LECTURE 26, 27

Time Delay

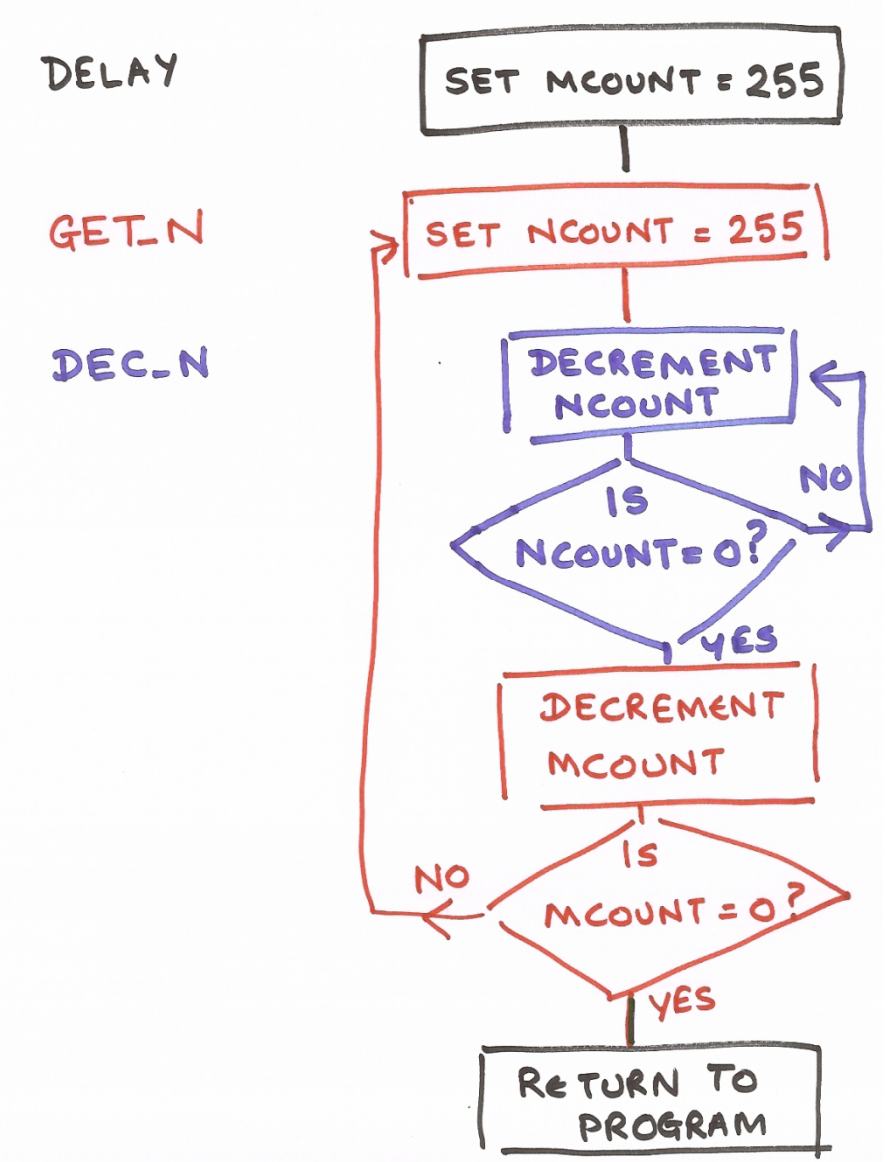
Topics to be covered

- Time Delay
- Flowchart
- Configuring inputs and outputs

Time Delay with Flowchart

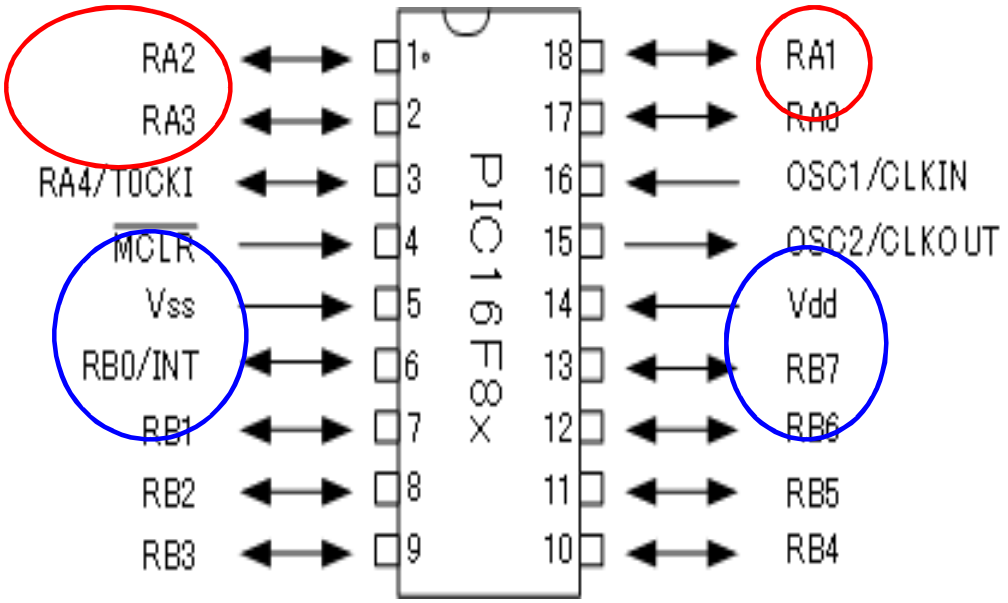
```
MCOUNT      EQU       0x0C
NCOUNT      EQU       0x0D

DELAY       MOVLW     0XFF
            MOVWF     MCOUNT
GET_N       MOVLW     0XFF
            MOVWF     NCOUNT
DEC_N       DECFSZ    NCOUNT,F
            GOTO      DEC_N
            DECFSZ    MCOUNT,F
            GOTO      GET_N
            RETURN
```



Configuring I/O : TRIS

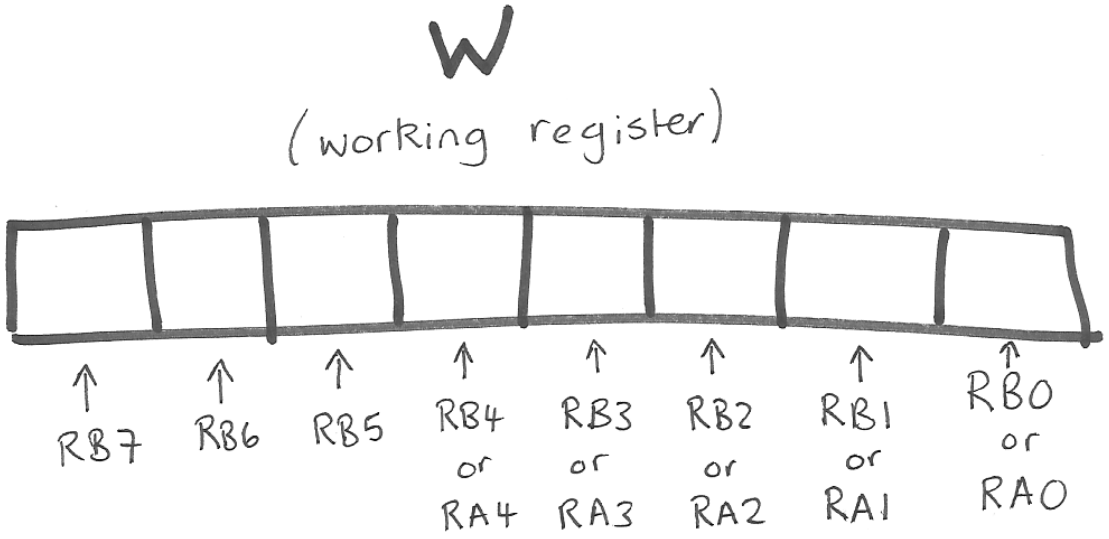
- TRIS is the (obsolete*) instruction for configuring inputs and outputs.
- The PIC16F84 has 13 lines that can be used as inputs or outputs.
  - The lines are grouped into 8 PORTB lines and 5 PORTA lines.
  - On the chip they are labelled RB7..RB0 and RA4..RA0.
- The programmer needs to tell the processor what inputs or outputs are needed.
- The simple way to do this uses the TRIS instruction



Using TRIS

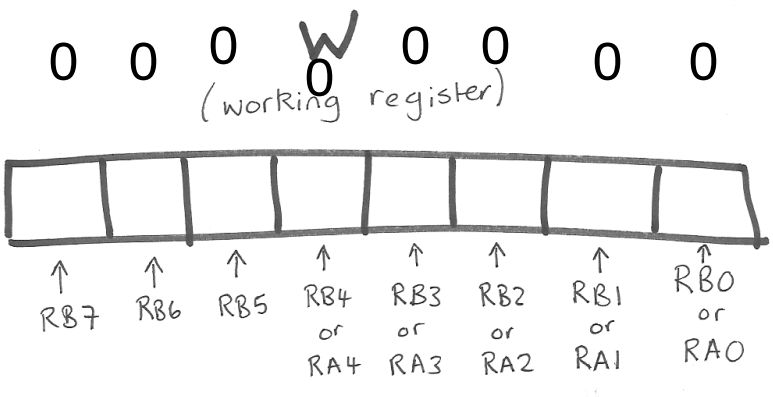
METHOD:

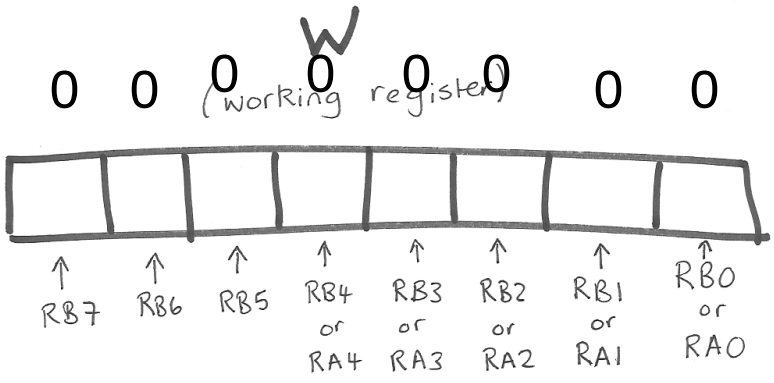
Put a pattern of bits in the working register. O's for outputs and 1's for inputs. Now use the TRIS instruction with PORTA or PORTB (whichever you want to configure)



TRIS uses the current bit pattern in W to define if lines are inputs or outputs

Using TRIS - examples





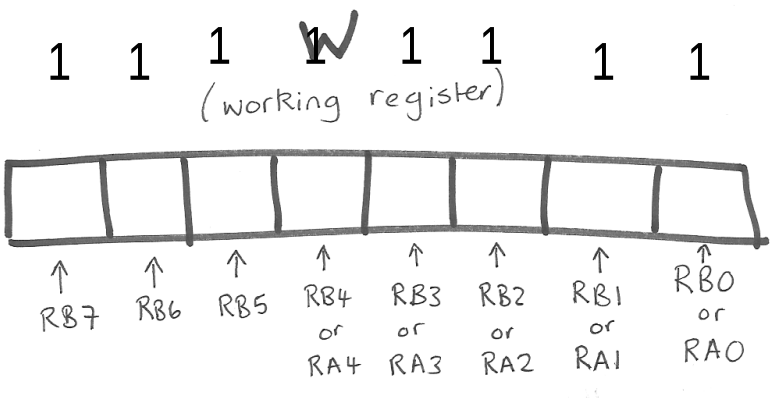
- Setting PORTB lines as all outputs

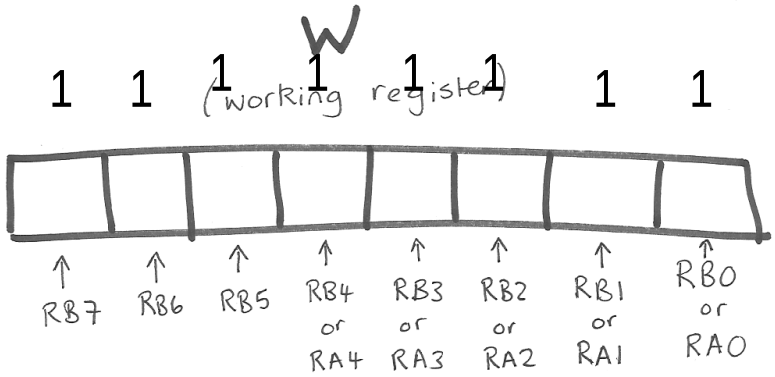
```
MOVLW   0x00
TRIS    PORTB
```

- Setting PORTA lines as all outputs

```
MOVLW   0x00
TRIS    PORTA
```

Using TRIS - examples





- Setting PORTB lines as all inputs

```
MOVLW   0xFF
TRIS    PORTB
```

- Setting PORTA lines as all inputs

```
MOVLW   0xFF
TRIS    PORTA
```

- Note - it doesn't matter that we have configured 3 lines that don't exist. But if you wanted you could use the value 1F (ie. 00011111)

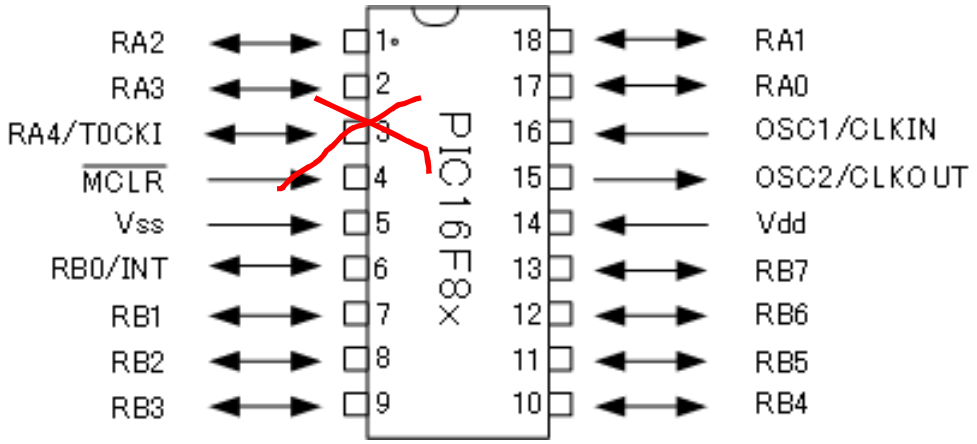
TRIS - what else?

- In our examples here we haven't cleared the values on the ports.

  (e.g., CLRF PORTB)

  - It's usually good practice to clear them, i.e., to make sure you know they start at zero or some other specified value.

- Advise against using RA4.
  - This is a special line. For example, it can be used for an external timer.



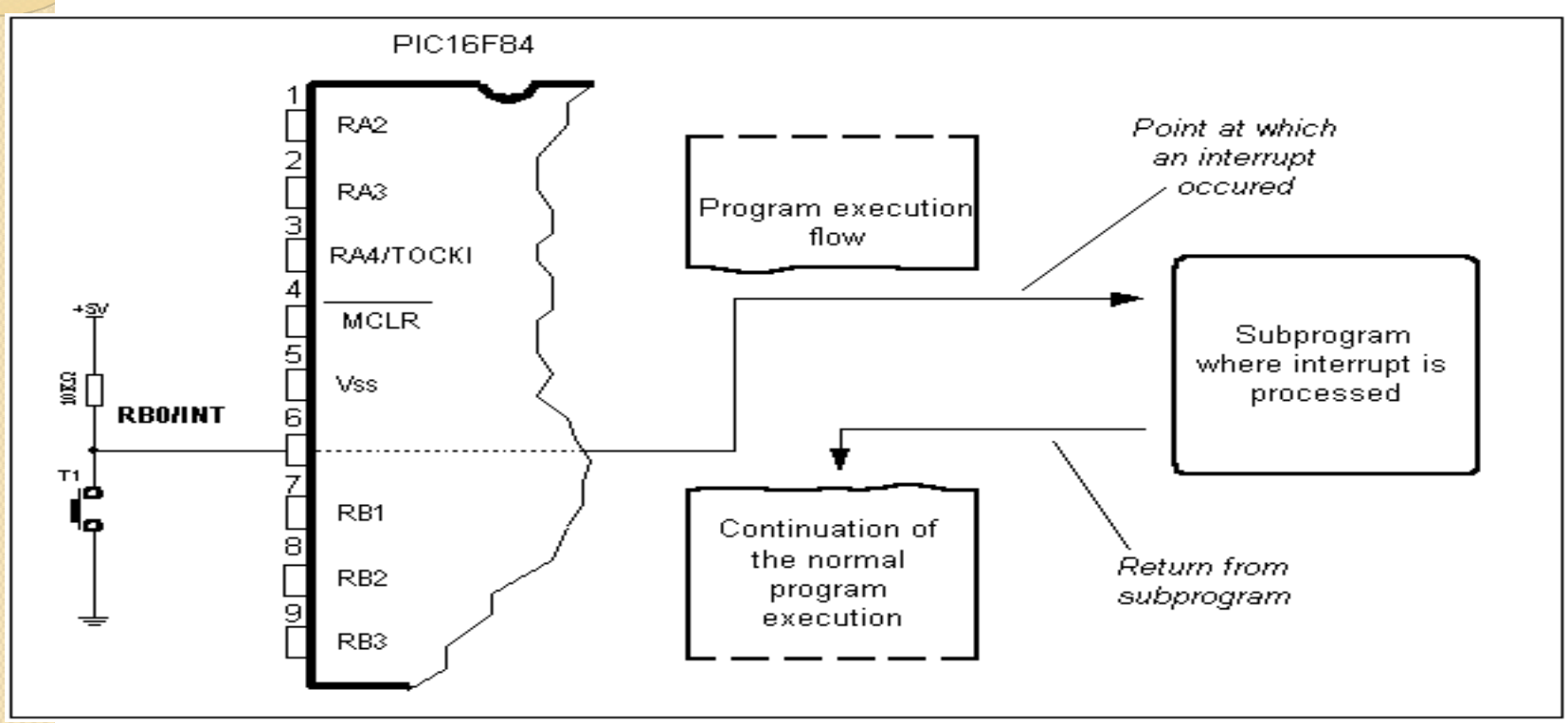
Using I/O Ports

```
; WRITTEN BY            VINAY NASSA
; DATE                  10/10/2010
; FILE SAVED AS         TEST.ASM
; DEVICE                16F84
; OSCILLATOR            XT (4MHZ)
; WATCHDOG              DISABLED
; FUNCTION              OUTPUTS THE VALUE 0XF1 TO 8 LEDS CONNECTED TO PORTB
; ----------------------   EQUATES    ----------------------------------
PORTB   EQU     0X06    ;ASSIGN THE PORTB REGISTER TO THE LABEL 'PORTB'
; ---------------------- MAIN PROGRAM ----------------------------------
START   ORG     0X00    ;'ORG' SPECIFIES THE MEMORY LOCATION OF THE PROGRAM
        MOVLW   0X00    ;MOVE THE VALUE 00, I.E., ALL 0'S TO W
        TRIS    PORTB   ;CONFIGURE PORTB WITH THE VALUE IN W (THE
                        ;WORKING REGISTER) 1=INPUT AND 0=OUTPUT.
                        ;SO 00 (ALL 0'S) MAKES ALL PORTB LINES OUTPUTS.
        CLRF    PORTB   ;CLEAR THE PORTB REGISTER
        MOVLW   0XF1    ;MOVE THE HEX VALUE F1 TO THE WORKING REGISTER
        MOVWF   PORTB   ;OUTPUT THE VALUE TO PORTB
LOOP    GOTO    LOOP
        END
```

Interrupts

- Interrupts are used to change the normal flow of a program so that it can perform another (specified) function.
- Interrupts allow external events to change (interrupt) the normal flow of the software, executing code specifically designed for a response to the change.
- Processors without in-built interrupt support require programs which regularly inspect selected input lines. This is called ‘polling’. Polling is very expensive in terms of processing.
- Interrupts enable processors to automatically respond to specified events and concentrate processing power on executing a main program.

Interrupts



One of the possible sources of interrupt and how it affects the main program

Interrupts

- When an interrupt occurs the instruction currently being executed is completed. The program counter then jumps to address 0x04 in program memory and executes the instruction stored there.

- Interrupts can be enabled or disabled (masked) individually or globally (all disabled regardless of source.) This is done via the **interrupt control register** (**INTCON**.)

- Using interrupts:

```
                ORG     0X00
                GOTO    START
;
                ORG     0X04
                GOTO    INT_SERV

INT_SERV                        ; INT. SERV ROUTINE HERE
                ……….
                RETFIE          ; RETURN FROM INTERRUPT
;
START                           ; MAIN PROGRAM GOES HERE
                ……….
                END
```

Interrupts

The PIC16F84 supports 4 in-built interrupt sources:

- **RB0 interrupt:** Edge-triggered interrupt (via RB0/INT pin)
- **Port B change** (bits 7-4): Port B logic level change on bits 7,6,5,4
- **TMR0 overflow:** Timer/counter overflow interrupt
- **EE**

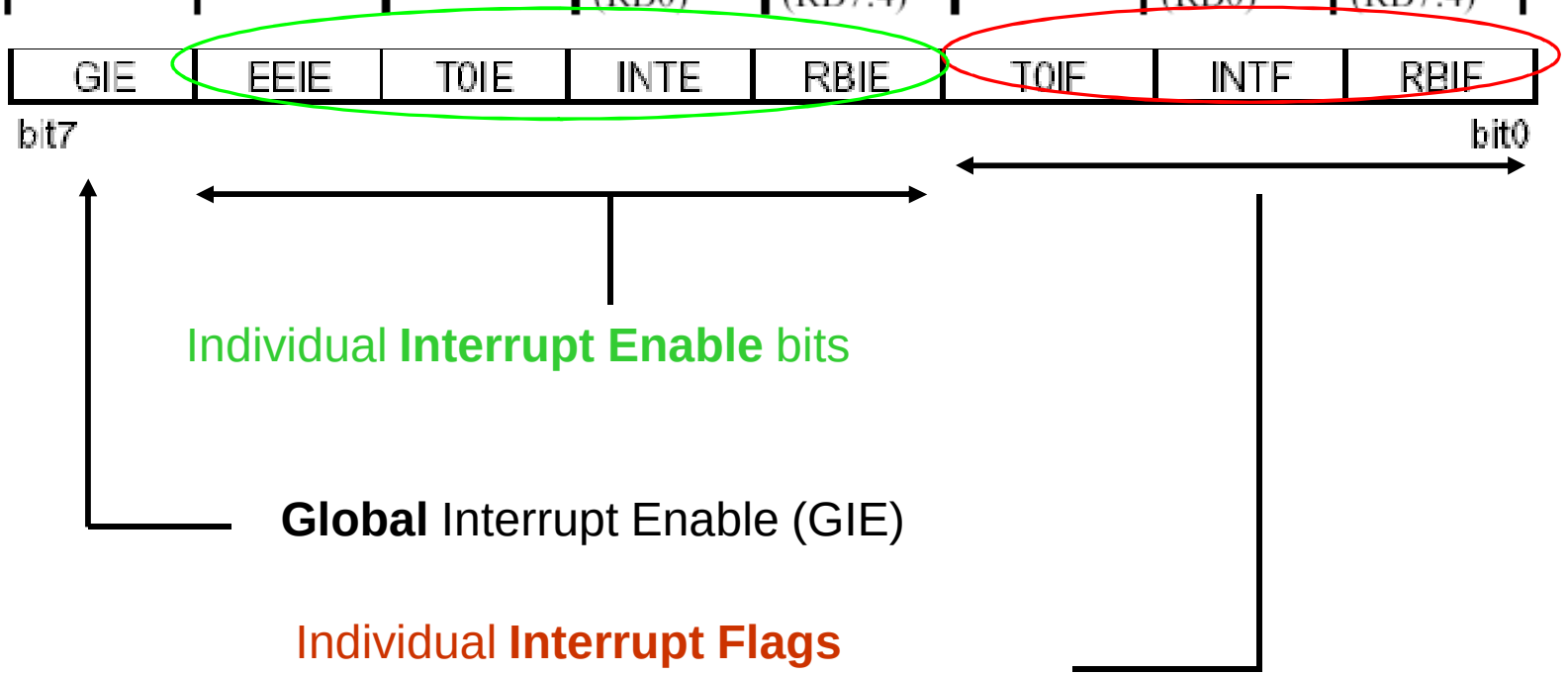
7	6	5	4	3	2	1	0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**
Global Interrupt Enable	EE write complete Interrupt Enable	Timer Overflow Interrupt Enable	INTerrupt Enable (RB0)	RB port change Interrupt Enable (RB7:4)	Timer Overflow Interrupt Flag	INTerrupt Flag (RB0)	RB port change Interrupt Flag (RB7:4)

- The RB0/INT is an edge-triggered interrupt. It can be enabled and disabled using the INTE flag in the INTCON register.
- PORTB pins (RB7 to RB4 inclusive) can be used as external interrupts. When configured as inputs, these pins can trigger an interrupt on change.
- Only 5 PORTB pins support interrupts. There are no interrupts on PORTA.

Interrupts

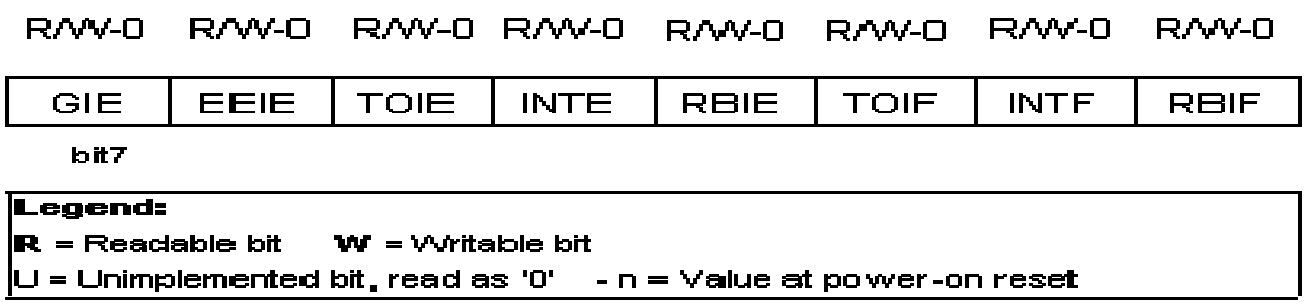
Interrupts are controlled by the **INTCON**

7	6	5	4	3	2	1	0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**
Global Interrupt Enable	EE write complete Interrupt Enable	Timer Overflow Interrupt Enable	INTerrupt Enable (RB0)	RB port change Interrupt Enable (RB7:4)	Timer Overflow Interrupt Flag	INTerrupt Flag (RB0)	RB port change Interrupt Flag (RB7:4)



Control register

- INTCON Register



- *Bit 7* **GIE** (*Global Interrupt Enable bit*) Bit which enables or disables all interrupts.
  1 = all interrupts are enabled
  0 = all interrupts are disabled
- *Bit 6* **EEIE** (*EEPROM Write Complete Interrupt Enable bit*) Bit which enables an interrupt at the end of a writing routine to EEPROM
  1 = interrupt enabled
  0 = interrupt disabled
  If EEIE and EEIF (which is in EECON1 register) are set simultaneously , an interrupt will occur

INTCON Register

- *bit 5* **T0IE** (*TMR0 Overflow Interrupt Enable bit*) Bit which enables interrupts during counter TMR0 overflow.
  1 = interrupt enabled
  0 = interrupt disabled
  If T0IE and T0IF are set simultaneously, interrupt will occur.
- *bit 4* **INTE** (*INT External Interrupt Enable bit*) Bit which enables external interrupt from pin RB0/INT.
  1 = external interrupt enabled
  0 = external interrupt disabled
  If INTE and INTF are set simultaneously, an interrupt will occur.
- *bit 3* **RBIE** (*RB port change Interrupt Enable bit*) Enables interrupts to occur at the change of status of pins 4, 5, 6, and 7 of port B.
  1 = enables interrupts at the change of status
  0 =interrupts disabled at the change of status
  If RBIE and RBIF are simultaneously set, an interrupt will occur.
- *bit 2* **T0IF** (*TMR0 Overflow Interrupt Flag bit*) Overflow of counter TMR0.
  1 = counter changed its status from FFh to 00h
  0 = overflow did not occur
  Bit must be cleared in program in order for an interrupt to be detected.

INTCON Register

- *bit 1* **INTF** (*INT External Interrupt Flag bit*) External interrupt occurred.
  1 = interrupt occurred
  0 = interrupt did not occur
  If a rising or falling edge was detected on pin RB0/INT, (which is defined with bit INTEDG in OPTION register), bit INTF is set.

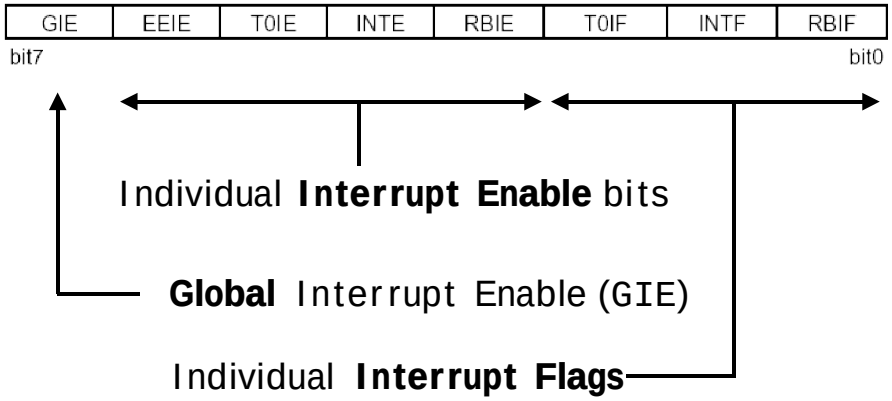
- *bit 0* **RBIF** (*RB Port Change Interrupt Flag bit*) Bit which informs about changes on pins 4, 5, 6 and 7 of port B.
  1 = at least one pin has changed its status
  0 = no change occurred on any of the pins
  Bit has to be cleared in an interrupt subroutine to be able to detect further interrupts.

Interrupts : Saving Context

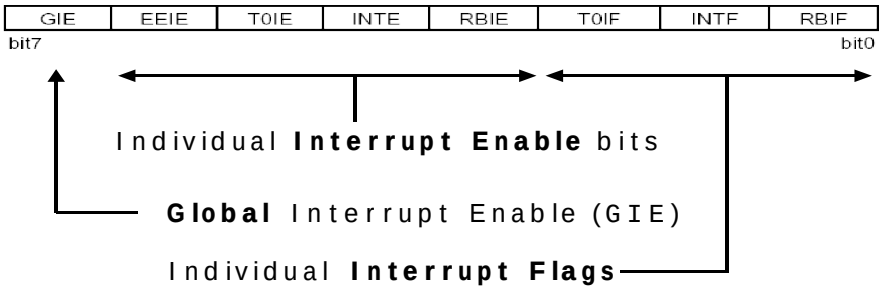
- We usually need to save context (save important current values) inside an interrupt service routine.
- These usually include the working register and the STATUS register.
- The values are backed up and restored inside the interrupt service routine.
- The values are backed up at the start and restored at the end.

Interrupts

- To use interrupts we need to set (enable) GIE.
- GIE is the Global Interrupt Enable and it 'lives' at bit number 7 in the interrupt control register.
- But enabling GIE doesn't enable any of the four types of interrupts.
- We still have to set each type of interrupt we want to use, by setting the corresponding enable bit.
- GIE is useful if we want to turn OFF interrupts. If we have all four types of interrupts enabled, then clearing GIE will stop all the different types of interrupts from having an effect.



Interrupts



EXAMPLE: To use the INTE interrupt.

- INTE is the edge-triggered interrupt available on RB0.
- We need to connect our interrupt source (a button perhaps) to RB0.
- We need to configure RB0 as an INPUT.
- We need to write an interrupt service routine. This is a special subroutine that will run whenever the interrupt happens.
- We need to set GIE and INTE.
- Now our interrupt service routine will run when the button is pressed, because this will set the interrupt flag.
- We must make sure our interrupt service routine clears the interrupt flag (INTF).
- **There needs to be a PORTB read or write prior to the clearing of RBIF.**

```
GIE       EQU     7
INTE      EQU     4
INTF      EQU     1

INTCON    EQU     0x0B

          ORG     0x00
          GOTO    START
          ORG     0x04
          GOTO    INT_SER

INT_SER   …
          BCF     INTCON,INTF
          …
          RETFIE

START     ….
          MOVLW   0x01            ;-RB0 is an I/P
          TRIS    PORTB           ;/
          BSF     INTCON, GIE   ;-Enable interrupt
          BSF     INTCON, INTE;/
```