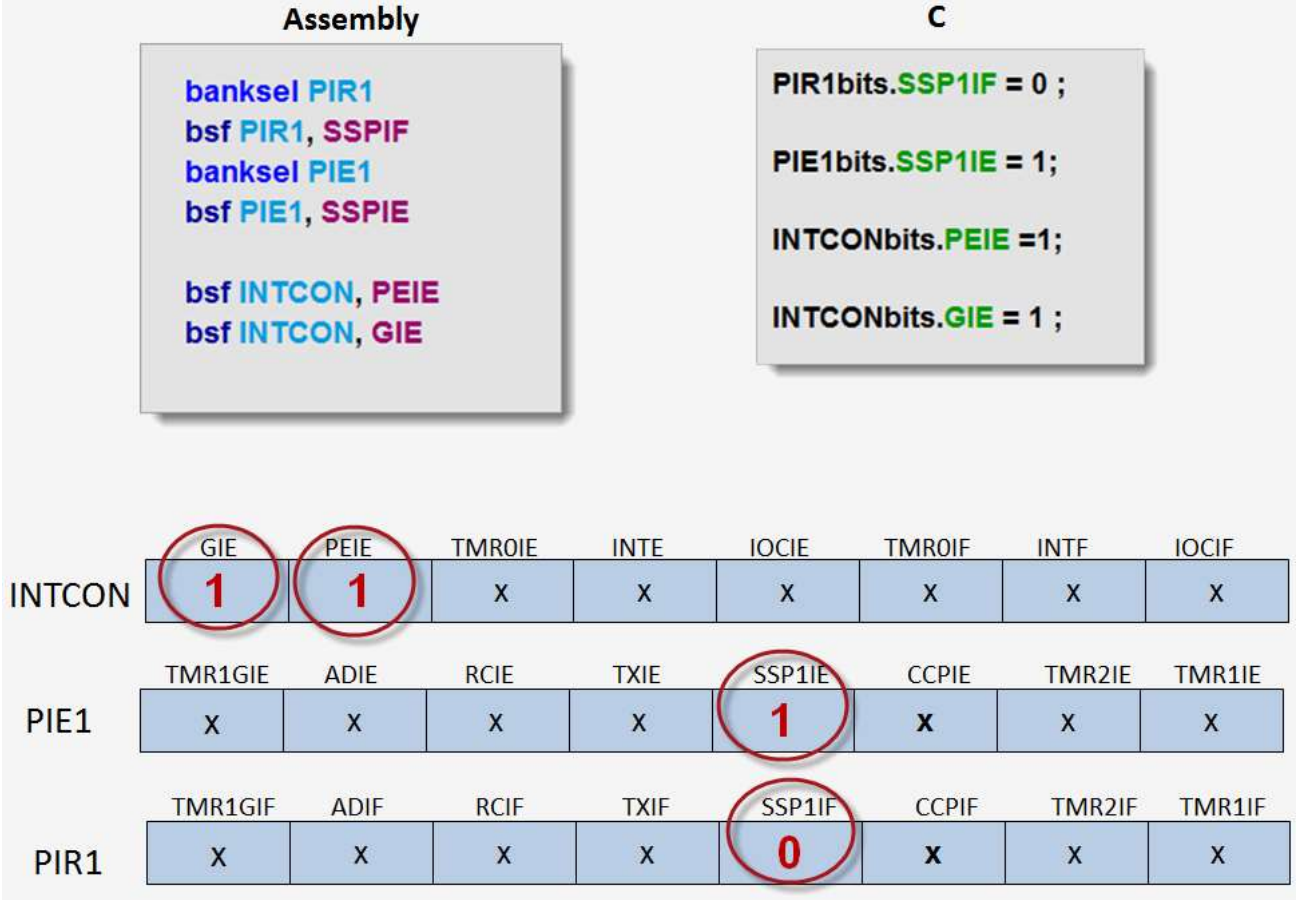


(/local--files/8bit:emr-interrupts/core-enable-code.png)

Clearing an Interrupt Request flag before setting the Interrupt Enable Flag prevents any pending interrupt requests from triggering an immediate interrupt.

Peripheral Interrupts

The PIC16F1xxx peripherals, capable of generating interrupt requests each, have their interrupt enable flags in one of the three PIE registers. To enable a peripheral interrupt, the individual interrupt flag, GIE, *and* PEIE must be all set.



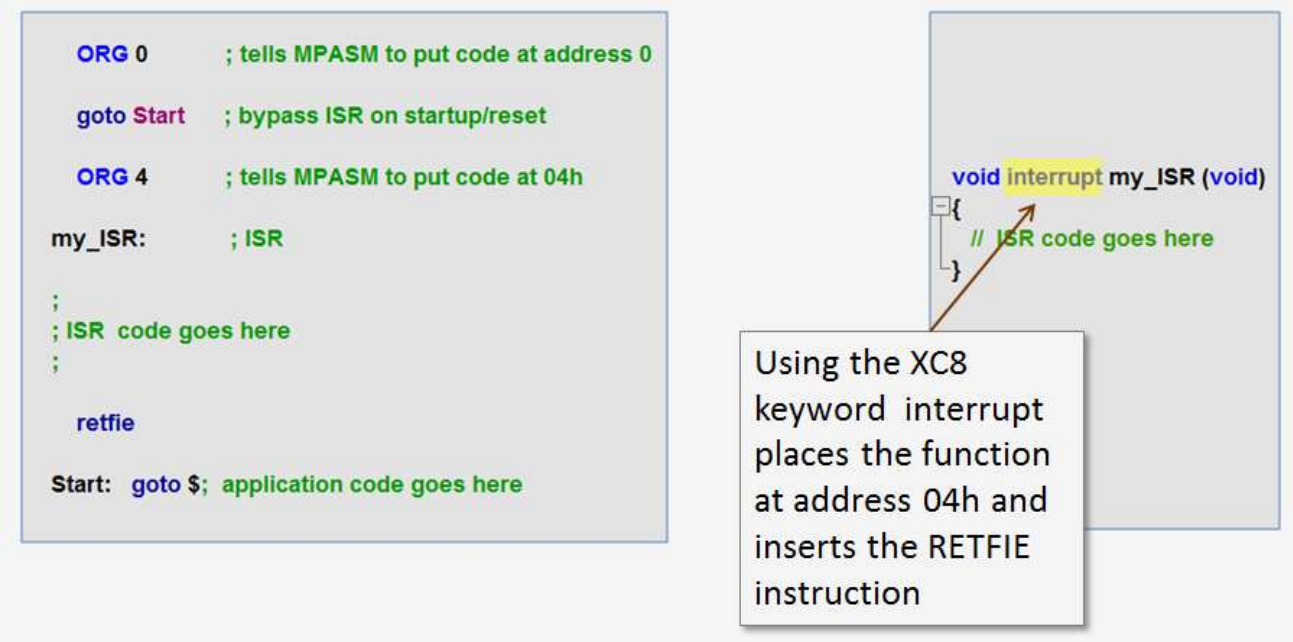
(/local--files/8bit:emr-interrupts/peie-code.png)

Servicing an Interrupt

ISR

The ISR is a user-written program that performs the necessary tasks when an interrupt occurs. The user is responsible for writing the ISR and placing it at address 04h. The last instruction executed by the ISR must be the RETFIE instruction. ISRs can be written in either C or assembly language.

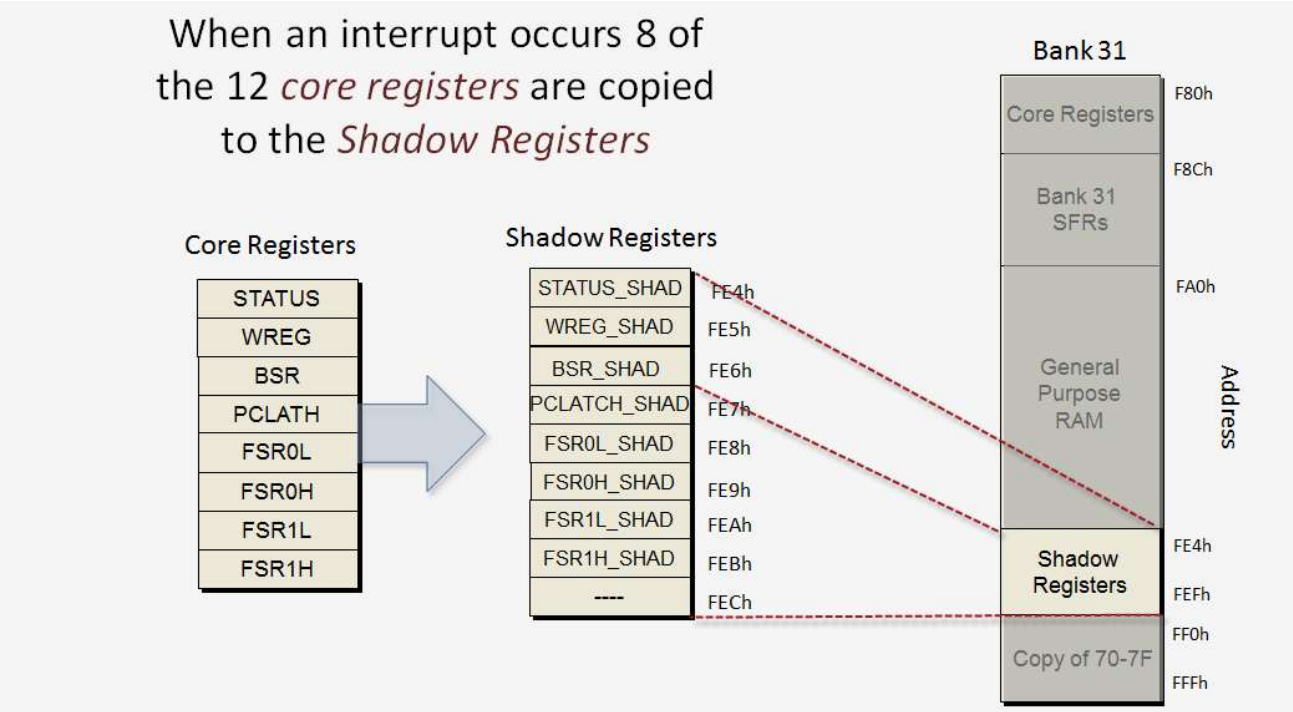
ISR in Assembly (left) and ISR in C (right)



(/local--files/8bit:emr-interrupts/isr-location.png)

Context Saving

The MCU's hardware will invoke the context saving mechanism when an interrupt occurs.



(/local--files/8bit:emr-interrupts/context-saving.png)

Verify Source of the Interrupt

There is one ISR which services all the interrupts for the application. The ISR needs to sequentially check the individual interrupt request flags to determine the source of the interrupt.

Clearing the Interrupt Request Flag

The Interrupt Request Flags are latched when set by the peripheral. They must be cleared by ISR. Failure of the ISR to reset a request flag will cause another interrupt to occur immediately after the ISR returns control to the main program.

```
void interrupt sample_isr (void)
{
   if (TMR2IF == 1)
   {
      // perfrom TMR2 Interrupt task
      TMR2IF = 0 ;
   }

   if (INTE == 1)
   {
     // External Interrupt tasks
      INTF = 0 ;
   }

   if (CCP1IF == 1)
      // CCP tasks
      CCP1IF =0;

}
```

(/local--files/8bit:emr-interrupts/sample-isr.png)

Sample Interrupt Code

Processing an Interrupt from Timer2

The animation below shows the code to setup and process an interrupt from Timer2 on the PIC16F1xxx enhanced mid-range MCU. The control icons at the bottom of the animation allow for the display to be paused and single stepped.

Further explanation of PIC16F1xxx interrupts can be found in the Interrupts (/mcu1102:interrupts) page.

The Timer2/4/6 (/8bit:timer2) page provides the details on the setup and operation of Timer2.
