INTERRUPTS, RESETS, AND CONFIGURATION

PIC Microcontroller: An Introduction to Software & Hardware Interfacing

Han-Way Huang

Thomson Delmar Learning, 2005

Chung-Ping Young

楊中平

Networked Embedded Applications and Technologies Lab

Department of Computer Science and Information Engineering

National Cheng Kung University, TAIWAN

Interrupts

- Without interrupts, the processor will execute programs by following the program logic flow.
- Interrupt is an event that will cause the CPU to stop the normal program execution and provide some service to the event.
- An interrupt can be generated inside and outside the microcontroller chip.
- An external interrupt is generated when the external hardware asserts an interrupt signal to the CPU.
- An internal interrupt can be generated by the hardware circuitry inside the chip and caused by software errors.
- Abnormal situations that occur during program execution, such as illegal opcodes, overflows, divide-by-zero, and underflow, are referred to as software interrupts.
- Software interrupts are also called **traps** and **exceptions**.
- A good analogy for interrupt is how one will act when there is a incoming phone call.

Applications of Interrupts

- Coordinate I/O activities and prevent CPU from being tied up during the data transfer process.
- Perform time-critical operation—one example is process control.
- Provide a graceful way to exit from the application when a software error occurred.
- Remind the CPU to perform routine tasks:
  1. Keep track of time of day
  2. Periodic data acquisition
  3. Task switching in a multi-tasking operating system
  4. Others

Interrupt Maskability

- Some interrupts are not desirable under some situations and should be ignored by the CPU. These interrupts are called **maskable** interrupt.
- Other interrupts represent events that the CPU shouldn't ignore and must take immediate action. These interrupts are called **nonmaskable** interrupts.
- A program can disable (mask) an interrupt by setting or clearing a bit for most processors.
- An interrupt is pending when it is active but not serviced by the CPU.

Interrupt Priority

- A microcontroller or microprocessor normally has multiple interrupt sources.
- More than one interrupt source can be pending at the same time.
- The CPU needs to decide which one of the pending interrupts to service.
- The solution for handling multiple pending interrupts is to prioritize all interrupts.
- The interrupt at the higher priority will be attended before the interrupt at the lower priority.
- Interrupt priority can be implemented by the hardware or software.
- The PIC18 supports two-level interrupt priority only and let the software to decide which pending interrupt to service.

Interrupt service

- The CPU provides service to an interrupt by executing an **interrupt service routine**.
- After providing service to the pending interrupt, the CPU must resume normal program execution.
- The complete interrupt service cycle involves the following:

1. Saving the program counter in the stack
2. Saving the CPU status in the stack (optional for some processor)
3. Identifying the interrupt source
4. Resolving the starting address of the corresponding interrupt service routine
5. Executing the interrupt service routine
6. Restoring the CPU status from the stack
7. Restoring the program counter from the stack
8. Resuming the interrupted program

- For maskable interrupts, the CPU starts to provide service to the interrupt after it completes the execution of the current instruction.
- For some maskable interrupts, the CPU may provide service to the interrupt without completing the current instruction.

Interrupt Vector

- The starting address of the interrupt service routine is called **interrupt vector**.
- There are three methods for determining the interrupt vector:

  1. Predefined
  2. Stored in a predefined memory location
  3. Execute an interrupt vector acknowledge cycle to fetch a vector number to locate the interrupt vector

- The PIC18 uses the first method to identify the interrupt vector.

Interrupt Programming

Step 1. Write the service routine

```
        org     ox08
        …                       ; interrupt service for high priority interrupts

        retfie
        org     0x18
        …                       ; interrupt service for low priority interrupts
        retfie
```

Step 2. Initialize the interrupt vector table
(not needed for PIC18)

Step 3. Enable interrupts to be serviced

The PIC18 Interrupt Sources

- Four edge-triggered INT pin (INT0…INT3) interrupts
- Port B pins change (any one of the upper four Port B pins)
- On-chip peripheral function interrupts. Not all the PIC18 members have the same number of peripheral function interrupts.

Registers Related to Interrupts

These registers enable/disable the interrupts, set the priority of the interrupts, and record the status of each interrupt source.

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2, and PIR3
- PIE1, PIE2, and PIE3
- IPR1, IPR2, and IPR3

Each interrupt source has three bits to control its operation:

- A flag bit
- An enable bit
- A priority bit

RCON register

7	6	5	4	3	2	1	0
IPEN	-	-	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

IPEN: Interrupt priority enable bit
0: Disable priority levels on interrupts
1: Enable priority levels on interrupts
$\overline{RI}$: RESET instruction flag bit
0: The reset instruction was executed causing a device reset
1: The reset instructionwas not executed
$\overline{TO}$: Watchdog timeout flag bit
0: A watchdog timeout occurred
1: After power-up, CLRWDT instruction, or SLEEP instruction
$\overline{PD}$: Power-down detection flag bit
0: By execution of the SLEEP instruction
1: After power up or by the CLRWDT instruction
$\overline{POR}$: Power-on reset status bit
0: A power-on reset has occurred
1: A power-on reset has not occurred
$\overline{BOR}$: Brown-out reset status bit (PIC18CX01 does not have this bit)
0: A brown-out reset has occurred
1: A brown-out reset has not occurred

Figure 6.1 The RCON Register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

GIE/GIEH: Global interrupt enable bit
  when IPEN (RCON<7>)= 0
   0: disables all interrupts
   1: enables all interrupts
  when IPEN = 1
   0: disables all interrupts
   1: enables all high priority interrupts
PEIE/GEIEL: Peripheral interrupt enable bit
  when IPEN = 0:
   0: disables all peripheral interrupts
   1: enables all peripheral interrupts
  when IPEN = 1
   0: disables all low priority interrupts
   1: enables all low priority interrupts
TMR0IE: TMR0 overflow interrupt enable bit
  0: disables TMR0 overflow interrupt
  1: enables TMR0 overflow interrupt
INT0IE: INT0 pin interrupt enable
  0: disables INT0 pin interrupt
  1: enables INT0 pin interrupt
RBIE: PORTB port change interrupt enable bit
  0: disables PORTB port change interrupt
  1: enables PORTB port change interrupt
TMR0IF: TMR0 overflow interrupt flag bit
  0: TMR0 has not overflowed
  1: TMR0 has overflowed
INT0IF: INT0 pin interrupt flag bit
  0: the INT0 pin interrupt did not occur
  1: the INT0 pin interrupt has occurred
PORTB port change interrupt flag bit
  0: none of the RB7:RB4 pins have changed state
  1: at least one of the RB7:RB4 pins change state

Figure 6.2a The **INTCON** register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMR0IP	INT3IP	RBIP

$\overline{\text{RBPU}}$: PORTB pull-up enable bit
0: all PORTB pull-ups are enabled
1: all PORTB pull-ups are disabled
INTEDG0..INTEDG3: INT0..INT3 interrupt pins edge select
0: interrupt on falling edge
1: interrupt on rising edge
TMR0IP: TMR0 overflow interrupt priority bit
0: low priority
1: high priority
INT3IP: INT3 interrupt priority bit (not available in P18FXX8 & P18CX01)
0: low priority
1: high priority
RBIP: PORTB change interrupt priority bit
0: low priority
1: high priority
Note. 1. PIC18FXX8 does not have INTEDG2 & INTEDG3)
2. PIC18C601/801 does not have INTEDG3

Figure 6.2b The INTCON2 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
INT2IP	INT1IP	INT3IE	INT2IE	INT1IE	INT3IF	INT2IF	INT1IF

INT2IP..INT1IP: INT2..INT1 interrupt priority bit
0: low priority
1: high priority
INT3IE..INT1IE: INT3..INT1 interrupt enable bit
0: disable interrupt
1: enable interrupt
INT3IF..INT1IF: INT3..INT1 interrupt flag bit
0: interrupt did not occur
1: interrupt occurred

Note. 1. PIC18FXX2 , PIC18CXX2, PIC18CXX8, and PIC18FXX8 do not have INT2 and INT3 enable and flag bits
2. PIC18C601/801 does not have INT3 enable and flag bits

Figure 6.2c The INTCON3 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
PSPIF(1)	ADIF	RC1IF	TX1IF	SSPIF	CCP1IF	TMR2IF	TMR1IF

PSPIF: Parallel slave port Read/Write interrupt flag bit (1)
0: no read or write has occurred
1: a read or write operation has taken place (must be cleared in software)
ADIF: A/D converter interrupt flag bit
0: the A/D conversion is not completed
1: an A/D conversion completed (must be cleared in software)
RCIF: USART receive interrupt flag bit
0 = the USART receive buffer is empty
1 = the USART receive buffer is full (cleared by reading RCREG)
TXIF: USART transmit tnterrupt flag bit
0 = the USART transmit buffer is full
1 = the USART transmit buffer is empty
SSPIF: Synchronous serial port interrupt flag bit
0 = waiting to transmit/receive
1 = the transmission/reception is complete (must be cleared in software)
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
0 = no TMR1 register compare match occurred
1 = a TMR1 register compare match occurred (must be cleared in software)
PWM mode
Unused in this mode
TMR2IF: TMR2 to PR2 match interrupt flag bit
0 = No TMR2 to PR2 match occurred
1 = TMR2 to PR2 match occurred (must be cleared in software)
TMR1IF: TMR1 overflow interrupt flag bit
0 = TMR1 register did not overflow
1 = TMR1 register overflowed (must be cleared in software)

Note 1. Enabled only in Microcontroller mode for the PIC18F8X20 devices
2. PIC18CX01 device does not have PSPIF flag bit

Figure 6.3a The PIC18 PIR1 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	CMIF	-	EEIF	BCLIF	LVDIF	TMR3IF	CCP2IF

CMIF: Comparator interrupt flag bit
  0: the comparator input has not changed
  1: the comparator input has changed (must clear in software)
EEIF: Data EEPROM/FLASH write operation interrupt flag bit
  0: the write operation is not complete
  1: the write operation is complete (must cleared in software)
BCLIF: Bus collision interrupt flag bit
  0: no bus collision
  1: a bus collision occurred while SSP module was transmission (I2C mode)
    (must be cleared in software)
LVDIF: low voltage detect interrupt flag bit
  0: the device voltage is above the low voltage detect trip point
  1: a low voltage condition occurred (must be cleared in software)
TMR3IF: TMR3 overflow interrupt flag bit
  0: TMR3 register did not overflow
  1: TMR3 register overflowed
CCP2IF: CCP2 interrupt flag bit
  Capture mode
  0: no TMR1 or TMR3 register capture occurred
  1: a TMR1 or TMR3 register capture occurred (must be cleared in software)
  Compare mode
  0: no TMR1 or TMR3 register compare match occurred
  1: TMR1 or TMR3 register compare match occurred (must be cleared in software)
  PWM mode:
  unused in this mode

Figure 6.3b The PIC18 PIR2 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	-	RC2IF	TX2IF	TMR4IF	CCP5IF	CCP4IF	CCP3IF

RC2IF: USART2 receive interrupt flag bit
0 = the USART receive buffer is empty
1 = the USART receive buffer is full (cleared when RCREG is read)
TX2IF: USART2 transmit interrupt flag bit
0 = the write operation is not complete
1 = the USART transmit buffer is empty (cleared when TXREG is written)
TMR4IF: TMR4 overflow interrupt flag bit
0 = TMR4 did not overflow
1 = TMR4 register overflowed (must be cleared in software)
CCPxIF: CCPx interrupt flag bit (x = 3, 4, 5)
Capture mode:
0 = No TMR1 or TMR3 register capture occurred
1 = A TMR1 or TMR3 register capture occured (must be cleared in software)
Compare mode:
0 = No TMR1 or TMR3 register compare match occurred
1 = A TMR1 or TMR3 register compare match occurred (must cleared in software)
PWM mode: (not used)

Figure 6.3c The PIR3 register (PIC18FXX20) (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
IRXIF	WAKIF	ERRIF	TXB2IF/ TXBnIF	TXB1IF	TXB0IF	RXB1IF/ RXBnIF	RXB0IF

IRXIF: Invalid message received interrupt flag bit
  0: an invalid message has not occurred on the CAN bus
  1: an invalid message has occurred on the CAN bus
WAKIF: Bus activity wakeup interrupt flag bit
  0: activity on the CAN bus has not occurred
  1: activity on the CAN bus has occurred
ERRIF: CAN bus error interrupt flag big
  0: an error has not occurred in the CAN module
  1: an error has occurred in the CAN module (multiple sources)
When CAN is in mode 0
TXB2IF..TXB0IF: Transmit buffer 2..0 interrupt flag bit
  0: transmit buffer 2..0 has not completed transmission of a message
  1: transmit buffer 2..0 has completed transmission of a message and may be reloaded
When CAN is in mode 1 or mode 2
TXBnIF: Any transmit buffer interrupt flag bit
  0: No message was transmitted
  1: One or more transmit buffer has completed transmission of a message and may reloaded
TXB1IF and TXB0IF are forced to 0 in mode 1 and mode 2
When CAN is in mode 0
RXB1IF..RXB0IF: Receive buffer 1..0 interrupt flag bit
  0: receive buffer 1 (or 0) has not received a new message
  1: receive buffer 1 (or 0) has received a new message
When CAN is in mode 1 or mdoe 2
RXBnIF: CAN receive buffer interupt flag big
  0: No receive buffer has received a new message
  1: One or more receive buffer has received a new message
RXB0IF is forced to 0 when in mode 1 or mode 2

Figure 6.3d The PIC18 PIR3 register (PIC18FXX8 or other devices with CAN)
(reprint with permission of Microchip)

7	6	5	4	3	2	1	0
PSPIE(1)	ADIE	RC1IE	TX1IE	SSPIE	CCP1IE	TMR2IE	TMR1IE

PSPIE: Parallel slave port Read/Write interrupt enable bit (1)
0: disables the PSP read/write interrupt
1: eables the PSP read/write interrupt
ADIE: A/D converter interrupt enable bit
0: disables the A/D interrupt
1: enables the A/D interrupt
RC1IE: USART1 receive interrupt enable bit
0 = disable the USART1 receive interrupt
1 = enable the USART1 receive interrupt
TX1IE: USART1 transmit interrupt enable bit
0 = disable the USART1 transmit interrupt
1 = enable the USART1 transmit interrupt
SSPIE: Synchronous serial port interrupt enable bit
0 = disables the MSSP interrupt
1 = enables the MSSP interrupt
CCP1IE: CCP1 interrupt enable bit
0 = disables the CCP1 interrupt
1 = enables the CCP1 interrupt
TMR2IE: TMR2 to PR2 match interrupt enable bit
0 = disables the TMR2 to PR2 match interrupt
1 = enables the TMR2 to PR2 match interrupt
TMR1IE: TMR1 overflow interrupt enable bit
0 = disables TMR1 register overflow interrupt
1 = enables TMR1 register overflow interrupt

Note 1. Enabled only in Microcontroller mode for the PIC18F8X20 devices
2. The PIC18C601/801 does not have PSPIE flag bit

Figure 6.4a The PIE1 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	CMIE	-	EEIE	BCLIE	LVDIE	TMR3IE	CCP2IE

CMIE: Comparator interrupt enable bit
0: disables the comparator interrupt
1: enables the comparator interrupt
EEIE: Data EEPROM/FLASH write operation interrupt enable bit
0: disables the write operation interrupt
1: enables the write operation interrupt
BCLIE: Bus collision interrupt enable bit
0: disables the bus collision interrupt
1: enables the bus collision interrupt
LVDIE: low voltage detect interrupt enable bit
0: disables the device voltage detect interrupt
1: enables the device voltage detect interrupt
TMR3IE: TMR3 overflow interrupt enable bit
0: disables the TMR3 overflow interrupt
1: enables the TMR3 overflow interrupt
CCP2IE: CCP2 interrupt enable bit
0: disables the CCP2 interrupt
1: enables the CCP2 interrupt
Note 1. The PIC18C601/801 has only the lower four bits of this register.
2. The PIC18FXX2 does not have the CMIF bit

Figure 6.4b The PIE2 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	-	RC2IE	TX2IE	TMR4IE	CCP5IE	CCP4IE	CCP3IE

RC2IE: USART2 receive interrupt enable bit
0: the comparator input has not changed
1: the comparator input has changed (must clear in software)
TX2IE: USART2 transmit interrupt enable bit
0: disables the USART2 transmit enable bit
1: enables the USART2 transmit enable bit
CCPxIE: CCPx interrupt enable bit (x = 3, 4, or 5)
0: disables the CCPx interrupt
1: enables the CCPx interrupt

Figure 6.4c The PIE3 register (PIC18FXX20) (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
IRXIE	WAKIE	ERRIE	TXB2IE/ TXBnIE	TXB1IE	TXB0IE	RXB1IE/ RXBnIE	RXB0IE

IRXIE: Invalid message received interrupt enable bit
  0: disables the invalid CAN message received interrupt
  1: enables the invalid CAN message received interrupt
WAKIE: Bus activity wakeup interrupt enable bit
  0: disables the bus activity wakeup interrupt
  1: enables the bus activity wakeup interrupt
ERRIE: CAN bus error interrupt enable big
  0: disables the CAN bus error interrupt
  1: enables the CAN bus error interrupt
When CAN is in mode 0:
TXB2IE..TXB0IE: Transmit buffer 2..0 interrupt enable bit
  0: disables transmit buffer 2..0 interrupt
  1: enables transmit buffer 2..0 interrupt
When CAN is in mode 1 or mode 2:
TXBnIE: CAN transmit buffer interrupt enable bit
  0: disable all transmit buffer interrupts
  1: enable transmit buffer interrupt; individual interrupt is enabled by TXBIE abd BIE0
When CAN is in mode 0:
TXB1IE and TXB0IE are forced to 0 in mode 1 and mode 2.
RXB1IE..RXB0IE: Receive buffer 1..0 interrupt enable bit
  0: disables receive buffer 1 (or 0) interrupt
  1: enables receive buffer 1 (or 0) interrupt
When CAN is in mode 1 or mdoe 2:
RXBnIE: CAN receive buffer interrupt enable bit
  0: disable all receive buffer interrupts
  1: enable receive buffer interrupts; individual interrupts is enabled by BIE0
RXB0IE is forced to 0 when in mode 1 and mode 2.

Figure 6.4d Contents of the PIC18 PIE3 register (PIC18FXX8 and other devices with CAN)
(reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	CMIP	-	EEIP	BCLIP	LVDIP	TMR3IP	CCP2IP

CMIP: Comparator interrupt priority bit
  0: low priority
  1: high priority
EEIP: Data EEPROM/FLASH write operation interrupt priority bit
  0: low priority
  1: high priority
BCLIP: Bus collision interrupt priority bit
  0: low priority
  1: high priority
LVDIP: low voltage detect interrupt priority bit
  0: low priority
  1: high priority
TMR3IP: TMR3 overflow interrupt priority bit
  0: low priority
  1: high priority
CCP2IP: CCP2 interrupt priority bit
  0: low priority
  1: high priority
Note 1. The PIC18C601/801 has only the lower four bits of this register.
2. The PIC18FXX2 does not have the CMIP bit

Figure 6.5b The IPR2 register (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
-	-	RC2IP	TX2IP	TMR4IP	CCP5IP	CCP4IP	CCP3IP

RC2IP: USART2 receive interrupt priority bit
0: low priority
1: high priority
TX2IP: USART2 transmit interrupt priority bit
0: low priority
1: high priority
CCPxIP: CCPx interrupt priority bit (CCP modules 3, 4, and 5)
0: low priority
1: high priority

Figure 6.5c The IPR3 register (PIC18FXX20) (reprint with permission of Microchip)

7	6	5	4	3	2	1	0
IRXIP	WAKIP	ERRIP	TXB2IP/ TXBnIP	TXB1IP	TXB0IP	RXB1IP/ RXBnIP	RXB0IP

IRXIP: Invalid message received interrupt priority bit
0: low priority
1: high priority
WAKIP: Bus activity wakeup interrupt enable bit
0: low priority
1: high priority
ERRIP: CAN bus error interrupt enable big
0: low priority
1: high priority
TXB2IP..TXB0IP: Transmit buffer 2..0 interrupt enable bit
In mode 0
0: low priority
1: high priority
In mode 1 and mode 2
TXBnIP is used as the priority bit for all CAN transmit buffers
0: low priority
1: high priority
Bit 3 and 2 are forced to 0 in CAN mode 1 and mode 2.
RXB1IP..RXB0IP: Receive buffer 1..0 interrupt enable bit
In mode 0
0: low priority
1: high priority
In mode 1 and 2
RXBnIP is used to enable the priority of all receive buffer
0: low priority
1: high priority
Bit 0 is forced to 0 in CAN mode 1 and mode 2.

Figure 6.5d The IPR3 register (Devices with CAN) (reprint with permission of Microchip)

PIC18 Interrupt Operation

- The interrupt priority scheme can be enabled or disabled.
- All interrupts are divided into **core group** and **peripheral group**.
- The priority bits of the interrupts sources in the core group are contained in one of the interrupt control registers.
- When the priority scheme is enabled, all high-priority interrupts are under the control of a two-level interrupt enabling mechanism.
- All low-priority interrupts are under the control of a three-level interrupt enabling scheme.
- When the priority scheme is disabled, all interrupts in the core group are under the control of a two-level interrupt enabling scheme.
- All interrupts in the peripheral group are under the control of a three-level enabling scheme.
- The following interrupts are in the core group:
  1. INT0…INT3 pin interrupts
  2. TMR0 overflow interrupt
  3. PORTB input pin (RB7…RB4) change interrupts

- When an interrupt is responded to, the GIE bit is cleared to disable further interrupt. the return address is pushed onto return address stack and the PC is loaded with the interrupt vector.

PIC18 Interrupts without Setting Priority

- The priority scheme is disabled by clearing the bit 7 of the RCON register.
- All interrupt sources share the common interrupt vector at 0x08.
- In order to identify the cause of interrupt, one need to check each individual interrupt flag bit.
- The interrupt sources in the **core group** can be enabled by setting the GIE bit and the corresponding enable bit of the interrupt source.
- To enable TMR0 interrupt, for example, one must set both the GIE and the TMR0IE bits to 1.
- The interrupts in the peripheral group can be enabled by setting the GIE, PEIE, and the associated interrupt enable bits.
- To enable A/D interrupt, one needs to set the GIE, PEIE, and the ADIE bits

PIC18 Interrupts with Priority Enabled

- Should be used when certain interrupts need to be serviced promptly.
- The PEIE bit of the INTCON register is used as the low-priority interrupt enable bit.
- A high-priority interrupt is enabled when the GIEH bit and its interrupt enable bit are set to 1.
- A low-priority interrupt is enabled when the GIEH, GIEL, and its associated interrupt enable bits are all set to 1.
- By default, all interrupts are placed in high priority.
- The pending interrupts in the high-priority group will always be served before the interrupts in the low-priority group.
- Interrupt flags must be cleared in the interrupt service routine to avoid recursive interrupts.

INT0…INT3 Pin Interrupts

- All INT pins interrupt are edge-triggered.
- The edge-select bits are contained in the INTCON2 register.
- When an edge-select bit is set to 1, the corresponding INT pin interrupt on the rising edge.

Port B Pins Input Change Interrupt

- An input change on pins RB7…RB4 sets the flag bit RBIF (INTCON<0>).
- If the RBIE bit is set, then the setting of the RBIF bit causes an interrupt.
- In order to use this interrupt, the RB7…RB4 pins must be configured for input.

TMR0 Overflow Interrupt

- The Timer0 module contains a 16-bit timer TMR0.
- Timer0 can operate in either 8-bit or 16-bit mode.
- When TMR0 rolls over from 0xFF to 0x00 or from 0xFFFF to 0x0000, it may trigger an interrupt.

PIC18 Interrupt Programming

Two steps are needed:

Step 1

Write the interrupt service routine and place it in the predefined program memory.

Step 2

Set the appropriate interrupt enable bits to enable the desired interrupt.

A circuit for illustrating interrupt programming is shown in Figure 6.6b.

- The circuit is available in SSE452, SSE8680, and SSE8720 demo boards.
- Connect the 1 Hz output from the function generator to the INT0 pin.
- By enabling the INT0 interrupt, the INT0 pin will generate an interrupt to the CPU once every second.
- The interrupt service routine simply increments a memory counter, outputs it to the Port D pins, and returns.
- An LED is turned on when its driving Port D pin outputs a low.

Example 6.1 Write an assembly program to handle the **INT0** interrupt for the circuit shown in Figure 6.6b.

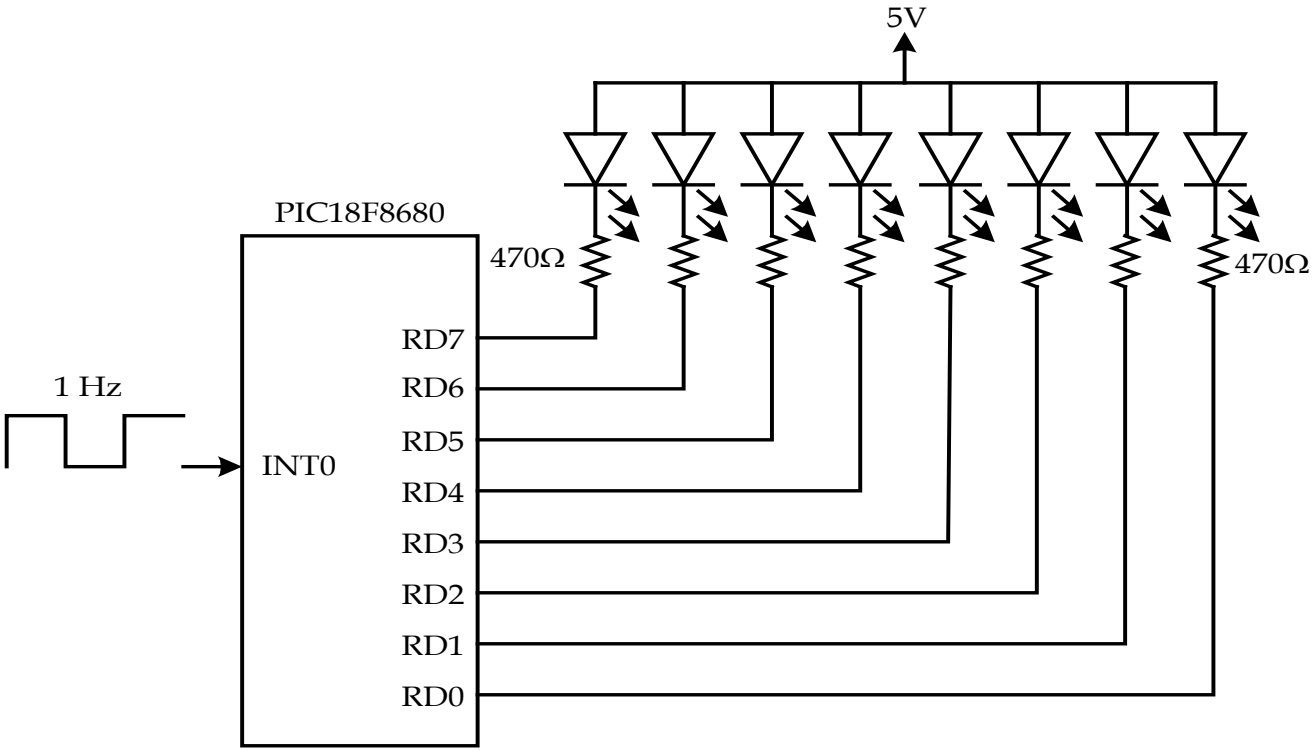

Figure 6.6b INT0 interrupt circuit and LEDs

```
            #include    <p18F8680.inc>
count       equ         0x00
            org         0x00
            goto        start
            org         0x08
            goto        ISR_hi
            org         0x18
            retfie
start       setf        count           ; count down from 255
            clrf        TRISD           ; configure port D for output
            movff       count,PORTD     ; output count to LEDs
            bsf         RCON,IPEN       ; enable priority interrupt
            movlw       0x90            ; enable INT0 interrupt
            movwf       INTCON          ; and clear INT0IF flag
forever     nop
            bra         nop

ISR_hi      btfss       INTCON,INT0IF   ; check interrupt source
            retfie                      ; not caused by INT0, return
            decf        count,F
            movff       count,PORTD     ; output count to LEDs
            retfie
```

Interrupt Programming Template in C

```
#include <p18Fxxx.h>
void low_ISR(void);
void high_ISR(void);
#pragma code high_vector = 0x08          // force the following statement to start at
void high_interrupt (void)               // 0x08
{
      _asm
      goto  high_ISR;
      _endasm
}

#pragma code                             //return to the default code section
#pragma interrupt high_ISR
void high_ISR (void)
{
      …                                  //handle high-priority interrupts
}
```

```
#pragma code low_vector = 0x18      //force the following statements to start at
void low_interrupt (void)           //0x18
{
        _asm
        goto  low_ISR;
        _endasm
}

#pragma code                        //return to the default code section
#pragma interruptlow low_ISR
void low_ISR (void)
{
        …                           //handle low-priority interrupts
}

void main (void)
{
        …
}
```

Example 6.2 Write a C language version of the program for example 6.1.

```
#include <p18Fxxx.h>
void low_ISR(void);
void high_ISR(void);
unsigned char  count;
#pragma code high_vector = 0x08  // force the following statement to start at
void high_interrupt (void)       // 0x08
{
      _asm
      goto  high_ISR;
      _endasm
}
#pragma code                     //return to the default code section
#pragma interrupt high_ISR
void high_ISR (void)
{
      if (INTCONbits.INT0IF){    //handle high-priority interrupts
            count++;
            PORTD = count;
      }
}
```

```
#pragma code low_vector = 0x18          //force the following statements to start at
void low_interrupt (void)         //0x18
{
        _asm
        goto  low_ISR;
        _endasm
}
#pragma code         //return to the default code section
#pragma interruptlow low_ISR
void low_ISR (void)
{
        _asm              // handle low-priority interrupts
        retfie            // simply return
        _endasm
}
```

```
void main (void)
{
        TRISD           = 0x00;     // Configure PORTD for output
        count           = 0xFF;     // turn off all LEDs initially
        PORTD           = count;    //      "
        RCONbits.IPEN               = 1;  // enable priority interrupt
        INTCON = 0x90;                    // enable INT0 interrupt
        while (1);                  // stay in an infinite loop
}
```

Context Saving During Interrupts

- When an interrupt occurs, the PIC18 MCU saves WREG, BSR, and STATUS in the fast register stack.
- The user can use the **retfie fast** instruction to retrieve these registers before returning from the interrupt service routine.
- One can save additional registers in the stack if the interrupt service needs to modify these registers. These registers must be restored before returning from the service routine.
- In C language, one can add a **save clause** to the **#pragma** statement to inform the C compiler to generate appropriate instructions for saving additional registers.

```
#pragmainterrupt   high_ISR   save    = reg1,…,regn
#pragma  interrupt low_ISR    save    = reg1,…,regn
```

- A whole section of data can be saved by the following statements:

```
#pragma  interrupt high_ISR   save    = section("section name")
#pragma  interrupt low_ISR    save    = section("section name")
```

Resets

- Resets can establish the initial values for the CPU registers, flip-flops, and control registers of the interface chips so that the computer can function properly.
- The reset service routine will be executed after the CPU leaves the reset state.
- The reset service routine has a fixed starting address and is stored in the read only memory.
- The PIC18 can differentiate the following types of reset:
  1. Power-on reset (POR)
  2. MCLR pin reset during normal operation
  3. MCLR pin reset during sleep mode
  4. Watchdog timer (WDT) reset (during normal operation)
  5. Programmable brown-out reset (BOR)
  6. RESET instruction
  7. Stack full reset
  8. Stack underflow reset