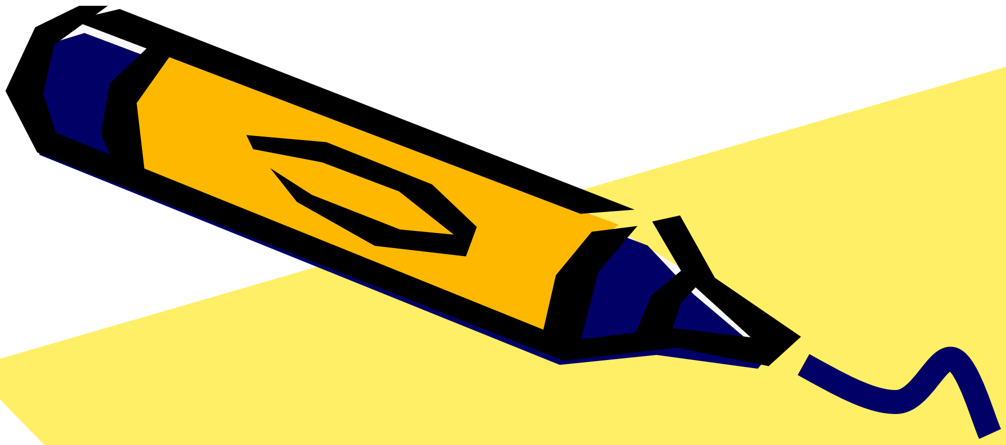
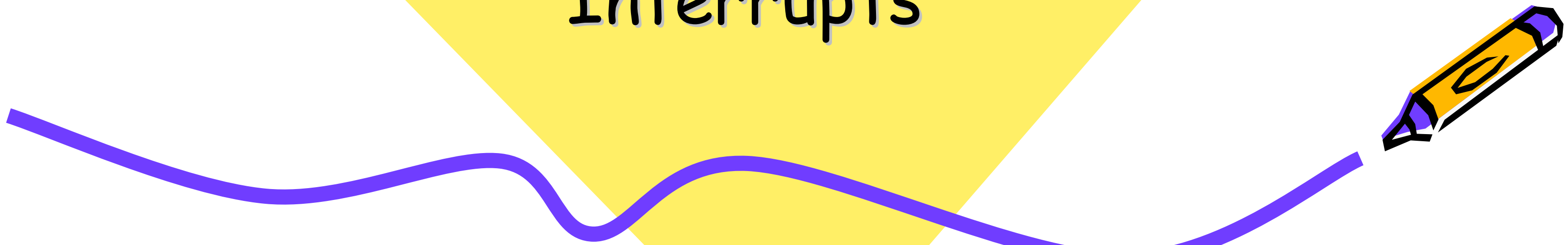


CENG-336 Introduction to Embedded Systems Development

Interrupts



Interrupts

Modern processors provide facilities for being interrupted by external devices.

This is more efficient than the processor asking devices if they need attention (polling).

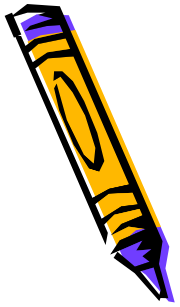
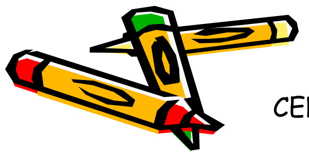
<u>Programmed I/O</u> - the status of the I/O device is checked by the program

<u>Interrupt I/O</u> - the I/O device request the interrupt, and an interrupt handling routine or interrupt service routine is needed

Interrupts

- An ***interrupt*** is a (temporary) break in the flow of execution of a program
  - the CPU is said to be "interrupted"

- When an interrupt occurs, the CPU deals with the interruption, then carries on where it was left off
  - ISR programmer should put the CPU back to normal operation.

Interrupt Example

1. Suppose you are sitting at home, chatting to someone.
2. Suddenly the telephone rings.
3. You stop chatting, and pick up the telephone to speak to the caller.
4. When you have finished your telephone conversation, you go back to chatting to the person before the telephone rang.

You can think of

- the main routine as you chatting to someone,
- the telephone ringing causes you to interrupt your chatting, and
- the interrupt routine is the process of talking on the telephone.
- When the telephone conversation has ended, you then go back to your main routine of chatting.

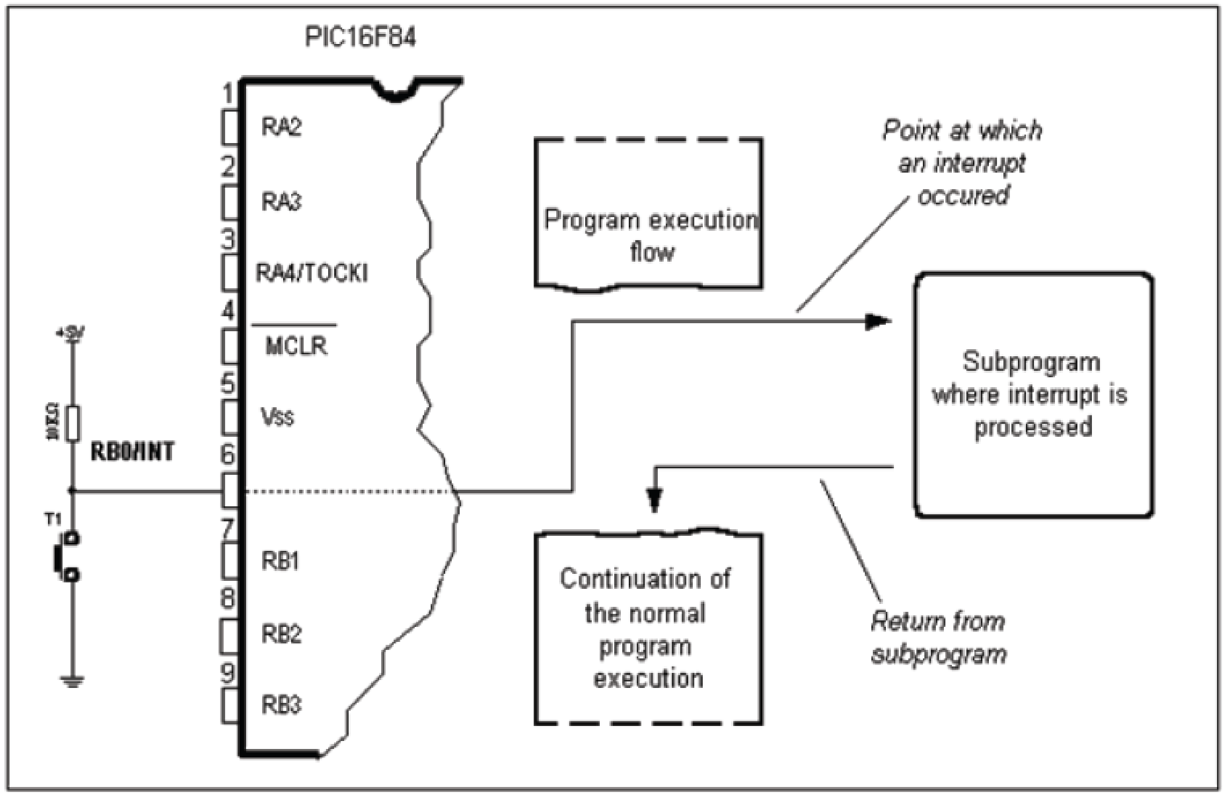
Interrupt Example

This example is exactly how an interrupt causes a processor to act.

- The main program is running, performing some function in a circuit,
  - but when an interrupt occurs the main program halts while another routine is carried out.
- When this routine finishes, the processor goes back to the main routine again.

PIC Interrupts

Generally, each interrupt changes the program flow, interrupts it and after executing an interrupt subprogram (interrupt –service- routine) it continues from that same point on.


PIC16F84
1
RA2
2
RA3
3
RA4/TOCKI
4
MCLR
5
Vss
6
7
RB1
8
RB2
9
RB3
+5V
10KΩ
RB0/INT
T1
Program execution flow
Point at which an interrupt occured
Subprogram where interrupt is processed
Continuation of the normal program execution
Return from subprogram

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

Sources of Interrupts

- Typical sources of interrupts on the PIC include
  - a positive (rising) or negative (falling) transition on the RB0/INT input,
  - a change on any of the inputs RB4 - RB7 or
  - a timer / counter overflow from a value of FFH to 00H.

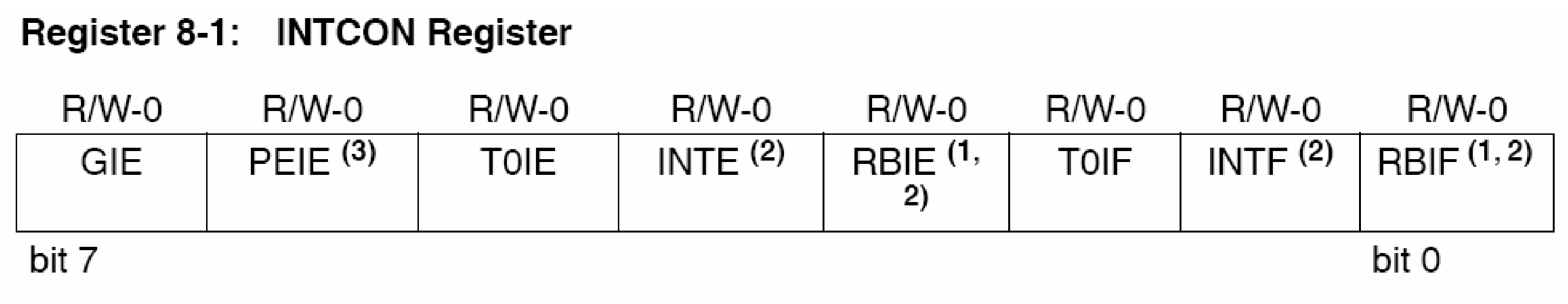
Sources of Interrupts ...

- These sources generally include one interrupt source for each peripheral module, though some modules may generate multiple interrupts (such as the USART module).

  The current interrupts are:

  - RB0/INT Pin Interrupt (external interrupt)
  - TMR0 Overflow Interrupt
  - PORTB Change Interrupt (pins RB7:RB4)
  - Comparator Change Interrupt
  - Parallel Slave Port Interrupt
  - USART Interrupts
  - Receive Interrupt
  - Transmit Interrupt
  - A/D Conversion Complete Interrupt
  - LCD Interrupt.
  - Data EEPROM Write Complete Interrupt
  - Timer1 Overflow Interrupt
  - Timer2 Overflow Interrupt
  - CCP Interrupt
  - SSP Interrupt

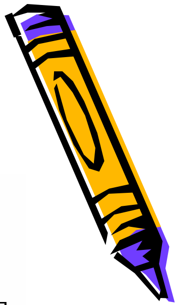
INTCON


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-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

- Inside the PIC there is a register called INTCON, and is at address 0Bh.
- Within this register there are 8 bits that can be enabled or disabled.
- Bit 7 of INTCON is called GIE.
  - This is the Global Interrupt Enable.
  - Setting this to 1 tells the PIC that we are going to use an interrupt.

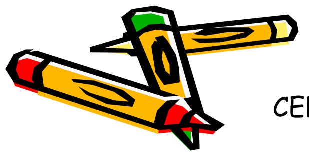
INTCON



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-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

- Bit 4 of INTCON is called INTE,
  - which means INTerrupt Enable. Setting this bit to 1 tells the PIC that RB0 will be an interrupt pin.
- Setting bit 3, called RBIE,
  - tells the PIC that we will be using Port B bits 4 to 7.



External interrupt on RB0/INT pin

- External interrupt on RB0/INT pin is triggered by
  - rising edge, if bit INTEDG=1 (in OPTION<6> register),
  - falling edge, if INTEDG=0.
- When correct signal appears on INT pin, INTF bit is set in INTCON register.
- INTF bit (INTCON<1>) must be reset in interrupt routine, so that interrupt would not occur again while going back to the main program.
  - This is an important part of the program, which the programmer must not forget, or program will constantly go into interrupt routine.
- Interrupt can be turned off by resetting INTE control bit (INTCON<4>).

TMR0 Counter Overflow Interrupt

- Overflow of TMR0 counter (from FFhto 00h) will set T0IF (INTCON<2>) bit.
- This is very important interrupt because many real problems can be solved using this interrupt.
- One of the examples is time measurement.
  - If we know how much time counter needs in order to complete one cycle from 00h to FFh, then a number of interrupts multiplied by that amount of time will yield the total of elapsed time.
  - In interrupt routine some variable would be incremented (in RAM memory). Value of that variable multiplied by the amount of time the counter needs to count through a whole cycle, would yield total elapsed time.
- Interrupt can be turned on/off by setting/resetting T0IE (INTCON<5>) bit.

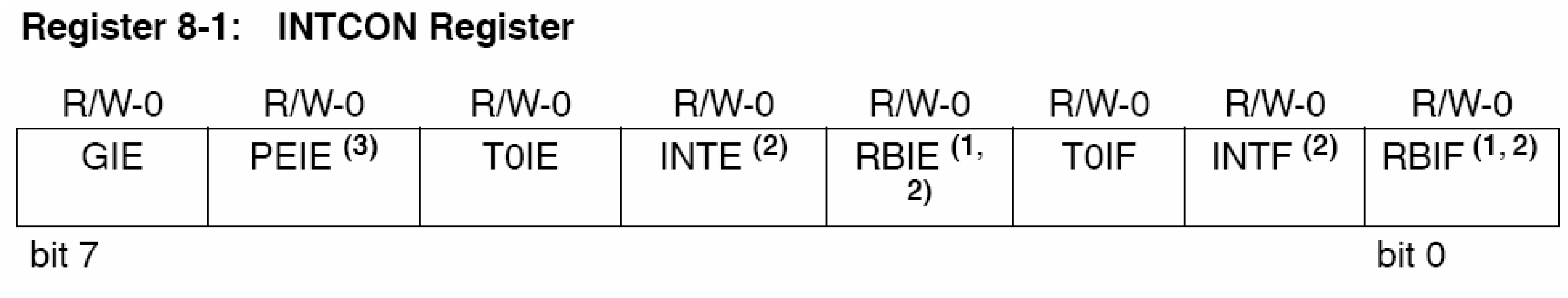
Interrupt [Pins 4-7 of Port B]

- Change of input signal on PORTB <7:4> sets RBIF (INTCON<0>) bit.
- Four pins RB7 to RB4 of port B, can trigger an interrupt which occurs when status on them changes from logic one to logic zero, or vice versa.
- For pins to be sensitive to this change, they must be defined as input.
- If any one of them is defined as output, interrupt will not be generated at the change of status.
- If they are defined as input, their current state is compared to the old value which was stored at the last reading from port B.
- Interrupt can be turned on/off by setting/resetting RBIE bit (bit 3) in INTCON register.

Interrupt upon finishing write-subroutine to EEPROM

- This interrupt is of practical nature only.
- Since writing to oneEEPROM location takes about 10ms (which is a long time in the notion of a microcontroller), it doesn't pay off to a microcontroller to wait for writing to the end.
- Thus interrupt mechanism is added which allows the microcontroller to continue executing the main program, while writing in EEPROM is being done in the background.
- When writing is completed, interrupt informs the microcontroller that writing has ended.
- EEIF bit, through which this informing is done, is found in EECON1 register.
- Occurrence of an interrupt can be disabled by resetting the EEIE bit in INTCON register.

INTCON



Register 8-1: INTCON Register

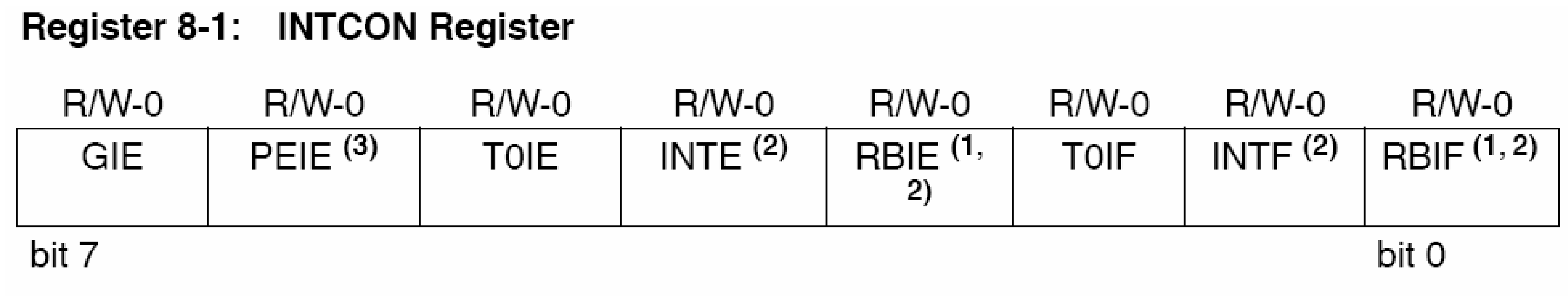
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

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 **PEIE** (*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

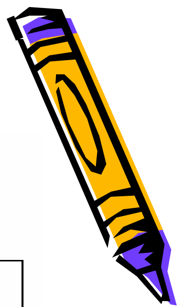
If PEIE and EEIF (which is in EECON1 register) are set simultaneously , an interrupt will occur.

INTCON



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-0
GIE	PEIE [3]	T0IE	INTE [2]	RBIE [1, 2]	T0IF	INTF [2]	RBIF [1, 2]
bit 7							bit 0



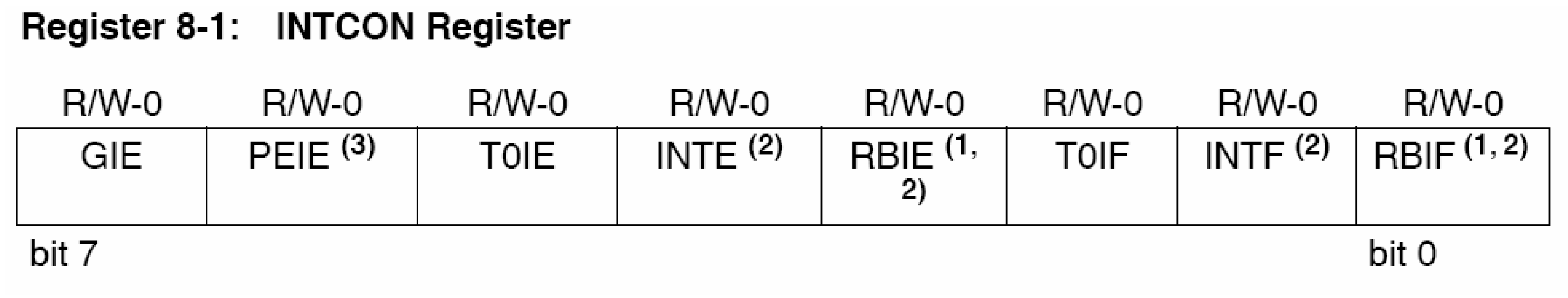
Bit 5 **T0IE** (*TMR0 Overflow Interrupt Enable bit*) Bit which enables interrupts during counter TMR0 overflow.
1 = interrupt enabled 0 = interrupt disabled

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.

INTCON


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-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

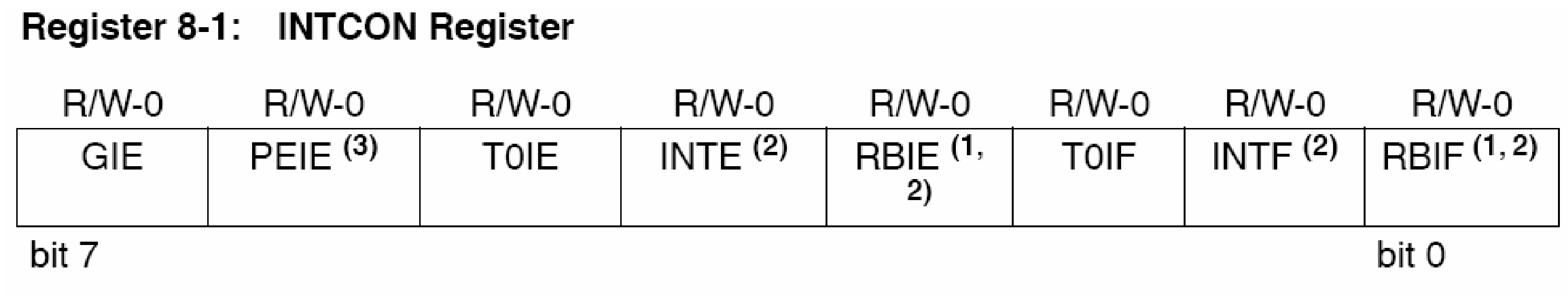
Bit 3 **RBIE** (*RB port change Interrupt Enable bit*) Enables interrupts to occur at the change of status of pins 4, 5, 6, and 7 of port B.
1 = enables interrupts at the change of status 0 =interrupts disabled at the change of status

If RBIE and RBIF are simultaneously set, an interrupt will occur.

Bit 2 **TOIF** (*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 8-1: INTCON Register

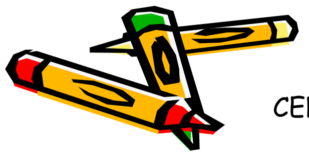
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE (3)	T0IE	INTE (2)	RBIE (1, 2)	T0IF	INTF (2)	RBIF (1, 2)
bit 7							bit 0

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.



Interrupt Initialization

- In order to use an interrupt mechanism of a microcontroller, some preparatory tasks need to be performed ("initialization").
- By initialization we define
  - to what interrupts the microcontroller will respond, and
  - which ones it will ignore.
- If we do not set the bit that allows a certain interrupt, program will not execute an interrupt subprogram.
- Through this we can obtain control over interrupt occurrence, which is very useful.

```
clrf  INTCON               ; all interrupts disabled
movlw B'00010000'          ; external interrupt only is enabled
bsf  INTCON, GIE           ; occurrence of interrupts allowed
```

What Happens?

- On interrupt,
  - the processor saves the return address on the stack and
  - program control is redirected to the interrupt service routine.
- It is important to note that aside from the return address no registers are saved.
- However the user may save such important registers as W and STATUS.

What happens in the program and in the PIC?

- First, a 'flag' is set.
  - This tells the internal processor of the PIC that an interrupt has occurred.
- Secondly, the program counter points to a particular address within the PIC.

External Interrupt

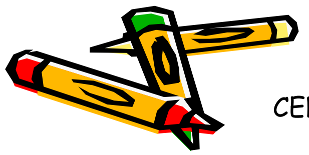
- When external interrupt occurs, INTF will be set to 1. While there isn't an interrupt, the flag is kept at 0.
- While this flag is set to 1, PIC cannot, and will not, respond to any other interrupt.
- If this flag was not set to 1,
  - PIC is allowed to keep responding to the interrupt,
  - then continually pulsing the pin will keep the PIC going back to the start of our interrupt routine, and never finishing it.
- Although PIC automatically sets this flag to 1, it doesn't set it back to 0!
  - That task has to be done by the programmer after the PIC has executed the interrupt routine.

Initially …

- When you first power up the PIC, or if there is a reset,
  - Program Counter points to address 0000h, which is right at the start of the program memory.
- When there is an interrupt, Program Counter will point to address 0004h.
- When we are writing our programs that are going to have interrupts,
  - first of all, we have to tell the PIC to jump over address 0004h, and
  - keep the interrupt routine which starts at address 0004h separate from the rest of the program.

Coding ...

- Start the program with a command called ORG.
  - This command means Origin, or start.
  - We follow it with an address.
  - Because the PIC will start at address 0000h, we type "ORG 0000h".
- Next we need to skip over address 0004h.
  - We do this by placing a GOTO instruction, followed by a label which points to our main program.



Coding …

- We then follow this GOTO command with another ORG, this time with the address 0004h.
  - It is after this command that we enter our interrupt routine.
  - Now, we could either type in our interrupt routine directly following the second ORG command, or we can place a GOTO statement which points to the interrupt routine.
- To tell the PIC that it has come to the end of the interrupt routine we need to place the command RTFIE at the end of the routine.
  - This command means return from the interrupt routine.
  - When the PIC see this, the Program Counter points to the last location the PIC was at before the interrupt happened.

Coding …

```
        ORG    0000h   ;PIC starts here on power up and reset
        GOTO  start    ;Goto our main program

        ORG    0004h   ;The PIC will come here on an interrupt
        :              ;This is our interrupt routine that we
        :              ;want the PIC to do when it receives
        :              ;an interrupt
        . . .
        . . .
        . . .
        RETFIE         ;End of the interrupt routine
start                  ;This is the start of our main program.
```

Or …

- On interrupt, program flow is directed to program location 004H.
- As “reset” directs program flow to 000H, a program using interrupts is usually structured as follows;

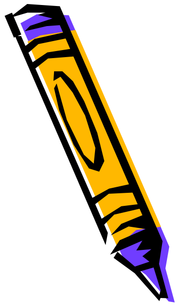
```
    ORG      000H              ; a reset redirects program to this point
    GOTO     MAIN

    ORG      004H              ; an interrupt redirects the program to here
    GOTO     INT_SERV

MAIN:
                               ; Your main program

                               ; end of main

INT_SERV:
                               ; your interupt service routine
```

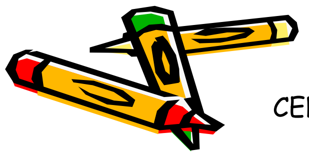
Coding …

- There are two things you should be aware of when using interrupts.
  - The first is that, if you are using the same register in your main program and the interrupt routine, bear in mind that the contents of the register will probably change when the interrupt occurs.
  - The way to get around this, is to temporarily store the contents of the w register before you use it again in the interrupt routine.

Coding …

- The <u>second</u> is that, there is a delay between when one interrupt occurs and when the next one can occur.
- PIC has an external clock, which can either be a crystal or it can be a resistor-capacitor combination.
  - Whatever the frequency of this clock, the PIC divides it by 4 and then uses this for it's internal timing.
  - This internal timing is called an Instruction Cycle.
- Data sheet states that you must allow 3 to 4 instruction cycles between interrupts.
  - Allow 4 cycles.
  - The reason for the delay is the PIC needs time to jump to the interrupt address, set the flag, and come back out of the interrupt routine.





Coding ...

- A point to remember is that if you use bits 4 to 7 of Port B as an interrupt:
  - You cannot select individual pins on Port B to serve as an interrupt.
  - So, if you enable these pins, then they are all available.
  - For example, you can't just have bits 4 and 5 – bits 6 and 7 will be enabled as well.

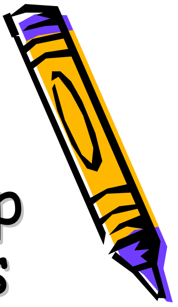
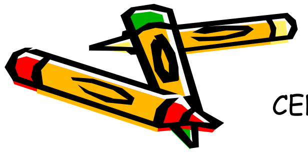
Example

- The program we are going to write will
  - count the number of times we turn a switch on,
  - and then display the number.
- The program will
  - count from 0 to 9, displayed on 4 LEDs in binary form,
  - and the input or interrupt will be on RB0.

Example ...

- The first thing we need to do is tell the PIC to jump over the address where the Program Counter points to when an interrupt occurs.

```
org         0x00        ;This is where the PC points to on power up
                        ;and reset
goto         main       ;Goto our main program
org         0x04        ;This is where our interrupt routine
                        ; will start
retfie                  ;This tells the PIC that the interrupt
                        ;routine has finished and the PC will
                        ;point back to the main program

main                    ;This is the start of our main program
```

Example ...

- Now we need to tell the PIC that we are going to use interrupts, and we are using RB0 pin 6 as an interrupt pin:

```
bsf      INTCON,7      ;GIE – Global interrupt enable
                       ;(1=enable)
bsf      INTCON,4      ;INTE - RB0 interrupt enable
                       ;(1=enable)
```

- We are going to clear the interrupt flag just in case

```
bcf      INTCON,1      ;INTF - Clear flag bit just in case
```

Example …

- Now we need to set up our two ports. Remember that as we are using RB0 as an interrupt pin, this must be set up as an input:

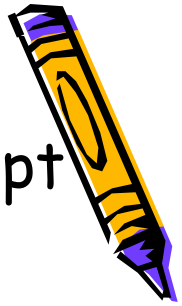
```
bsf      STATUS,5    ;Switch to Bank 1
movw     0x01        ;
movwf    TRISB       ;Set RB0 as input
movlw    0x10        ;
movwf    TRISA       ;Set the first 4 pins on PortA as output
bcf      STATUS,5    ;Come back to Bank 0
```

Example ...

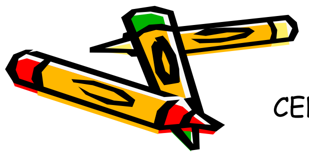
- We are going to use a variable called COUNT to store the number of switch counts.

```
loop
        movf    COUNT,0         ;Move the contents of COUNT into W
        movwf   PORTA           ;Now move it to Port A
        goto    loop            ;Keep on doing this
        end                     ;End of our program
```

Example ...



- We need to tell the PIC what to do when an interrupt happens.
- In this instance, our interrupt is going to be the switch.
- What we want the PIC to is add one to the variable COUNT each time the switch is closed.
- However, we only want to display the number of times the switch closes from 0 to 9.
- Port A has 5 bits, and if we just simply incremented the port, we will have a maximum count of 31.
- There is a reason why we chose not to go up to 31.
  - We are going to use a 7-segment display, which can at the most only go from 0 to 15 (0 to F in hex).



Example …

- Now the first thing we need to do is temporarily store the contents of our w register, as we are using this to transfer the contents of COUNT to PORTA.
- If we don't store it, then we could send a completely different number as a result of our arithmetic. So let's do that first:

```
movwf       TEMP      ;Store w register in a temporary location
```

- Next we want to add 1 to our variable COUNT:

```
incf        COUNT,1   ;Increment COUNT by 1, and put the result
                      ;back into COUNT
```

- Next we want to do a check on COUNT to see if we have gone past the value of 9. The way we can do this is to subtract it from 10.

```
movlw       0x0A      ;Move the value 10 into w
subwf       COUNT,0   ;Subtract w from COUNT, and put the
                      ;result in w
```

Example ...

- If we subtract a large number from a small number a Carry flag will be set.
- This flag will also be set if the numbers are equal, and we subtract them.

```
btfss     STATUS,0     ;Check the Carry flag. It will be set if
                       ;COUNT is equal to, or is greater than w,
                       ;and will be set as a result of the subwf
                       ;instruction
```

Example ...

- Now we know if the value of COUNT is 9 or more.
- What we want to do now is if COUNT is greater than 9, put it back to 0, otherwise go back to the main program so that we can send it to Port A.
- BTFSS command as you know will skip the next instruction if the carry flag is set i.e. COUNT = 10:

```
    goto     carry_on        ;If COUNT is <10, then we can carry on
    goto     clear           ;If COUNT is >9, then we need to clear it

carry_on
    bcf      INTCON,0x01     ;We need to clear this flag to enable
                             ;more interrupts
    movfw    TEMP            ;Restore w to the value before the interrupt
    retfie                   ;Come out of the interrupt routine

clear
    clrf     COUNT           ;Set COUNT back to 0
    bcf      INTCON,1        ;We need to clear this flag to enable
                             ;more interrupts
    retfie                   ;Come out of the interrupt routine
```

Altogether

```
org       0x00            ;This is where we come on power up and reset
;*******************SETUP CONSTANTS*******************

INTCON   EQU   0x0B   ;Interrupt Control Register
PORTB    EQU   0x06   ;Port B register address
PORTA    EQU   0x05   ;Port A register address
TRISA    EQU   0x85   ;TrisA register address
TRISB    EQU   0x86   ;TrisB register address
STATUS   EQU   0X03   ;Status register address
COUNT    EQU   0x0c   ;This will be our counting variable
TEMP     EQU   0x0d   ;Temporary store for w register
  goto   main         ;Jump over the interrupt address
```

Altogether …

```
;***************INTERRUPT ROUTINE***************
 org      0x04          ;This is where PC points on an interrupt
 movwf    TEMP          ;Store the value of w temporarily
 incf     COUNT,1       ;Increment COUNT by 1, and put the result
                        ;back into COUNT
 movlw    0x0A          ;Move the value 10 into w
 subwf    COUNT,0       ;Subtract w from COUNT, and put the result in w
 btfss    STATUS,0      ;Check the Carry flag. It will be set if COUNT is
                        ;equal to, or is greater than w, and will be set as a
                        ;result of the subwf instruction
 goto     carry_on      ;If COUNT is <10, then we can carry on
 goto     clear         ;If COUNT is >9, then we need to clear it
carry_on
 bcf      INTCON,0x01   ;We need to clear this flag to enable
                        ;more interrupts
 movfw    TEMP          ;Restore w to the value before the interrupt
 retfie                 ;Come out of the interrupt routine
clear
 clrf     COUNT         ;Set COUNT back to 0
 bcf      INTCON,1      ;We need to clear this flag to enable
                        ;more interrupts
 retfie                 ;Come out of the interrupt routine
```

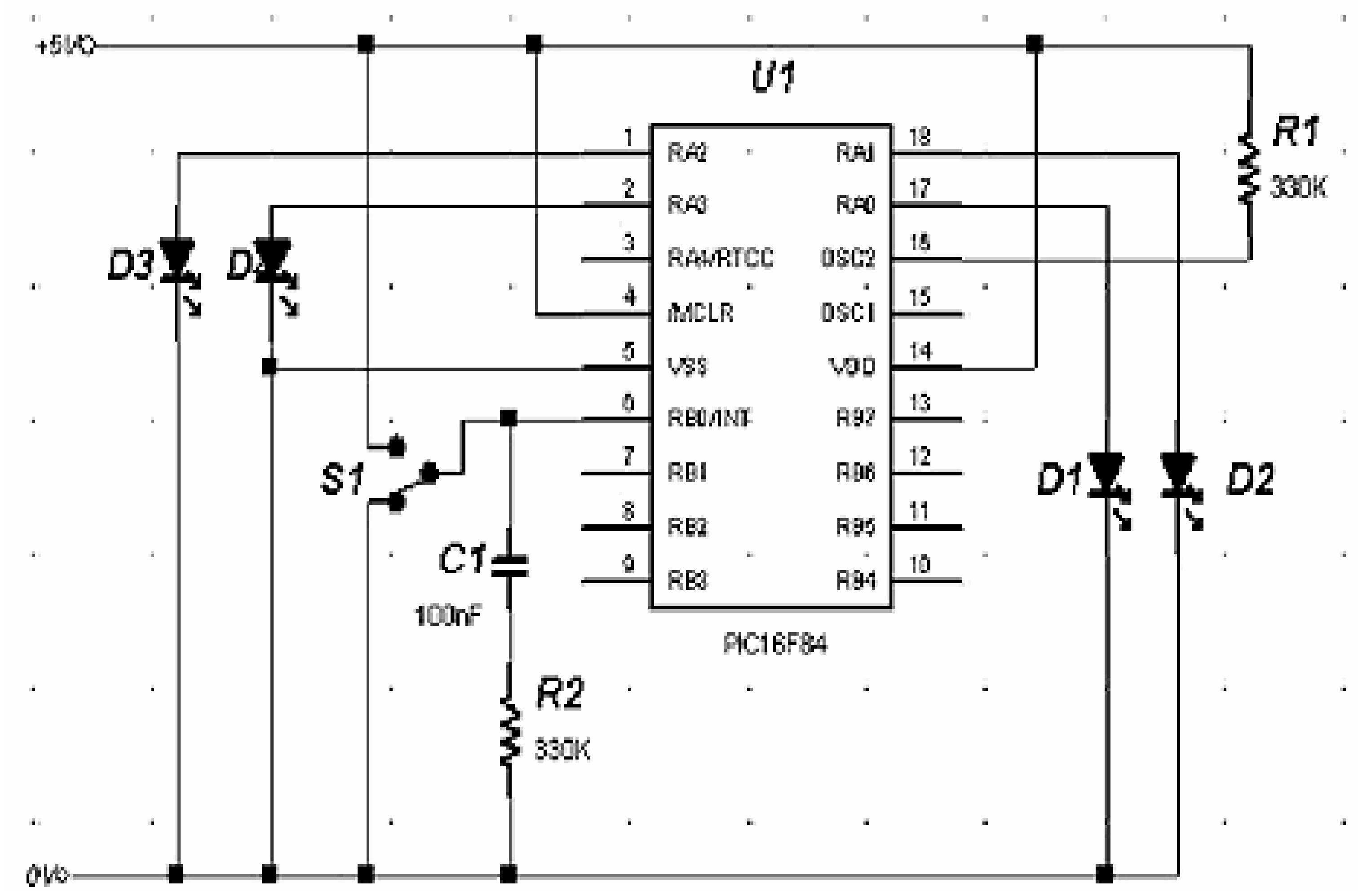
Altogether …

```
;******************** Main Program *********************
main
;************** Set Up The Interrupt Registers ***********
 bsf      INTCON,7     ;GIE – Global interrupt enable (1=enable)
 bsf      INTCON,4     ;INTE - RB0 Interrupt Enable (1=enable)
 bcf      INTCON,1     ;INTF - Clear FLag Bit Just In Case
;****************** Set Up The Ports *****************
 bsf      STATUS,5     ;Switch to Bank 1
 movlw    0x01
 movwf    TRISB        ;Set RB0 as input
 movlw    0x10
 movwf    TRISA        ;Set R 0 to RA3 on PortA as output
 bcf      STATUS,5     ;Come back to Bank 0
;********* Now Send The Value Of COUNT To Port A ********
loop
 movf     COUNT,0      ;Move the contents of Count into W
 movwf    PORTA        ;Now move it to Port A
 goto     loop         ;Keep on doing this
 end                   ;End Of Program
```

Notes ...

- Next is the circuit diagram that will work for the code above.
- First, we have not included a timing capacitor in the oscillator circuit.
  - This is a clever little trick that you can try if you run out of capacitors.
  - The capacitance comes from the stray capacitance between the oscillator pin and ground.
  - so, with the resistor and the stray capacitance, we have an RC oscillator.
- Secondly, We have included a de-bouncing circuit across the switch.
  - This is needed because every time you flick a switch, the contacts will bounce.
  - This will make the PIC think there have been more than one switches.
  - With the de-bouncing circuit, when the switch goes high, the capacitor charges up.
  - no matter how many times the switch goes to +5V, the capacitor will only charge once.
  - The capacitor is discharged when the switch is thrown the other way.

Circuit Diagram



Notes ...

- The user may control the sources of interrupts. For example;

```
BSF    INTCON, INTE    ; enable interupts on RB0/INT
BSF    INTCON, RBIE    ; enable change in RB4 - RB7 interrupt
BSF    INTCON, TOIE    ; enable timer interrupt
```

- Any of these may be used alone, or several sources may be enabled, depending on your application.

Notes …

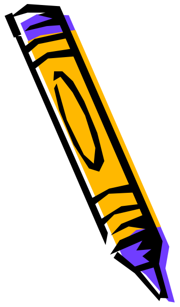
- In addition to INTCON, if the device has peripheral interrupts, then it will have registers to enable the peripheral interrupts and registers to hold the interrupt flag bits. Depending on the device, the registers are:
  - PIE1 Peripheral Interrupt Enable Register
  - PIR1 Peripheral Interrupt Flag Register
  - PIE2
  - PIR2

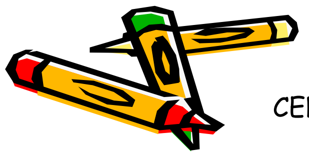
PIE Register(s)

- Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Enable registers (PIE1, PIE2).
- These registers contain the individual enable bits for the Peripheral interrupts.
- These registers will be generically referred to as PIE.
- If the device has a PIE register, The PEIE bit must be set to enable any of these peripheral interrupts.

PIE Register(s)

- Although, the PIE register bits have a general bit location with each register, future devices may not have consistent placement.
- Bit location inconsistencies will not be a problem if you use the supplied Microchip Include files for the symbolic use of these bits.
- This will allow the Assembler/Compiler to automatically take care of the placement of these bits by specifying the correct register and bit name.
- **Note:**
  - Bit PEIE (INTCON<6>) must be set to enable any of the peripheral interrupts.





PIE Register

- **TMR1IE** : TMR1 Overflow Interrupt Enable bit
  1 = Enables the TMR1 overflow interrupt 0 = Disables the TMR1 overflow interrupt bit
- **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
- **CCP1IE**: CCP1 Interrupt Enable bit
  1 = Enables the CCP1 interrupt 0 = Disables the CCP1 interrupt bit
- **CCP2IE**: CCP2 Interrupt Enable bit
  1 = Enables the CCP2 interrupt 0 = Disables the CCP2 interrupt bit
- **SSPIE**: Synchronous Serial Port Interrupt Enable bit
  1 = Enables the SSP interrupt 0 = Disables the SSP interrupt bit
- **RCIE**: USART Receive Interrupt Enable bit
  1 = Enables the USART receive interrupt 0 = Disables the USART receive interrupt
- 8 more

PIR

- These registers contain the individual flag bits for the peripheral interrupts.
- Very similar to PIE

Edge Triggering

- Indicate PIC to interrupt when the signal goes
  - from low to high, or
  - from high to low.
- By default, this is set up to be on the rising edge (low to high)
- Edge 'triggering' is set up in another register called the OPTION register, at address 81h.
  - The bit we are interested in is bit 6, which is called INTEDG.
  - Setting this to 1 will cause the PIC to interrupt on the rising edge (default state) and
  - setting it to 0 will cause the PIC to interrupt on the falling edge.
  - If you want the PIC to trigger on the rising edge, then you don't need to do anything to this bit.
- Option register is in Bank 1, which means that
  - we have to change from bank 0 to bank 1,
  - set the bit in the Option register,
  - then come back to bank 0.

INT and External Interrupts

- The external interrupt on the INT pin is edge triggered:
  - either rising if the INTEDG bit (OPTION<6>) is set, or
  - falling, if the INTEDG bit is clear.
- When a valid edge appears on the INT pin, the INTF flag bit (INTCON<1>) is set.
- This interrupt can be enabled/disabled by setting/clearing the INTE enable bit (INTCON<4>).
- The INTF bit must be cleared in software in the interrupt service routine before re-enabling this interrupt.
- The INT interrupt can wake-up the processor from SLEEP, if the INTE bit was set prior to going into SLEEP.
- The status of the GIE bit decides whether or not the processor branches to the interrupt vector following wake-up.
- See the "**Watchdog Timer and Sleep Mode**" section for details on SLEEP and for timing of wake-up from SLEEP through INT interrupt.

Saving the STATUS and W Registers in RAM

```
MOVWF   W_TEMP           ; Copy W to a Temporary Register
                         ; regardless of current bank
SWAPF   STATUS,W         ; Swap STATUS nibbles and place
                         ; into W register
BCF     STATUS,RP0       ; Change to Bank0 regardless of
                         ; current bank
MOVWF   STATUS_TEMP      ; Save STATUS to a Temporary register
                         ; in Bank0
     :
     : (Interrupt Service Routine (ISR) )
     :
SWAPF   STATUS_TEMP,W    ; Swap original STATUS register value
                         ; into W (restores original bank)
MOVWF   STATUS           ; Restore STATUS register from
                         ; W register
SWAPF   W_TEMP,F         ; Swap W_Temp nibbles and return
                         ; value to W_Temp
SWAPF   W_TEMP,W         ; Swap W_Temp to W to restore original
                         ; W value without affecting STATUS
```

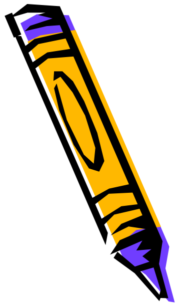
More …

- Once in the interrupt service routine, the source(s) of the interrupt can be determined by polling (inspecting) the interrupt flag bits.
- Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

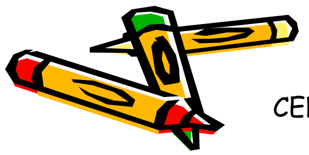
Initialization and enabling of device interrupts, where PIE1_MASK1 value is the value to write into the interrupt enable register

```
PIE1_MASK1   EQU    B'01101010'      ; This is the Interrupt
   Enable
                                     ; Register mask value
:
CLRF         STATUS         ; Bank0
CLRF         INTCON         ; Disable interrupts and clear some flags
CLRF         PIR1           ; Clear all flag bits
BSF          STATUS, RP0    ; Bank1
MOVLW        PIE1_MASK1     ; This is the initial masking for PIE1
MOVWF        PIE1           ;
BCF          STATUS, RP0    ; Bank0
BSF          INTCON, GIE    ; Enable Interrupts
```

Register Saving / Restoring as Macros



```
PUSH_MACRO      MACRO                          ; This Macro Saves register
    contents
                MOVWF  W_TEMP                  ; Copy W to a Temporary Register
                                               ; regardless of current bank
                SWAPF  STATUS,W                ; Swap STATUS nibbles and place
                                               ; into W register
                MOVWF  STATUS_TEMP             ; Save STATUS to a Temporary
                                               ; register in Bank0
ENDM                                           ; End this Macro
                                               ;
POP_MACRO       MACRO                          ; This Macro Restores register
                                               ;contents
                SWAPF  STATUS_TEMP,W ; Swap original STATUS register
    value
                                               ; into W (restores original bank)
                MOVWF  STATUS                          ; Restore STATUS
    register from
                                               ; W register
                SWAPF  W_TEMP,F                ; Swap W_Temp nibbles and return
                                               ; value to W_Temp
```



```
org  ISR_   ADDR                 ;
     PUSH_  MACRO                 ; MACRO that saves required context registers,
                                  ; or in-line code
     CLRF   STATUS                ; Bank0
     BTFSC  PIR1, TMR1IF          ; Timer1 overflow interrupt?
     GOTO T1_INT                  ; YES
     BTFSC  PIR1, ADIF            ; NO, A/D interrupt?
     GOTO AD_INT                  ; YES, do A/D thing
      :                           ; NO, do this for all sources
      :                           ;
     BTFSC  PIR1, LCDIF                         ; NO, LCD interrupt
     GOTO LCD_INT                 ; YES, do LCD thing
     BTFSC INTCON, RBIF           ; NO, Change on PORTB interrupt?
     GOTO PORTB_INT               ; YES, Do PortB Change thing
INT_ERROR_LP1                     ; NO, do error recovery
     GOTO INT_ERROR_LP1           ; This is the trap if you enter the ISR
                                  ; but there were no expected
                                  ; interrupts
T1_INT                            ; Routine when the Timer1 overflows
      :                           ;
     BCF    PIR1, TMR1IF          ; Clear the Timer1 overflow interrupt flag
     GOTO   END_ISR               ; Ready to leave ISR (for this request)
     AD_INT                       ; Routine when the A/D completes
      :                           ;
     BCF    PIR1, ADIF            ; Clear the A/D interrupt flag
     GOTO   END_ISR               ; Ready to leave ISR (for this request)
     LCD_INT                      ; Routine when the LCD Frame begins
      :                           ;
     BCF    PIR1, LCDIF                ; Clear the LCD interrupt flag
     GOTO   END_ISR               ; Ready to leave ISR (for this request)
```