

PIC18F Timers & Interrupts

CTEC1904/2014W

Microprocessor Technology

Copyright Notice

Contains material from:

"CTEC1630 Embedded Systems Design", Fall 2010. © 2010 Mark S. Csele, P. Eng., Niagara College Canada

Embedded Design with the PIC18F452 Microcontroller by John B. Peatman, © 2003 by Pearson Education, Inc. ISBN 0-13-0462-6;

"PIC18FXX2" Data Sheet, © 2006 Microchip Technology Inc. Document # **DS39564C** downloaded from www.microchip.com

"PIC18F8722 Family Data Sheet", © 2008 Microchip Technology Inc. Document # **DS39646C** downloaded from www.microchip.com

"MPASM/MPLINK/MPLIB User's Guide", © 2009 Microchip Technology Inc. Document # **DS33014K** downloaded from www.microchip.com

"PICkit™ 3 Programmer/Debugger User's Guide", © 2009 Microchip Technology Inc. Document # **DS51795B** downloaded from www.microchip.com

"101 TLS" and "102 ASP" (Microchip Regional Training Centres training materials), © 2006 Microchip Technology Inc.

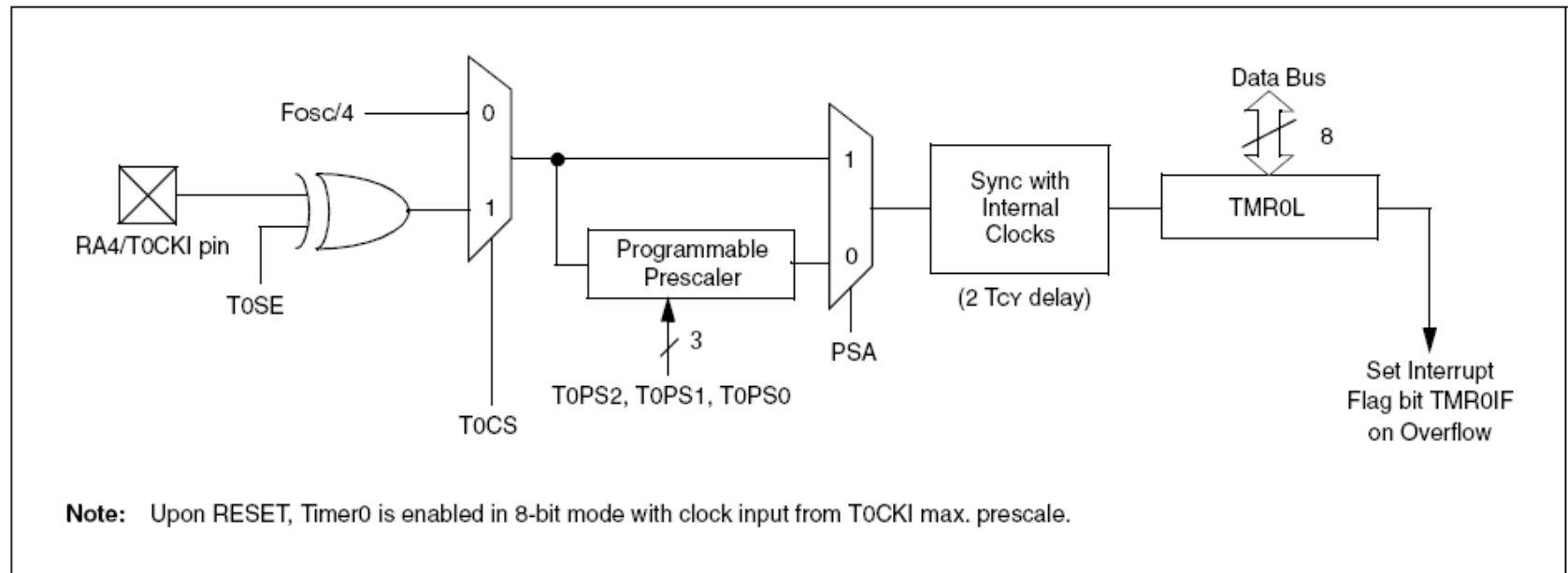
and other sources as acknowledged on certain slides.

On-Chip Peripherals: Timers

- The 18F452 and 18F4520 chips contain four timers; the 18F8722 contains five timers
- Timers can be 8- and/or 16-bit
- Timers can generate interrupts

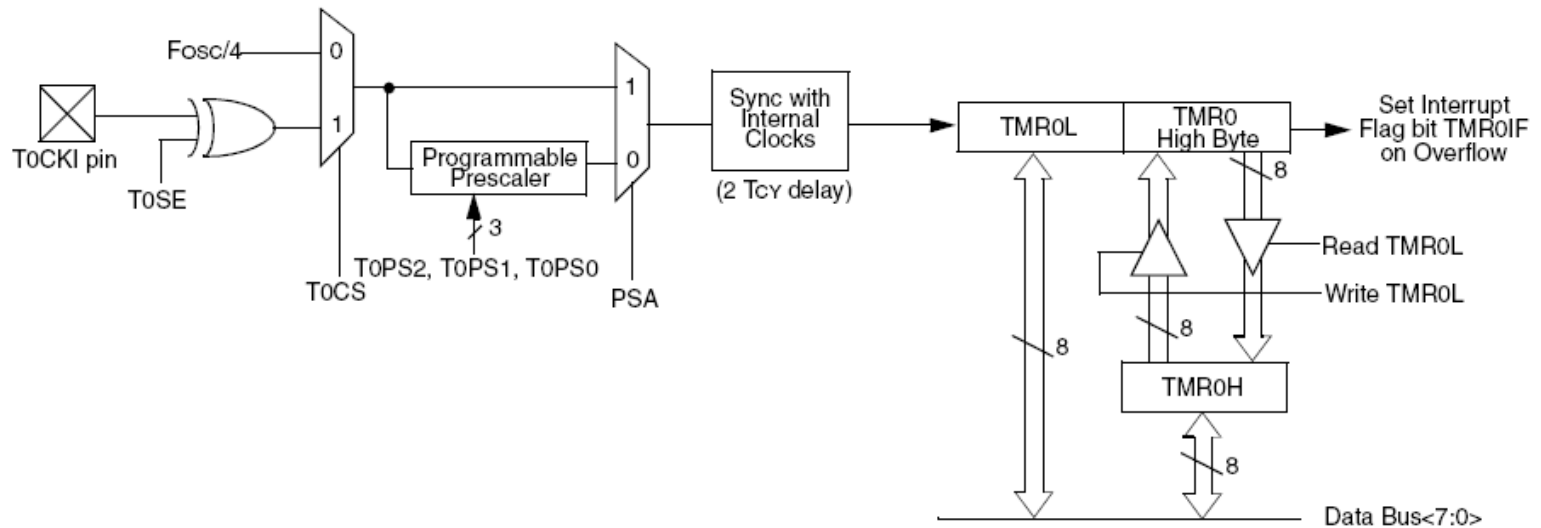
Timer0 Block Diagram (8-bit mode)

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



Timer0 Block Diagram (16-bit mode)

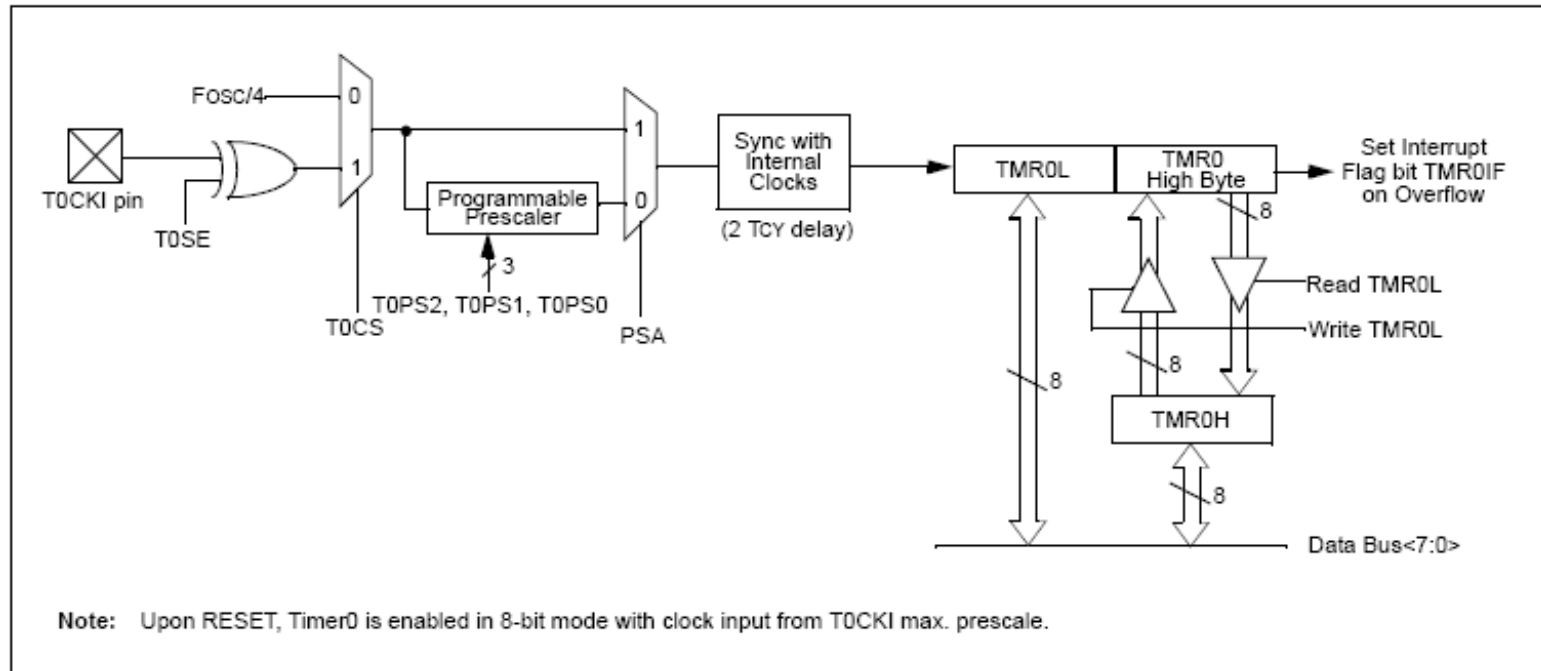
FIGURE 10-2: TIMER0 BLOCK DIAGRAM IN 16-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

Timer 0

- Consider TIMER0 in 16-bit mode

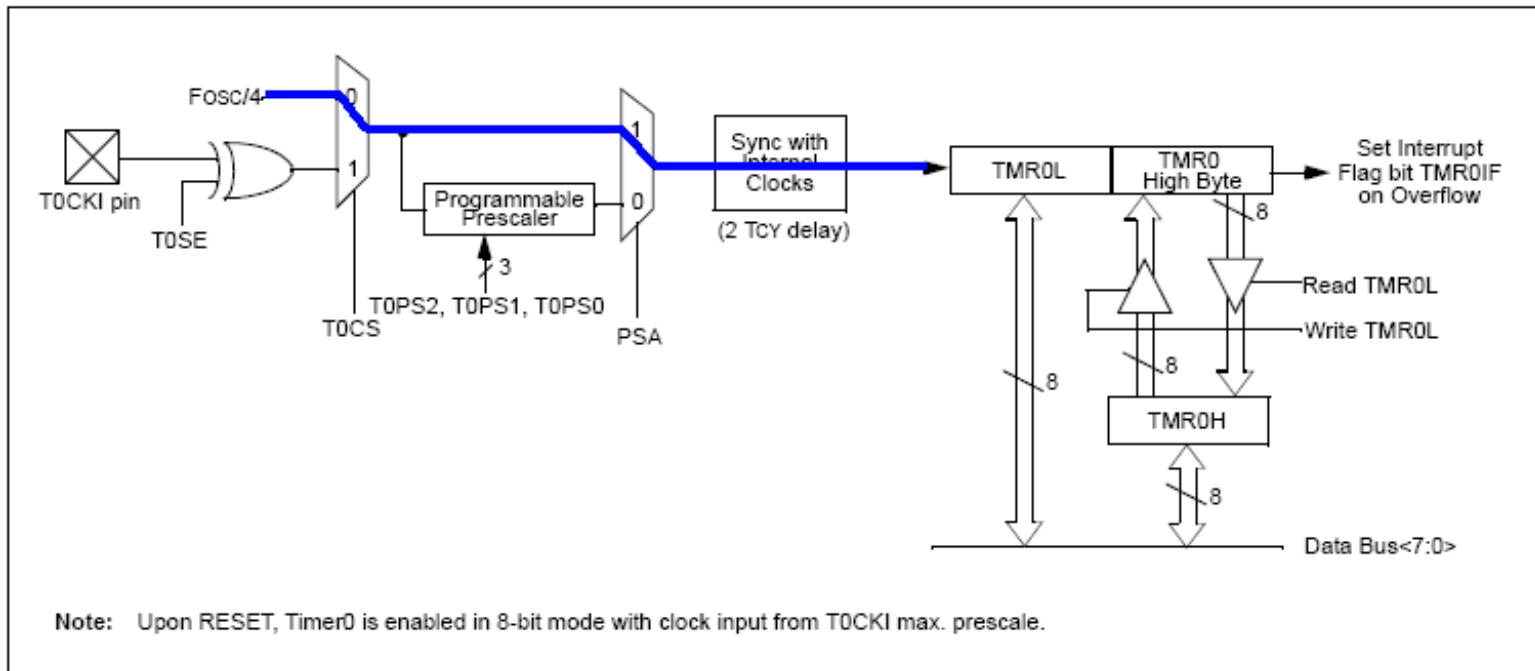


- Bits (e.g. T0CS, PSA, etc) program operation of the timer

Timer 0 (2)

Set the **T0CS** (Timer0 Clock Select) bit to 0 to select the internal clock and the **PSA** (PreScaler Assign) bit to 1 to bypass the prescaler. If a 4MHz clock were employed, the timer would count at the rate of 1MHz.

The bits are set in the **T0CON** (Timer0 Control) register



Timer 0

- To use Timer 0 as a hardware timer we can use it in the interrupt mode or simply test a bit.
- To setup the Timer 0 use the **T0CON** register.

TABLE 10-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
TMR0L	Timer0 Module Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISA	—	PORTA Data Direction Register							-111 1111	-111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

T0CON

- Bit 7: turn off/on
- Bit 6: 8- or 16- bit timer (use 16 bit)
- Bit 5: clock source, use internal clock source
- Bit 3: assign prescaler to Timer0 to slow down count
- Bit 2,1,0: set the prescaler value to maximum

REGISTER 10-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

T0CON Bits (1)

- bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0
- bit 6 **T08BIT:** Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter
- bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)
- bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin
- bit 3 **PSA:** Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

TOCON Bits (2)

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits

111 = 1:256 prescale value

110 = 1:128 prescale value

101 = 1:64 prescale value

100 = 1:32 prescale value

011 = 1:16 prescale value

010 = 1:8 prescale value

001 = 1:4 prescale value

000 = 1:2 prescale value

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

Remember to control bits one at a time rather than placing a 8 bit value into a control register. Use labels over numbers for easier troubleshooting

Timer 0 (3)

- Timer is typical on on-chip peripherals, i.e., configuration is similar
- use external pin (**T0CKI**) for counter
- $FOSC/4 = XTAL \div 4 = \text{machine cycle}$
- configured with **T0CS** bit
- prescaler = $\div n$ where n is binary 2 ... 256
- sync: line up with pipeline stages

Timer 0 – Interrupt Mode

- To setup the interrupt use the **INTCON** register.
- Bit 7: **GIE** bit set to 1 to enable interrupts
- Bit 5: **TMROIE** bit set to 1 to enable Timer 0 interrupt
- Bit 2: **TMROIF** clear this bit and remember to clear in software.

INTCON (1)

REGISTER 8-1: INTCON 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
bit 7							bit 0

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

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

INTCON (2)

- 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 = 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
- Note:** A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Timer 0 (4)

- **TMR0IF** (interrupt flag) on overflow -> generates interrupt
- **T0CS** = 0, **PSA** = 1, read **TMR0L** as a pseudo-random number generator (PRNG)

Timer 0 (5)

- $F_{XTAL} = 4 \text{ MHz}$

$\div 4$ \rightarrow $f_{cy} = 1 \text{ MHz}$

PSA=1

Counter

0- \rightarrow 65536

PSA=0

$\div n$

1 MHz

Interrupt

e.g.

$\div 2 = 500 \text{ kHz}$

$1 \text{ MHz} \div 65536 \approx 15.2 \text{ Hz}$

Timer Preload

- Pre-load counter to divide by 10000, for example.
- Pre-load value = $65536 - \text{Divisor}$
 $= 65536 - 10000 = 55536$

Timer Counts (1)

- In 16-bit mode, the timer can count from zero up to **65535** ($2^{16}-1$) and back to zero again. (The overflow causes the interrupt.)
- With the 3.6864 MHz oscillator selected, and since the internal processor clock divides the oscillator clock by 4, there are 921,600 clock cycles per second.

Timer Counts (2)

- With the timer running in 1:1 mode (i.e., *no* prescaler), 65536 timer ticks will take around 71 ms.
- If we use the maximum prescaler value, the timer runs in 1:256 mode: *for every 256 processor clock cycles, the timer will count down by 1.*

Timer Counts (3)

- The timer in 1:256 mode can count through its entire range in:

$$65536 * 256 / 921600 = 18.2 \text{ seconds}$$

- For a shorter delay, we can either adjust the prescaler (coarse grain control) or **preload** the timer registers, so it does not have to count down from the top each time (fine grain control).

Preloading the Timer (1)

- If we want a 0.5s delay, we can tell the timer to only count enough values to take up 0.5s.
- The timer clock frequency is simply the oscillator frequency divided by 4 divided by the prescaler value:

$$3686400 \text{ Hz} / (4 * 256) = 3600 \text{ Hz}$$

Preloading the Timer (2)

- Therefore, a 0.5s delay requires

$$0.5 \text{ s} * 3600 \text{ Hz} = 1800 \text{ clock cycles}$$

The preload value for the timer is then:

$$65536 - 1800 = 63736$$

- The easiest way to code this is with a constant:


```
TimerPreload    equ    .63736    ; 0.5s
```

Preloading the Timer (3)

- To preload the timer:

```
movlw  high      TimerPreload
movwf  TMR0H
movlw  low       TimerPreload
movwf  TMR0L
      ; restarts the timer
```


"Weird" Crystal Frequencies

- $7.3728 \text{ MHz} \div 4 = 1.8432 \text{ MHz} \div 256 = 7200 \text{ Hz}$
 $65536 - 7200 = 1 \text{ Hz interrupt!}$
- e.g., "Colorburst" TV crystal 3.579545 MHz
See <http://en.wikipedia.org/wiki/Colorburst>
- on PICPROTO2 board: 10 MHz & 3.6864 MHz
- on PICTRAINER board: 32768 kHz & 4.9152 MHz

"Weird" Crystal Frequencies (2)

- Scenario #1: PICPROTO2 board with 3.6864MHz crystal selected ...
- $3.6864 \text{ MHz} \div 4 =$
 $921.6 \text{ kHz} \div 256 =$
 $3600 \text{ Hz} \div 240 = 15 \text{ Hz}$
 $256 - 240 = \text{16} \text{ pre-load value (8-bit mode)}$

Timer 0 Setup (8-bit)

- Set prescaler & bits
T0CON

Setup {

```
movlw    .16           ; 15Hz using 3.6864MHz XTAL
movwf    TMR0L
bcf       INTCON, T0IF  ; Clear TMR0 INT flag
bsf       INTCON, T0IE  ; Set TMR0 INT ON
bsf       INTCON, GIE   ; Global INT enable
```

"Weird" Crystal Frequencies (3)

Scenario #2: To achieve the same 15 Hz timer interrupt on the PICTRAINER board using the 4.9152MHz crystal...

- $4.9512 \text{ MHz} \div 4 =$
 $1.2288 \text{ MHz} \div 256 =$
 $4800 \text{ Hz} \div \mathbf{320} = 15 \text{ Hz}$

**But $320 > 255$, so
you need to switch
to 16-bit mode!**

"Weird" Crystal Frequencies (3A)

Scenario #2: To achieve the same 15 Hz timer interrupt on the PICTRAINER board using the 4.9152MHz crystal...

- $4.9152 \text{ MHz} \div 4 =$
 $1.2288 \text{ MHz} \div 256 =$
 $4800 \text{ Hz} \div 320 = 15 \text{ Hz}$
 $65536 - 320 = \text{65216}$ pre-load value
(16-bit mode)

Timer 0 Setup (16-bit)

- Set prescaler & bits
T0CON

; 15Hz using 4.9152MHz XTAL

TimerPreload **equ** .65216


Setup {
 movlw high Preload ; load high 8 bits first
 movwf TMR0H
 movlw low Preload ; then load low 8 bits
 movwf TMR0L
 ...

Timer 0 Interrupt Service Routine (ISR)

movwf	ISR_WREG_Temp	}	Context Save: High-priority interrupt auto-saves
swapf	STATUS, w		
movwf	ISR_STATUS_Temp		
bcf	INTCON, T0IE	; Disable interrupt	
bcf	INTCON, T0IF	; Clear INT flag	
bsf	INTCON, T0IE	; Re-enable interrupt	
.			
.	processing		
.			

Timer 0 Interrupt Service Routine (ISR) [2]

```
swapf STATUS_SAVE, w  
movwf STATUS ; restore W and STATUS  
swapf W_SAVE, f  
swapf W_SAVE, w
```



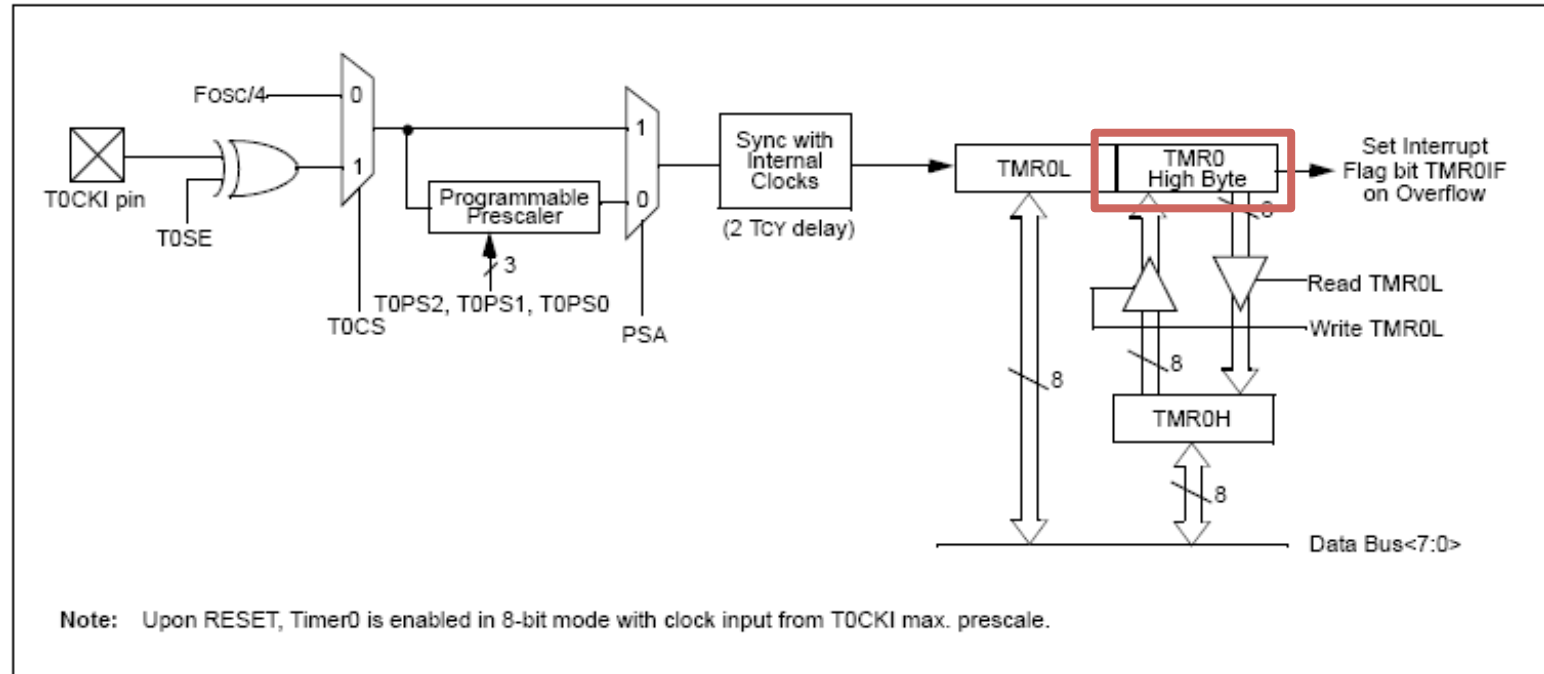
Context
Restore

```
retfie      ; return from interrupt and  
            ; re-enable interrupts
```


Polling vs. Interrupts

- P1.asm polls the Timer 0 interrupt flag to check for roll over, but Interrupt Controller is not actually enabled (the flag gets set anyway)
- Useful technique for RS-232
- P1.asm pads out "12 + 2" to compensate instruction count

Reading and Writing Timer 0



- Read **TMR0L** and write **TMR0H** (buffer, not real register) = snapshot of "***TMR0 high byte***" (real reg.) at the time you read **TMR0L**

Reading and Writing Timer 0 (2)

- To **read** TMR0: read TMR0L **first**
 - this latches value of TMR0H
 - read TMR0H second
- To **write** to TMR0: write TMR0H **first**
 - data goes into buffer
 - write TMR0L **second**
 - latches TMR0H into "TMR0 high byte"

16-bit Operations on an 8-bit MCU

- **16-bit add on 8-bit:**

addwf
addwfc



includes carry from previous add

16-bit Operations on an 8-bit MCU

(2)

- **low, high** are *assembler directives*:
 - processed at compile time
 - use Disassembly window to view assembler output
- you must clear interrupt flag manually (**bcf**)
- problem with polling: *may miss more than one event!*

Interrupts

- P1Int.asm
- **Interrupt** = *asynchronous subroutine*
 - hardware-triggered, e.g., RTC
 - guaranteed, real-time performance
 - externally triggered, e.g., comms

Interrupts (2)

1. When Interrupt occurs, PC+2 (i.e., the return address) is pushed onto stack (after finished executing current instruction).
2. Load PC with address of ISR.
3. Pop PC (via retfie).

Interrupts (3)

- Use GPRs as global variables to communicate between ISR and main code.
- Main program has no knowledge or control that interrupt has been triggered.

Interrupts (4)



Don't waste time in ISR,
esp. waiting for an event to
occur, e.g., a key press.

- Service I/O or hardware and leave!

Interrupt Vectors

- User must set up reserved locations in Program Memory.

```
    org 0x000000
goto    Main
    org 0x000008
goto    ISRHi           ; High-priority interrupt
    org 0x000018
goto    ISRLow          ; Low-priority interrupt

Main                    ; Main program
. . .
```

Low-Priority Interrupt

```
org          0x000000  
nop  
nop  
goto        Main
```

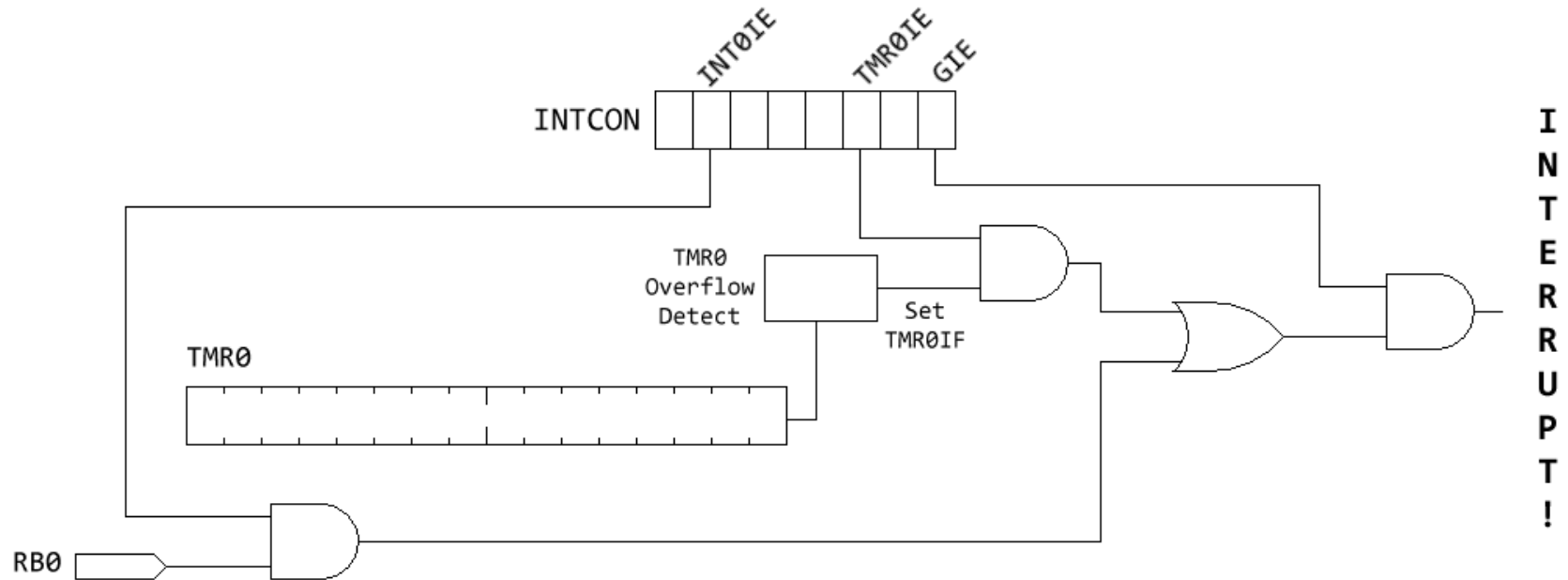
```
ISR  
org          0x000018  
.  
.  
.  
retfie
```

```
Main  
.  
.  
.
```

Interrupt Initialization

bsf
bsf

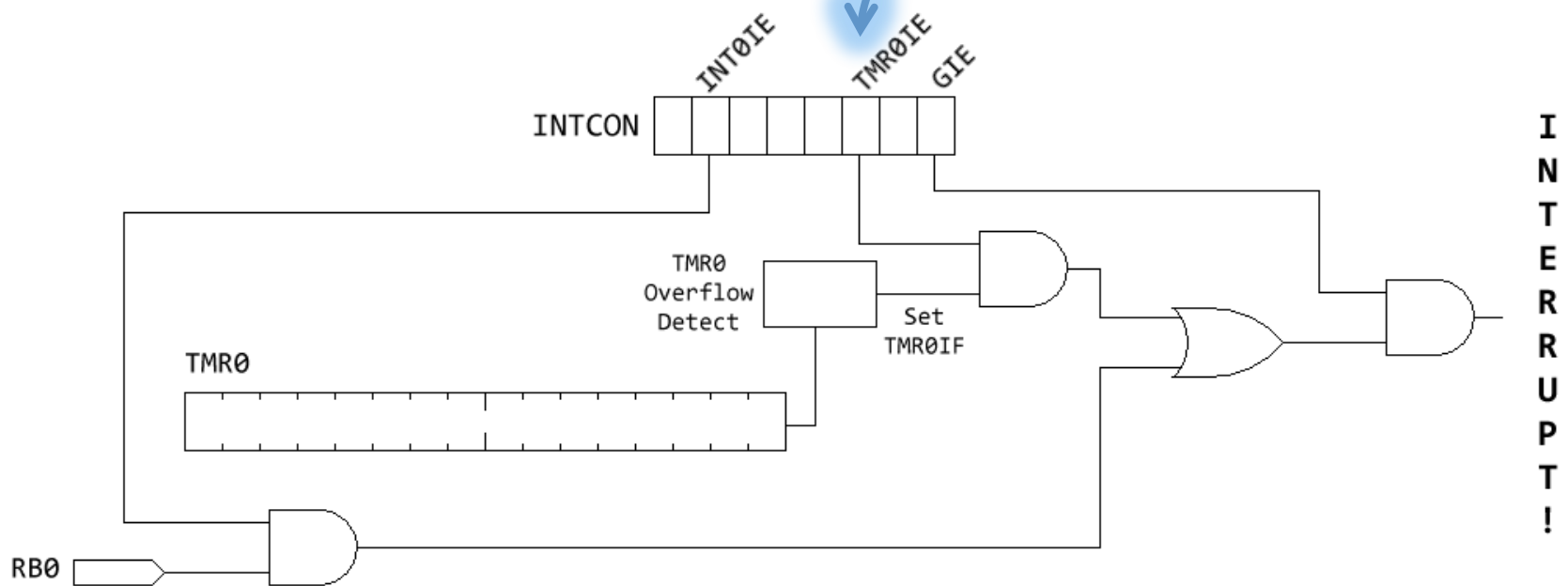
INTCON, TMR0IE
INTCON, GIE



Interrupt Initialization (2)

bsf
bsf

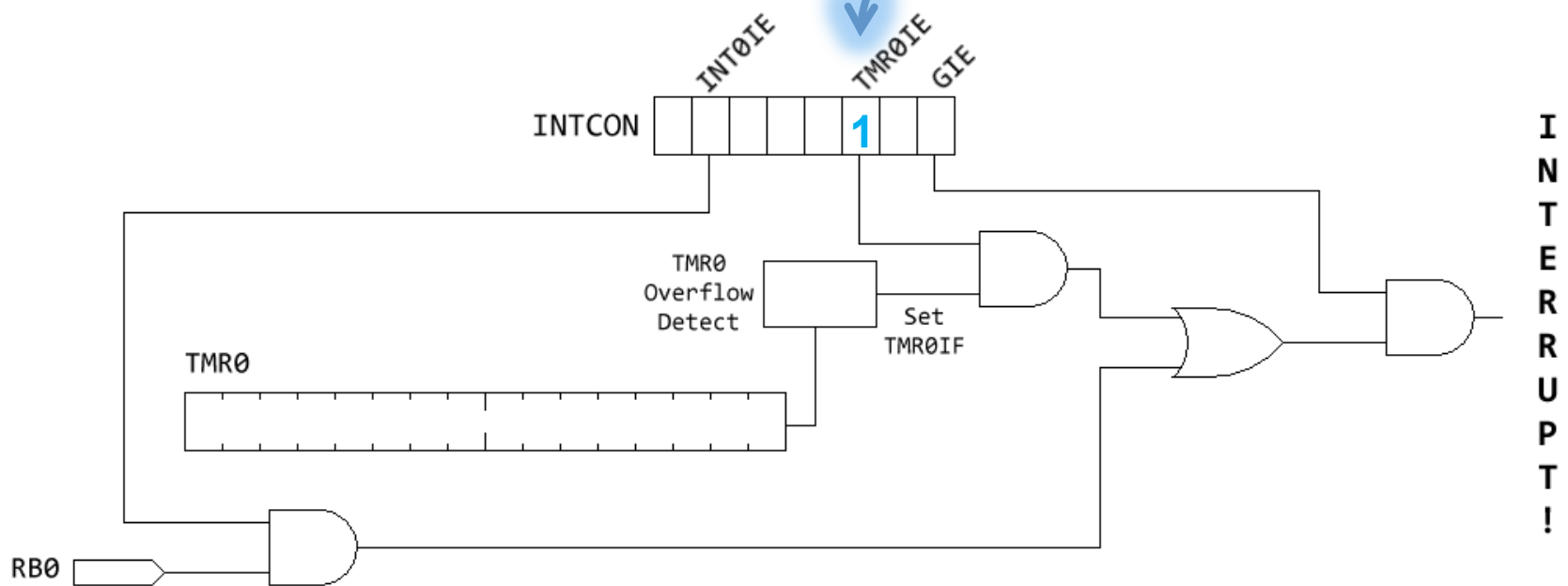
INTCON, TMR0IE
INTCON, GIE



Interrupt Initialization (3)

bsf
bsf

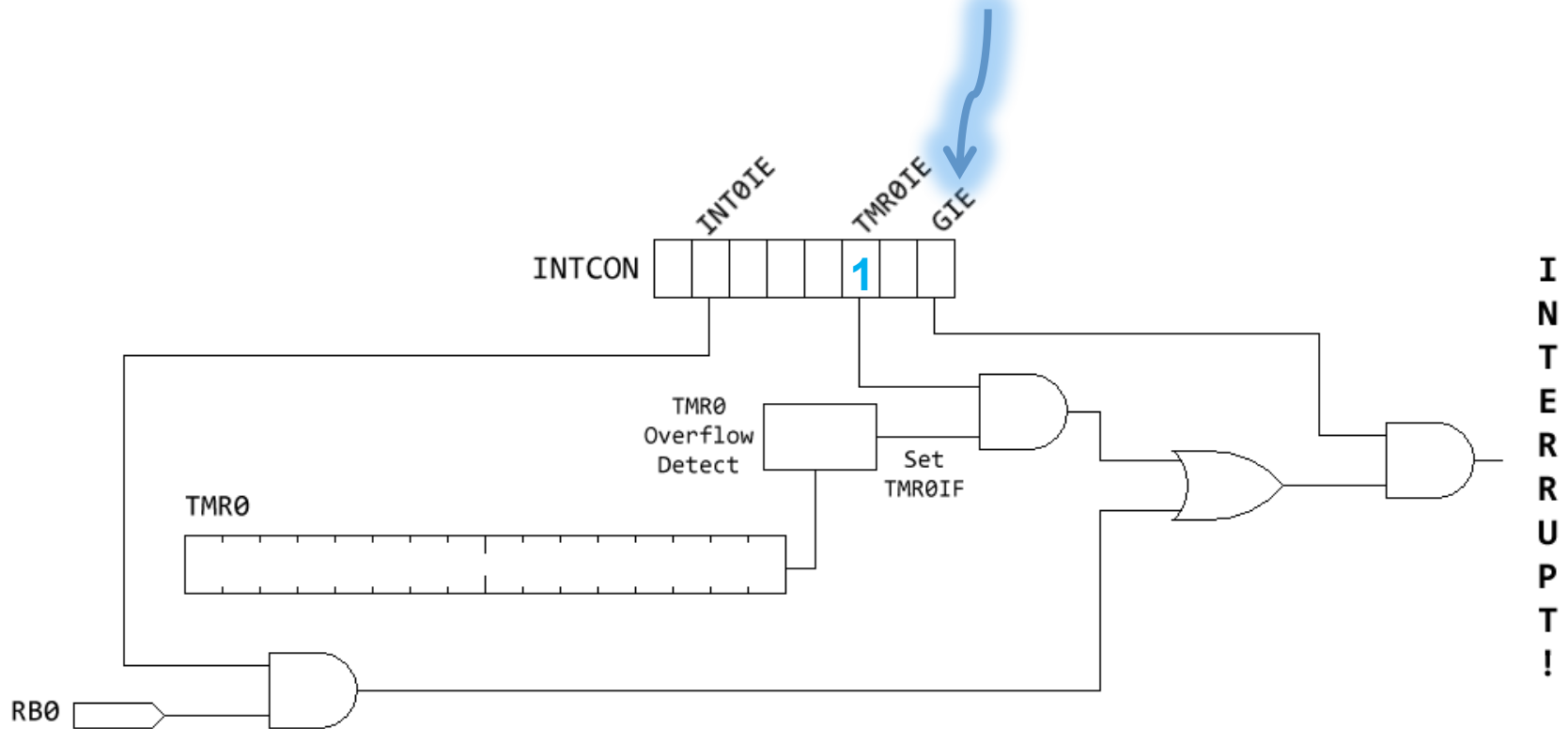
INTCON, TMR0IE
INTCON, GIE



Interrupt Initialization (4)

bsf
bsf

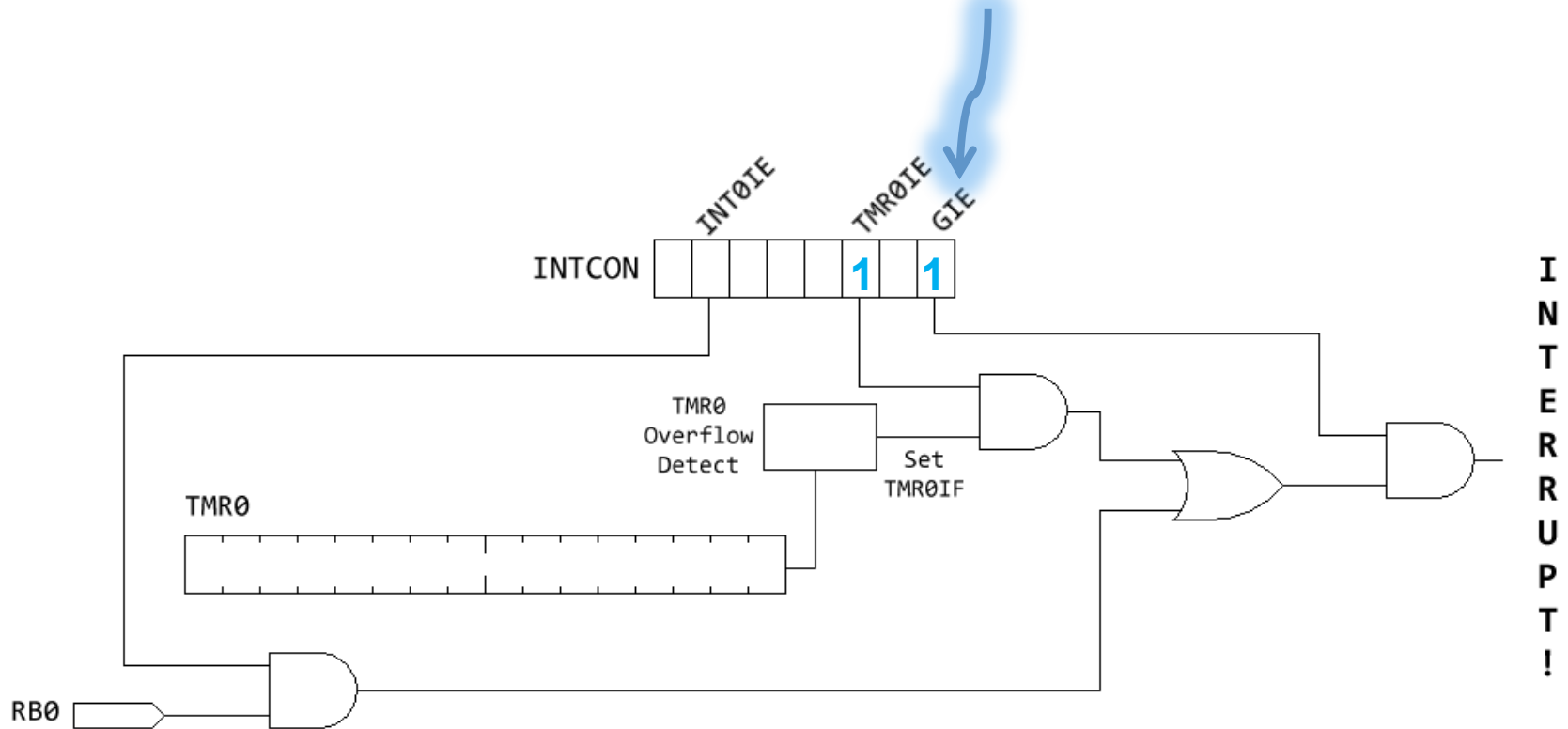
INTCON, TMR0IE
INTCON, GIE



Interrupt Initialization (5)

```
bsf  
bsf
```

```
INTCON, TMR0IE  
INTCON, GIE
```

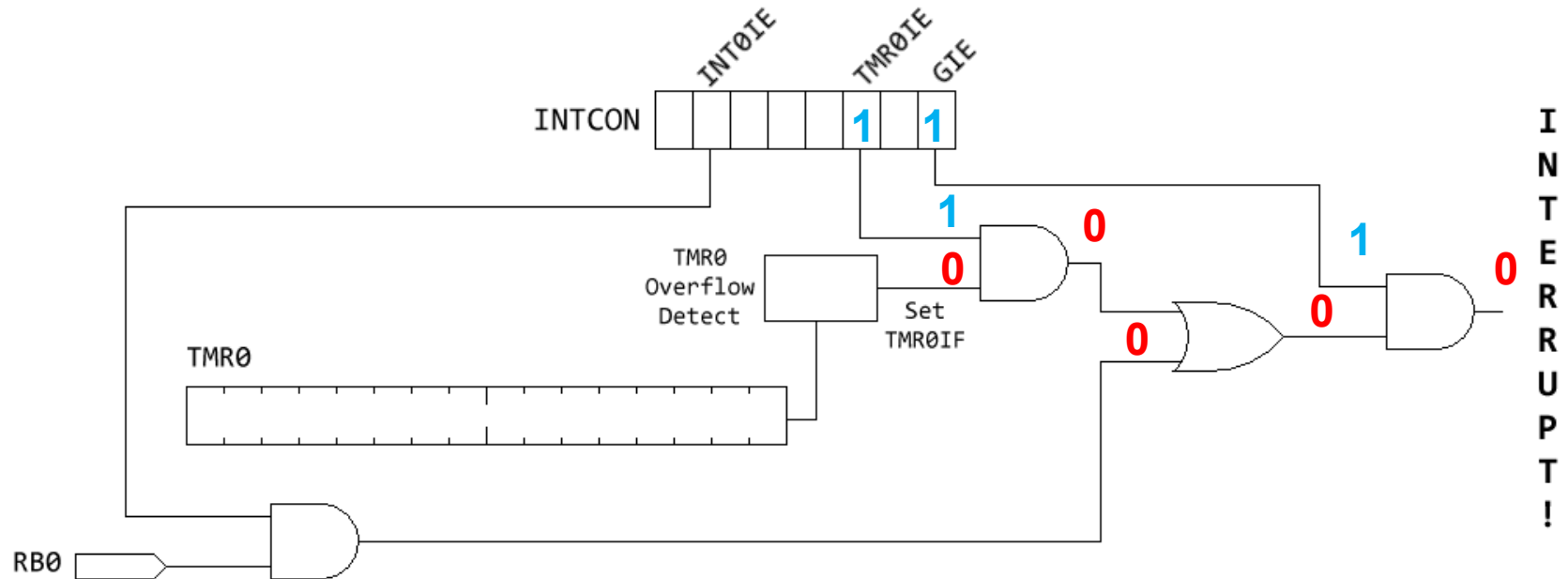


Interrupt Triggering

bsf
bsf

INTCON, TMR0IE
INTCON, GIE

Initial state

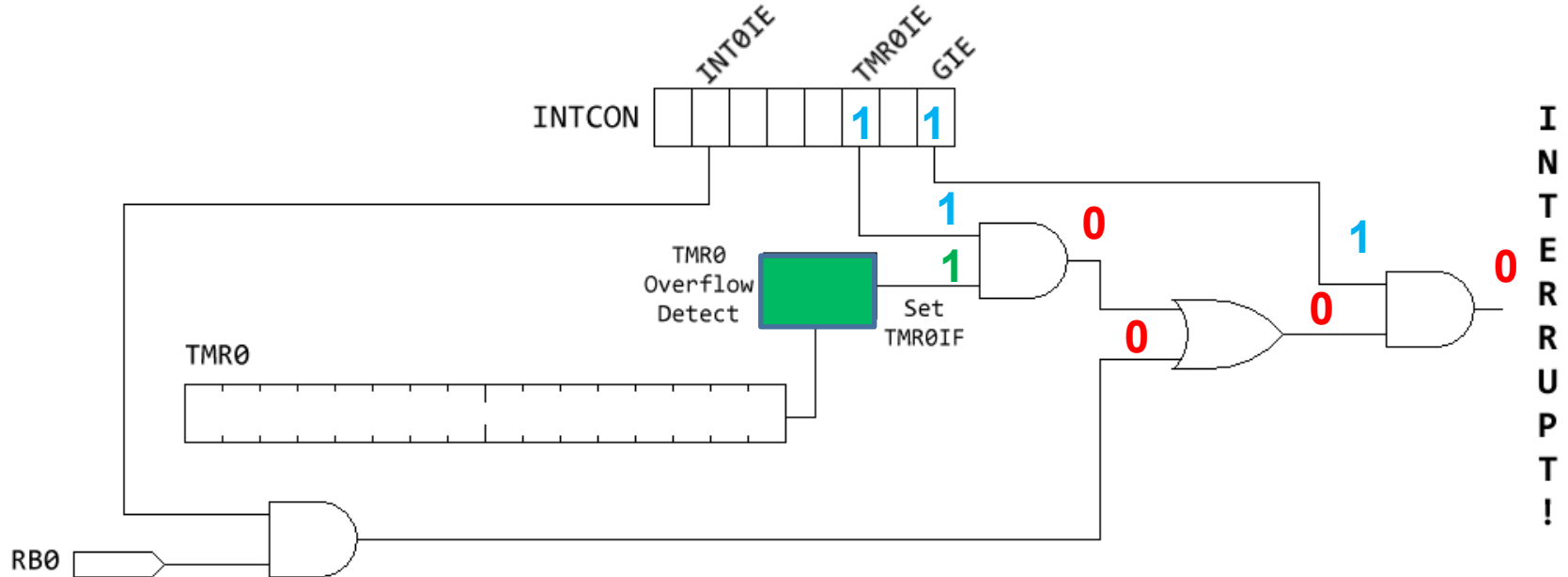


Interrupt Triggers (1)

```
bsf  
bsf
```

```
INTCON, TMR0IE  
INTCON, GIE
```

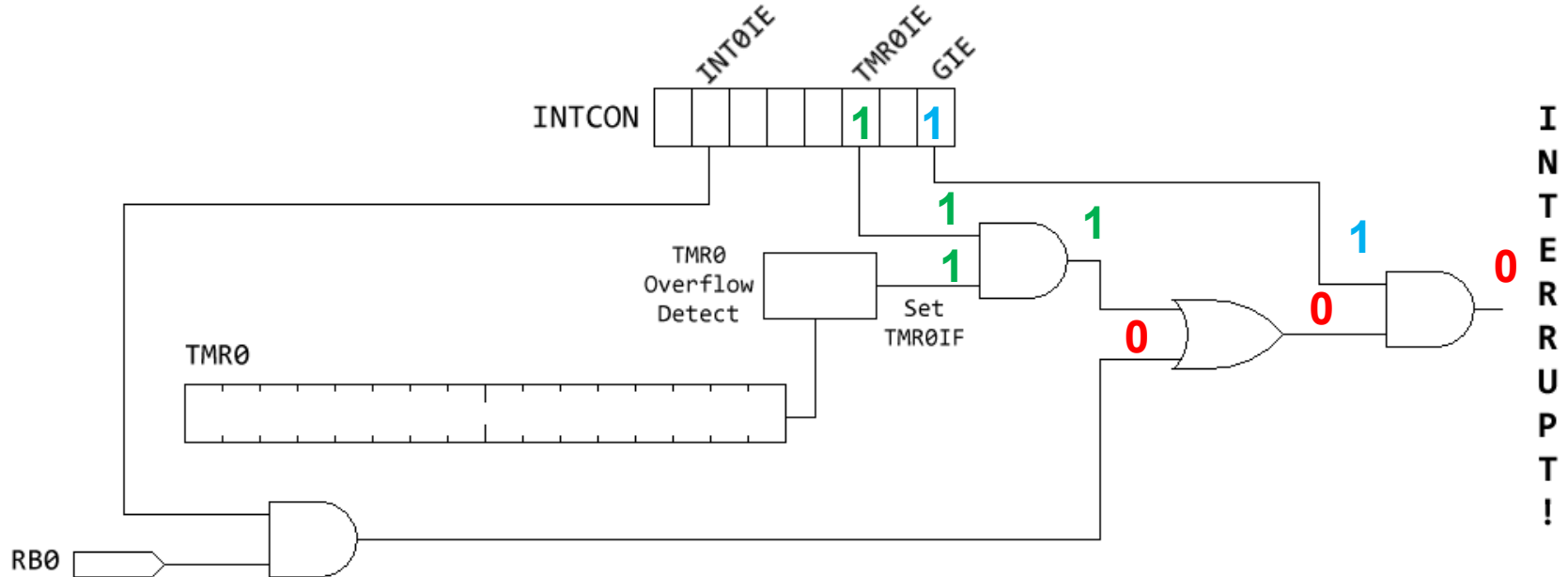
Timer 0 Overflow detected



Interrupt Triggering (2)

bsf INTCON, TMR0IE
bsf INTCON, GIE

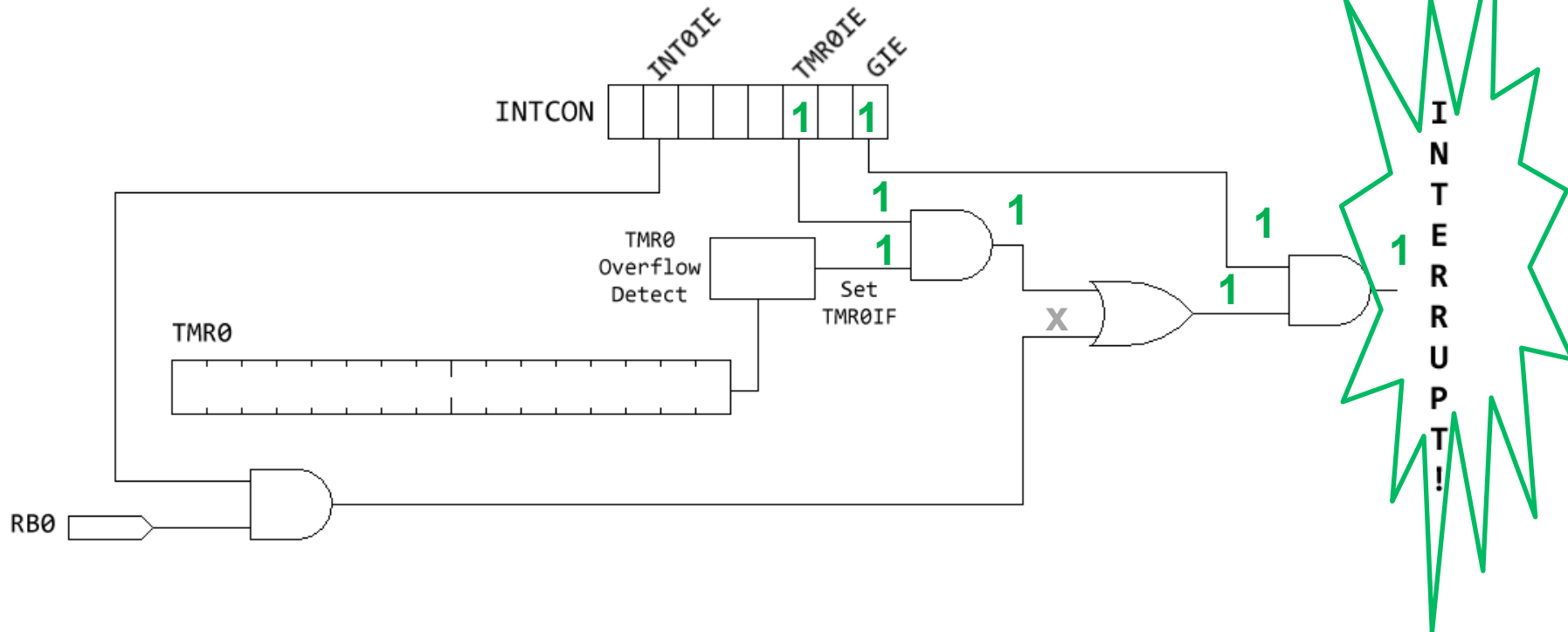
Overflow propagates due to Interrupt Enable bit



Interrupt Triggering (3)

`bsf` `INTCON, TMR0IE`
`bsf` `INTCON, GIE`

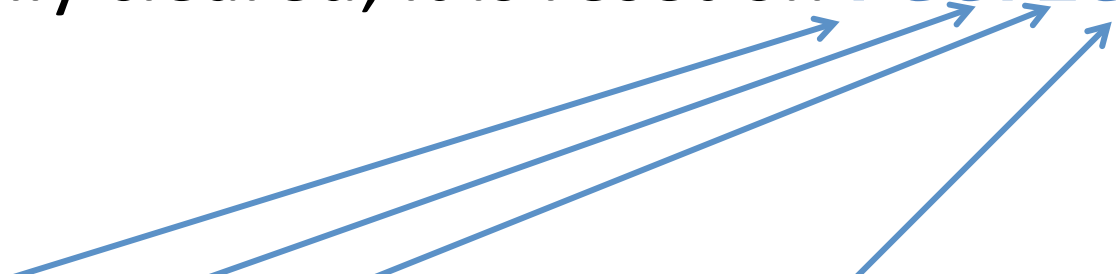
Interrupt triggers due to Global Interrupt Enable bit



Interrupt Enable

- When the interrupt occurs, **GIE** gets automatically cleared; it is reset on **retfie**

return from interrupt and enable



- This makes the ISR an **atomic operation**
- Equivalent code:

```
pop    PC
bsf    INTCON, GIE
```

Inside The ISR

- e.g., P1Int.asm has 1 Hz interrupts -> fast enough
- save W, STATUS at minimum (also BSR if you are using bank switching)
- also save GPRs that you modify that are not being used for return values (e.g., on 16-bit PICs there are 16 W registers)

Inside The ISR (2)

ISR

movwf

WRegTemp

movff

STATUS, StatusTemp

movwf doesn't affect
STATUS bits

-
-
-

older 16F PIC workaround
(no movff instruction):
swapf, then swapf again

On 16-bit PICs (24F, dsPIC30F, dsPIC33F)
there is a proper stack with push and
pop instructions

Inside The ISR (3)

-
-
-

You have to clear the Interrupt Flag manually because the ISR doesn't know which flag caused the interrupt.

```
bcf      INTCON, TMR0IF
movf     WRegTemp, w
movff    StatusTemp, STATUS
retfie
```


Interrupt Controller

- See DS, Ch. 10
- More **INTCON** registers, e.g., for USART
- Low-priority interrupt scheme is used by most processors.

10.1 INTCON Registers

The INTCON Registers are readable and writable registers, which contain various enable, priority and flag bits.

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

REGISTER 10-1: 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
bit 7							bit 0

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
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 = 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

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

REGISTER 10-2: INTCON2: INTERRUPT CONTROL REGISTER 2

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBP}}\text{U}$	INTEDG0	INTEDG1	INTEDG2	INTEDG3	TMR0IP	INT3IP	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

$\overline{\text{RBP}}\text{U}$: 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

INTEDG3: External Interrupt 3 Edge Select bit

1 = Interrupt on rising edge

0 = Interrupt on falling edge

bit 2

TMR0IP: TMR0 Overflow Interrupt Priority bit

1 = High priority

0 = Low priority

bit 1

INT3IP: INT3 External Interrupt Priority bit

1 = High priority

0 = Low priority

bit 0

RBIP: RB Port Change Interrupt Priority bit

1 = High priority

0 = Low priority

REGISTER 10-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INT2IP	INT1IP	INT3IE	INT2IE	INT1IE	INT3IF	INT2IF	INT1IF
bit 7							bit 0

- 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 **INT3IE:** INT3 External Interrupt Enable bit

1 = Enables the INT3 external interrupt

0 = Disables the INT3 external interrupt
- 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 **INT3IF:** INT3 External Interrupt Flag bit

1 = The INT3 external interrupt occurred (must be cleared in software)

0 = The INT3 external interrupt did not occur
- 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 enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

Interrupt Priorities

- See DS for priority order (Section 10.4).
- Second interrupt controller for nested "fast" interrupts.
- e.g., high-priority fast communications like Ethernet and USB 2.0