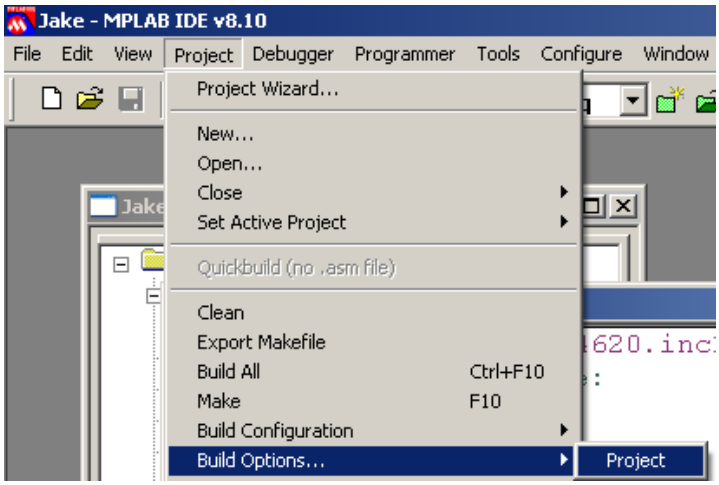
Binary Outputs and Timing

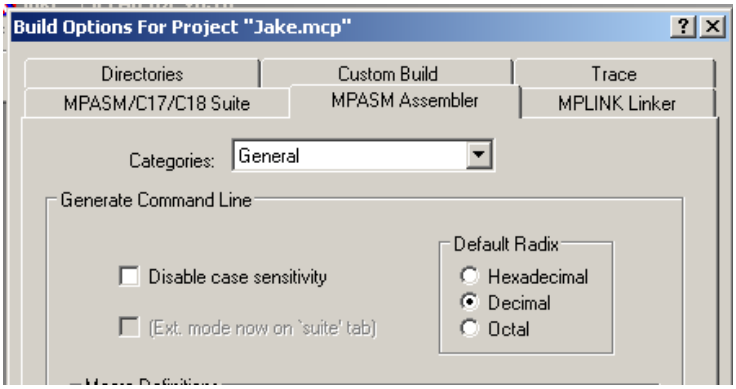
Each of the I/O pins on a PIC can be inputs or ourputs

- As an input, the pin is high impedance (meaning it is passive and draws very little current). If you apply 0V to that pin, it is read as logic 0, 5V is logic 1
- As an output, the pin is active. It will try to force the output pin to 0V (logic 0) or 5V (logic 1) A PIC chip has its limits: it is limited to sourcing or sinking 25mA (max).

Note: To run these programs, make sure the default in MPLAB is decimal. Check this by going to Project - Build Options - Project



MPASM: Decimal



Binary Outputs: Hardware

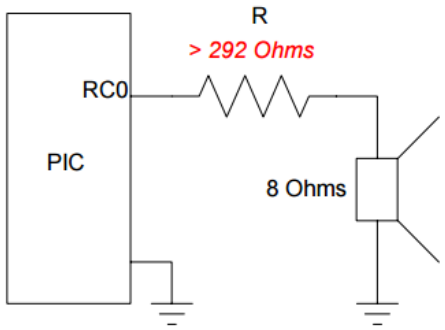
Problem: Connect an 8-Ohm speaker to your PIC board.

Solution #1: You can't connect an 8-Ohm speaker directly: it draws more than 25mA at 5V

$$I = \frac{5V}{8\Omega} = 625mA$$

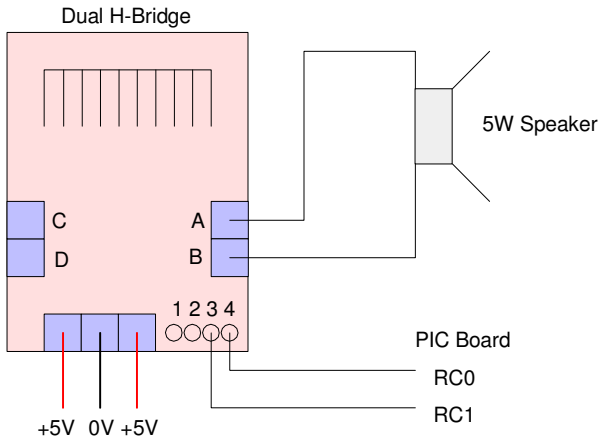
(Actually, you can connect an 8-ohm speaker to an I/O pin. It tends to burn out that I/O pin, however.) To limit the current, add a resistor in series:

$$R_{total} = \frac{5V}{25mA} = 200\Omega$$



Connecting and 8 Ohm Speaker toa PIC: Current is limited to 25mA

Solution #2: If you want a really *loud* speaker, use the H-bridge in your lab kit (and a diferent speaker rated at 3W or more...)



This H-bridge allows you to drive loads

- Up to 46VDC, and
- Up to 3A (max), 2A (continuous)

The inputs (RC0 and RC1) control the votlage Vab

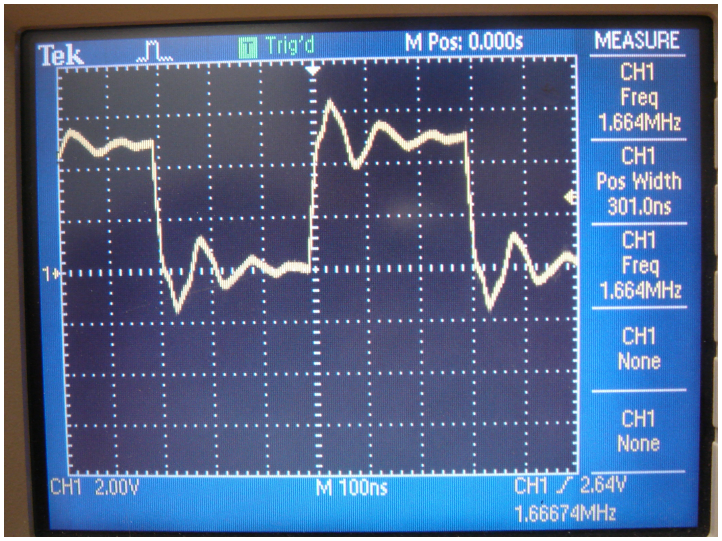
RC0 (IN4)	RC1 (IN3)	Vab
0	0	0V
0	1	-3.27V
1	0	+3.27V
1	1	0V

(note: you don't get +5V or -5V out due to a slight drop across the transistors in the H-bridge)

Software & Timing

Count on PORTC (counts really fast)

```
 #include <p18f4620.inc>

; Start of code:
      org 0x800
      clrf TRISC
      clrf PORTC

      movlw 0x0F
      movwf ADCON1
Loop:
      incf  PORTD,F
      goto Loop
      end
```



Signal on RD0: Loop Time = 301ns (3 clocks)

Problem: Play 261Hz on a speaker

Solution: You can't connect a speaker to a constant (DC) source: all that does is push the cone out. To make noise, you need to move the cone back and forth to create pressure waves (i.e. sound). From the PIC's standpoint, you need to output a square wave on RC0: the frequency of the square wave is the frequency of the should you hear.

From before, if you want to play the note C4 (261Hz), you need to output a square wave with a frequency of 261Hz on RC0:

$$\#\ \text{Clocks} = \frac{10{,}000{,}000}{2xHz}$$

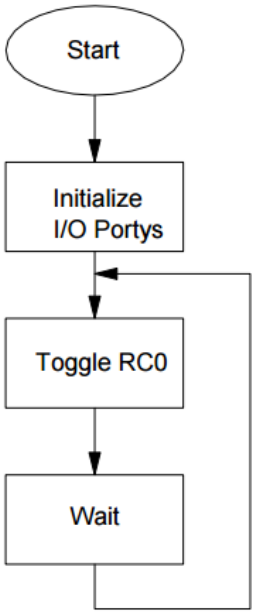
At 261Hz

$$\#\ \text{Clocks} = 19{,}157$$

Ideally, there should be 19,157 clocks between each time you toggle RC0.

To do that, add a wait loop:

```
Wait:
        movlw   19
        movwf   CNT1
Loop1:
        movlw   100
        movwf   CNT0
Loop0:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  CNT0,F
        goto    Loop0

        decfsz  CNT1,F
        goto    Loop1

        return
```



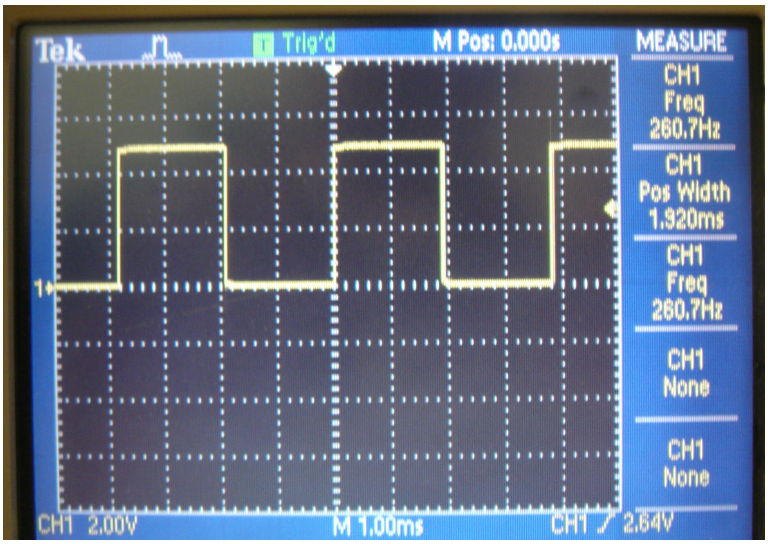
Note the following:

- The wait routine (Wait) is a little off. It *should* wait 19,157 clocks. It actually waits

  $\#\ \text{Clocks} = (10*100 + 5) * 19 + 5$

  $= 19{,}100$ (0.29% low)
- Pin RC0 is always outputing a square wave. It's kind of annoying.

On the oscilloscope, you can check the frequency:

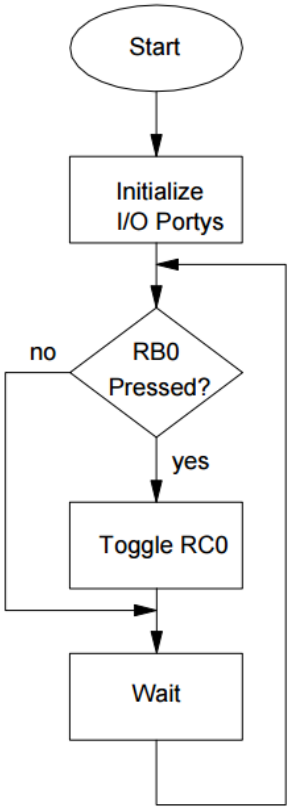


Signal on RC0 for 260.7Hz (loop time = 19,157 clocks = 1.9157ms)

One Key Piano: Suppose instead you want to play this note only when you press RB0. One way to do this is to check if RB0 is pressed:

- If RB0 = 1 (button pressed), toggle RC0
- Otherwise, leave RC0 alone

This results in a square wave appearing on RC0 only when RB0 is pressed.




```
; This program toggles RC0 at 261 Hz (note C4)

#include <p18f4620.inc>

; Variables

CNT0 EQU 1
CNT1 EQU 2

; Program

        org 0x800
        call Init
Loop:
        btfsc  PORTB,0
        call Toggle

        call Wait
        goto Loop


( same as before )
```

Note that if RB0 is not pressed, it skips over the toggle command.

4-Key Piano: Finally, design a 4-key piano: Whan you press RB0..RB3, the note should be:

- RB0: 261 Hz (C4)
- RB1: 293 Hz (D4)
- RB2: 329 Hz (E4)
- RB3: 349 Hz (F4)

There are several ways to do this. One way is to use four wait loops: one for each frequency. If you get fancy, you can also write a single wait loop and adjust the period - but why get too fancy.

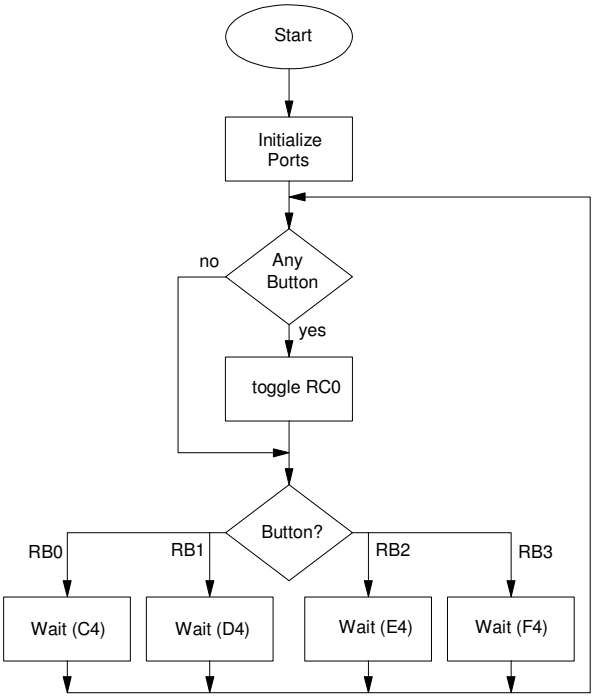
The number of clocks to wait is

$$\#\ \text{Clocks} = \frac{10{,}000{,}000}{2 \times \text{Hz}}$$

The clocks for each wait loop are then:

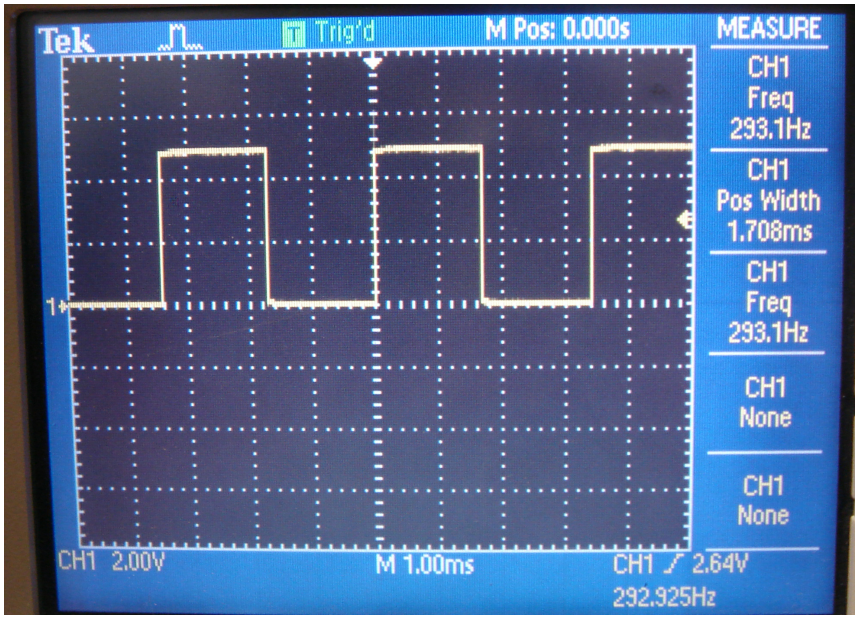
Hz	261	293	329	349
# Clocks (ideal)	19,157.09	17,064.85	15,197.57	14,326.65
A	239	243	253	239
B	8	7	6	6
# Clocks (actual)	19,165	17,050	15,215	14,375

```
; --- Piano2.asm ----
; This program plays notes C4 / D4 / E4 / F4

#include <p18f4620.inc>

; Variables

CNT0 EQU 1
CNT1 EQU 2

; Program
       org 0x800
       call Init
Loop:
       movf     PORTB,W
       btfss    STATUS,Z
       call     Toggle

       btfsc    PORTB,0
       call     Wait_C4

       btfsc    PORTB,1
       call     Wait_D4

       btfsc    PORTB,2
       call     Wait_E4

       btfsc    PORTB,3
       call     Wait_F4

       goto     Loop
```



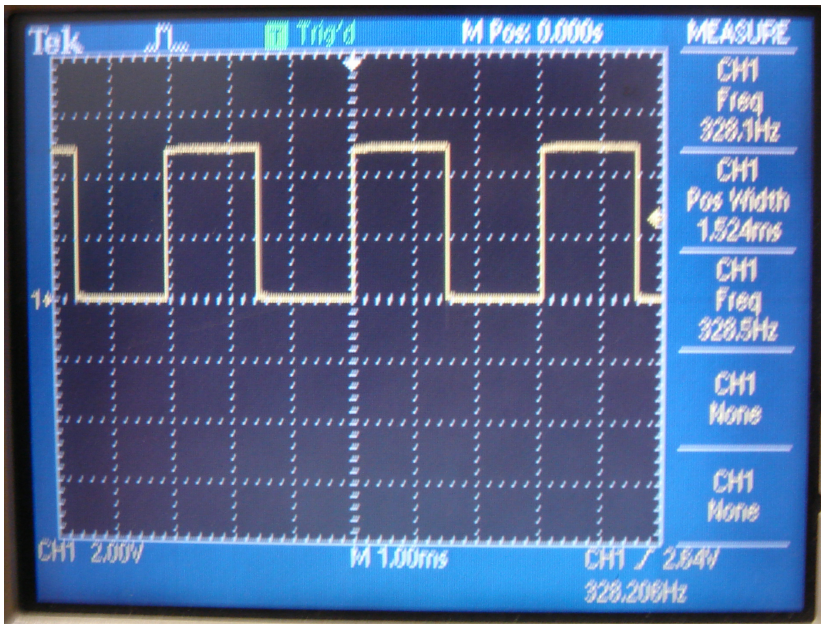
The wait routines set the delay for each note:

```
Wait_D4:                    ; Wait 17,064 clocks
        movlw   7
        movwf   CNT1
Loop1:
        movlw   243
        movwf   CNT0
Loop0:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  CNT0,F
        goto    Loop0

        decfsz  CNT1,F
        goto    Loop1

        return
```



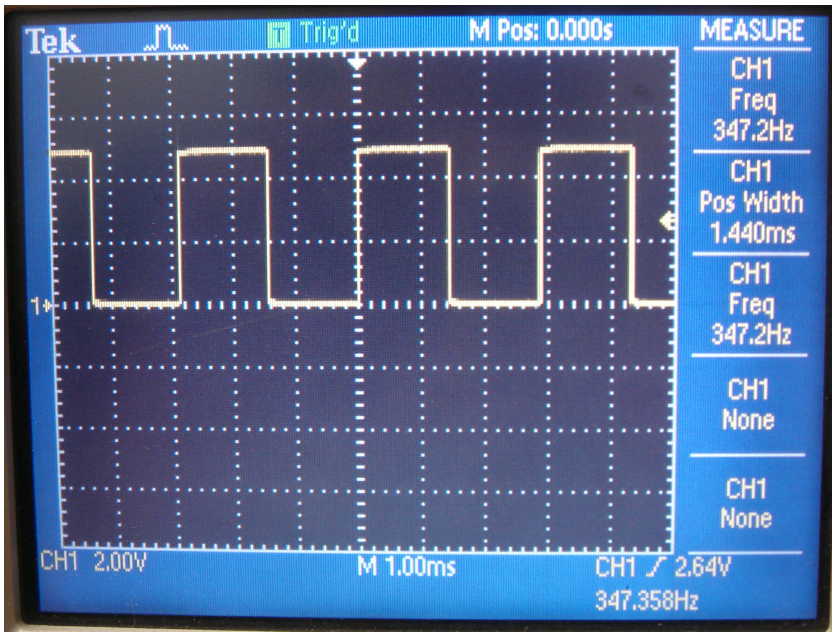
Output on RC0 for D4 (293.00Hz ideally)

```
Wait_E4:                    ; Wait 15,197 clocks
        movlw   6
        movwf   CNT1
Loop1:
        movlw   253
        movwf   CNT0
Loop0:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  CNT0,F
        goto    Loop0

        decfsz  CNT1,F
        goto    Loop1

        return
```



Output on RC0 for note E4 (329.00Hz ideally)

```
Wait_F4:                    ; Wait 14,326 clocks
        movlw   6
        movwf   CNT1
Loop1:
        movlw   239
        movwf   CNT0
Loop0:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  CNT0,F
        goto    Loop0

        decfsz  CNT1,F
        goto    Loop1

        return
```



Signal on RC0 for F4 = 349Hz (14,326 clocks / cycle = 1.4326ms)