Microprocessors and Microcontrollers

Interrupts

EE3954

by
Maarten Uijt de Haag, Tim Bambeck
Interrupts

Definitions

• Polling and interrupts are used when interfacing and communicating with internal and external devices and programs

• **Polling:**
  - Internal or external devices are checked on a regular basis to find out if they need service; if they require a particular function to be executed.

• **Interrupts:**
  - An event from either an internal or external source where a processor will suspend the execution of its current process (program) and switch to a different instruction sequence.
Polling

If the switch is closed, then turn the LED on!

- **PIC16F877**
  - **VDD**
  - **VSS**
- **RB4**
- **RB0**

**Flowchart:**

1. **Polling the Button**
2. **Set up the I/O pins**
3. **Switch Closed?**
   - **Y**
     - **Turn LED on**
   - **N**
     - **Continue polling**
PORTB   equ  0x06      ; in Bank 0 & 2
TRISB   equ  0x06      ; in Bank 1 & 3
STATUS equ  0x03      ; in all Banks
RP0    equ  d’ 5’
RP1    equ  d’ 6’

org    0x000      ; program starts at address 0
nop

INIT:  bcf    STATUS, RP1  ; access Bank 1…
       bsf    STATUS, RP0  ; … so we can get to TRIS Regs.
       bsf    TRISB, 0    ; set pin 0 of PORTB to input
       bcf    TRISB, 4    ; set pin 4 of PORTB to output
       bcf    STATUS, RP0  ; access bank 0
       bcf    PORTB, 4    ; initially, turn the LED off

POLL:  btfss  PORTB, 0    ; check if PORTB pin 0 is ‘0’ or ‘1’
       goto   LED_ON     ; goto TURN LED on
       goto   POLL      ; go back and keep polling

LED_OFF  bcf    PORTB, 4    ; Turn LED off, (set RB4= ‘0’) 
         goto   POLL      ; go back and keep polling

LED_ON:  bsf    PORTB, 4    ; Turn LED on, (set RB4= ‘1’) 
         goto   POLL      ; go back and keep polling

end
Interrupt Usage of a System

“Push Button” Interrupt

“Timer Interrupts”

“Serial Interrupt”

= Main Program Loop
Interrupts

If the switch is closed, Then turn the LED on!

If the switch is closed
Set up I/O pins
Set up Interrupts
Process A

If the switch is closed
Set up I/O pins

Interrupt Service Routine
Set up I/O pins
Switch Closed?

N
Turn LED off
Go Back to Process A (RETFIE)

Y
Turn LED on
Interrupts
Transfer of Control

Program control is transferred to the first instruction of the interrupt service routine (ISR)

Return of the program control from the end of the ISR to the next (N+1) instruction of the main program
PIC Microcontroller

- Analog-to-Digital Converter (ADC),
- Timers and counters (8-bit, 16-bit),
- Parallel port,
- Serial communication,
  - RS232 (UART), I2C, SSP
- Comparator,
- Pulse-Width-Modulation (PWM)
- External signal or interrupt

Microcontroller has various on-chip peripheral devices that can interrupt the main program.
Interrupts

Multiplexing of I/O Pins

ADC
- Signal: on-change

Parallel port
- Signal: edge

Comparator

PWM

Pin Diagram from Datasheet
Example:

On-change interrupt (RB4, RB5, RB6, RB7):

If the input on the pin is changed ("0" => "1" or "1" => "0") an interrupt will occur
Interrupt

So what happens when an interrupt occurs

In other words, the current value of the PC is stored on the STACK and 0x0004 is put into PC: next instruction is at 0x0004
Interrupt

So what happens when an interrupt occurs

Remember from the Labs

Interrupt Vector

org 0x000

goto INIT

org 0x004

goto ISR

INIT:

...  

MAIN:

...

ISR:

...  

...  

retfie

Interrupt Service Routine (ISR)
Works just like a subroutine

instead of ‘return’
Interrupt
Returning from an ISR

Situation 4 - RETURN, RETFIE, or RETLW Instruction

Put the value on top of the STACK back into the Program Counter (PC); in other words, return to whatever the main program was doing.
Interrupt Service Routine

Example

Suppose that we want to write an ISR that XORs the values stored in memory locations 0x20 through 0x3f and write the result to PORTE.

```
org 0x000
goto INIT
org 0x004
goto SUMISR

INIT:
...

MAIN:
...
goto MAIN
...

SUMISR:
movlw 0x20
movwf FSR
movlw 0x00

NXT:
xorwf INDF,W
incf FSR,F
btfss FSR,6
goto NXT
movwf PORTE
retfie
```
Interrupts

- INT Pin Interrupt (external interrupt-RB0)
- Timer overflow interrupt,
- Comparator change interrupt,
- PORTB change interrupt (RB4-RB7),
- Receive (communication) interrupt,
- Transmit (comm.) interrupt,
- A/D conversion complete interrupt,
- Parallel slave port interrupt,
- Etc.
Interrupts

- **INTCON** (address 0x0B, 0x8B, 0x10B, 0x18B)
  - Control and Status of interrupts

```
<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>
</tr>
</thead>
<tbody>
<tr>
<td>GIE</td>
<td>PEIE</td>
<td>T0IE</td>
<td>INTE</td>
<td>RBIE</td>
<td>T0IF</td>
<td>INTF</td>
<td>RBIF</td>
</tr>
</tbody>
</table>
```

**Legend:**
- Ending with an **E** indicates an enable bit: if the bit is set, the corresponding interrupt is enabled.
- Ending with an **F** indicates a flag: if the bit is set, the corresponding interrupt has occurred.
## Register 8-1: INTCON Register

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GIE</td>
<td>PEIE (3)</td>
<td>T0IE</td>
<td>INTE (2)</td>
<td>RBIE (1, 2)</td>
<td>T0IF</td>
<td>INTF (2)</td>
<td>RBIF (1, 2)</td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GIE**: Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

**PEIE**: Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

**T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

**INTE**: INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

**RBIE (1)**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

**T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

**INTF**: INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

**RBIF (1)**: 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
Interrupts

Control Registers

Used for: Configuration for Peripherals

- **PIE1 & PIE2** (address 0x8C and 0x8D respectively)
  - Contain bits to enable/disable peripheral interrupts for use

- **PIR1 & PIR2** (address 0x0C and 0x0D respectively)
  - Contain bits to identify which interrupt occurs (flags):
  - Corresponding bits are set when the interrupt occurred;
### PIE & PIR

from PIC16F877 datasheet

<table>
<thead>
<tr>
<th>8Bh(3)</th>
<th>INTCON</th>
<th>GIE</th>
<th>PEIE</th>
<th>T0IE</th>
<th>INTE</th>
<th>RBIE</th>
<th>T0IF</th>
<th>INTF</th>
<th>RBIF</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8Ch</td>
<td>PIE1</td>
<td>PSpIE(2)</td>
<td>ADIE</td>
<td>RCIE</td>
<td>TXIE</td>
<td>SSpIE</td>
<td>CCP1IE</td>
<td>TMR2IE</td>
<td>TMR1IE</td>
<td>0000 0000</td>
</tr>
<tr>
<td>8Dh</td>
<td>PIE2</td>
<td>—</td>
<td>(5)</td>
<td>—</td>
<td>EEIE</td>
<td>BCLIE</td>
<td>—</td>
<td>—</td>
<td>CCP2IE</td>
<td>-r-0 0--0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0Bh(3)</th>
<th>INTCON</th>
<th>GIE</th>
<th>PEIE</th>
<th>T0IE</th>
<th>INTE</th>
<th>RBIE</th>
<th>T0IF</th>
<th>INTF</th>
<th>RBIF</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0Ch</td>
<td>PIR1</td>
<td>PSpIF(3)</td>
<td>ADIF</td>
<td>RCIF</td>
<td>TXIF</td>
<td>SSpIF</td>
<td>CCP1IF</td>
<td>TMR2IF</td>
<td>TMR1IF</td>
<td>0000 0000</td>
</tr>
<tr>
<td>0Dh</td>
<td>PIR2</td>
<td>—</td>
<td>(5)</td>
<td>—</td>
<td>EEIF</td>
<td>BCLIF</td>
<td>—</td>
<td>—</td>
<td>CCP2IF</td>
<td>-r-0 0--0</td>
</tr>
</tbody>
</table>

---

*From page 15 and 16 of PIC16F877 datasheet (DS)*
Interrupts.

For PIC16F877
(See section 12 of DS)
In General for Mid-range devices

Note 1: This shows all current Interrupt bits (at time of manual printing) for all PICmicro Mid-Range MCUs. Which bits pertain to a specific device is dependent upon the device type and peripherals implemented. See specific device data sheet.

2: Some of the original Mid-Range devices had only one peripheral module. These devices do not have the PEIE bit, and have the module enable bit in the INTCON register.
Let’s Look at an Example

• Use the PORTB on-change interrupt to turn on the LED.
Step 1

Set up the interrupt in main program

INTCON Register (0x0B in all Banks):

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>GIE</th>
<th>PEIE</th>
<th>T0IE</th>
<th>INTE</th>
<th>RBIE (1, 2)</th>
<th>T0IF</th>
<th>INTF</th>
<th>RBIF (1, 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
<td>R/W-0</td>
</tr>
</tbody>
</table>

Diagram showing the connections between various interrupt flags and the wake-up mechanism for the CPU.
Step 1
Set up the interrupt in main program

INTCON equ 0x0B
TRISB equ 0x06
PORTB equ 0x06

org 0x000
goto INIT
org 0x004
goto ISR

INIT:
clrf STATUS ; access
bsf STATUS,5 ; bank 1
bsf TRISB, 4 ; pin 4 of PORTB: input
bcf TRISB, 1 ; pin 1 of PORTB: output
bcf STATUS,5 ; access bank 0
bsf INTCON,3 ; enable PORTB on-change interrupt
bcf INTCON,0 ; YOU MUST CLEAR THE FLAG

... (see next slide)
Step 2
Write the ISR

MAIN:

... goto MAIN

ISR:

btfss PORTB, 4 ; check if pin 4 changed from “0” to “1”
goto LED_ON ; if not so, turn on LED (“1” to “0”)

LED_OFF:

bcf PORTB, 1 ; otherwise, turn off LED
bcf INTCON,0 ; YOU MUST CLEAR THE FLAG
retfie ; return to whatever you were doing

LED_ON:

bsf PORTB, 1 ; turn on LED
bcf INTCON,0 ; YOU MUST CLEAR THE FLAG
retfie ; return to whatever you were doing

The flag is the only indication for the microcontroller that the interrupt occurred, if you do not clear it in the ISR the interrupt condition will remain!!!
Step 3

Enable the Global interrupt in **main program**

```
INTCON    equ  0x0B
TRISB     equ  0x06
PORTB     equ  0x06

org 0x000
goto INIT
org 0x004
goto ISR

INIT:
  clrf STATUS ; access
  bsf STATUS,5 ; bank 1
  bsf TRISB, 4 ; pin 4 of PORTB: input
  bcf TRISB, 0 ; pin 0 of PORTB: output
  bcf STATUS,5 ; access bank 0
  bsf INTCON,3 ; enable PORTB on-change interrupt
  bcf INTCON,0 ; YOU MUST CLEAR THE FLAG

MAIN:
  bsf INTCON,7 ; enable all interrupts

  ...
  ...
```
Interrupt Latency

Time from interrupt event (the interrupt flag bit gets set) to the 
time that the instruction at address 0x004 starts execution

Synchronous (typical internal) events: \( \text{latency is } 3T_{\text{CY}} \)

Asynchronous (typical external) events: \( \text{latency is } 3T_{\text{CY}} \text{ to } 3.75T_{\text{CY}} \)

with the instruction cycle time

In general:
see the individual sections of the peripheral devices for the exact latency
Interrupt Latency

Figure 8-2: INT Pin and Other External Interrupt Timing
What can go wrong with the combination of the following main program and ISR?

MAIN:
```
movf TEMP1, W
addlw 0x23
movwf TEMP2
...
...
goto MAIN
```

ISR:
```
movlw 0x20
movwf ISR_TMP,F
retfie
```
Oops ... Again

What can go wrong with the combination of the following main program and ISR?

MAIN:

```
movf  TEMP1, W
sublw 0x23
btfss STATUS,Z
...  
...  
goto MAIN
```

ISR:

```
movlw 0x20
addwf ISR_TMP,F
retn  
```

suppose that the interrupt occurs here

Instruction alters the ‘Z’ bit
Context Saving
During Interrupts

• Save the CONTEXT:
  - at least W and STATUS,
  - but also all Registers you plan to use in the ISR that are used in the Main Program.

• Save CONTEXT in RAM Common Area
  - 0x71 - 0x7F (recall 0x70 used by ICD2)
Common RAM Area

Advantage of this common area is that you DO NOT have to change banks.

Remember this Data Memory area from Lab #1
Context Saving

Why Not:

ISR:

`movwf W_TEMP ; DOES NOT affect STATUS`
`movf STATUS,W ; DOES affect STATUS`
`movwf STATUS_TEMP ; DOES NOT affect STATUS`
...
...
...
`movf STATUS_TEMP,W ; DOES affect STATUS`
`movwf STATUS ; DOES NOT affect STATUS`
`movf W_TEMP,W ; DOES affect STATUS`

So, while you are saving ‘W’ and ‘STATUS’ you actually changing the contents of STATUS. You do not want this to happen so you use SWAPF which does not affect any flags in STATUS

Example: suppose STATUS = 0x00?
## Context Saving

### Why Not:

<table>
<thead>
<tr>
<th>Label</th>
<th>Instr.</th>
<th>Argument</th>
<th>Contents After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>ISR:</td>
<td>movwf</td>
<td>W_TEMP</td>
<td>0x12</td>
</tr>
<tr>
<td></td>
<td>movf</td>
<td>STATUS,W</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>STATUS_TEMP</td>
<td>0x00</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>movf</td>
<td>STATUS_TEMP, W</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>STATUS</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>movf</td>
<td>W_TEMP,W</td>
<td>0x12</td>
</tr>
</tbody>
</table>

Before this program segment: \( W = 0x12 \) and \( STATUS = 0x00 \)
## Context Saving

### Why Not:

<table>
<thead>
<tr>
<th>Label</th>
<th>Instr.</th>
<th>Argument</th>
<th>W</th>
<th>STATUS</th>
<th>W_TEMP</th>
<th>STATUS_TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISR:</td>
<td>movwf</td>
<td>W_TEMP</td>
<td>0x00</td>
<td>0x12</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>movf</td>
<td>STATUS,W</td>
<td>0x12</td>
<td>0x12</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>STATUS_TEMP</td>
<td>0x12</td>
<td>0x12</td>
<td>0x00</td>
<td>0x12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>movf</td>
<td>STATUS_TEMP,W</td>
<td></td>
<td>0x12</td>
<td>0x??</td>
<td>0x00</td>
<td>0x12</td>
</tr>
<tr>
<td>movwf</td>
<td>STATUS</td>
<td></td>
<td>0x12</td>
<td>0x12</td>
<td>0x00</td>
<td>0x12</td>
</tr>
<tr>
<td>movf</td>
<td>W_TEMP,W</td>
<td></td>
<td>0x00</td>
<td>0x16</td>
<td>0x00</td>
<td>0x12</td>
</tr>
</tbody>
</table>

Before this program segment: $W = 0x00$ and $STATUS = 0x12$
So ...

What instruction moves to/from register without affecting any flags?

\textbf{swapf}

Problem is that this instruction swaps the nibbles of the byte

\textbf{Solution:}
Just swap them back
## Context Saving

<table>
<thead>
<tr>
<th>Label</th>
<th>Instr.</th>
<th>Argument</th>
<th>Contents After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>movwf</td>
<td>W_TEMP</td>
<td>W: 0x00 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: -</td>
</tr>
<tr>
<td>ISR:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>swapf</td>
<td>STATUS,W</td>
<td>W: 0x12 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: -</td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>STATUS_TEMP</td>
<td>W: 0x12 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: 0x21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>swapf</td>
<td>STATUS_TEMP, W</td>
<td>W: 0x12 STATUS: - W_TEMP: 0x00 STATUS_TEMP: 0x21</td>
</tr>
<tr>
<td></td>
<td>movwf</td>
<td>STATUS</td>
<td>W: 0x12 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: 0x21</td>
</tr>
<tr>
<td></td>
<td>swapf</td>
<td>W_TEMP,F</td>
<td>W: 0x12 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: 0x21</td>
</tr>
<tr>
<td></td>
<td>swapf</td>
<td>W_TEMP,W</td>
<td>W: 0x00 STATUS: 0x12 W_TEMP: 0x00 STATUS_TEMP: 0x21</td>
</tr>
</tbody>
</table>

Before this program segment: \( W = 0x00 \) and \( STATUS = 0x12 \)
Context Saving During Interrupts

ISR:

```
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

... 

... 

... 

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
```
Multiple Interrupts

What to do when the Microcontroller has to deal with interrupts for multiple sources, e.g. timer and PORTB change interrupt?

Answer:

Check for the flags
Multiple Interrupts

Example: suppose we could get an interrupt for
1) a PORTB pin level change,
2) a TIMER 0 overflow (0xFF -> 0x00)

Check:
Multiple Interrupts

```
org 0x0000
goto INIT
org 0x0004
btfsc INTCON, RBIF
goto RB_ISR
btfsc INTCON, TOIF
goto TO_ISR
retfie

INIT: ...
...
MAIN: ...
...
RB_ISR: ...
...
... retfie
TO_ISR: ...
...
retfie
```
Multiple Interrupts

Or

org 0x0000
goto INIT
org 0x0004
btfsc INTCON, RBIF
goto RB_ISR
goto TO_ISR

INIT:
  ...

MAIN:
  ...

RB_ISR:
  ...

TO_ISR:
  ...
  retfie
  ...
  retfie
Multiple Interrupts

Or

```asm
org 0x0000
goto INIT
org 0x0004
goto ISR

INIT:
...

MAIN:
...

ISR:
btfsc INTCON, RBIF
goto RB_ISR

T0_ISR:
...

RB_ISR:
...

retfie
```
Example in the Timer Section