Timer2 Interrupts

ECE 376 Embedded Systems

Jake Glower - Lecture #18

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Timing in Assembler and C:

You can set the timing of a routine in assembler and C

- Assembler: Count instructions
- C: Trial and Error (Oscilloscope helps)

Assembler: Count every 10ms Solution

```
Loop   incf   PORTC,F
       call  Wait
       goto  Loop

Wait   movlw   80
       movwf   CNT1

Loop1  movlw   250
       movwf   CNT2

Loop2  nop
       nop
       decfsz  CNT2
       goto   Loop2
       decfsz  CNT1
       goto   Loop1
       return
```

C: Count every 10ms

```
void Main(void)
{
   unsigned int i;

   COUNTER = 0;
   TRISC  = 0;

   do {
      COUNTER += 1;
      PORTC = COUNTER;
      for(i=0; i<6170; i++);
      } while (1>0);
   }
```

Problems with Timing

The timing is slightly off

- It's hard to get the number of clocks per loop to be *exactly* 100,000

It's inefficient

- 99.9% of the time is spent in the wait loop

It makes code changes a pain

- If you add/ remove anything to the code, the timing is off

TIMER Interrupts

- Interrupts solve all of these problems

Interrupts are similar to subroutines except that

- Subroutines are routines called by software (such as the 10ms wait loop from before)
- Interrupts are routines called by hardware (such as a certain time elapses)

Timer Interrupts are useful: four are avablable of a PIC18F4626:

- TIMER0: Interrupt after N events (or N clocks). N = 1 to 2^{24} (1.67 seconds)
- TIMER1: Interrupt after N events (or N clocks). N = 1 to 2^{19} (52 milliseconds)
- TIMER2: Interrupt every N clocks. N = 1 to 2^{16} (6.5 millisecond)
- TIMER3: Interrupt after N events (or N clocks). N = 1 to 2^{19} (52 milliseconds)

Defaults:

- Default is interrupts are turned off
- You have to turn them on to use them.

If an interrupt occurs,

- The present instruction is completed
- The processor inserts a *call 0x08* into the program

The interrupt service routine *must* be located at address 0x08

What happens on an interrupt?

Save the W and STATUS register.

- Interrupts can be called at any time (i.e. middle of an *if* statement)

Clear TMR2IF.

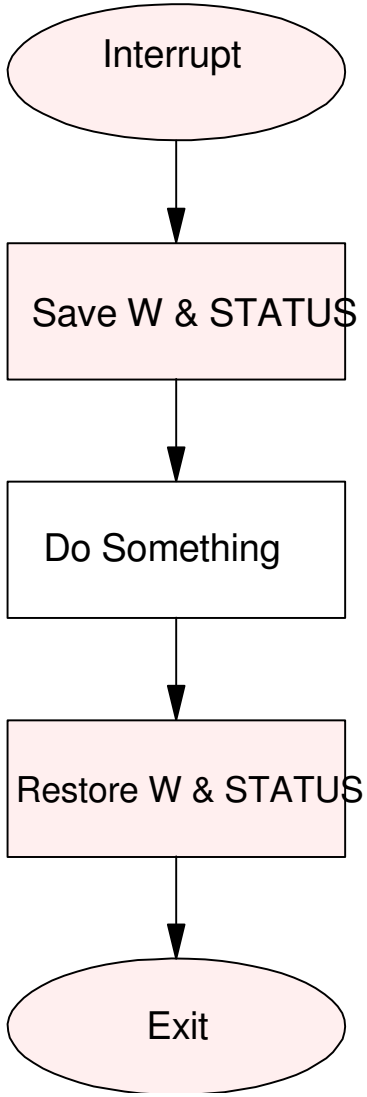
- Disables interrupts.
- Interrupts cannot interrupt another interupt.

Do something

- Optional.

Exit with *retfie*

- Return from interrupt
- Restores store W and STATUS
- Return the processor to its prior state



Timer2 Interrupts

Interrupts every N clocks

- 1 < N < 65,536 (6.55ms)

N = A * B * C

- A = 1..16
- B = 1..256
- C = 1, 4, or 16

Measure time to 1.000ms

- N = 10,000

Output 440Hz

- N = 11,364

Source	N	Enable Bits	Flag
OSC/4 (10MHz)	N = A*B*C PR2 = B-1 T2CON = xaaaa1cc aaaa = 0000: A=1 aaaa = 0001: A=2 ::: aaaa = 1110: A=15 aaaa = 1111: A=16 cc = 00: C = 1 cc = 01: C = 4 cc = 10: C = 16 cc = 11: C = 16	TMR2ON = 1 TMR2IE = 1 TMR2IP = 1 PEIE = 1	TMR2IF

Timer2 Interrupts vs. the Main Routine

- The main routine can do whatever
  - Drive the LCD display
  - Make lights bounce back and forth
  - Read the push buttons
- Interrupts run in the background
  - They have no affect on the main routine
  - The main routine has no affect on the interrupts

Photo: PIC board displaying time

Procedure to Turn On Timer2 Interrupts

Step 1: Turn on the enable bits (x4)

- TMR2ON = 1;
- TMR2IE = 1;
- PEIE = 1;
- TMR2IP = 1;

Plus a Global Interrupt Enable

- GIE = 1;

Forget any of these and interrupts won't happen

Source	N	Enable Bits	Flag
OSC/4 (10MHz)	N = A*B*C PR2 = B-1 T2CON = xaaaa1cc aaaa = 0000: A=1 aaaa = 0001: A=2 ::: aaaa = 1110: A=15 aaaa = 1111: A=16 cc = 00: C = 1 cc = 01: C = 4 cc = 10: C = 16 cc = 11: C = 16	TMR2ON = 1 TMR2IE = 1 TMR2IP = 1 PEIE = 1	TMR2IF

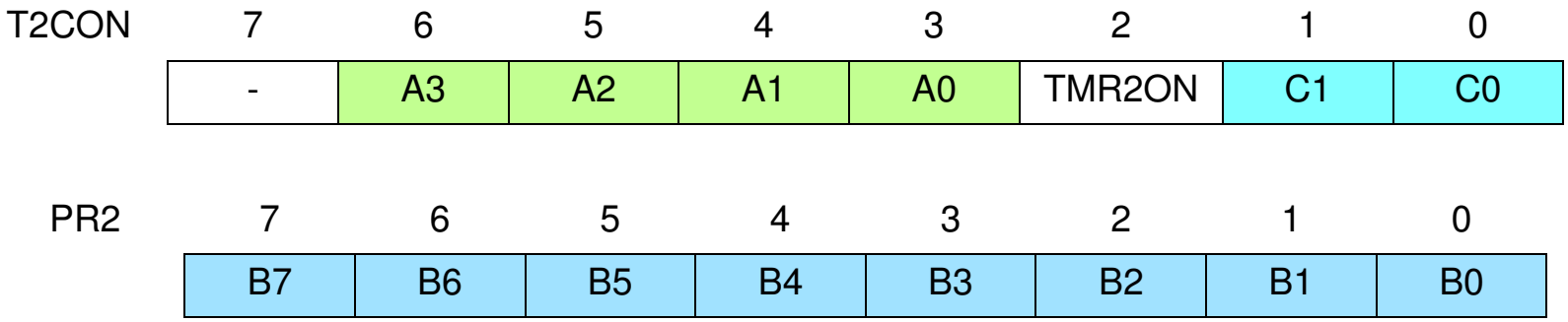
Procedure to Use Timer2 Interupts

Step 2: Set the Conditions for the interrupt (N)

Interrupt every N clocks

- N = A * B * C

A, B, and C are defined by registers T2CON and PR2:



Setting A, B, and C

- N = A * B * C
- Maximum value = 65,536 (6.5536ms)

PostScalar A		Main Scalar B		Prescalar C	
A3:A2:A1:A0	A	B7:B0	B	C1:C0	C
0000	1	0000 0000	1	00	1
0001	2	0000 0001	2	01	4
				10	16
1110	15	1111 1110	255	11	16
1111	16	1111 1111	256		

Example: Toggle RC0 every 6.5536 ms (65,536 clocks)

- N = 16 * 256 * 16

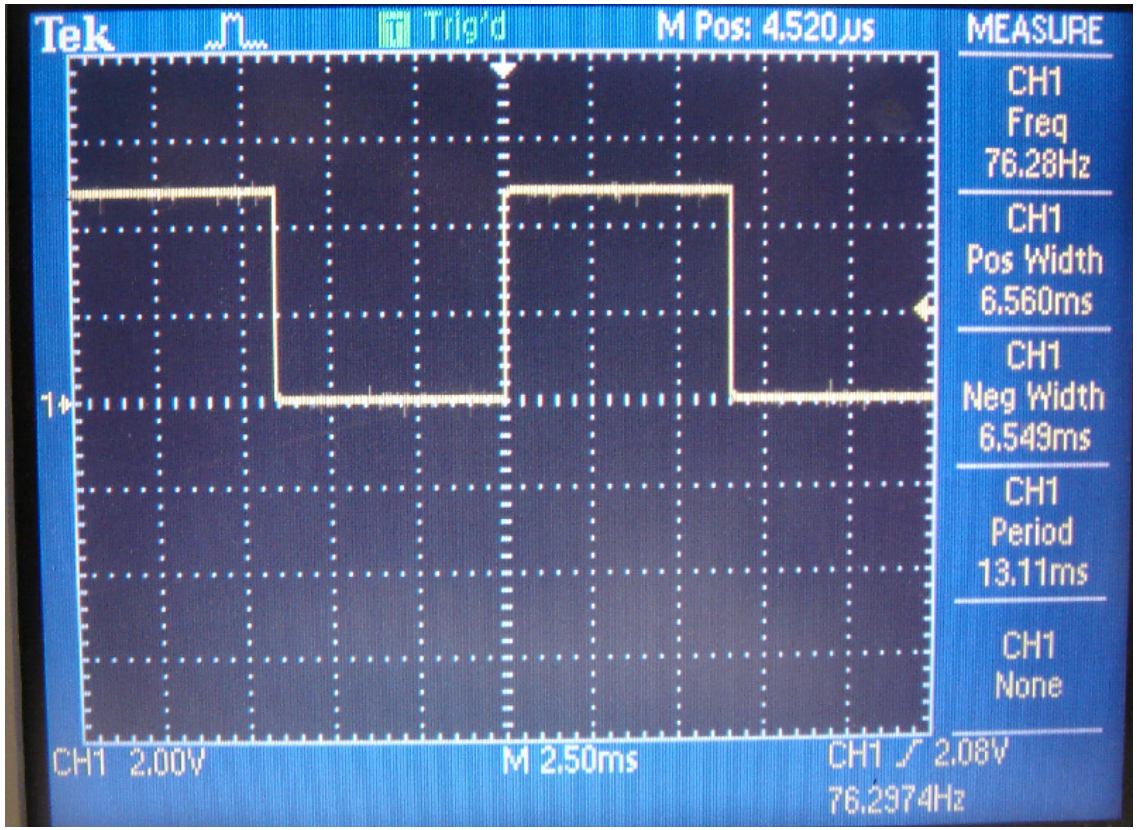
```
void interrupt timer2(void)
{
    RC0 = !RC0;
    TMR2IF  = 0;
    }

// initialize Timer2

    T2CON   = 0xFF;
    PR2     = 255;
    TMR2IE  = 1;
    PEIE    = 1;
    TMR2ON  = 1;
    TMR2IP  = 1;

// Turn on all interrupts

    GIE = 1;
```



Example 2: Toggle RC0 every 1.000ms

- N = 10 * 250 * 4 = 10,000 (1.000ms)

or

- PR2 = 249
- T2CON = 0x4D

T2CON	7	6	5	4	3	2	1	0
	-	A3	A2	A1	A0	T2E	C1	C0
(A=9, C=1)	0	1	0	0	1	1	0	1
		A = b1001 + 1 = 10					C = 4	

Toggle RC0 every 1.000ms (cont'd)

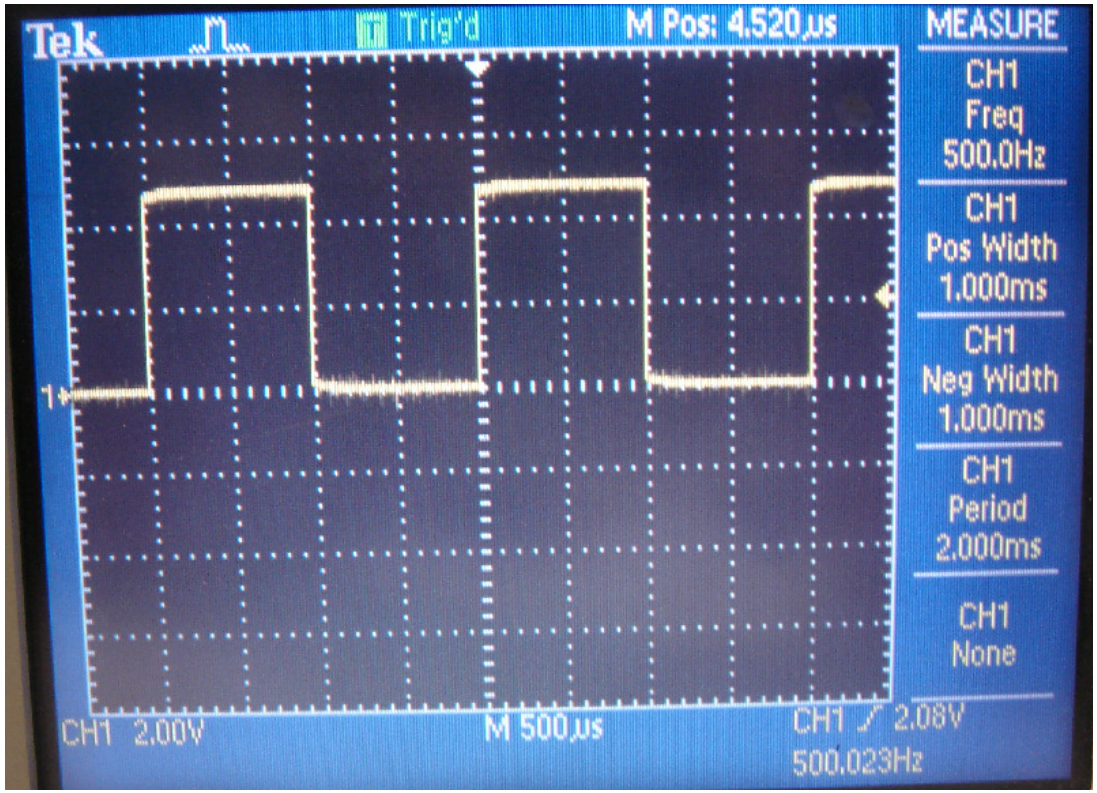
```
void interrupt timer2(void)
{
   RC0 = !RC0;
   TMR2IF  = 0;
   }

// initialize Timer2

   T2CON   = 0x4D;
   PR2     = 249;
   TMR2IE  = 1;
   PEIE    = 1;
   TMR2ON  = 1;
   TMR2IP  = 1;

// Turn on all interrupts

   GIE = 1;
```



Example 3: Play 440Hz

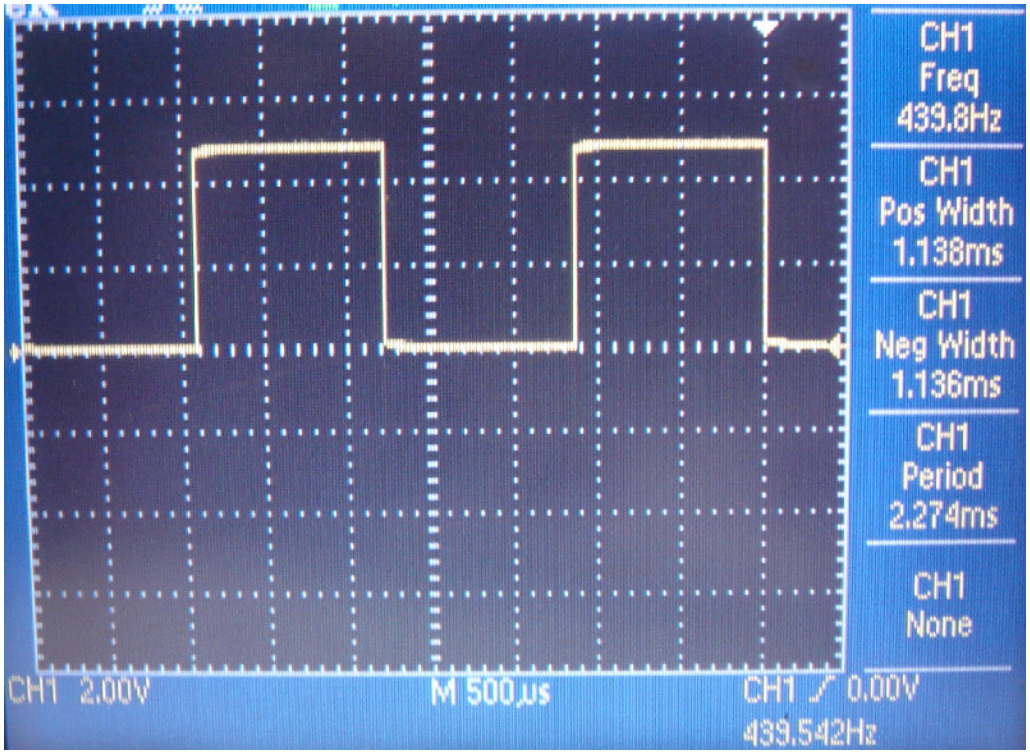
N = 11,364

- A = 12, B = 237, C = 4

```
void interrupt timer2(void)
{
   RC0 = !RC0;
   TMR2IF  = 0;
   }

// initialize Timer2

   T2CON   = 0x5D;
   PR2     = 236;
   TMR2IE  = 1;
   PEIE    = 1;
   TMR2ON  = 1;
   TMR2IP  = 1;
   GIE = 1;
```



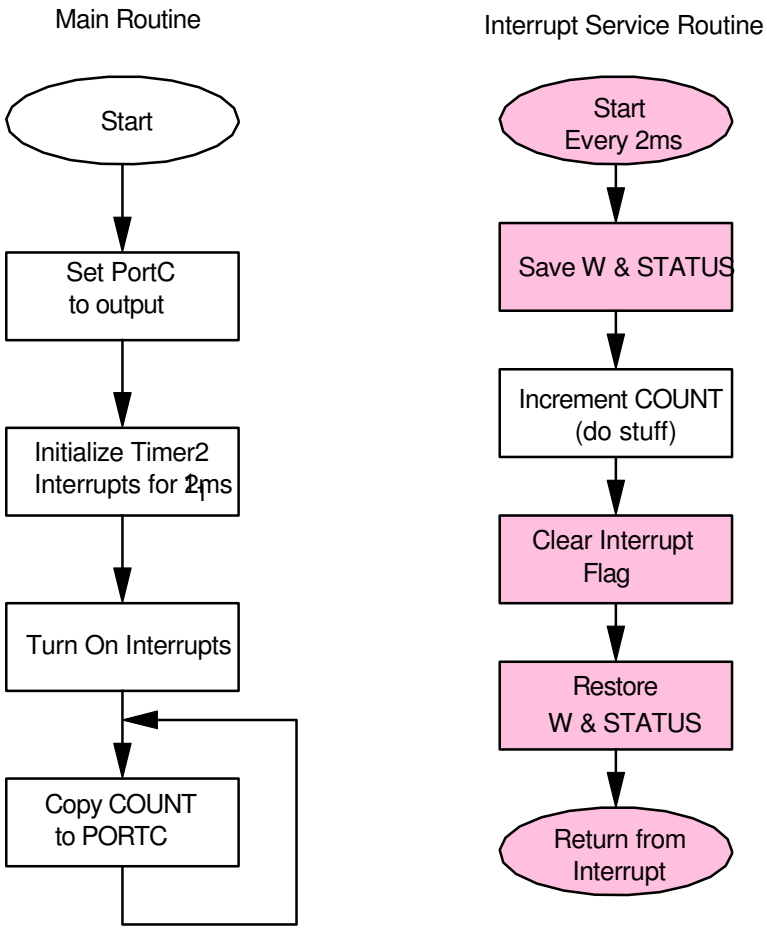
Interrupts and Flow Charts

You almost *have* to use parallel flow charts:

- The main routine starts executing on reset
- The interrupt routine is called every N clocks
- We have no idea when the interrupt is called

Note that

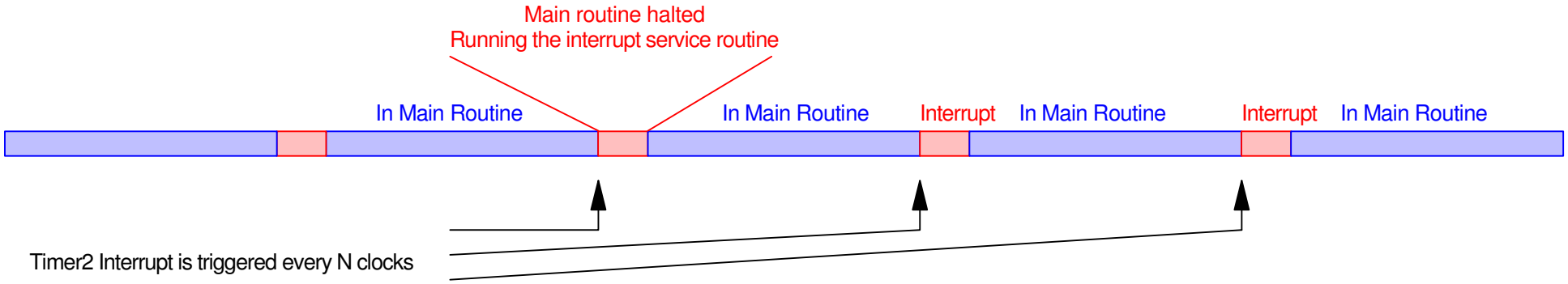
- The main routine simply watches COUNTER and sends it to PORTC.
- The Interrupt routine is responsible for changing COUNTER every 1ms



Interrupt Constraints

Timer2 interrupts are a way to keep track of time.

- The PIC is running at 10 million instructions / second (10MHz)
- Every N clocks, a Timer2 interrupt is triggered
- When the interrupt is triggered (every N clocks)
  - The main routine is halted
  - The interrupt routine executes, then
  - You return back to the main routine



Minimum Time Between Interrupts

It takes about 50 clocks to call an interrupt

- More if the interrupt does something
- The interrupt steals cycles from the main routine

N cannot be less than 50

- It takes about 50 clocks to call an interrupt and return
- If N is less than 50, it acts as if N = 50

Interrupt Time (clock resolution)	Clocks / Interrupt (N)	# Clocks Spent in the Interrupt	# Clocks Left for the Main Routine	Processor 'Speed'
1ms	10,000	50	9,950	99.5%
100 us	1,000	50	950	95%
10 us	100	50	50	50%
1 us	10	50	-50	0%

Maximum Time Between Interrupts

a) Maximum value for N is 65,536

- A = 16, B = 256, C = 16
- 6.5536ms

b) There is no maximum

- Instead of couting every interrupt, count every 10th interrupt
- There is no maximum size for a counter
- The counter must be a global variable

```
// Global Variables
unsigned int COUNTER

void interrupt timer2(void)
{
   COUNTER += 1;
   if (COUNTER >= 10000)
      COUNTER = 0;
      RC0 = !RC0;
      }
   TMR2IF  = 0;
   }
```

What Happens If....

- If you forget to include this line of code...

RC0 toggles every 50 clocks

- Upon exit, the main routine sees that TMR2IF=1
- This triggers another interrupt (RC0 toggles)
- Upon exit, the main routine sees that TMR2UIF = 1
- This triggers anotehr interrupt (RC0 toggles)
- etc.

The program is stuck inside the interrupt

```
void interrupt timer2(void)
{
    RC0 = !RC0;
    TMR2IF  = 0;
    }

// initialize Timer2

    T2CON   = 0x4D;
    PR2     = 249;
    TMR2IE  = 1;
    PEIE    = 1;
    TMR2ON  = 1;
    TMR2IP  = 1;

// Turn on all interrupts

    GIE = 1;
```

What Happens If....

- If you forget to include this line of code...

Timer2 interrupts are being called

A, B, and C have *some* value

- Whatever they were set to last time you ran a program
- You just don't know what they are

RC0 toggles at an unknown freqnecy

```
void interrupt timer2(void)
{
   RC0 = !RC0;
   TMR2IF  = 0;
   }

// initialize Timer2

   T2CON   = 0x4D;
   PR2     = 249;
   TMR2IE  = 1;
   PEIE    = 1;
   TMR2ON  = 1;
   TMR2IP  = 1;

// Turn on all interrupts

   GIE = 1;
```

What Happens If....

- If you forget to include this line of code...

Interrupts are not enabled.

- If they are not enabled, they don't happen

RC0 never changes

```
void interrupt timer2(void)
{
   RC0 = !RC0;
   TMR2IF  = 0;
   }

// initialize Timer2

   T2CON   = 0x4D;
   PR2     = 249;
   TMR2IE  = 1;
   PEIE    = 1;
   TMR2ON  = 1;
   TMR2IP  = 1;

// Turn on all interrupts

   GIE = 1;
```

Programming Style when Using Interrupts

Keep the interrupt routine short

- No do/while loops
- No for loops
- Just get in, do something, get out

The next interrupt is coming up

- If you spend too much time in the interrupt, you'll miss interrupts

Summary

Timer2 Interrupts are a way to

- Keep precise track of time
  - With a maximum resolution of 100us (N = 1000)
- Output a precise frequency

This is in parallel to the main routine

- The processor can now do two things at once