Computer interfacing (CI)

5. Interrupts

5.1 Need of the interrupt mechanism

CPU and I/O speed mismatch:

CPUs are able to perform billions of instructions per second. I/O devices (buttons, keyboards, ADCs) provide 0.1 – 100K units of data per second.

Interrupts is a synchronization mechanism between an I/O device or peripheral and the CPU. The CPU will manage I/O data only when it is available.

Typical interrupt sources are:

- Input pins, port changes, IRQ lines.
- Timers (overflow), other time managing devices.
- Analog to Digital Converter (ADC), when data available.
- Serial communication ports, data available, port ready.

(Note that all of them are related to hardware events).

5.1 Need of the interrupt mechanism

Synchronization mechanisms:

(1) Blind cycle.

Waiting a fixed amount of time and assuming the I/O will complete the process. Useful when I/O speed is short or predictable.

Example:
```
LCD_print("Hola")
delay(10);
```

(2) Gadfly, busy waiting or polling.

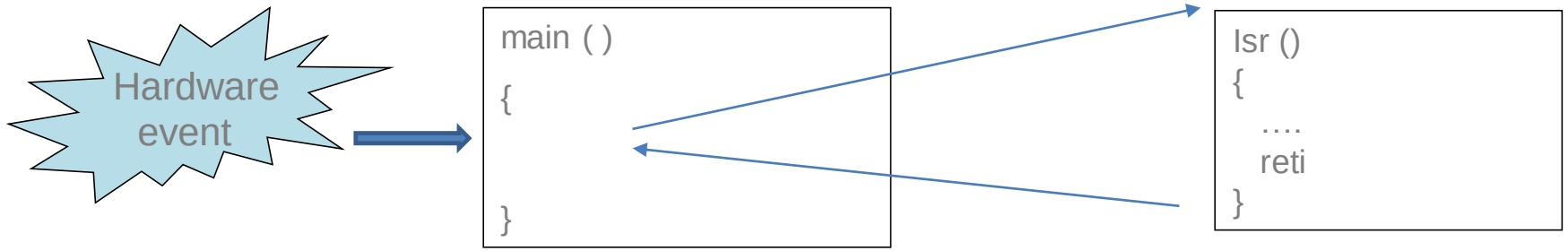
Waiting until an event arrives. Used when time response is not important (CPU can wait)

Example:
```
while ( PORTAbits.RA0 == 0);
```

5.1 Need of the interrupt mechanism

(3) Interrupt.

The program counter is changed to a special piece of code often named ISR (Interrupt Service Routine). Afterwards, program execution is restored.



Optimal strategy in CPU usage and time response. Needs hardware support!

(4) Periodic polling.

Using a clock interrupt for every given amount of time, check the status of all the devices. Several OS use this strategy. Not real time response.

(5) DMA.

Using a hardware coprocessor to transfer data between memory and devices (topic 9).

5.1 Need of the interrupt mechanism

Do **not confuse** Interrupts with other execution breaking mechanisms!!

An I/O interrupt is **asynchronous** with respect to instruction execution:

- I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
- I/O interrupt does not prevent any instruction from completion

Other execution-break mechanisms:

(1) Exceptions: signal marking that something "out of the ordinary" has happened and needs to be handled. Examples: divide by zero, Stack Overflow.

(2) Traps: synchronous events triggered by the software. Example (code): IRQx = 1;

5.1 Need of the interrupt mechanism

(3) Resets: Force the CPU to restart from scratch.

- 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 has a fixed starting address and is stored in the read only memory (PIC18F = 0x00000).

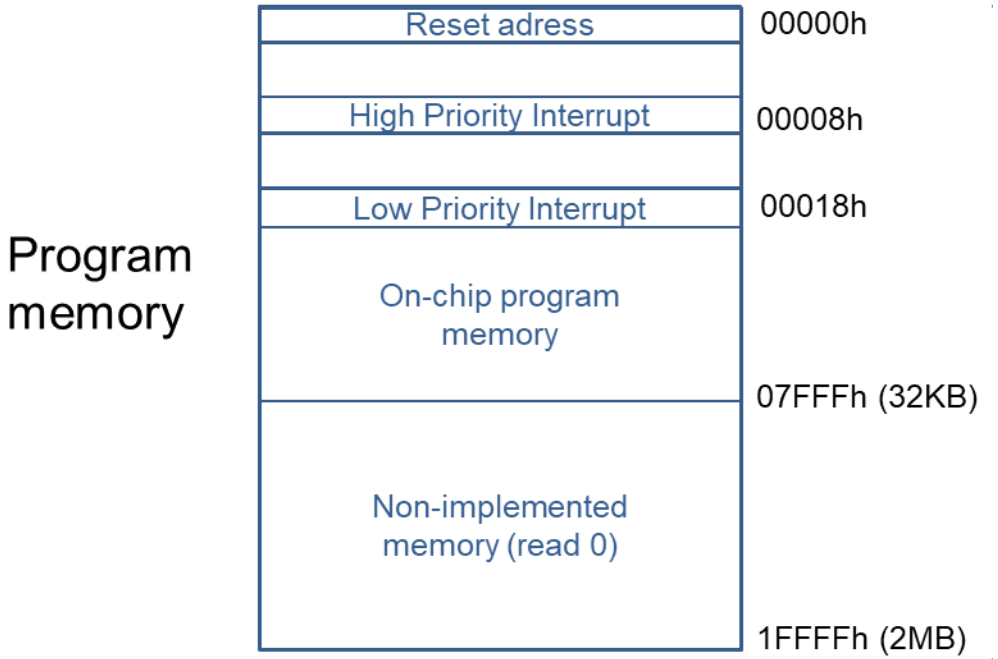
- The PIC18 can differentiate (Reset Register) the following causes of reset:

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

5.2 Hardware support for interrupts

Microprocessors must **provide** some support for the interrupt mechanism:

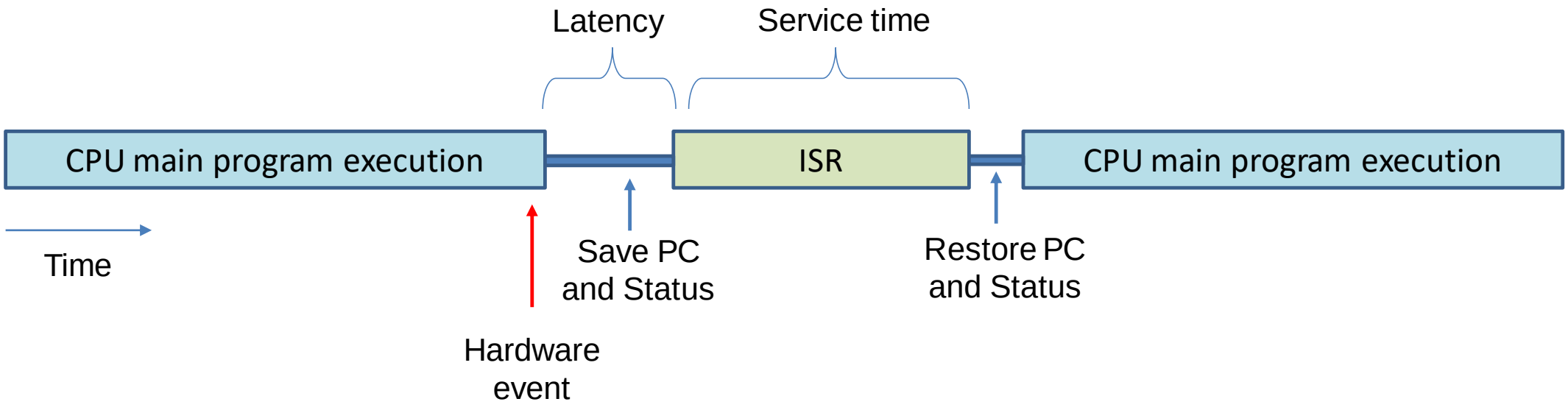
- Save the PC address for later return. Where? Special register, Stack…
- Save some special registers: Processor status, Working register…
- Determine the cause of the interrupt:
  Cause register or Flags, must be available.
- Provide a mechanism to place Interrupt Service Routines (ISR): Fixed addresses (0x08, 0x18), Interrupt vector…
- Priority mechanism: Which interrupt will be attended first ?
- Masking mechanism: Can an interrupt be ignored? Can they be globally ignored?



5.2 Hardware support for interrupts

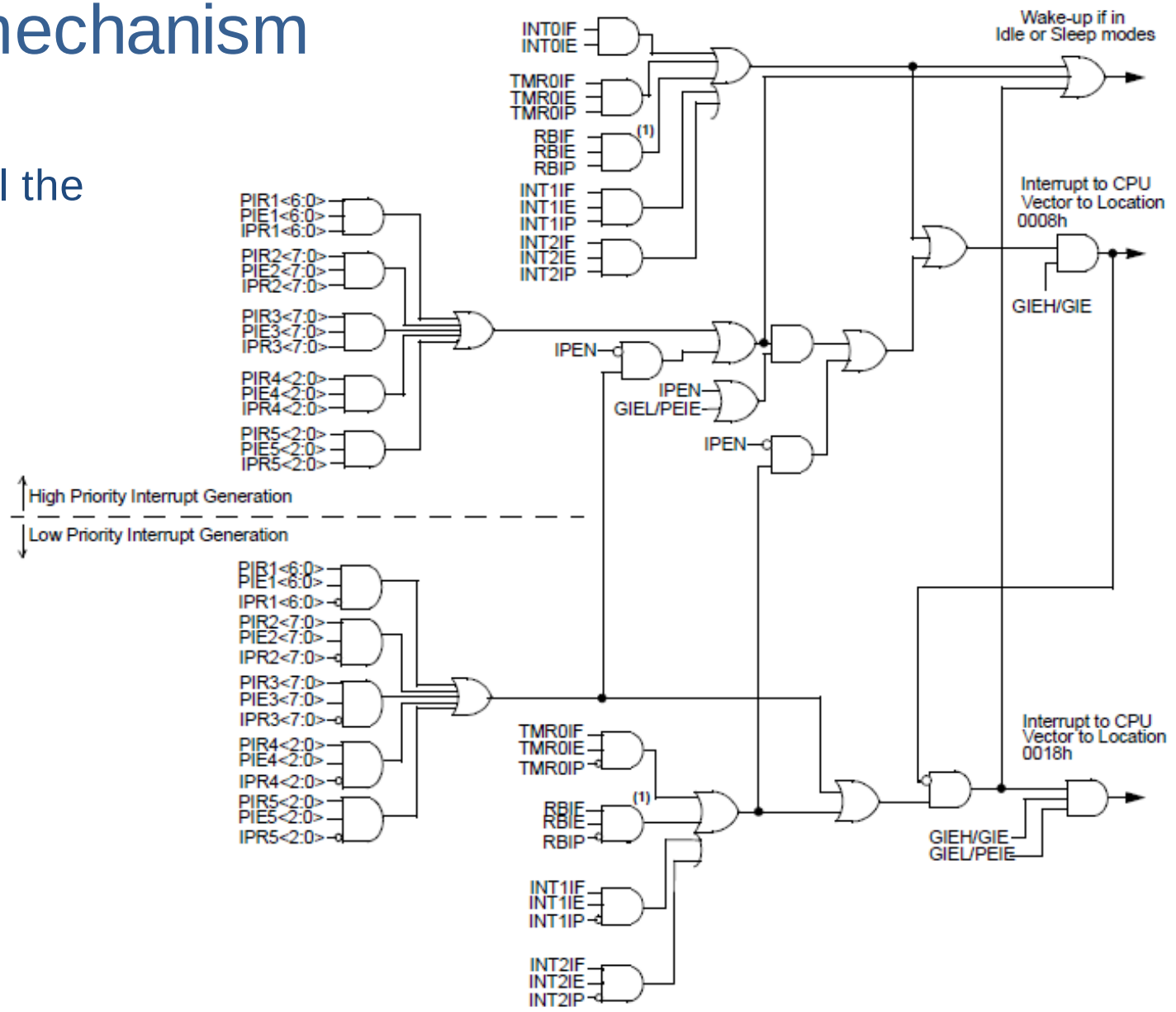
The hardware mechanism determine two main parameters of the interrupts:

- Latency: How long does it take to attend the interrupt ?
- Service time: How long takes the interrupt to be served ?



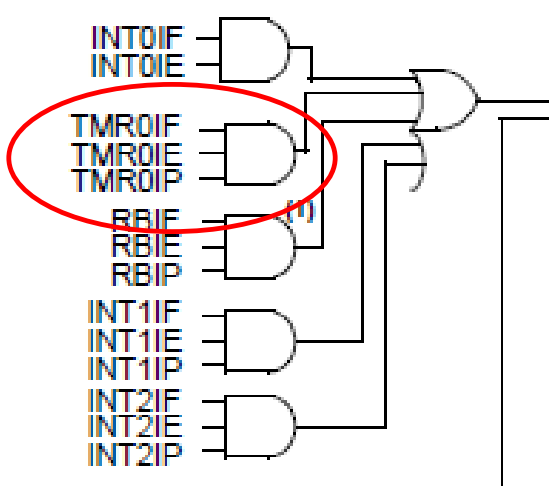
5.3 PIC18 interrupt mechanism

This logic scheme summarizes all the hardware support:

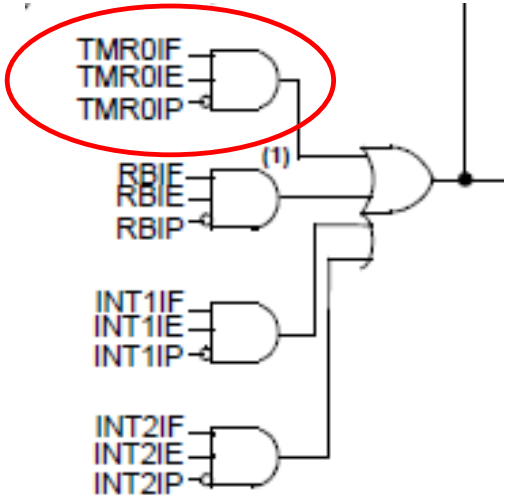


5.3 PIC18 interrupt mechanism

For every interrupt source we have an enable bit (IE) and a priority bit (IP).

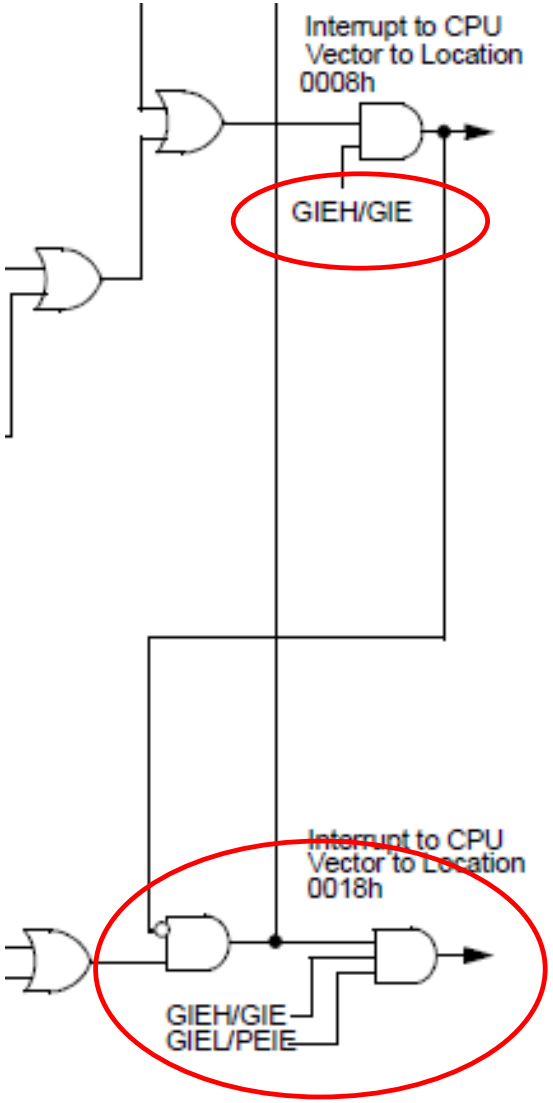


High Priority Path



Low Priority Path

There is a Global Interrupt Enable/ Disable bit.



High priority interrupts disable low priority interrupts

5.3 PIC18 interrupt mechanism

Interrupt operation

- All interrupts are divided into core group and peripheral group. The following interrupts are in the core group:

    1. INT0…INT2 pin interrupts
    2. TMR0 overflow interrupt
    3. PORTB input pin (RB7…RB4) change interrupts

- The interrupt sources in the core group can be enabled by setting the GIE bit and the corresponding enable bit of the interrupt source.

  Ex: To enable TMR0 interrupt, 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.

  Ex: To enable A/D interrupt, one needs to set the GIE, PEIE, and the ADIE bits

5.3 PIC18 interrupt mechanism

Interrupt operation

- In order to identify the cause of interrupt, one need to check each individual interrupt flag bit.

- 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.

- **Interrupt flags must be cleared** in the interrupt service routine to avoid reiterative interrupts.

5.3 PIC18 interrupt mechanism

Some interrupt sources:

INT0…INT2 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 interrupts 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.

5.3 PIC18 interrupt mechanism

PIC18 registers related to the interrupts:

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

RCON	Reset Control Register
INTCON	Interruption Control Registers
INTCON2	
INTCON3	
PIR1, PIR2, PIR3, PIR4 and PIR5	Peripheral IF's
PIE1, PIE2, PIE3, PIE4 and PIE5	Peripheral IE's
IPR1, IPR2, IPR3, IPR4 and IPR5	Peripheral IP's

Each interrupt source has three bits to control its operation:

An Interruption flag (IF) bit
An Interruption enable (IE) bit
An Interruption priority (IP) bit

5.3 PIC18 interrupt mechanism

REGISTER 9-10: RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(2)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX Compatibility mode)

bit 6 **SBOREN:** BOR Software Enable bit(1)
For details of bit operation, see Register 4-1.

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit
For details of bit operation, see Register 4-1.

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 4-1.

bit 2 **$\overline{PD}$:** Power-Down Detection Flag bit
For details of bit operation, see Register 4-1.

bit 1 **$\overline{POR}$:** Power-on Reset Status bit(2)
For details of bit operation, see Register 4-1.

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
For details of bit operation, see Register 4-1.

5.3 PIC18 interrupt mechanism

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high-priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low-priority peripheral interrupts (if GIE/GIEH = 1)
0 = Disables all low-priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

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

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit(1)
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

5.3 PIC18 interrupt mechanism

REGISTER 9-2: INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Note: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global interrupt enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

5.3 PIC18 interrupt mechanism

REGISTER 9-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Note: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global interrupt enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

5.3 PIC18 interrupt mechanism

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'.

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared by software)
0 = The A/D conversion is not complete or has not been started

bit 5 **RC1IF:** EUSART1 Receive Interrupt Flag bit
1 = The EUSART1 receive buffer, RCREG1, is full (cleared when RCREG1 is read)
0 = The EUSART1 receive buffer is empty

bit 4 **TX1IF:** EUSART1 Transmit Interrupt Flag bit
1 = The EUSART1 transmit buffer, TXREG1, is empty (cleared when TXREG1 is written)
0 = The EUSART1 transmit buffer is full

bit 3 **SSP1IF:** Master Synchronous Serial Port 1 Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared by software)
0 = Waiting to transmit/receive

bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR register capture occurred (must be cleared by software)
0 = No TMR register capture occurred
Compare mode:
1 = A TMR register compare match occurred (must be cleared by software)
0 = No TMR register compare match occurred
PWM mode:
Unused in this mode

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

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

5.3 PIC18 interrupt mechanism

REGISTER 9-9: PIE1: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 1

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'.

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RC1IE:** EUSART1 Receive Interrupt Enable bit
1 = Enables the EUSART1 receive interrupt
0 = Disables the EUSART1 receive interrupt

bit 4 **TX1IE:** EUSART1 Transmit Interrupt Enable bit
1 = Enables the EUSART1 transmit interrupt
0 = Disables the EUSART1 transmit interrupt

bit 3 **SSP1IE:** Master Synchronous Serial Port 1 Interrupt Enable bit
1 = Enables the MSSP1 interrupt
0 = Disables the MSSP1 interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

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

5.3 PIC18 interrupt mechanism

REGISTER 9-14: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6 **ADIP:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RC1IP:** EUSART1 Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TX1IP:** EUSART1 Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSP1IP:** Master Synchronous Serial Port 1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP:** CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

5.3 PIC18 interrupt mechanism

Interrupt register summary:

TABLE 9-1: REGISTERS ASSOCIATED WITH INTERRUPTS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
ANSELB	—	—	ANSB5	ANSB4	ANSB3	ANSB2	ANSB1	ANSB0	150
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	109
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	110
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	111
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	—	—	—	—	153
IPR1	—	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP	121
IPR2	OSCFIP	C1IP	C2IP	EEIP	BCL1IP	HLVDIP	TMR3IP	CCP2IP	122
IPR3	SSP2IP	BCL2IP	RC2IP	TX2IP	CTMUIP	TMR5GIP	TMR3GIP	TMR1GIP	123
IPR4	—	—	—	—	—	CCP5IP	CCP4IP	CCP3IP	124
IPR5	—	—	—	—	—	TMR6IP	TMR5IP	TMR4IP	124
PIE1	—	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE	117
PIE2	OSCFIE	C1IE	C2IE	EEIE	BCL1IE	HLVDIE	TMR3IE	CCP2IE	118
PIE3	SSP2IE	BCL2IE	RC2IE	TX2IE	CTMUIE	TMR5GIE	TMR3GIE	TMR1GIE	119
PIE4	—	—	—	—	—	CCP5IE	CCP4IE	CCP3IE	120
PIE5	—	—	—	—	—	TMR6IE	TMR5IE	TMR4IE	120
PIR1	—	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF	112
PIR2	OSCFIF	C1IF	C2IF	EEIF	BCL1IF	HLVDIF	TMR3IF	CCP2IF	113
PIR3	SSP2IF	BCL2IF	RC2IF	TX2IF	CTMUIF	TMR5GIF	TMR3GIF	TMR1GIF	114
PIR4	—	—	—	—	—	CCP5IF	CCP4IF	CCP3IF	115
PIR5	—	—	—	—	—	TMR6IF	TMR5IF	TMR4IF	116
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	148
RCON	IPEN	SBOREN	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$	56

5.4 Programming considerations

Assembly programming:

Step 1. Write the interrupt service routine and place it in the predefined program memory.

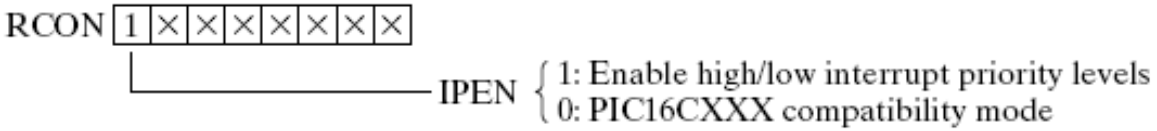
```
org     ox08
…                       ; interrupt service for high priority interrupts
retfie

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

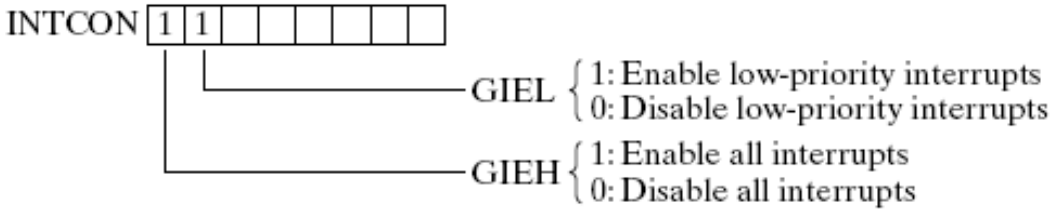
Step 2. Set the appropriate interrupt enable bits to enable the desired interrupt.

5.4 Programming considerations

- The interrupt priority scheme can be enabled or disabled using IPEN bit
- IPEN = 1 ==> enable priority levels
  IPEN = 0 ==> single interrupt level (all interrupts are high priority)



(a) Initialization for two levels of interrupt priority

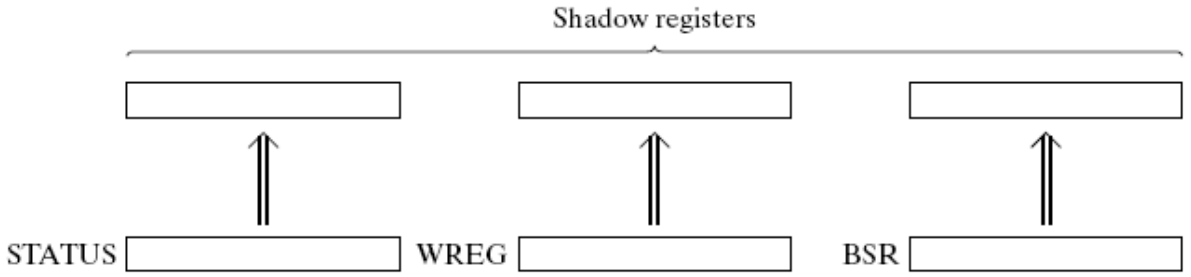


(b) Global interrupt enable bits

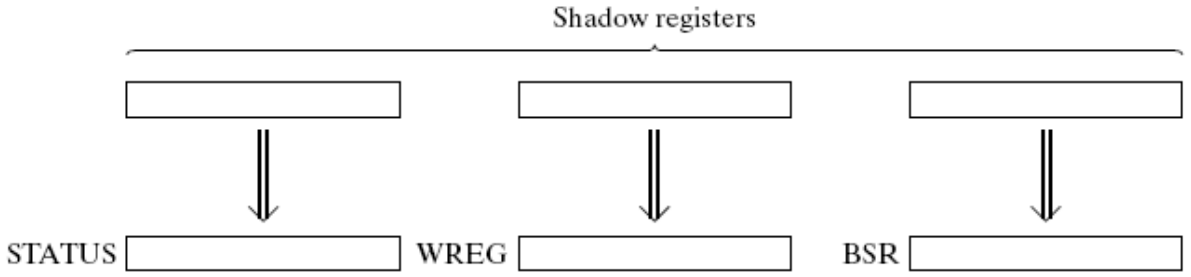
- GIEL/GIELH bit is automatically cleared when a low/high priority RSI is launched.
- *retfie* sets again the GIEL/H bit, and pops the PC from stack
- Each interrupt source has associated with it a priority bit. The default state of all priority bits is 'high' at power-on reset

5.4 Programming considerations

- High-priority interrupts are able to suspend the execution of low-priority ones
- When a high-priority interrupt occurs, STATUS, WREG and BSR registers are automatically copied to **shadow** registers
- retfie FAST restores the shadow registers, the PC, and sets GIEH bit
- **Never** use retfie FAST with low-priority interrupts



(a) Automatic setting aside of **STATUS**, **WREG**, and **BSR** when a high-priority interrupt occurs



(b) Automatic restoration of **STATUS**, **WREG**, and **BSR** in response to the "**retfie FAST**" instruction.

5.4 Programming considerations

- One can save additional registers if the interrupt service needs to modify these registers. These registers must be restored before returning from the service routine.

- In C language, some compilers allow the programmer to add a save clause to the #pragma statement to inform the compiler to generate appropriate instructions for saving additional registers.

```
#pragma  interrupt      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")
```

5.4 Programming considerations

C programming interrupt template:

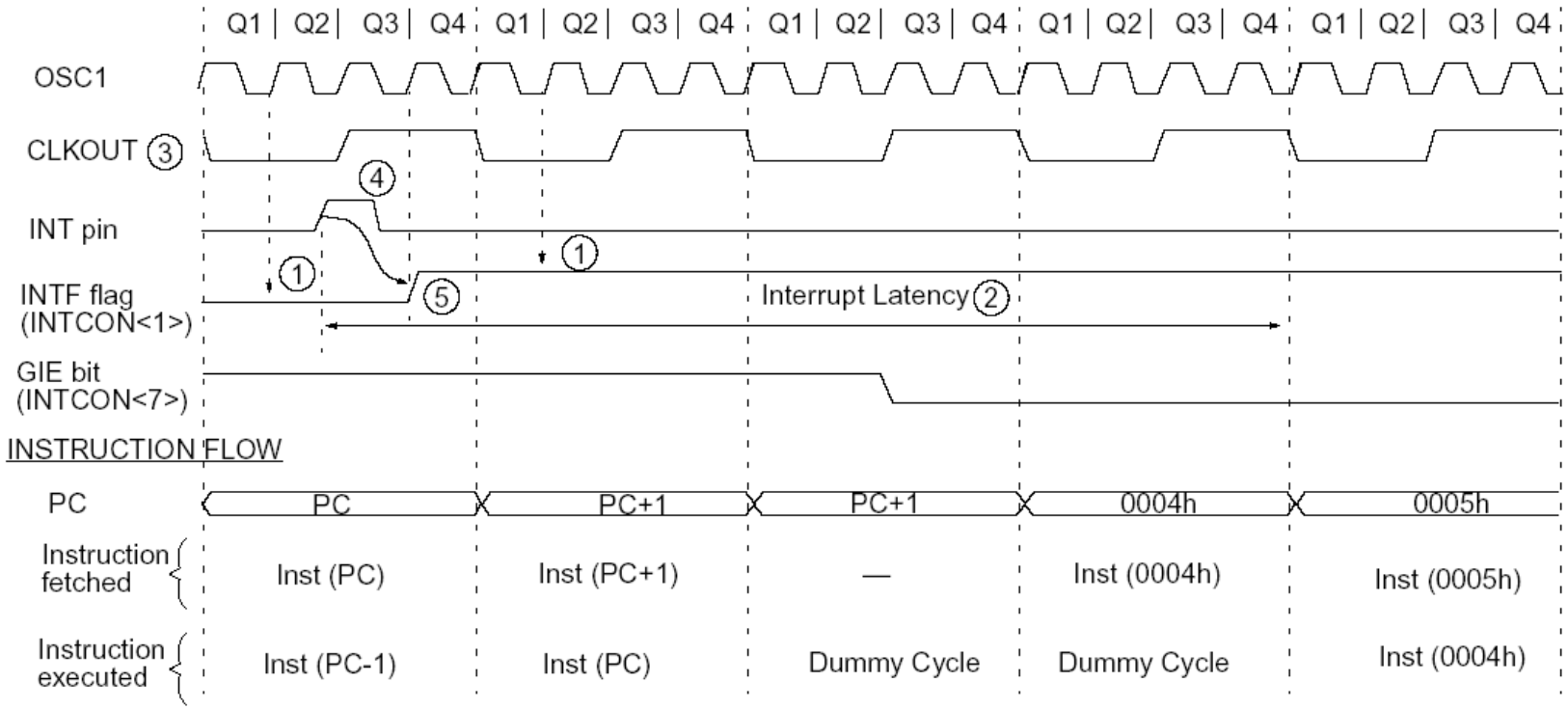
```
void interrupt tc_int(void)
{
      if (TMR0IE && TMR0IF)
      {
                  TMR0IF=0;
                  tick_count++;
      }
// process other interrupt sources here, if
// required
}
```

```
void interrupt low_priority tc_clr(void)
{
            if (TMR1IE && TMR1IF)
            {
                        TMR1IF=0;
                        tick_count = 0;
            }
// process any other low priority sources here, if
// required
}
```

C compilers place the interrupts in the right memory addresses and add the retfie instructions at the end of the code.

5.4 Programming considerations

Latency calculations:



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.

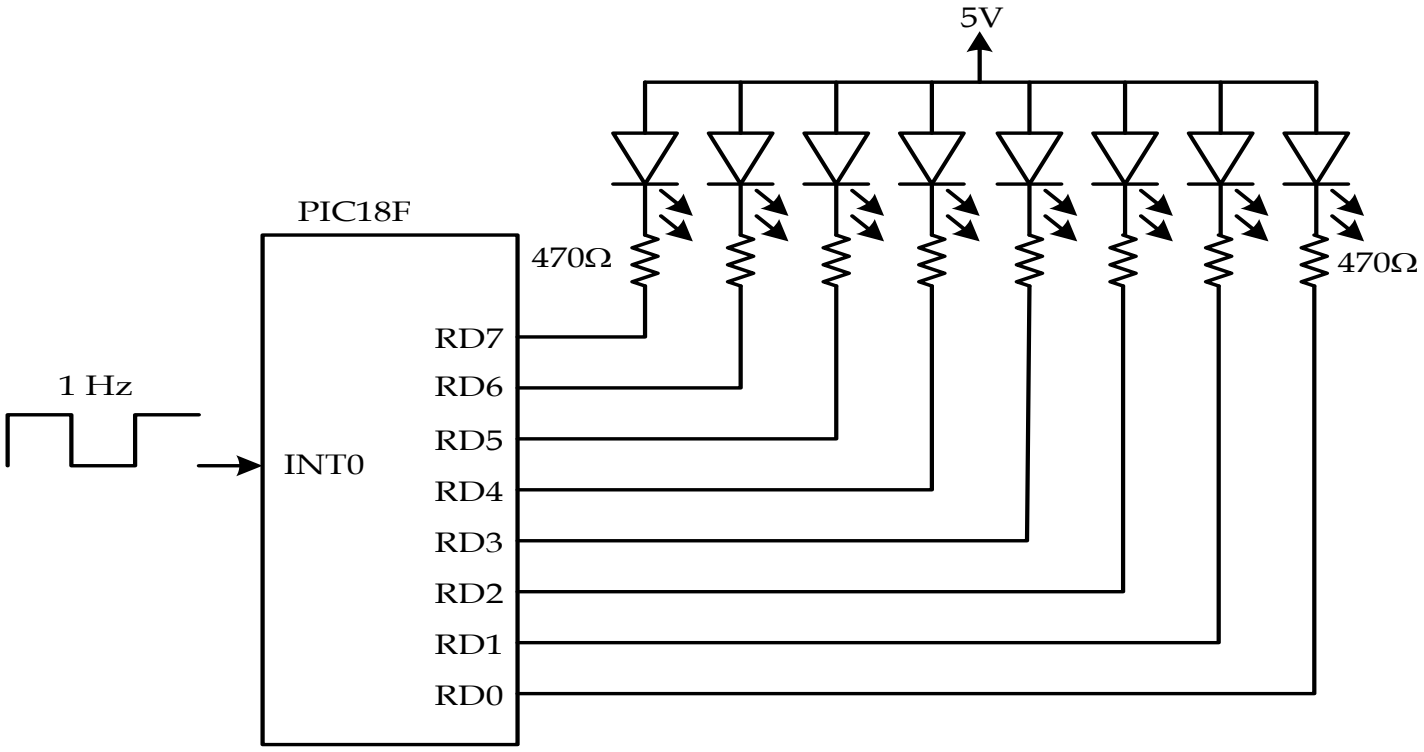
PIC-18 MCUs take 12 to 20 clock cycles to get to the ISR — depending on the type of instruction that was in progress at interrupt time.

Then, in the ISR, the compiler will add instructions to determine where the interrupt originated and to push some registers.

Absolute minimum will be 3 instructions or 12 clock cycles.

5.5 Programming Example

We put a 1Hz clock source to pin INT0. We want to enable an interrupt associated with this pin and program an interrupt service routine that increases a counter and shows it in PORTD.



5.5 Programming Example

```
#include <xc.h>

unsigned char count;

void interrupt service_routine_HighP (void)
{
      if (INTCONbits.INT0IF && INTCONbits.INT0IE)
      {
            INTCONbits.INT0IF=0;
            count++;
            PORTD = count;
      }
    // process other interrupt sources here, if required
}

void interrupt low_priority service_routine_LowP(void)
{    // Nothing to do for LP INT's in this example
}
```

```
void main(void)
 {
     // Init PIC
     ANSELD=0x00;    // PORT digital
     TRISD = 0x00;     // Configure PORTD for output
     ANSELB &=0xFE: // Bit 0 (INT0) digital
     TRISB |= 0x01     // Bit 0 (INT0) input
      count = 0xFF;    // turn off all LEDs initially
      PORTD = count;                          "
      RCONbits.IPEN = 1;  // enable priority interrupt
      INTCON = 0x90;   // enable GIEH and INT0 inter

       while (1);   // Main loop, is an idle loop
                    //  that waits for interruptions
 }
```