Basic Concepts in Interrupts

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
  - The MPU acknowledges the request
  - Attends to the request
  - Goes back to processing where it was interrupted
Types of Interrupts

- **Hardware interrupts**
  - Maskable: can be masked or disabled
  - Two groups: external and internal interrupts
    - External through designated I/O pins
    - Internal by Timers, A/D, etc.
  - Non-maskable: cannot be disabled

- **Software interrupts**: generally used when the situation requires **stop** processing and start all over
  - Examples: divide by zero or stack overflow
  - Generally, microcontrollers do **not** include software interrupts
When the interrupt process is enabled, the MPU, during execution, checks the interrupt request flag just before the end of each instruction.

If the interrupt request is present, the MPU:
- **Completes** the execution of the instruction
- **Resets** the interrupt flag
- **Saves** the address of the program counter on the stack
  - Some interrupt processes also save contents of MPU registers on the stack.
- **Stops** the execution
To restart the execution, the MPU needs to be redirected to the memory location where the interrupt request can be met. Accomplished by interrupt vectors.

The set of instructions written to meet the request (or to accomplish the task) is called an interrupt service routine (ISR).

Once the request is accomplished, the MPU should find its way back to the instruction, next memory location where it was interrupted. Accomplished by a specific return instruction.

Main code
....
Setup interrupt vectors
....
HERE: GOTO HERE

ORG 0x100
INT1_ISR: ISR code
....
RETFIE

END
Interrupt Vectors

- Direct the MPU to the location where the interrupt request is accomplished.
- They are:
  - Defined memory location where a specific memory location/s is assigned to the interrupt request.
  - Defined vector location where specific memory locations assigned to store the vector addresses of the ISRs.
  - Specified by external hardware: The interrupt vector address (or a part of it) is provided through external hardware using an interrupt acknowledge signal.
Interrupt Service Routine (ISR)

- A group of instructions that accomplishes the task requested by the interrupting source
- Similar to a subroutine except that the ISR must be terminated in a Return instruction specially designed for interrupts
  - The Return instruction, when executed, finds the return address on the stack and redirects the program execution where the program was interrupted.
  - Some Return instructions are designed to retrieve the contents of MPU registers if saved as a part of the interrupts.
  - RETFIE FAST (1/0)
Interrupt Priorities

- **Rationale for priorities**
  - Multiple interrupt sources exist in a system, and more than one interrupt requests can arrive simultaneously.
  - Example: A/D and Timer0
  - When one request is being served (meaning when the MPU is executing an ISR), another request can arrive.
  - → the interrupt requests must be prioritized.
  - Most MCUs (and MPUs) include an interrupt priority scheme. Some are based on hardware and some use software.

\[
\begin{array}{c|c|c}
  \text{IIN1} & \text{INT1_ISR} & \text{INT1 has higher priority than INT2} \\
  \text{INT2} & & \\
\end{array}
\]
Reset as a Special Purpose Interrupt

- **Reset** is an external signal that enables the processor to begin execution or interrupts the processor if the processor is executing instructions.

- There are at least two types of resets in microcontroller-based systems.
  - Power-on reset and manual reset

- When the reset signal is activated, it establishes or reestablishes the initial conditions of the processor and directs the processor to a specific starting memory location.
PIC18 Interrupts

- PIC18 Microcontroller family
  - Has multiple sources that can send interrupt requests
  - Does not have any non-maskable or software interrupts; all interrupts are maskable (can be disabled)
  - Has a priority scheme divided into two groups
    - High priority and low priority
  - Uses many Special Function Registers (SFRs) to implement the interrupt process
PIC18 Interrupt Sources

- Divided into two groups
  - External sources and internal peripheral sources on the MCU chip
  - External sources
    - Three pins of PORTB - RB0/INT0, RB1/INT1, and RB2/INT2 (edge driven)
    - Change in logic levels of pins RB4-RB7 of PORTB can be recognized as interrupts
  - Internal sources
    - Use SFRs to setup the interrupt process...
PIC18 Interrupt Sources

- Internal peripheral sources
  - Examples: Timers, A/D Converter, Serial I/O, and Low-Voltage Detection Module

- SFRs
  - Used to setup the interrupt process:
    - RCON: Register Control (global priority)
    - INTCON: Interrupt Control
    - INTCON2: Interrupt Control2
    - INTCON3: Interrupt Control3
    - PIR1 and PIR2: Peripheral Interrupt Register 1 & 2
    - PIE1 and PIE2: Peripheral Interrupt Enable 1 & 2
    - IPR1 and IPR2: Interrupt Priority Register 1 & 2

Click here: Summery of Interrupt Registers
PIC18 Interrupt Sources

- To recognize the occurrence of an interrupt request, the MPU needs to check the following **three bits**:
  - The **flag** bit to indicate that an interrupt request is present
  - The **enable** bit to redirect the program execution to the interrupt vector address
  - The **priority** bit (if set) to select priority
In PIC interrupt are controlled by three bits in three different registers.

- The **IE** bit is the interrupt **enable** bit used to enable the interrupt.
- The **IP** bit is the interrupt **priority** bit which selects the **priority** (high or low).
- The **IF** bit is the interrupt **flag** that indicates the interrupt has occurs. This bit **must be cleared** in the interrupt service function or no future interrupt will ever take effect.
PIR (flag)

1 = Enable
0 = Disable

IPR (priority)
1 = High Prio
0 = Low Prio

PIE (peripheral)
1 = Enable
0 = Disable
Interrupt Priorities and RCON Register (1 of 2)

- Any interrupt can be set up as high-priority or low-priority.
  - All high-priority interrupts are directed to the interrupt vector location **000008H**.
  - All low-priority interrupts are directed to the interrupt vector location **000018H**.
  - A high-priority interrupt can interrupt a low-priority interrupt in progress.
The interrupt priority feature is enabled by Bit7 (IPEN) in RCON register.
External Interrupts and INTCON Registers (1 of 3)

- Three registers with interrupt bit specifications primarily for external interrupt sources. INTCON (3)
Example

- Write an instruction to setup INT1 as the high priority interrupt. (INT1 → RB1)

High Priority Interrupt Vector:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0008</td>
<td>EF80</td>
<td>GOTO 0x100</td>
</tr>
<tr>
<td>6</td>
<td>000A</td>
<td>F000</td>
<td>NOP</td>
</tr>
<tr>
<td>7</td>
<td>000C</td>
<td>8E6D</td>
<td>BSF Cxf0, 0x7, ACCESS</td>
</tr>
<tr>
<td>8</td>
<td>000E</td>
<td>8FF2</td>
<td>BSF Cxff2, 0x7, ACCESS</td>
</tr>
<tr>
<td>9</td>
<td>0010</td>
<td>9001</td>
<td>BCF Cxff1, 0x5, ACCESS</td>
</tr>
<tr>
<td>10</td>
<td>0012</td>
<td>8CFD</td>
<td>BSF Cxff0, 0x6, ACCESS</td>
</tr>
<tr>
<td>11</td>
<td>0014</td>
<td>86F0</td>
<td>BSF Cxff0, 0x3, ACCESS</td>
</tr>
<tr>
<td>12</td>
<td>0016</td>
<td>0E0A</td>
<td>MOVLW 0xa</td>
</tr>
<tr>
<td>13</td>
<td>0018</td>
<td>6E01</td>
<td>MOVLW 0x1, ACCESS</td>
</tr>
<tr>
<td>14</td>
<td>001A</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
</tbody>
</table>

ISR:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
<td>0100</td>
<td>0601</td>
<td>DECF 0x1, F, ACCESS</td>
</tr>
<tr>
<td>130</td>
<td>0102</td>
<td>E102</td>
<td>BNI 0x108</td>
</tr>
<tr>
<td>131</td>
<td>0104</td>
<td>0E0A</td>
<td>MOVLW 0xa</td>
</tr>
<tr>
<td>132</td>
<td>0106</td>
<td>6E01</td>
<td>MOVLW 0x1, ACCESS</td>
</tr>
<tr>
<td>133</td>
<td>0108</td>
<td>0011</td>
<td>RETFIE 0x1</td>
</tr>
<tr>
<td>134</td>
<td>010A</td>
<td>FFFF</td>
<td>NOP</td>
</tr>
</tbody>
</table>
Example

Software Setting

If ZERO no priority (early versions)
If enabled → high priority → gates all H/L interrupts to the CPU (remember: GIEL enables all low-priorities Interrupts to the CPU)

If 1 → rising edge (RB1)
If 1 → High Priority

When INT1 goes from high to low value of REG1 decrements!

NOTE: INTCON3: FLAG bit is read only!

Convert to C code!
## Interrupt Setting (INT0 high priority only)

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority Bit</th>
<th>Local Enable Bit</th>
<th>Local Flag Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT0 external interrupt</td>
<td>*</td>
<td>INTCON,INT0IE</td>
<td>INTCON,INT0IF</td>
</tr>
<tr>
<td>INT1 external interrupt</td>
<td>INTCON3,INT1IP</td>
<td>INTCON3,INT1IE</td>
<td>INTCON,INT1IF</td>
</tr>
<tr>
<td>INT2 external interrupt</td>
<td>INTCON3,INT2IP</td>
<td>INTCON3,INT2IE</td>
<td>INTCON,INT2IF</td>
</tr>
<tr>
<td>RB port change interrupt</td>
<td>INTCON2,RBIP</td>
<td>INTCON,RBIE</td>
<td>INTCON,RBIF</td>
</tr>
<tr>
<td>TMR0 overflow interrupt</td>
<td>INTCON2,TMR0IP</td>
<td>INTCON,TMR0IE</td>
<td>INTCON,TMR0IF</td>
</tr>
<tr>
<td>TMR1 overflow interrupt</td>
<td>IPR1,TMR1IP</td>
<td>PIE1,TMR1IE</td>
<td>PIR1,TMR1IF</td>
</tr>
<tr>
<td>TMR3 overflow interrupt</td>
<td>IPR2,TMR3IP</td>
<td>PIE2,TMR3IE</td>
<td>PIR2,TMR3IF</td>
</tr>
<tr>
<td>TMR2 to match PR2 int.</td>
<td>IPR1,TMR2IP</td>
<td>PIE1,TMR2IE</td>
<td>PIR1,TMR2IF</td>
</tr>
<tr>
<td>CCP1 interrupt</td>
<td>IPR1,CCP1IP</td>
<td>PIE1,CCP1IE</td>
<td>PIR1,CCP1IF</td>
</tr>
<tr>
<td>CCP2 interrupt</td>
<td>IPR2,CCP2IP</td>
<td>PIE2,CCP2IE</td>
<td>PIR2,CCP2IF</td>
</tr>
<tr>
<td>A/D converter interrupt</td>
<td>IPR1,ADIP</td>
<td>PIE1,ADIE</td>
<td>PIR1,ADIF</td>
</tr>
<tr>
<td>USART receive interrupt</td>
<td>IPR1,RCIP</td>
<td>PIE1,RCIE</td>
<td>PIR1,RCIF</td>
</tr>
<tr>
<td>USART transmit interrupt</td>
<td>IPR1,TXIP</td>
<td>PIE1,TFIE</td>
<td>PIR1,TFIF</td>
</tr>
<tr>
<td>Sync. serial port int.</td>
<td>IPR1,SSIP</td>
<td>PIE1,SSIE</td>
<td>PIR1,SSIF</td>
</tr>
<tr>
<td>Parallel slave port int.</td>
<td>IPR1,PSIP</td>
<td>PIE1,PSIE</td>
<td>PIR1,PSIF</td>
</tr>
<tr>
<td>Low-voltage detect int.</td>
<td>IPR2,LVDIP</td>
<td>PIE2,LVDIE</td>
<td>PIR2,LVDIF</td>
</tr>
<tr>
<td>Bus-collision interrupt</td>
<td>IPR2,BCLIP</td>
<td>PIE2,BCLIE</td>
<td>PIR2,BCLIF</td>
</tr>
</tbody>
</table>
C Code Example

When a pulse is generated on INT0 the high priority interrupt is generated!

```
ADCON1 = 0x0F; // make ports pins digital
TRISB = 1; // make RB0 input
RCONbits.IPEN = 1; // IPEN = 1
INTCON2bits.INTEDG0 = 0; // make INT0 negative edge triggered
INTCONbits.INT0IE = 1; // enable INT0
INTCONbits.GIEH = 1; // enable high priority interrupts

// INT0 is now armed and active
```
C Code Example – Burglar Alarm Circuit
C Code Example – Burglar Alarm Code

#include <p18cxxx.h>

/* Set configuration bits
 * - set RC oscillator
 * - enable watchdog timer
 * - disable low voltage programming
 * - disable brownout reset
 * - enable master clear
 */
#pragma config OSC = RC
#pragma config WDT = ON
#pragma config WDTPS = 4
#pragma config LVP = OFF
#pragma config BOR = OFF
#pragma config MCLRE = ON

void MyHighInt(void);
void MyLowInt(void);
#pragma interrupt MyHighInt
#pragma code high_vector=0x08
void high_vector(void)
{
    _asm GOTO MyHighInt _endasm
}
#pragma interrupt low MyLowInt
#pragma code low_vector=0x18
// low_vector is the vector at 0x18
void low_vector(void)
{
    _asm GOTO MyLowInt _endasm
}

void MyHighInt(void);
void MyLowInt(void);

// main program
void main (void)
{
    ADCON1 = 0x0F; // make ports pins digital
    TRISB = 0x24; // make RB2 and RB5 inputs
    // make RB4 and output
    PORTB = 0x00; // alarm off
    INTCON2bits.RBPU = 1; // Port B pullups on
    RCONbits.IPEN = 1; // IPEN = 1
    INTCON2bits.INTEDG2 = 1; // make INT2 positive edge-trig
    INTCON3bits.INT2IP = 0; // make INT2 low priority
    INTCON3bits.INT2IE = 1; // enable INT2
    INTCONbits.GIEH = 1; // enable high priority interrupts
    INTCONbits.GIEL = 1; // enable low priority interrupts
    while( 1 ) // main program loop
    {
        ClrWdt(); // pet spot (woof/pant)
        if ( PORTBbits.RB5 == 0 ) // pushbutton pressed
            PORTBbits.RB4 = 1; // turn alarm ON
        PORTBbits.RB4 = 0; // alarm off
    }
}

Do it in the LAB with LEDs and SWs
Another practical Example

Suppose you are given a circuit as shown below. Write a main program and an INT0 interrupt service routine in assembly language. The main program initializes a counter to 0, enables the INTO interrupt, and then stays in a while-loop to wait forever. The INT0 interrupts service routine simply increments the counter by 1 and outputs it to the LEDs. Whenever is incremented to 15, the service routine resets it to 0. Choose appropriate component that the PIC18 receives an INTO interrupt roughly every second.
Handling Multiple Interrupt Sources

- In PIC18 MCU, all interrupt requests are directed to one of two memory locations:
  - 000008H (high-priority) or 000018 (low-priority)
- When multiple requests are directed to these locations, the interrupt source must be identified by checking the interrupt flag through software instructions.
Example

BSF INTCON, GIEL ;Enable global low-priority - INTCON,6>
BCF IPR1, TMR1IP ;Set Timer1 as low-priority
BSF PIE1, TMR1IE ;Enable Timer1 overflow interrupt
BCF IPR1, TMR2IP ;Set Timer2 as low-priority
BSF PIE1, TMR2IE ;Enable Timer2 match interrupt

BCF PIR1, TMR1IF ;Clear TMR1 flag
CALL TMR1L ;Call service subroutine

BCF PIR1, TMR2IF ;Clear TMR2 flag
CALL TMR2 ;Call service subroutine
### Example of using multiple interrupts

**INT1=High Priority / TMR1 and TMR2 Low Priority**

<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the High Priority Interrupt Vector</td>
</tr>
<tr>
<td>Assign the Low Priority Interrupt Vector</td>
</tr>
<tr>
<td>Setup the interrupt registers for external interrupt</td>
</tr>
<tr>
<td>Setup the interrupt registers for internal interrupts</td>
</tr>
</tbody>
</table>
Example of using multiple interrupts
INT1=High Priority / TMR1 and TMR2 Low Priority

```
ORG 0x00
GOTO MAIN

ORG 0x0008
INTCK:
GOTO INT1_ISR

ORG 0x00018
TIMERCK:
BTSC PIR1, TMR1IF
GOTO TMR1_ISR
BTSC PIR1, TMR2IF
GOTO TMR2_ISR

MAIN:
```

```
BSF RCON, IPEN
BSF INTCON, GIEH
BSF INTCON2, INTEGD1
BSF INTCON3, INT1IP
BSF INTCON3, INT1IE
BSF INTCON, GIEL
BCF IPR1, TMRIIP
BSF PIR1, TMRIIE
BCF IPR1, TMR2IP
BSF PIR1, TMR2IE

1 0000  EF12  GOTO 0x24
2 0002  F000  NOP
3 0004  FFFF  NOP
4 0006  FFFF  NOP
5 0008  EF00  GOTO 0x100
6 000A  F000  NOP
7 000C  FFFF  NOP
8 000E  FFFF  NOP
9 0010  FFFF  NOP
10 0012  FFFF  NOP
11 0014  FFFF  NOP
12 0016  FFFF  NOP
13 0018  B09E  BTSC 0xf9e, 0, ACCESS
14 001A  EF87  GOTO 0x10e
15 001C  F000  NOP
16 001E  B29E  BTSC 0xf9e, 0x1, ACCESS
17 0020  EF91  GOTO 0x122
18 0022  F800  NOP

MAIN
19 0E3F  MOVLW 0x3f
20 6E93  MOVF 0xf93, ACCESS
21 0028  SED0  BSF 0xfdo, 0x7, ACCESS
22 002A  SEF2  BSF 0xff2, 0x7, ACCESS
23 002C  S&F1  BSF 0xff1, 0x5, ACCESS
24 002E  SCF0  BSF 0xff0, 0x6, ACCESS
25 0030  S&F0  BSF 0xff0, 0x3, ACCESS
26 0032  SCF2  BSF 0xff2, 0x6, ACCESS
27 0034  909F  BCF 0xf9f, 0, ACCESS
28 0036  809D  BSF 0xf9d, 0, ACCESS
29 0038  829F  BCF 0xf9f, 0x1, ACCESS
30 003A  829D  BSF 0xf9d, 0x1, ACCESS
31 003C  0E01  MOVLW 0xa
32 003E  6E01  MOVF 0x1, ACCESS
33 0040  EF20  GOTO 0x40
```
PIC18 Resets

When the reset signal is activated:

- The MPU goes into a reset state during which the initial conditions are established.
- The program counter is cleared to 000000 which is called the reset vector.
- The MPU begins the execution of instructions from location 000000.
On Chip reset circuit for PIC18

- Power-on reset (POR)
- MCLR pin reset during normal operation
- MCLR pin reset during SLEEP
- Watchdog timer (WDT) reset (during normal operation)
- Programmable brown-out reset (BOR)
- RESET instruction
- Stack full reset
- Stack underflow reset
PIC18 Resets

- PIC18 MCU can be reset by external source such as the push-button key, or when power is turned-on, or by various internal sources.
  - Resets categorized as follows:
    - External Manual Reset Key
    - Power-on Reset (POR)
    - Watchdog Timer Reset (WDT)
    - Programmable Brown-Out Reset (BOR)
    - RESET and SLEEP Instructions
    - Stack Full and Underflow Reset

Find MCLR pin!
Example of Reset Programming

- Identifying a power-on reset
  - IF_ RCON, NOT_POR == 0 ; POR has occurred
  - setf RCON ; Reinitialize all reset flags after power on
  - <take action particular to power-on reset>
  - ENDIF_

- Identifying a reset due to execution of a “reset” instruction
  - IF_ RCON, NOT_RI == 0 ; reset’ instruction has been executed
  - bsf RCON, NOT_RI ; Set bit to distinguish from other resets
  - <take appropriate action in response to “reset” instruction>
  - ENDIF