FIB UPC
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Interrupcions

Dpt. Enginyeria de Sistemes, Automàtica i Informàtica Industrial

Processor - I/O speed mismatch

- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 30,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device may not be ready to accept data as fast as processor stores it
- What to do?

Synchronization: Mechanisms

Blind Cycle: Software simply waits for a fixed amount of time and assumes the I/O will complete after that fixed delay.

Usage?

Where I/O speed is short and predictable

Gadfly (busy waiting, polling): is a software loop that checks the I/O status waiting for done status.

Usage?

When real time response is not important (CPU can wait)

Interrupts: uses hardware to cause special software execution i.e. input device will cause interrupt when it has new data!

Usage?

When real time response is crucial

Periodic Polling: Uses a clock interrupt to periodically check the I/O Status (i.e. The MCU or CPU will check the status)

Usage?

In situations that require interrupts but the I/O device does not support requests

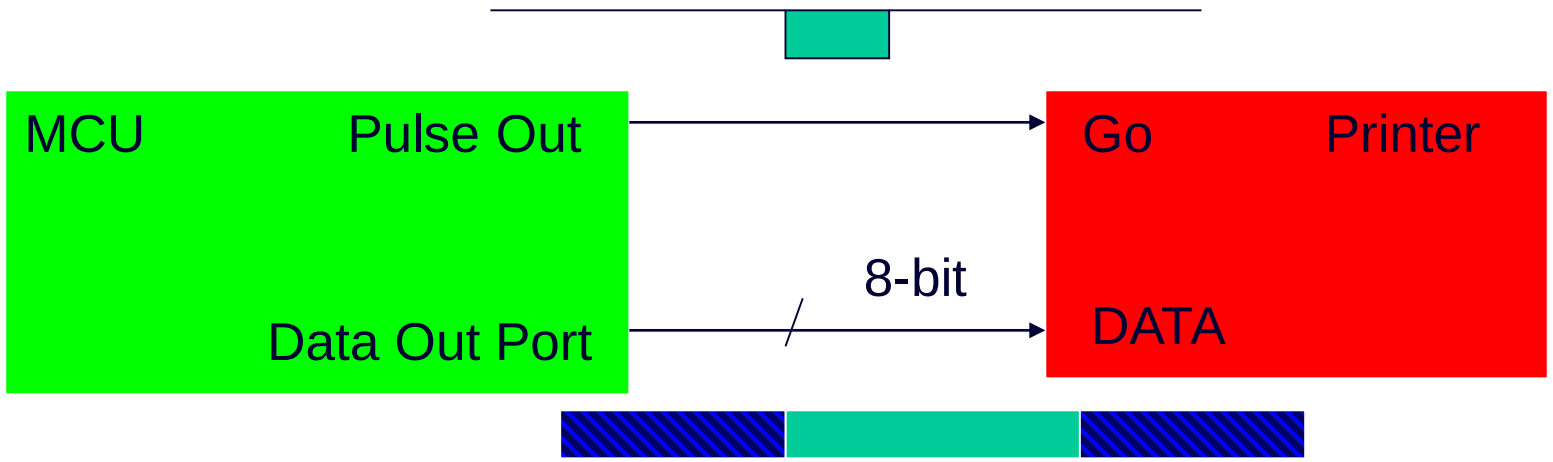
DMA: Transfer data directly to/from memory or I/O without CPU intervention.

Usage?

In situations where Bandwidth and latency are important.

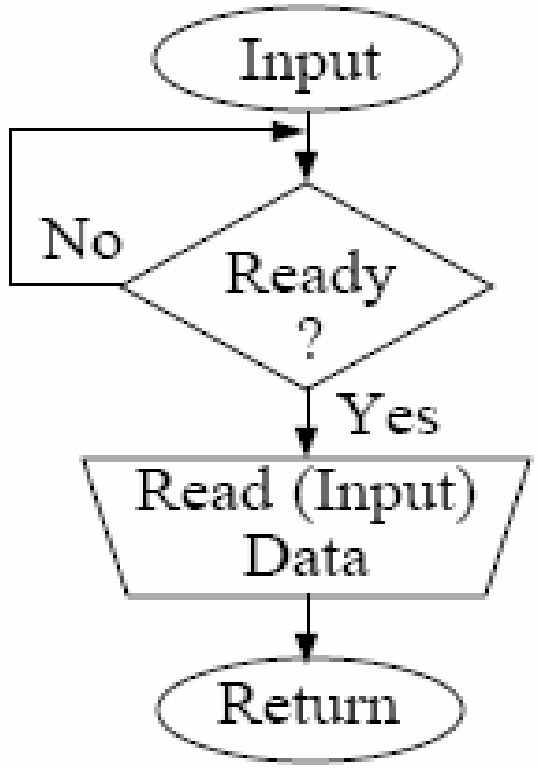
Blind Cycle Synchronization: Example

Printer can put 10 characters per second
With Blind Cycle there is no printer status signal from printer!
A simple software interface would be to output a character then wait 100 ms for it to finish.
Advantage?
Simple
Disadvantage?
If output rate is variable, then time delay is wasted.

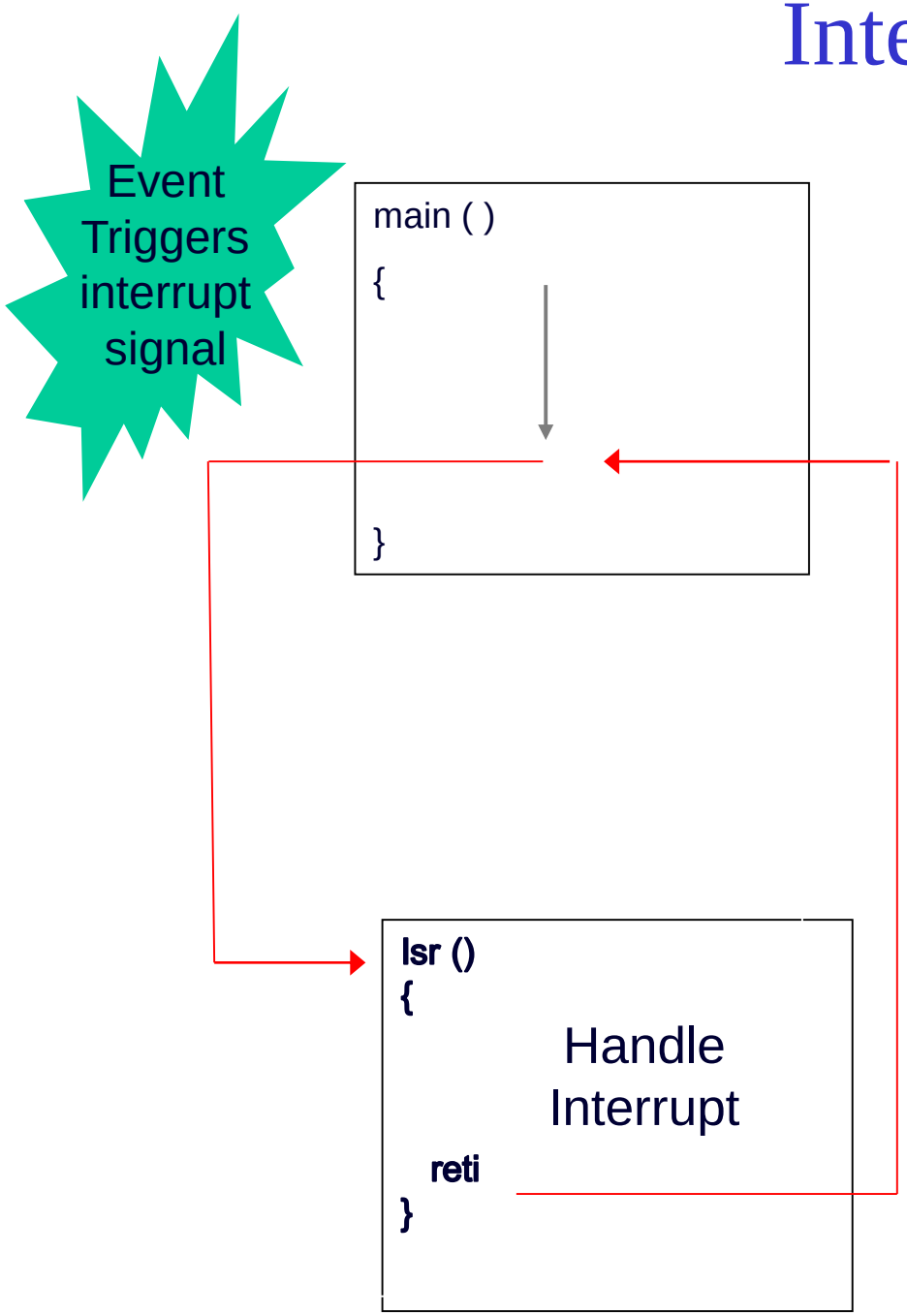


Gadfly (Busy Waiting)

- Software checks a status bit in the I/O device and loops back until device is ready.
- The **Gadfly loop must precede the data transfer for an input** device.



Interrupts



- Several steps have to be completed by the processor to return to the instruction following the instruction that was interrupted.

Support for interrupts

- Save the PC for return
  - But where?
- Save other special registers (Status…)
  - But where?
- Where to go when interrupt occurs?
  - MCU defines location. Ex: 0x08

- Determine cause of interrupt?
  - MCU has Cause Register, some bit field gives cause of exception

Questions

- Which I/O device caused exception?
  - Needs to convey the identity of the device generating the interrupt
- Can avoid interrupts during the interrupt routine?
  - What if more important interrupt occurs while servicing this interrupt?
  - Allow interrupt routine to be entered again?
- Who keeps track of status of all the devices, handle errors, know where to put/supply the I/O data?

Some parameters

- *Priority*: Which interrupt will be attended first ?
- *Masking*: Can an interrupt be ignored ?
- *Latency*: How long does it take to attend the interrupt ?
- *Service time*: How long takes the interrupt to be served ?

Interrupts

- An I/O interrupt is like overflow exceptions except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed

- 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

Interrupts, traps & exceptions

- Exception: signal marking that something “out of the ordinary” has happened and needs to be handled
- Interrupt: asynchronous exception
- Trap: synchronous exception

Interrupts: sources

Interrupts from on-chip resources.

examples: Serial Interface Module, timer overflow, ADC, …

External Interrupts.

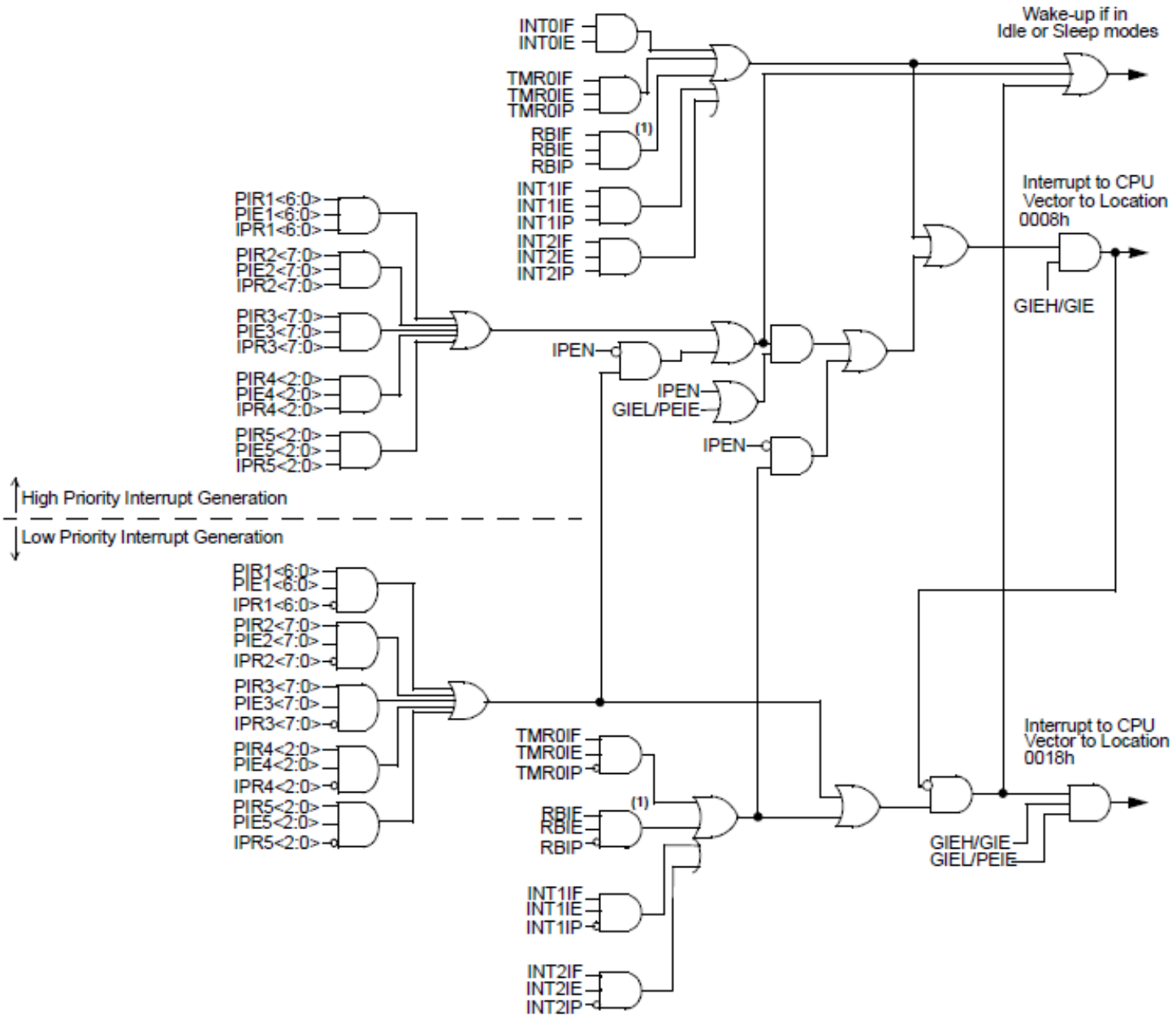
Examples: Input ports, IRQ line…

Software Interrupts.

Exceptions.

Examples: Opcode Trap, stack overflow, divide by zero…

PIC18 interrupt logic



PIC18 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 → 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

INTCON

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

INTCON2

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.

INTCON3

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.

RCON

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.

PIR1

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

IPR1

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

PIE1

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

Resum

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

Legend: — = unimplemented locations, read as '0'. Shaded bits are not used for Interrupts.

PIC18 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
- 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.

Interrupt sources. Examples

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.

Interrupt Programming

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

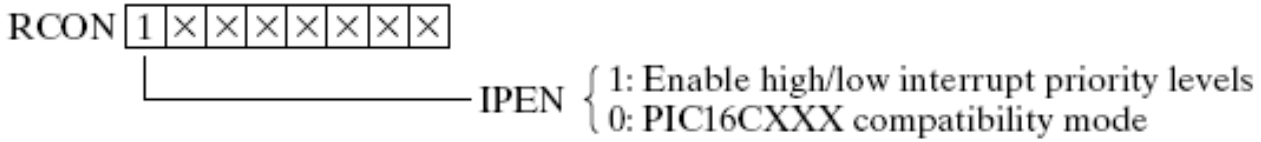
```
        org     ox08
        …                       ; interrupt service for high priority interrupts
        retfie
        org     0x18
        …                       ; interrupt service for low priority interrupts
        retfie
```

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

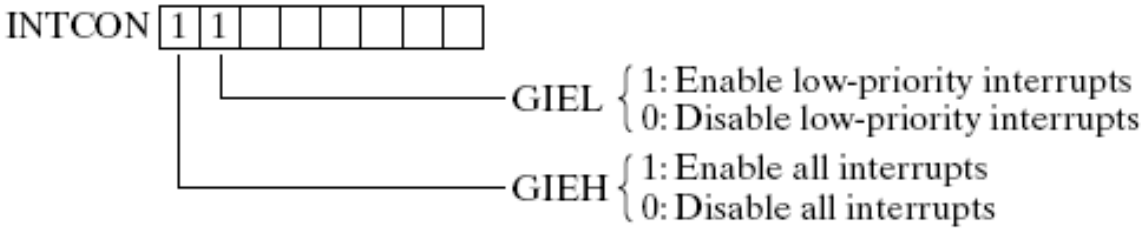
Step 3. Set the appropriate interrupt enable bits to enable the desired interrupt.

Low-priority interrupt structure

- 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

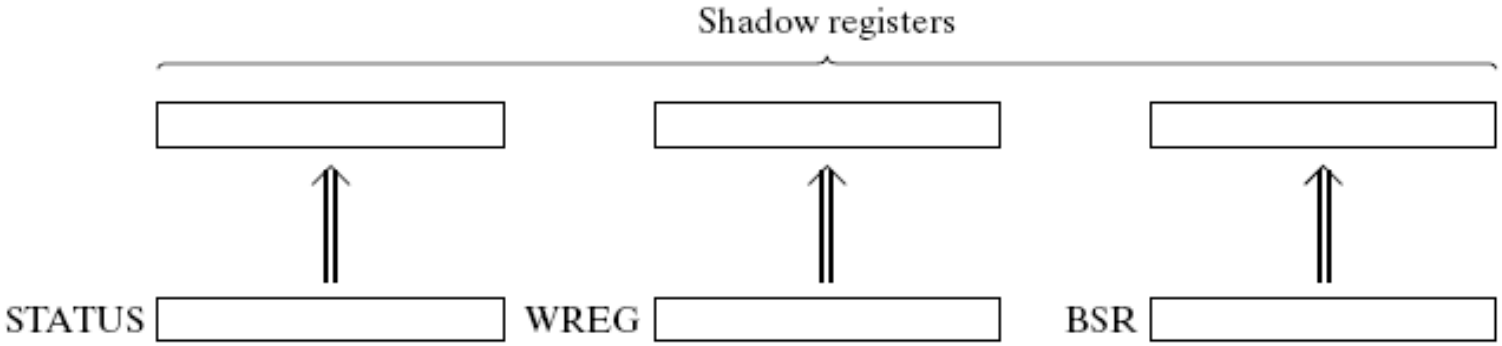


(b) Global interrupt enable bits

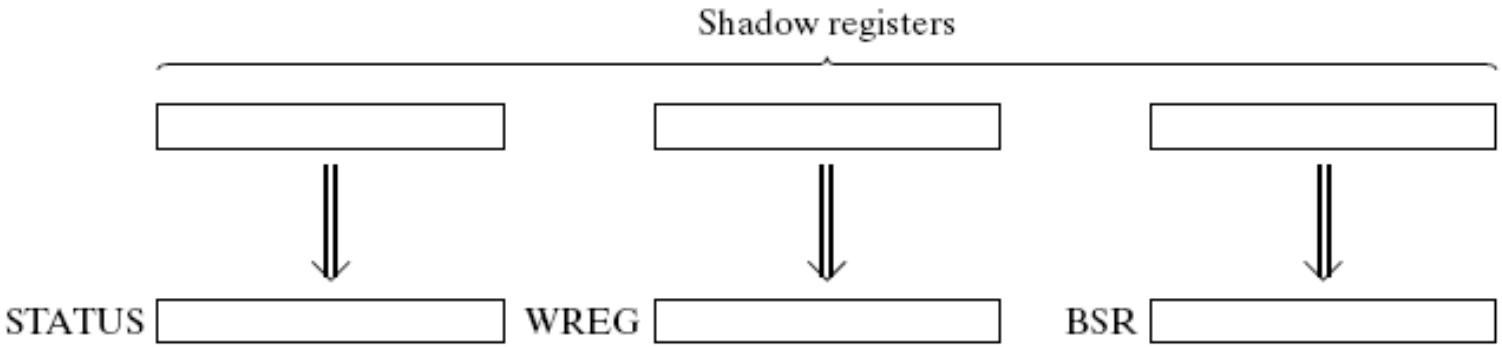
- GIEL is automatically cleared when a low-priority interrupt occurs
- *retfie* sets again the GIEL bit, and pops the PC from stack
- Each interrupt source has associated with it a priority bit
- The default state of all priority bits is 'high' at power-on reset

High-priority interrupt structure

- 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

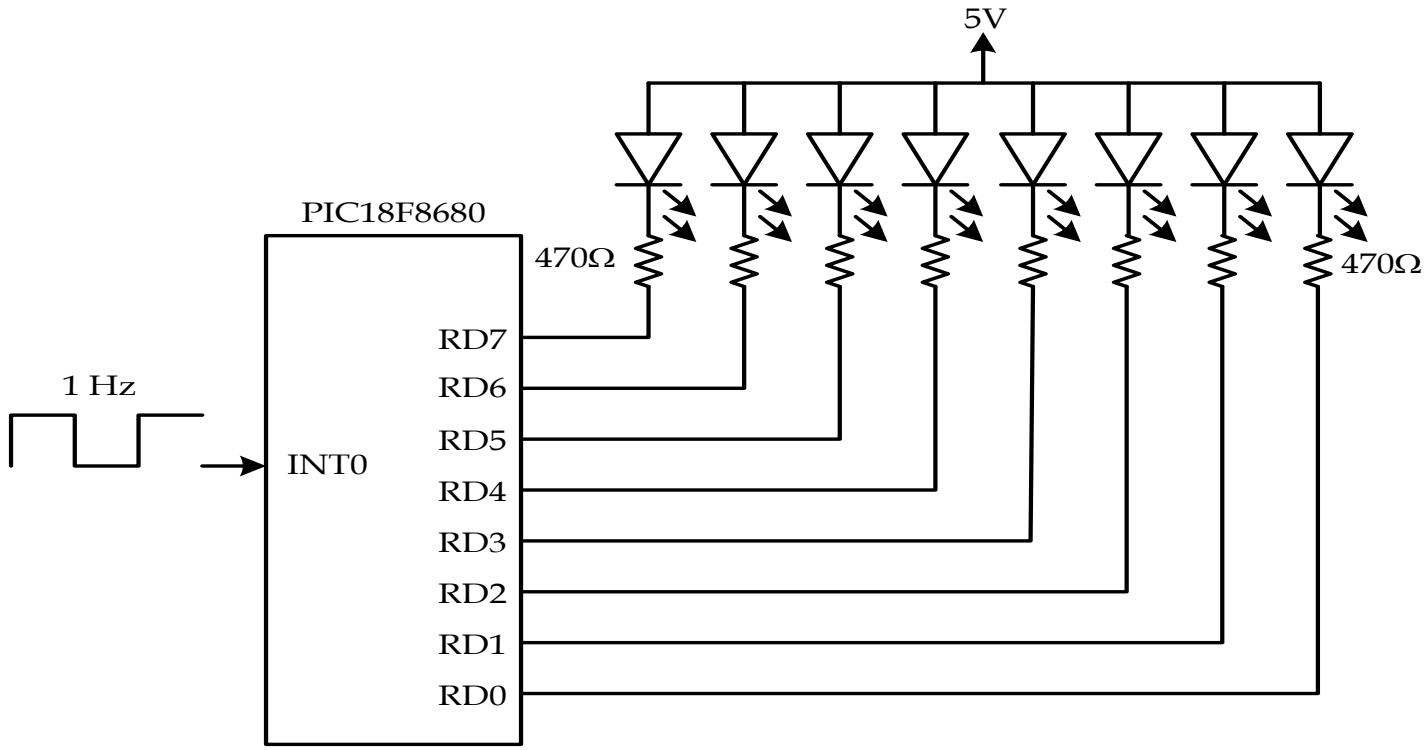


(a) Automatic setting aside of **STATUS**, **WREG**, and **BSR** when a high-priority interrupt occurs



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

Example



- 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.
- A LED is turned on when its driving Port D pin outputs a low.

example

```
        #include    <p18F45K22.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         forever

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

Interrupt Programming Template in C

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

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

}
```

Example -C language version-

```
#include <xc.h>

unsigned char count;

void interrupt service_routine_HP (void)
{
        if (INTCONbits.INT0IF && INTCONbits.TMR0IE)
        {
                INTCONbits.INT0IF=0;
                count--;
                PORTD = count;
        }
// process other interrupt sources here, if required
}

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

```
void main(void)
 {
  // Init PIC
  TRISD = 0x00;          // Configure PORTD for output
  count = 0xFF;          // turn off all LEDs initially
  PORTD = count;         //          "
  RCONbits.IPEN = 1;  // enable priority interrupt
  INTCON = 0x90;       // enable GIEH and INT0 interrupt

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

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.
  #pragma interrupt high_ISR save = reg_1,…, reg_n
  #pragma interrupt low_ISR save = reg1,…, reg_n

- 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")

Saving context in RAM

If retfie fast is not used

EXAMPLE 9-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF     W_TEMP                        ; W_TEMP is in virtual bank
MOVFF     STATUS, STATUS_TEMP           ; STATUS_TEMP located anywhere
MOVFF     BSR, BSR_TEMP                 ; BSR_TMEP located anywhere
;
; USER ISR CODE
;
MOVFF     BSR_TEMP, BSR                 ; Restore BSR
MOVF      W_TEMP, W                     ; Restore WREG
MOVFF     STATUS_TEMP, STATUS           ; Restore STATUS
```

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

Resets vs. Interrupts

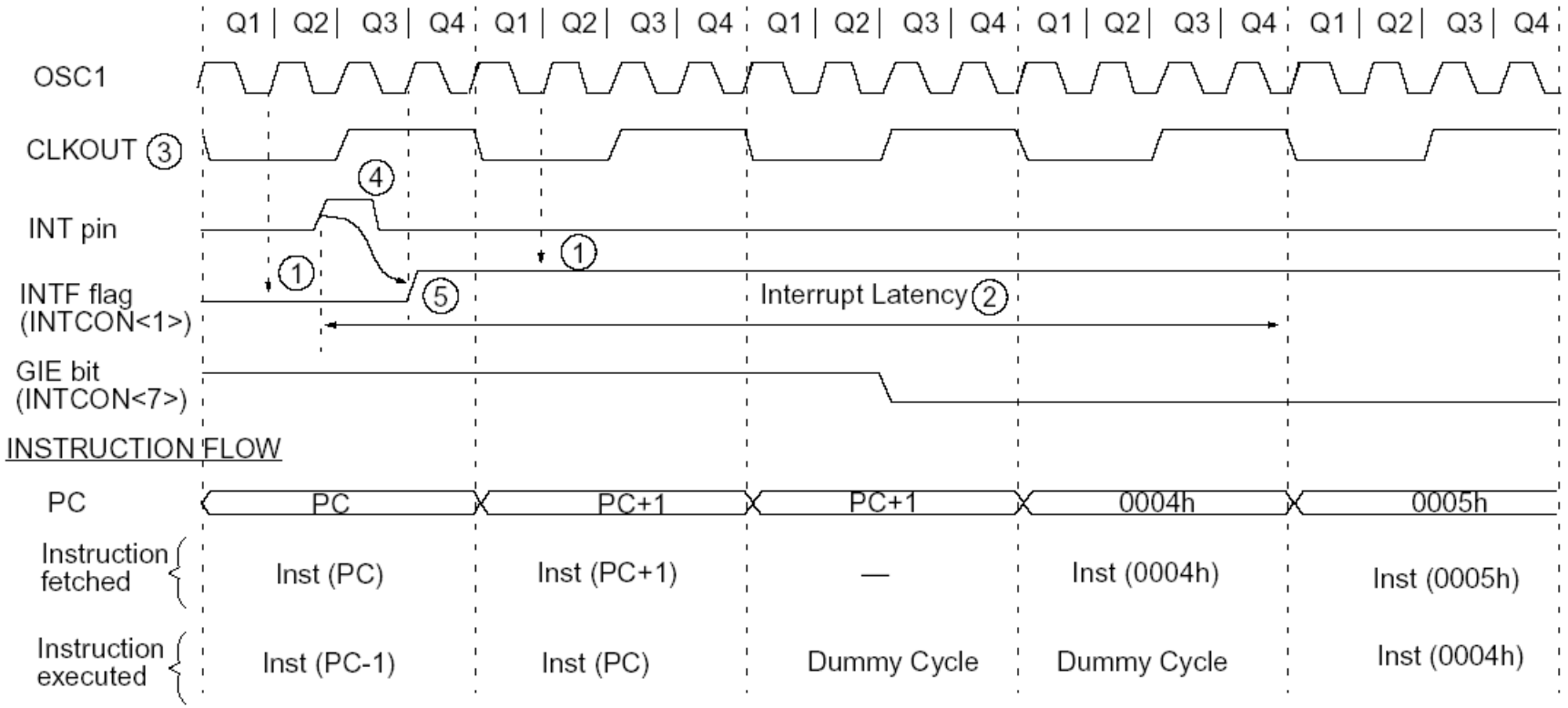
Reset/interrupts: What are the similarities and differences?

- Similarities:
  1. Both are extraordinary events that cause the MPU to deviate from fetch & decode (i.e., asynchronous)
  2. Both cause the MPU to copy a special address to its Program Counter and execute a different program (ISR).

- Differences:
  1. Resets cause MPU to abort its normal fetch and execute then prepares it to start from scratch
  2. Interrupts cause MPU to abort its normal fetch and execute but returns the MPU to the instruction following the one that was interrupted.

Interrupt latency

The 8-bit PIC-16/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.



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.