Microprocessors B (17.384)

Spring 2011

Lecture Outline

Class # 04

February 15, 2011

Dohn Bowden

Today's Lecture

- Administrative
- Microcontroller Hardware and/or Interface
- Programming/Software
- Lab
- Homework

Course Admin

Administrative

- Admin for tonight …
  - Syllabus Highlights
    - Lab #1 is due next Tuesday (February 22nd) … ***Changed***
  - We shall finish Lab #1 this week and, time permitting we will start Lab #2
  - NO lecture … ***Lab only***

Syllabus Review

Week	Date	Topics	Lab	Lab Report Due
~~1~~	~~01/25/11~~	~~PIC pin out, C programming, Watchdog Timer, Sleep~~		
~~2~~	~~02/01/11~~	~~General-purpose IO, LED/switch IO, FSM~~	~~1~~	
~~3~~	~~02/08/11~~	~~Lab~~	~~1 con't~~	
4	02/15/11	Interrupts, Timers, interrupt-driven IO	2	
5	02/22/11	Lab	2 con't	1
6	03/01/11	Asynchronous and Synchronous Serial IO (UART, I^2C, SPI)	3	2
7	03/08/11	**Examination 1**		
X	03/15/11	**No Class – Spring Break**		
8	03/22/11	Lab	3 con't	
9	03/29/11	Serial EEPROM operation, DAC, DC motor control, Servos, Stepper motor control	4	3
10	04/05/11	Lab	4 con't	
11	04/12/11	Advanced Hardware Topics	Project	4
12	04/19/11	**Examination 2**		
13	04/26/11	Work on Course Project	Project	
14	05/03/11	Final Exam/Course Project Brief and Demonstration	Demo	

Chat Page

- Still looking into what is available through the school
  - More to follow as it develops

Microcontroller Hardware and / or Interfaces

Chapter 9 …

Timers…

Timers

- A Timer is …
  - Just a counter
- The Timer2/3 feature has …
  - 32-bit timers that can also be configured as …
    - Two independent 16-bit timers with selectable operating modes

Timers

- As a 32-bit timer, the Timer2/3 feature permits operation in three modes …
  - Two Independent 16-bit timers (Timer2 and Timer3) with all 16-bit operating modes (except Asynchronous Counter mode)
  - Single 32-bit timer (Timer2/3)
  - Single 32-bit synchronous counter (Timer2/3)

Timers

- The Timer2/3 feature also supports …
  - Timer gate operation
  - Selectable Prescaler Settings
  - Timer operation during Idle and Sleep modes
  - Interrupt on a 32-bit Period Register Match
  - Time Base for Input Capture and Output Compare Modules (Timer2 and Timer3 only)
  - ADC1 Event Trigger (Timer2/3 only)

Timers

- Time can be converted from elapsed Timer Ticks (Ticks) by …
  - Multiplying by the clock period (Ttmr) of the timer …

$$Time = Ticks \times Ttmr$$

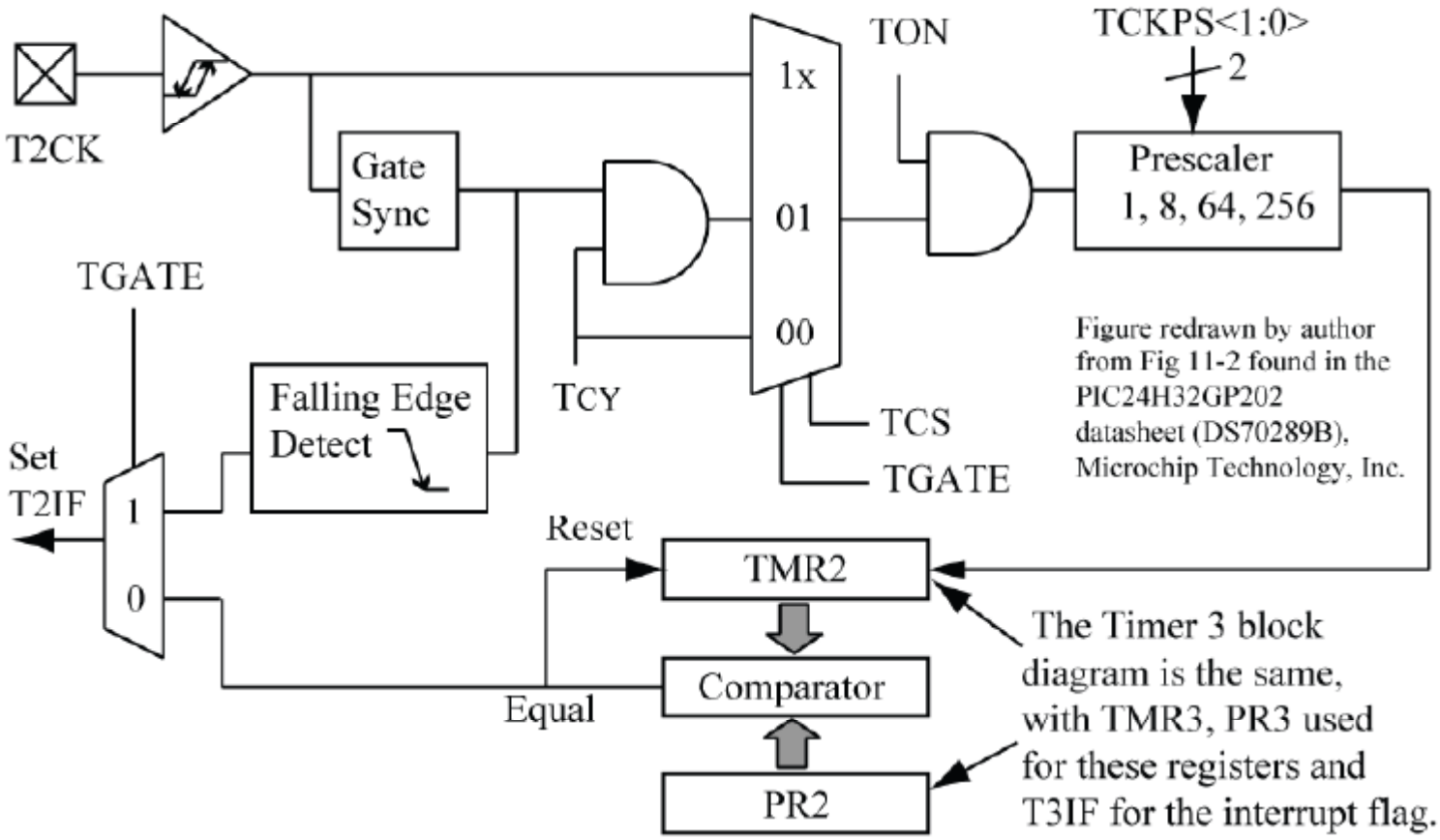
Timers

- If a timer is a 16-bit timer … and …
  - It is clocked at the FCY = 40 MHz … then …
    - It will count from 0x0000 to 0xFFFF (65536 ticks) in …

      Time = 65536 x (1/40 MHz)

      = 65536 x 25 ns = 1638400 ns = 1638.4 us = 1.6384 ms

Timer2 Block Diagram



T2IF Period

- The T2IF flag is set at the following period (T_{t2if}) …

$$T_{t2if} = (PR2+1) \times PRE \times Tcy = (PR2+1) \times PRE/Fcy$$

- Observe that because Timer2 is a 16-bit timer, if PR2 is its maximum value of 0xFFFF (65535), and the prescaler is '1', this is just:

$$T_{t2if} = 65536 \times 1/Fcy$$

- We typically want to solve for T_{t2if}, given a PRE value …

$$PR2 = (T_{t2if} \times Fcy / PRE) - 1$$

Example T2IF Periods

PR2/PRE Values for T_{t2if} = 15 ms, Fcy = 40 MHz

	PRE=1	**PRE=8**	**PRE=64**	**PRE=256**
PR2	600000	75000	9375	2344
	(invalid)	(invalid)		

The PR2 for PRE=1, PRE=8 are invalid because they are greater than 65535 (PR2 is a 16-bit register).

Configuring Timer2 to interrupt every T_{t2if} period is called a **PERIODIC INTERRUPT**.

32-Bit Timer Configuration …

To configure Timer2 … 32-Bit Operation

- To configure the Timer2 feature for 32-bit operation …
  1. Set the corresponding T32 control bit
  2. Select the prescaler ratio for Timer2 using the TCKPS<1:0> bits.
  3. Set the Clock and Gating modes using the corresponding TCS and TGATE bits
  4. Load the timer period value. PR3 contains the most significant word of the value, while PR2 contains the least significant word
  5. Set the interrupt enable bit T3IE, if interrupts are required. Use the priority bits T3IP<2:0> to set the interrupt priority. While Timer2 controls the timer, the interrupt appears as a Timer3 interrupt.
  6. Set the corresponding TON bit

To configure Timer2 … 32-Bit Operation

- Therefore …

1. Set the corresponding T32 control bit …

```
T3CONbits.TON = 0; // Stop any 16-bit Timer3 operation

T2CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation

T2CONbits.T32 = 1; // Enable 32-bit Timer mode
```

To configure Timer2 … 32-Bit Operation

2. Select the prescaler ratio for Timer2 using the TCKPS<1:0> bits …

```
T2CONbits.TCKPS = 0b00// Select 1:1 Prescaler
```

To configure Timer2 … 32-Bit Operation

3. Set the Clock and Gating modes using the corresponding TCS and TGATE bits …

```
T2CONbits.TCS = 0; // Select internal instruction cycle clock

T2CONbits.TGATE = 0; // Disable Gated Timer mode
```

To configure Timer2 … 32-Bit Operation

4. Load the timer period value. PR3 contains the most significant word of the value, while PR2 contains the least significant word …

```
PR3 = 0x0002; // Load 32-bit period value (msw)

PR3 = 0x0000; // Load 32-bit period value (lsw)
```

To configure Timer2 … 32-Bit Operation

5. Set the interrupt enable bit T3IE, if interrupts are required. Use the priority bits T3IP<2:0> to set the interrupt priority. While Timer2 controls the timer, the interrupt appears as a Timer3 interrupt …

```
IPC2bits.T3IP = 0x01; // Set Timer3 Interrupt Priority Level

IFS2bits.T3IF = 0; // Clear Timer3 Interrupt Flag

IEC0bits.T3IE = 1; // Enable Timer3 interrupt
```

To configure Timer2 … 32-Bit Operation

6. Set the corresponding TON bit …

```
T2CONbits.TON = 1; // Start 32-bit Timer
```

In Summary …
To configure Timer2 in 32-Bit Operation

```
T3CONbits.TON = 0; // Stop any 16-bit Timer3 operation
T2CONbits.TON = 0; // Stop any 16/32-bit Timer3 operation
T2CONbits.T32 = 1; // Enable 32-bit Timer mode
T2CONbits.TCS = 0; // Select internal instruction cycle clock
T2CONbits.TGATE = 0; // Disable Gated Timer mode
T2CONbits.TCKPS = 0b00// Select 1:1 Prescaler
TMR3 = 0x00; // Clear 32-bit Timer (msw)
TMR2 = 0x00; // Clear 32-bit Timer (lsw)
PR3 = 0x0002; // Load 32-bit period value (msw)
PR3 = 0x0000; // Load 32-bit period value (lsw)
IPC2bits.T3IP = 0x01; // Set Timer3 Interrupt Priority Level
```

Continued next page ….

In Summary …
To configure Timer2 in 32-Bit Operation

```
IFS2bits.T3IF = 0; // Clear Timer3 Interrupt Flag
IEC0bits.T3IE = 1; // Enable Timer3 interrupt
T2CONbits.TON = 1; // Start 32-bit Timer

/* Example code for Timer3 ISR*/
void __attribute__((__interrupt__, __shadow__)) _T3Interrupt(void)
{
/* Interrupt Service Routine code goes here */

IFS0bits.T3IF = 0; //Clear Timer3 interrupt flag
}
```

In Summary …
To configure Timer2 in 32-Bit Operation

- The timer value at any point is stored in …
  - The register pair TMR3:TMR2
- TMR3 always contains the …
  - Most significant word (msw) of the count
  - TMR2 contains the least significant word (lsw)

16-Bit Timer Configuration …

To configure Timer2 … 16-Bit Operation

- To configure any of the timers for …
  - Individual 16-bit operation …
    - The process is similar
      - Refer to the Family reference manual for timers
        - » Section 11

Where do we head from here?

Interrupts

- To utilize the interrupt feature of the …
  - Timers and other module …
    - We need to have an understanding of …
      - Interrupts … and …
      - How they work!

Interrupt Basics

Polled Input / Output

- Polled Input / Output (IO) …
  - Processor continually checks IO device to see if it is ready for data transfer
    - Inefficient, processor wastes time checking for ready condition
    - Either checks too often or not often enough

PIC24 µC Interrupt Operation

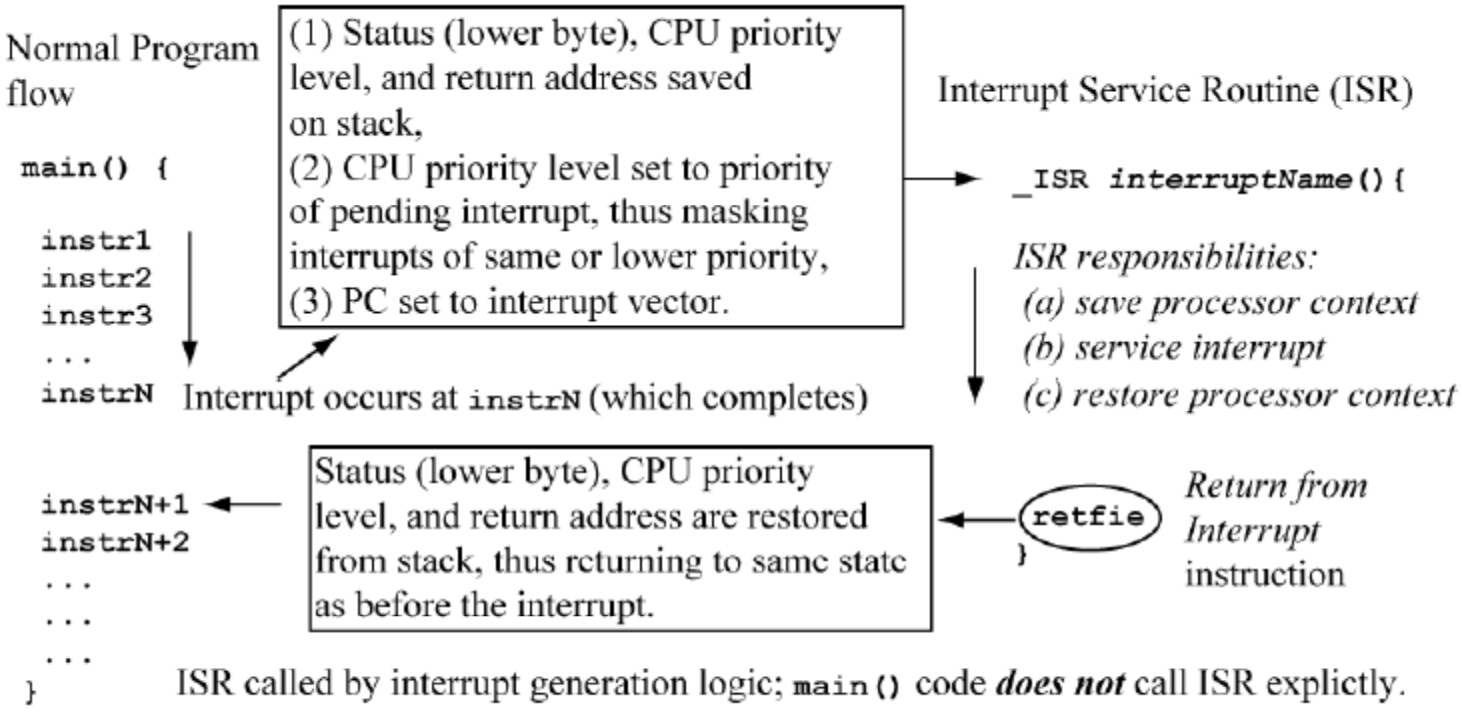
- The normal program flow … main … is referred to as …
  - The foreground code
- The interrupt service routine (ISR) is referred to as …
  - The background code

PIC24 µC Interrupt Operation

- Code that executes when the interrupt occurs is …
  - The interrupt service routine (ISR)
- The ISR …
  - Responds to whatever event triggered the interrupt

PIC24 µC Interrupt Operation

- **During normal program execution …**
  - **If we have an interrupt … the following will occur …**



Interrupt Driven Input / Output

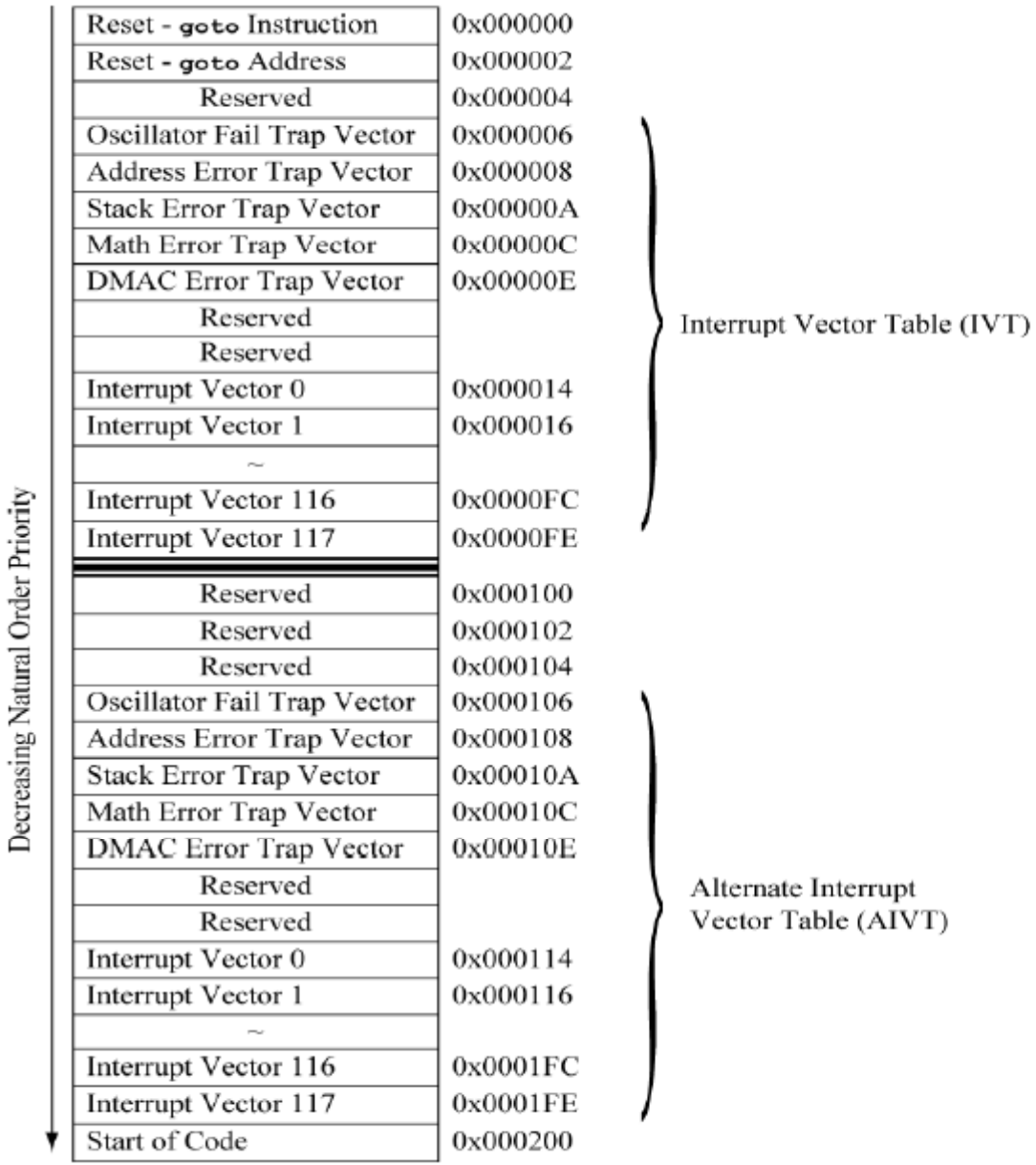
- Interrupt Driven Input / Output (IO) …
  - IO device interrupts processor when it is ready for data transfer
    - Processor can be doing other tasks while waiting for last data transfer to complete – very efficient
    - All IO in modern computers is interrupt driven

Vector Tables

Vector Tables

- Vector Tables …
  - Contains the starting address of the ISR for each interrupt source
- Interrupt Vector Table (IVT) …
  - Group of program memory locations
- Alternate Interrupt Vector Table (AIVT) …
  - Group of program memory locations

Vector Table



Interrupt Sources

PIC24HJ32GP202 - Interrupt Sources

- Interrupt sources depends on …
  - On-chip peripherals
- The following slide gives the sources for our processor
  - NOTE … Vector Num column gives the value for the lower seven bits of the following special function register …

    INTTREG (INTTREG<6:0>)
- INTTREG (INTTREG<11:8>) contains the interrupt priority

IVT Address	Vector Num	PIC24 Compiler Name	Vector Function
0x000006	1	_OscillatorFail	Oscillator Failure
0x000008	2	_AddressError	Address Error
0x00000A	3	_StackError	Stack Error
0x00000C	4	_MathError	Math Error
0x000014	8	_INT0Interrupt	INT0 – External Interrupt
0x000016	9	_IC1Interrupt	IC1 – Input Capture 1
0x000018	10	_OC1Interrupt	OC1 – Output Compare 1
0x00001A	11	_T1Interrupt	T1 – Timer1 Expired
0x00001E	13	_IC2Interrupt	IC2 – Input Capture 2
0x000020	14	_OC2Interrupt	OC2 – Output Compare 2
0x000022	15	_T2Interrupt	T2 – Timer2 Expired
0x000024	16	_T3Interrupt	T3 – Timer3 Expired
0x000026	17	_SPI1ErrInterrupt	SPI1E – SPI1 Error
0x000028	18	_SPI1Interrupt	SPI1 – SPI1 transfer done
0x00002A	19	_U1RXInterrupt	U1RX – UART1 Receiver
0x00002C	20	_U1TXInterrupt	U1TX – UART1 Transmitter
0x00002E	21	_ADC1Interrupt	ADC1 – ADC 1 convert done
0x000034	24	_SI2C1Interrupt	SI2C1 – I2C1 Slave Events
0x000036	25	_MI2CInterrupt	MI2C1 – I2C1 Master Events
0x00003A	27	_CNInterrupt	Change Notification Interrupt
0x00003C	28	_INT1Interrupt	INT1 – External Interrupt
0x000040	30	_IC7Interrupt	IC7 – Input Capture 7
0x000042	31	_IC8Interrupt	IC8 – Input Capture 8
0x00004E	37	_INT2Interrupt	INT2 – External Interrupt
0x000096	73	_U1ErrInterrupt	U1E – UART1 Error

Interrupt Sources

Serial data has arrived

CNx Pin has changed state

Interrupt Priorities

Interrupt Priorities

- An interrupt can be assigned a priority from 0 to 7
  - Normal instruction execution is priority 0
- An interrupt MUST have a higher priority than 0 to interrupt normal execution
- Assigning a priority of 0 to an interrupt masks (disables) than interrupt
- An interrupt with a higher priority can interrupt a currently executing ISR with a lower priority

Interrupt Priorities

- If simultaneous interrupts of the SAME priority occur …
  - Then the interrupt with the …
    - LOWER VECTOR NUMBER … is first in the interrupt vector table … has the higher *natural priority*
      - For example …
        - » The INT0 interrupt has a higher natural priority than INT1

Enabling an Interrupt

- Each interrupt source generally has …
  - FLAG bit
  - PRIORITY bits … and …
  - An ENBLE bit
- The flag bit is set whenever the flag condition is true …
  - Which varies by the interrupt
- The priority bits set the interrupt priority

Enabling an Interrupt

- The enable bit must be ‘1’ for the ISR to be executed …
  - NOTE … the enable bit *does not* have to be a ‘1’ for …
    - The flag bit to be set
- One of the things that must be done by the ISR is to …
  - Clear the flag bit … or else …
    - The ISR will get stuck in an infinite loop

Enabling an Interrupt

- By default …
  - All priority bits and enable bits are ‘0’ … so …
    - Interrupt ISRs are disabled from execution

Traps vs. Interrupts

Traps vs. Interrupts

- **A Trap is a special type of interrupt, is non-maskable, has higher priority than normal interrupts. Traps are always enabled!**
- **Hard trap …**
  - **CPU stops after instruction at which trap occurs**
- **Soft trap …**
  - **CPU continues executing instructions as trap is sampled and acknowledged**

Traps vs. Interrupts

Trap	Category	Priority	Flag(s)
Oscillator Failure	Hard	14	_OSCFAIL (oscillator fail, INTCON1<1>), _CF (clock fail, OSSCON<3>)
Address Error	Hard	13	_ADDRERR (address error, INTCON1<3>)
Stack Error	Soft	12	_STKERR (stack error, INTCON1<2>)
Math Error	Soft	11	_MATHERR (math error, INTCON1<4>)
DMAC Error	Soft	10	_DMACERR (DMA conflict write, INTCON1<5>)

GOLDEN RULE …

- An ISR should …
  - Do its work as quickly as possible

- When an ISR is executing …
  - It is keeping other ISRs of equal priority and lower from executing … as well as … the main code

INTx External Interrupts

INTx External Interrupts

- INTx interrupt inputs are another source for interrupts
- INT0 is assigned to pin 16
- INT1 and INT2 are not assigned …
  - Therefore … they must be mapped to an external pin
    - RPn
      - Remappable pins

INTx External Interrupts

- They can be configured to be …
  - Rising edge triggered … or …
  - Falling-edge triggered … by …
    - Using an associated INT*xEP* bit
      - '1' is falling edge
      - '0' is rising edge

Remappable Inputs

Input Name	Function Name	Example Assignment mapping inputs to RP*n*
External Interrupt 1	INT1	_INT1R = *n*;
External Interrupt 2	INT2	_INT2R = *n*;
Timer2 Ext. Clock	T2CK	_T2CKR = *n*;
Timer3 Ext. Clock	T3CK	_T3CKR = *n*;
Input Capture 1	IC1	_IC1R = *n*;
Input Capture 2	IC2	_IC2R = *n*;
UART1 Receive	U1RX	_U1RXR = *n*;
UART1 Clr To Send	U1CTS	_U1CTSR = *n*;
SPI1 Data Input	SDI1	_SDI1R = *n*;
SPI1 Clock Input	SCK1	_SCK1R = *n*;
SPI1 Slave Sel. Input	SS1	_SS1R = *n*;

Remappable Inputs continued

Output Name	Function Name	RP*n*R<4:0> Value	Example Assignment
Default Port Pin	NULL	0	_RP*n*R = 0;
UART1 Transmit	U1TX	3	_RP*n*R = 3;
UART1 Rdy. To Send	U1RTS	4	_RP*n*R = 4;
SPI1 Data Output	SDO1	7	_RP*n*R = 7;
SPI1 Clock Output	SCK1OUT	8	_RP*n*R = 8;
SPI1 Slave Sel. Out.	SS1OUT	9	_RP*n*R = 9;
Output Compare 1	OC1	18	_RP*n*R = 18;
Output Compare 2	OC2	19	_RP*n*R = 19;

Mapping outputs to RPx pins.

Interrupt Service Routines …

Specifying Attributes of Functions

- In the compiler, you declare certain things about functions called in your program which help the compiler optimize function calls and check your code more carefully
- The keyword __attribute__ allows you to …
  - Specify special attributes when making a declaration
- This keyword is followed by an attribute specification inside double parentheses

Specifying Attributes of Functions

- The following attribute is currently supported for functions …

  interrupt [([save(list)] [, irq(irqid)] [,
  altirq(altirqid)] [, preprologue(asm)])]

- Use this option to indicate that the specified function is an interrupt handler

- The compiler will generate function prologue and epilogue sequences suitable for use in an interrupt handler when this attribute is present

INTERRUPT SERVICE ROUTINE CONTEXT SAVING

- Interrupts … by their very nature … can occur at unpredictable times …
  - Therefore … the interrupted code must be able to resume with the same machine state that was present when the interrupt occurred
- To properly handle a return from interrupt …
  - The setup (*prologue*) code for an ISR function automatically saves the compiler-managed working and special function registers on the stack for later restoration at the end of the ISR

INTERRUPT SERVICE ROUTINE CONTEXT SAVING

- You can use the optional save parameter of the interrupt attribute to specify additional variables and special function registers to be saved and restored

- In certain applications … it may be necessary to insert assembly statements into the interrupt service routine immediately prior to the compiler-generated function prologue

INTERRUPT SERVICE ROUTINE CONTEXT SAVING

- For example … it may be required that a semaphore be incremented immediately on entry to an interrupt service routine

- A semaphore is a …

  – Flag set by an ISR to signal foreground task that an I/O action has occurred

- This can be done as follows …

```
void __attribute__((__interrupt__(__preprologue__("inc _semaphore"))))
        isr0(void);
```

Specifying Attributes of Functions

- The optional parameter save specifies …
  - A list of variables to be saved and restored in the function prologue and epilogue
- The optional parameters irq and altirq specify …
  - Interrupt vector table ID's to be used
- The optional parameter preprologue specifies …
  - Assembly code that is to be emitted before the compiler-generated prologue code

Specifying Attributes of Functions

- Program Space Visibility … (PSV) …
  - Allocate the variable in program space
- When using the interrupt attribute …
  - You should specify either …
    - auto_psv … or …
    - no_auto_psv
  - If none is specified a warning will be produced and auto_psv will be assumed

Syntax for Writing ISRs

- The syntax of the interrupt attribute is …

```
__attribute__((interrupt [(
        [ save(symbol-list)]
        [, irq(irqid)]
        [, altirq(altirqid)]
        [, preprologue(asm)]
        )]
    ))
```

An Example …

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void)

{

        if ( flag ==0 )
        {LATBbits.LATB2 =1;

                }
        else
        {
        LATBbits.LATB2=0;

        }
        flag =flag  ^ 1;        //  Toggle flag

      // 1.2 clear the interrupt flag
      _T1IF = 0;

} //T1Interrupt
```

Interrupt Summary …

Dividing Work between the ISR and main()

- There are usually multiple ways to divide work between the ISR and main()
- The ‘right’ choice is the one that services the I/O event in a timely manner, and there can be more than right choice
- Golden Rules:
  - The ISR should do its work as fast as possible
  - Do not put long software delays into an ISR
  - An ISR should never wait for I/O, the I/O event should trigger the ISR
  - An ISR is never called as a subroutine

Programming/Software

Flashing LED1…

Flashing LED - Using Timers and Interrupts

- Next Lab will be to update current Lab #1 programs such …
  - That we use Timers
  - And use Interrupts

Summary …

Summary

- Timers
- Interrupts
- Input / Output using the above

Lab

Peer Review of Software …

Peer Review of Software Developed

- How did you write your code?
- What problems did you encountered?
- Any questions that you need resolved?

Lab #2 …

Lab #2

- Update current Lab #1 programs such …
  - That we use Timers
  - And use Interrupts

Next Class

Next Class Topics

- **Lab #2 Start/continue**

Homework

Homework

- Read …
  - Material covered in today's lecture (may want to re-read)
    - Chapter 9, pages 317 - 362
  - Material for lecture in two weeks …
    - Chapter 10, pages 367 - 437
- Labs …
  - Lab #1 Report
  - Code development for Lab #2

Time to start the lab …

Lab

- **Continue Lab #1, if needed**
- **Start Lab #2**

White Board …

References

References

1. None