Chapter 6

Dr. Iyad Jafar
Outline

- Introduction
- Interrupts
- Timer/Counter
- Watchdog Timer
- Sleep Mode
- Summary
Introduction

- Microcontroller should be able to deal with time
  - Respond in a timely manner to external events
  - Measure time between events
  - Generate time-based activity

- For this purpose, microcontrollers are usually provided with timers and support interrupts
Interrupts

- An interrupt is an event that causes the microcontroller to halt the normal flow of the program and execute another program called the **interrupt service routine**

- Interrupts can be thought of as *hardware-initiated subroutine calls*

- Usually, interrupts are generated by I/O devices such as timers or external devices
Interrupts vs Polling

- **Advantages**
  - Immediate response to I/O service request
  - Normal execution continues until it is known that I/O service is needed

- **Disadvantages**
  - Coding complexity for interrupt service routines
  - Extra hardware needed
  - Processor’s interrupt system I/O device must generate an interrupt request
General Hardware Structure for Interrupts

- Interrupts sources can be *external and internal*
- Two types of interrupts: *maskable* and *non-maskable*
  - Maskable can be enabled/disabled by setting/clearing some bits
  - Non-maskable interrupts can not be disabled and they always interrupt the CPU
- Usually, each interrupt has a flag (a bit) that is set whenever the interrupt occurs
The 16F84A Interrupt Structure

- **Sources of interrupts**
  - *External interrupt*
    - The only external interrupt input
    - The input is multiplexed with RB0 pin of port B
    - It is edge triggered
  - *Timer overflow interrupt*
    - It is an internal interrupt that occurs when the 8-bit timer overflows
  - *Port B interrupt change*
    - An interrupt occurs when a change is detected on any of the upper 4 bits of port B
  - *EEPROM write complete interrupt*
The 16F84A Interrupt Structure

- **Interrupt Hardware Structure**

  - Timer Overflow Flag
  - Timer Overflow Interrupt Enable
  - External Interrupt
  - Port B Change
  - EEPROM Write Complete
  - Global Interrupt Enable
  - TOIF
  - TOIE
  - INTF
  - INTE
  - RBIF
  - RBIE
  - EEIF
  - EEIE
  - GIE

  **Wake-up (if in Sleep mode)**
  **Interrupt to CPU**

No non-maskable interrupts in 16F84A
The 16F84A Interrupt Structure

- The INTCON Register

### INTCON REGISTER (ADDRESS 0Bh, 8Bh)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><strong>GIE</strong>: Global Interrupt Enable bit</td>
</tr>
<tr>
<td></td>
<td>1 = Enables all unmasked interrupts</td>
</tr>
<tr>
<td></td>
<td>0 = Disables all interrupts</td>
</tr>
<tr>
<td>6</td>
<td><strong>EEIE</strong>: EE Write Complete Interrupt Enable bit</td>
</tr>
<tr>
<td></td>
<td>1 = Enables the EE Write Complete interrupts</td>
</tr>
<tr>
<td></td>
<td>0 = Disables the EE Write Complete interrupt</td>
</tr>
<tr>
<td>5</td>
<td><strong>T0IE</strong>: TMR0 Overflow Interrupt Enable bit</td>
</tr>
<tr>
<td></td>
<td>1 = Enables the TMR0 interrupt</td>
</tr>
<tr>
<td></td>
<td>0 = Disables the TMR0 interrupt</td>
</tr>
<tr>
<td>4</td>
<td><strong>INTE</strong>: RB0/INT External Interrupt Enable bit</td>
</tr>
<tr>
<td></td>
<td>1 = Enables the RB0/INT external interrupt</td>
</tr>
<tr>
<td></td>
<td>0 = Disables the RB0/INT external interrupt</td>
</tr>
<tr>
<td>3</td>
<td><strong>RBIE</strong>: RB Port Change Interrupt Enable bit</td>
</tr>
<tr>
<td></td>
<td>1 = Enables the RB port change interrupt</td>
</tr>
<tr>
<td></td>
<td>0 = Disables the RB port change interrupt</td>
</tr>
<tr>
<td>2</td>
<td><strong>T0IF</strong>: TMR0 Overflow Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td>1 = TMR0 register has overflowed (must be cleared in software)</td>
</tr>
<tr>
<td></td>
<td>0 = TMR0 register did not overflow</td>
</tr>
<tr>
<td>1</td>
<td><strong>INTF</strong>: RB0/INT External Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td>1 = The RB0/INT external interrupt occurred (must be cleared in software)</td>
</tr>
<tr>
<td></td>
<td>0 = The RB0/INT external interrupt did not occur</td>
</tr>
<tr>
<td>0</td>
<td><strong>RBIF</strong>: RB Port Change Interrupt Flag bit</td>
</tr>
<tr>
<td></td>
<td>1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)</td>
</tr>
<tr>
<td></td>
<td>0 = None of the RB7:RB4 pins have changed state</td>
</tr>
</tbody>
</table>
The 16F84A Interrupt Structure

- The Option Register (81H) – interrupt related bit

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2-0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RBPU</strong>: PORTB Pull-up Enable bit</td>
<td><strong>INTEDG</strong>: Interrupt Edge Select bit</td>
<td><strong>T0CS</strong>: TMR0 Clock Source Select bit</td>
<td><strong>T0SE</strong>: TMR0 Source Edge Select bit</td>
<td><strong>PSA</strong>: Prescaler Assignment bit</td>
<td><strong>PS2:PS0</strong>: Prescaler Rate Select bits</td>
</tr>
<tr>
<td>0 = PORTB pull-ups are enabled by individual port latch values</td>
<td>0 = Interrupt on falling edge of RB0/INT pin</td>
<td>0 = Internal instruction cycle clock (CLKOUT)</td>
<td>0 = Increment on low-to-high transition on RA4/T0CKI pin</td>
<td>0 = Prescaler is assigned to the WDT</td>
<td>Bit Value</td>
</tr>
<tr>
<td>1 = PORTB pull-ups are disabled</td>
<td>1 = Interrupt on rising edge of RB0/INT pin</td>
<td>1 = Transition on RA4/T0CKI pin</td>
<td>1 = Increment on high-to-low transition on RA4/T0CKI pin</td>
<td>1 = Prescaler is assigned to the Timer0 module</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
</tr>
</tbody>
</table>

Select the transition type on input RB0/INT that will cause an interrupt.
The 16F84A Interrupt Structure

- **Interrupt Operation**

```
Interrupt detected
Complete current instruction
Save Program Counter on Stack
Clear GIE
Reload PC with 0004_H
Continue program execution

Instruction is RETFIE?
Yes
Set GIE to 1
Load PC from Stack
Continue program execution

No

Main program is running
ISR execution starts
Main program continues
```
The 16F84A Interrupt Structure

How to use interrupts?

1. Start the interrupt service routine at 0x0004
2. Clear the flag of the used interrupt in the INTCON register (if it is not cleared on reset, e.g. RBIF)
3. Enable the corresponding interrupt by setting its bit in INTCON register
4. Enable global interrupts by setting the GIE bit
5. End the interrupt subroutine with RETFIE instruction to resume program execution
The 16F84A Interrupt Structure

**Example**

Write a PIC16F84 program that continuously adds the content of memory location 0x0A until an external interrupt is observed on RB0. In this case the result is stored in location 0x10 and the working register is cleared. The interrupt should be configured on the arrival of rising edge.
The 16F84A Interrupt Structure

**Example**

```assembly
#include p16F84A.inc ; include the definition file for 16F84A
org 0x0000 ; reset vector
goto START
org 0x0004 ; define the ISR
goto ISR
org 0x0006 ; Program starts here

START
bsf STATUS , RP0 ; select bank 1
bsf INTCON , INTE ; enable external interrupt on RB0/INT
bsf OPTION_REG , INTEDG ; select to interrupt on rising edge
bsf INTCON , GIE ; enable global interrupts
bcf STATUS , RP0 ; select bank 0
movlw 0x00 ; clear W

ADD
addwf 0x0A , 0 ; add the contents of 0x0A to W
goto ADD ; keep adding until an interrupt occurs

ISR
movwf 0x10 ; on interrupt store the accumulated result
clw ; clear working register
bcf INTCON , INTF ; clear the interrupt flag
retfie ; return from the ISR

end
```
Context Saving

• What if the main program is to preserve the W register and interrupt uses it?
  • Save it temporarily in memory at the beginning of the ISR
    MOVWF    TEMP ; push
  • Restore the value at the end of ISR
    MOVF    TEMP, W ; pop

• What if we want to preserve some memory location such as the STATUS register on interrupt?
  • Save it temporarily in memory at the beginning of the ISR
    SWAPF   STATUS,0 ; push
    MOVWF   TEMP
  • Restore the value at the end of ISR
    SWAP   TEMP, 0 ; pop
    MOVWF   STATUS
The 16F84A Interrupt Structure

- **Multiple Interrupts**
  - Note that there is only one interrupt vector for all types of interrupts
  - In other words, regardless of the interrupt type, the microcontroller will start executing from location 0x0004 on any interrupt
  - How to determine the source of interrupt?
  - Check the interrupt flag bits in the INTCON register at the beginning of the interrupt service routine to determine what source of the interrupt!

```Assembly
<table>
<thead>
<tr>
<th>Interrupt_SR</th>
<th>Instruction</th>
<th>Source Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>btfsc intcon,0</td>
<td>test RBIF</td>
</tr>
<tr>
<td></td>
<td>goto portb_int</td>
<td>Port B Change routine</td>
</tr>
<tr>
<td></td>
<td>btfsc intcon,1</td>
<td>test INTF</td>
</tr>
<tr>
<td></td>
<td>goto ext_int</td>
<td>external interrupt routine</td>
</tr>
<tr>
<td></td>
<td>btfsc intcon,2</td>
<td>test T0IF</td>
</tr>
<tr>
<td></td>
<td>goto timer_int</td>
<td>timer overflow routine</td>
</tr>
<tr>
<td></td>
<td>btfsc eecon1,4</td>
<td>test EEPROM write complete flag</td>
</tr>
<tr>
<td></td>
<td>goto eeprom_int</td>
<td>EEPROM write complete routine</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-x</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIE</td>
<td>EEIE</td>
<td>T0IE</td>
<td>INTE</td>
<td>RBIE</td>
<td>T0IF</td>
<td>INTF</td>
<td>RBIF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bit 0</td>
</tr>
</tbody>
</table>

**Legend**
- GIE: Global interrupt enable
- EEIE: EEPROM write enable
- T0IE: Timer 0 interrupt enable
- INTE: INT enable
- RBIE: RB0 interrupt enable
- T0IF: Timer 0 interrupt flag
- INTF: INT interrupt flag
- RBIF: RB0 interrupt flag
Counters and Timers

- Digital counters can be built with flip-flops. They can count up or down, reset, or loaded with initial value.
- When the most significant bit changes from 1 to 0, this indicates an overflow. This signal can be used to interrupt the microcontroller.
- *If the counter operates using a clock with known frequency we can use it as a timer*
Counters and Timers

- **Timer applications**
  
  (a) Measure the time between two events
  
  (b) Measure the time between two pulses
  
  (c) Measure a pulse duration

Use **polling** or **interrupts**
The 16F84A Timer 0 Module

- **8-bit counter**, memory address 0x01
- Configurable counter using the OPTION register (0x81)
- Two sources for the timer clock: *instruction cycle clock (Fosc/4)* or *RA4/T0CKI*
- The programmable prescaler is shared with the Watchdog Timer WDT
- The value of frequency division is determined by PS2, PS1, and PS0 bits in the OPTION register
The 16F84A Timer 0 Module

- The Option Register – Timer related bits

<table>
<thead>
<tr>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
<th>R/W-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBPU</td>
<td>INTEDG</td>
<td>T0CS</td>
<td>T0SE</td>
<td>PSA</td>
<td>PS2</td>
<td>PS1</td>
<td>PS0</td>
</tr>
</tbody>
</table>

**bit 7**
- **RBPU**: PORTB Pull-up Enable bit
  - 1 = PORTB pull-ups are disabled
  - 0 = PORTB pull-ups are enabled by individual port latch values

**bit 6**
- **INTEDG**: Interrupt Edge Select bit
  - 1 = Interrupt on rising edge of RB0/INT pin
  - 0 = Interrupt on falling edge of RB0/INT pin

**bit 5**
- **T0CS**: TMR0 Clock Source Select bit
  - 1 = Transition on RA4/T0CKI pin
  - 0 = Internal instruction cycle clock (CLKOUT)

**bit 4**
- **T0SE**: TMR0 Source Edge Select bit
  - 1 = Increment on high-to-low transition on RA4/T0CKI pin
  - 0 = Increment on low-to-high transition on RA4/T0CKI pin

**bit 3**
- **PSA**: Prescaler Assignment bit
  - 1 = Prescaler is assigned to the WDT
  - 0 = Prescaler is assigned to the Timer0 module

**bit 2-0**
- **PS2:PS0**: Prescaler Rate Select bits

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>TMR0 Rate</th>
<th>WDT Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1:2</td>
<td>1:1</td>
</tr>
<tr>
<td>001</td>
<td>1:4</td>
<td>1:2</td>
</tr>
<tr>
<td>010</td>
<td>1:8</td>
<td>1:4</td>
</tr>
<tr>
<td>011</td>
<td>1:16</td>
<td>1:8</td>
</tr>
<tr>
<td>100</td>
<td>1:32</td>
<td>1:16</td>
</tr>
<tr>
<td>101</td>
<td>1:64</td>
<td>1:32</td>
</tr>
<tr>
<td>110</td>
<td>1:128</td>
<td>1:64</td>
</tr>
<tr>
<td>111</td>
<td>1:256</td>
<td>1:128</td>
</tr>
</tbody>
</table>
The 16F84A Timer 0 Module

- Timer Timing

Note 1: Interrupt flag bit T0IF is sampled here (every Q1).
Note 2: Interrupt latency = 4TCY where TCY = instruction cycle time.
Note 3: CLKOUT is available only in RC oscillator mode.
The 16F84A Timer 0 Module

- **Example:** Write a program that generates a 5 ms delay using the TMR0 module without using interrupts. Assume the clock frequency is 800 KHz.
  - \( F_{osc} = 800 \text{ KHz} \rightarrow \) the timer internal clock = \( F_{osc}/4 = 200 \text{ KHz} \rightarrow \) instruction cycle = 5 us \( \rightarrow \) timer increment every 5 us
  - For these settings, the timer generates an interrupt after 256 * 5 us = 1280 us only ?!
  - How about changing the prescale factor ?
    - 256 x prescale x 5 us = 5 ms \( \rightarrow \) prescale = 3.9 \( \approx \) 4
    - This will generate a delay of 4 x 256 x 5 us = 5.12 ms
  - What if we need more accurate delay !! We can play around with the count value (we don’t have to start from 0 always)
    - \( N \times \text{prescale} \times 5 \text{ us} = 5 \text{ ms} \rightarrow N \times \text{prescale} = 1000 \rightarrow \) we can select the prescale 8 and the count \( N \) to be 125
    - We have to load TMR0 with 256 – 125 = 131 as initial value
The 16F84A Timer 0 Module

- Example – cont’d

```
#include p16f84A.inc

org 0x0000

start . . .

goto start

org 0x0010

start . . .

call delay5

. . .

delay5 movlw D'131' ; preload T0, it overflows after 125 counts

movwf TMR0

bsf STATUS, RP0 ; select memory bank 1

movlw B'00000010' ; set up T0 for internal input, prescale by 8

movwf OPTION_REG

bcf STATUS, RP0 ; select bank 0

del1 btfss intcon,T0IF ; test for Timer Overflow flag

goto del1 ; loop if not set (no timer overflow)

bcf intcon,T0IF ; clear Timer Overflow flag

return
```
Watchdog Timer

- Special timer internal to the microcontroller that is continually counting up
- Can be used to reset the Microcontroller if a program fails or gets stuck
- If enabled and it overflows, the microcontroller is reset
- Properties
  - The WDT timer is enabled/disabled by the WDTE bit in the configuration word
  - It has its own internal RC oscillator
  - The nominal time-out period is 18 ms
  - It can be extended through the prescaler bits in the OPTION register (up to 128x18 ms= 2.3 sec)
  - The WDT timer can be cleared by software using the CLRWDT instruction

- How does the watchdog timer know if the program is stuck???!!!
Sleep Mode

- An important way to save power!
- The microcontroller can be put in sleep mode by using the `SLEEP` instruction
- Once in sleep mode, the microcontroller operation is almost suspended
  - The oscillator is switched off
  - The WDT is cleared. If the WDT is enabled, it continues running
  - Program execution is suspended
  - All ports retain their current settings
  - $PD$ and $TO$ bits are cleared and set respectively
  - Power consumption falls to a negligible amount
- To exit the sleep mode
  - Interrupt occurs (even if GIE = 0)
  - WDT wake-up
  - External reset the MCLR pin

Program continues execution from PC+1
MC is reset !
Summary

- Microcontrollers can deal with time by using timers and interrupts

- Interrupts saves the microcontrollers computational power as they require its attention when they occur only

- Most interrupts are configurable

- Hardware timer can be used as a counter or a timer and it is very useful in measuring time