Binary Outputs and Timing

ECE 376 Embedded Systems

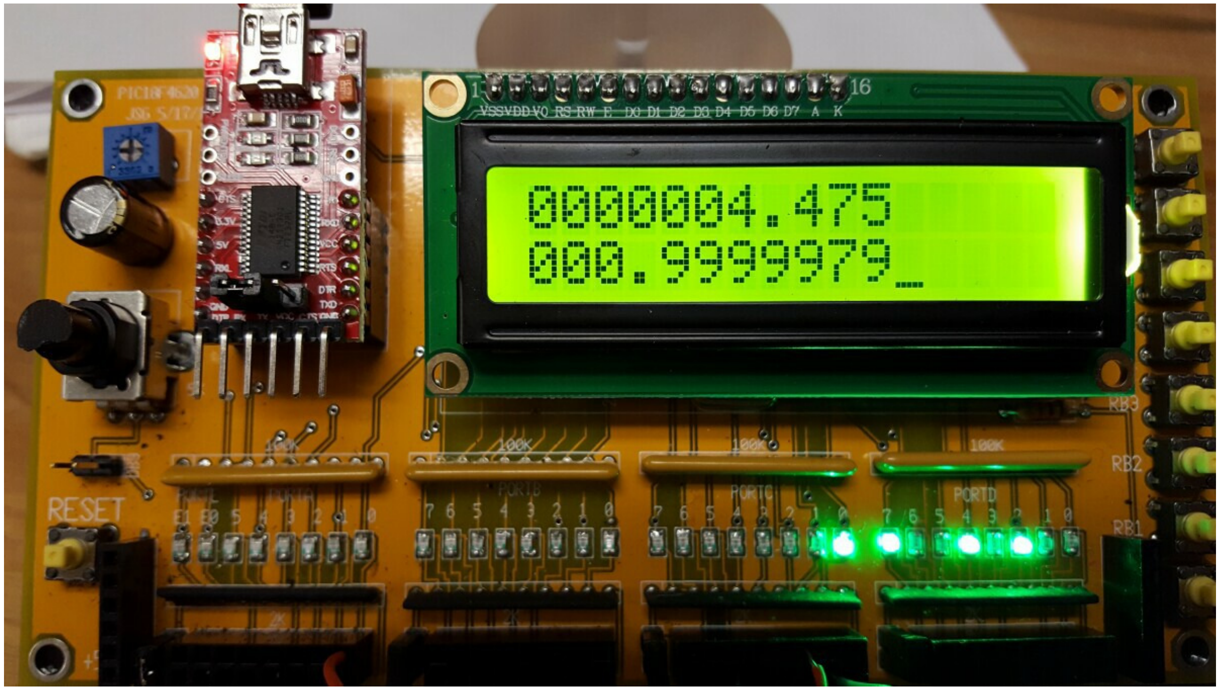
Jake Glower - Lecture #5

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

Binary Outputs and Timing

Each I/O pin can be input or output

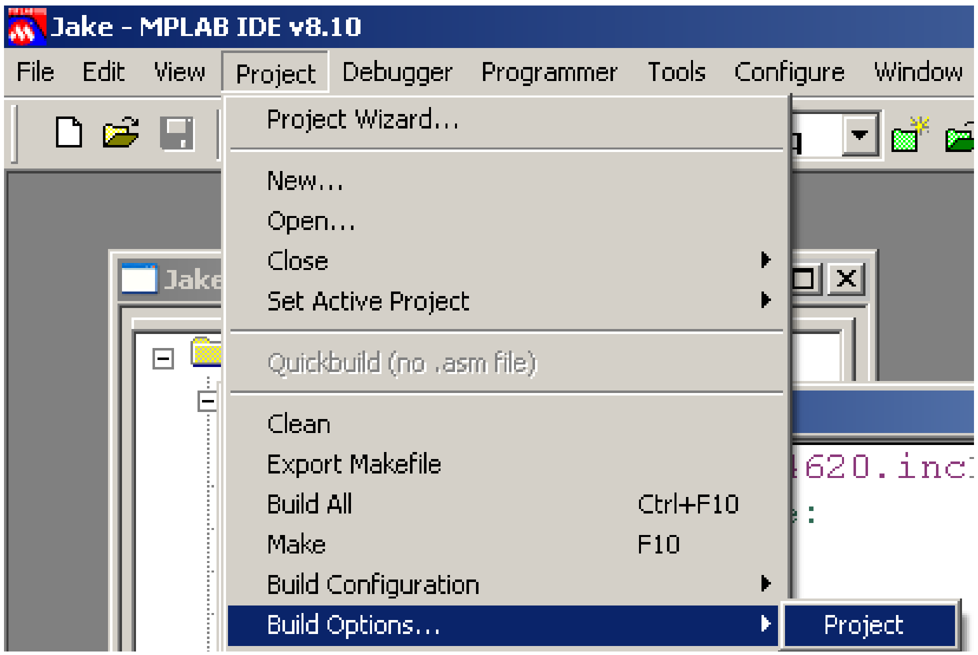
- Input:
  - High Impedance
  - 0V reads logic 0
  - 5V reads logic 1
- Output:
  - Logic 0 = 0V
  - Logic 1 = 5V
  - Capable of up to 25mA

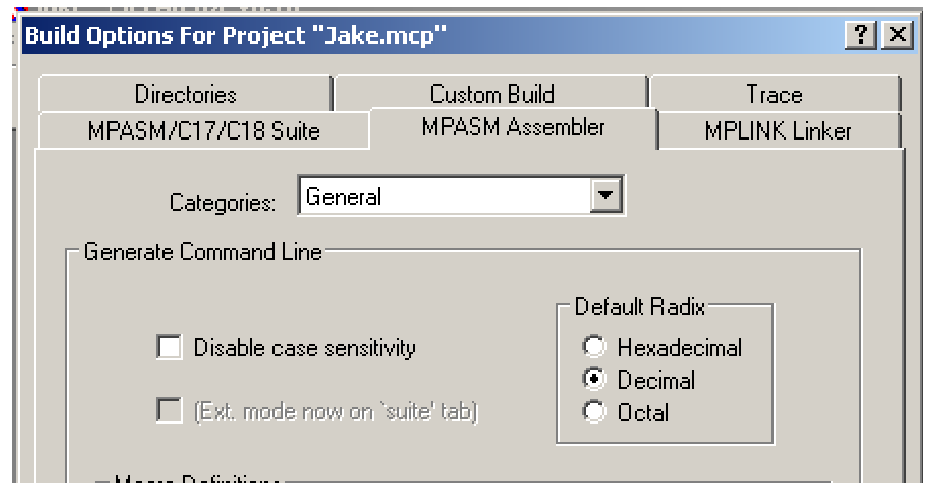


Note: Make sure default is decimal

- Timing will be off for these programs otherwise
- Project - Build Options - Project

MPASM: Decimal





Timing in Assembler

Each line of assembler takes 1 clock (100ns)

- By counting instructions, you can set the timing of a program

With what we know now, you can

- Keep track of time in seconds, or
- Output a precise frequency

Later on, you can do more than one thing at a time

- That requires the use of interrupts (a future topic)

Timing in Seconds: Binary Clock

- A clock that only engineers can read
- Binary Coded Decimal (BCD)

PORTD

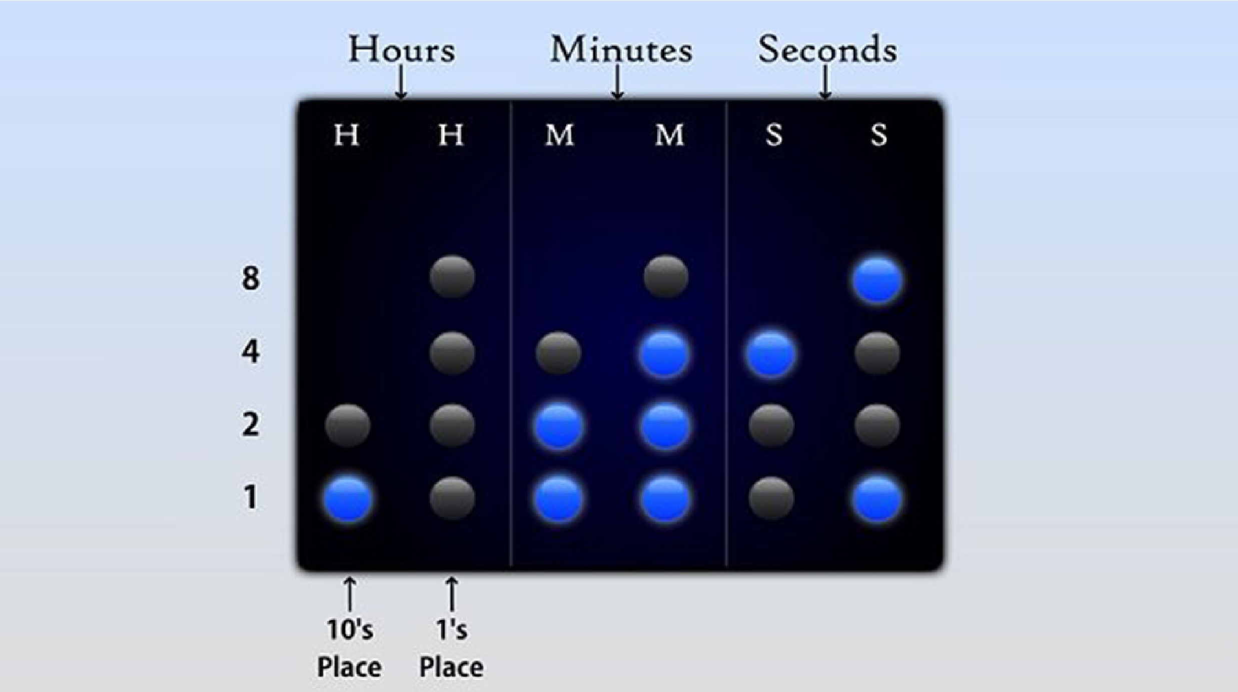
- Seconds x 1 bits 0..3
- Seconds x 10 bits 4..7

PORTC

- Minutes x 1 bits 0..3
- Minutes x 10 bits 4..7

PORTB

- Hours x 1 bits 0..3
- Hours x 10 bits 4..7

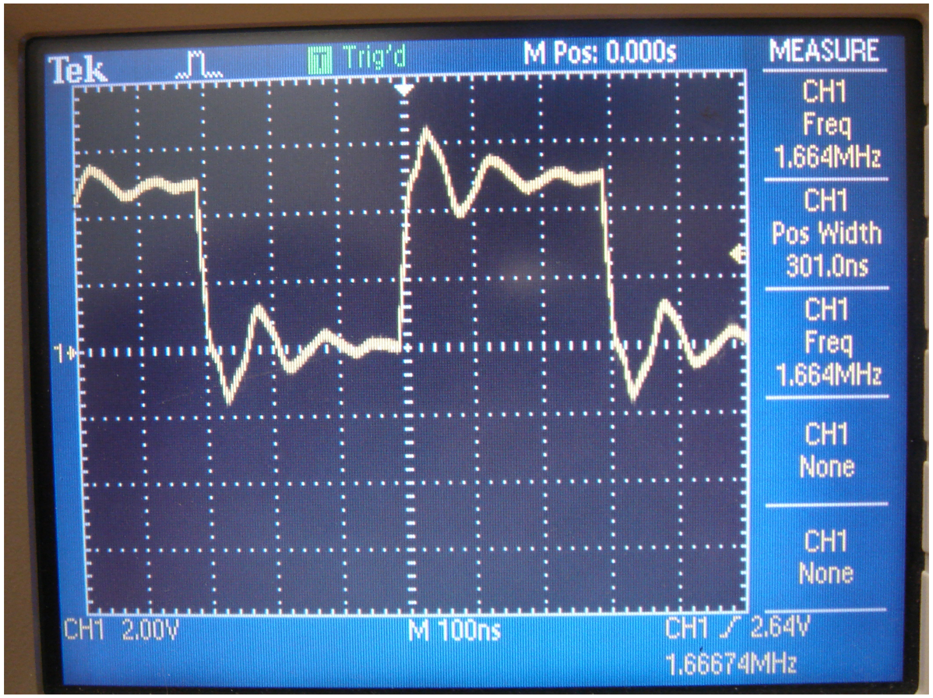


Step 1: Get it to count

Only engineers get excited when a light blinks

- Your code compiled
- Your code downloaded
- Your code is running

```
#include <p18f4620.inc>

; Start of code:
      org 0x800
      clrf TRISD
      clrf PORTD

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



Step 2: Get it to count once per second

1 second = 10,000,000 clocks

- Actual # clocks = 10,050,504 (1.0050504 seconds)

```
Wait:                                    N = 4
     movlw  100
     movwf  CNT2
Loop2:                                    N = 5 * 100
         movlw  100
         movwf  CNT1
Loop1:                                    N = 5 * 100 * 100
             movlw  200
             movwf  CNT0
Loop0:                                    N = 5 * 200 * 100 * 100
                 nop
                 nop
                 decfsz  CNT0,F
                 goto    Loop0
             decfsz CNT1,F
             goto   Loop1
         decfsz CNT2
         goto   Loop2
     return
```

Step 3: Get it to count in BCD

1st Nibble:

- PORTD bits 0..3
- Count 0..9 & repeat

2nd Nibble:

- PORTD bits 4..7
- Increment when 1st nibble get to ten

```
SEC equu 0
; Start of code:
      org 0x800
      clrf TRISD
      clrf SEC
      movlw 0x0F
      movwf ADCON1
Loop:
      incf   SEC,F
      movf   SEC,W
      andlw  0x0F
      movwf  TEMP
      movlw  10
      cpfseq TEMP
      goto   L2
      movlw  6
      addwf  SEC,F
L2:
      movff  SEC, PORTD
      call   Wait
      goto Loop
      end
```

Step 4: Get the minutes to count

- Doesn't have to be in real time
- Speed up the Wait loop for test purposes

When SEC = 60

- SEC goes back to zero
- Increment MIN

```
Loop:
        incf    SEC,F
        movf    SEC,W
        andlw   0x0F
        movwf   TEMP
        movlw   10
        cpfseq  TEMP
        goto    L2
        movlw   6
        addwf   SEC,F
L2:
        movf    0x60
        cpfseq  SEC
        goto    L3
        clrf    SEC
        incf    MIN,F
L3:
        movff   SEC, PORTD
        movff   MIN, PORTC
        call    Wait
        goto Loop
```

Timing in Seconds:
Hungry-Hungry Hippo (take 2)

Count button presses on

- RB0 (player 1)
- RB7 (player 2)
- Same as before

Start the game when RB0 = 1

- Something new

Stop counting after 10 seconds

- Something New

Concept for Timing

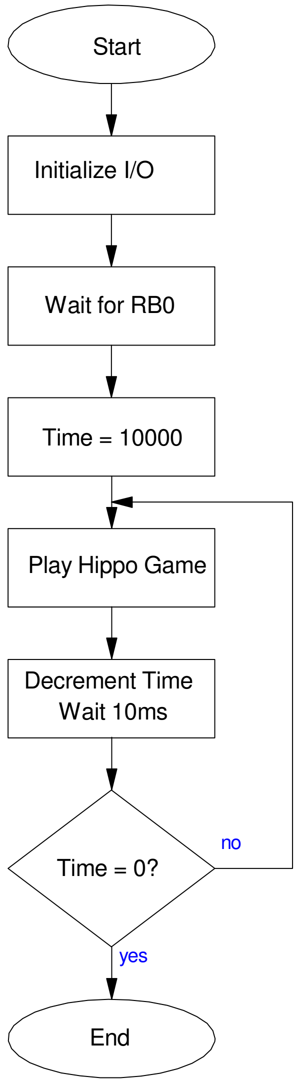
The previous *Hungry Hungry Hippo* code has 13 lines of assembler

- Takes 14 clocks (1.4us) to execute

Add a 10ms wait loop at the end

- 14 clocks for the *hungry hungry hippo* code
- 100,000 clocks for the wait loop

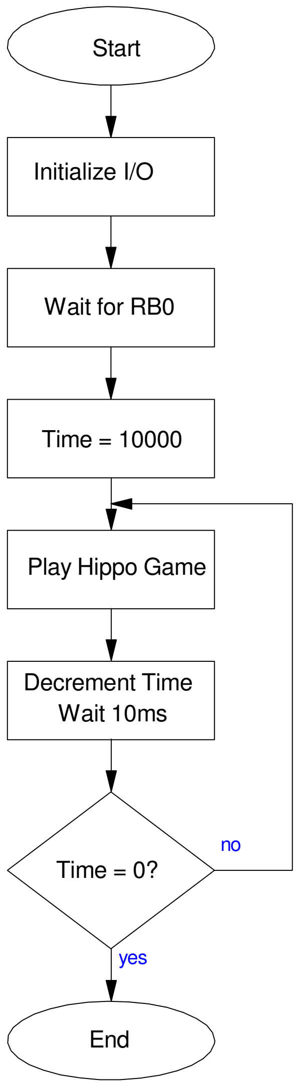
10 seconds is 1000 loops (approximately)



Assembler Coding

- Top-Down Programming

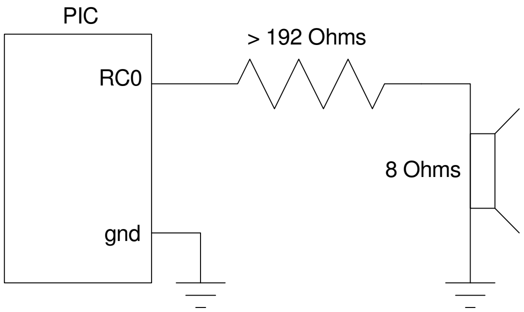
```
      org 0x800

      call  Initialize
      call  WaitForRB0
      movlw 0x03
      movwf TIMEH
      movlw 0xE8
      movwf TIMEL
Loop:
      call  Hippo
      call  DecrementTime
      call  Wait10ms
      movlw 0
      cpfseq TIMEH
      goto  Loop
      cpfseq TIMEL
      goto  Loop
End:
      goto  End
```



Playing Notes witha PIC

Hardware

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



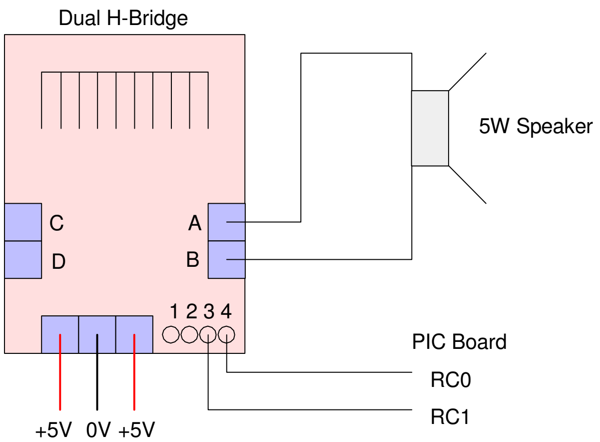
Option #1: Add 200 Ohms in series to limit the current

- max(I) = 25mA
- $R_{total} > \frac{5V}{25mA} = 200\Omega$

Option #2: Use an H-bridge (in your lab kit)

- Up to 46VDC, and
- Up to 3A (max), 2A (continuous)
- Requires a 3W speaker if using 5V

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

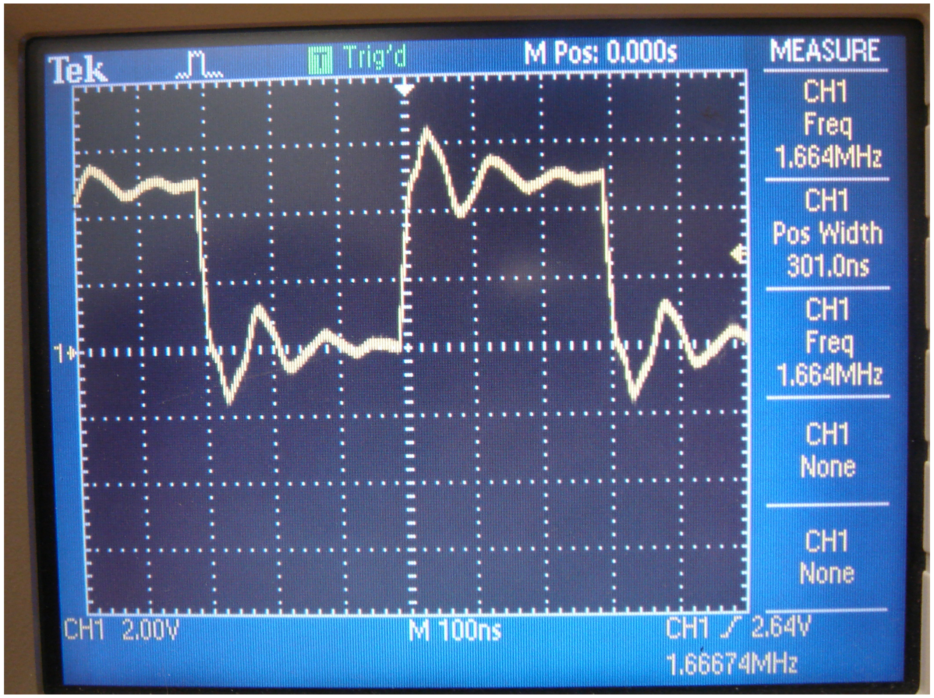


Software and Timing:

Count on PORTC really really fast

- Main loop takes 3 clocks
- 300ns / toggle
- 1.67MHz

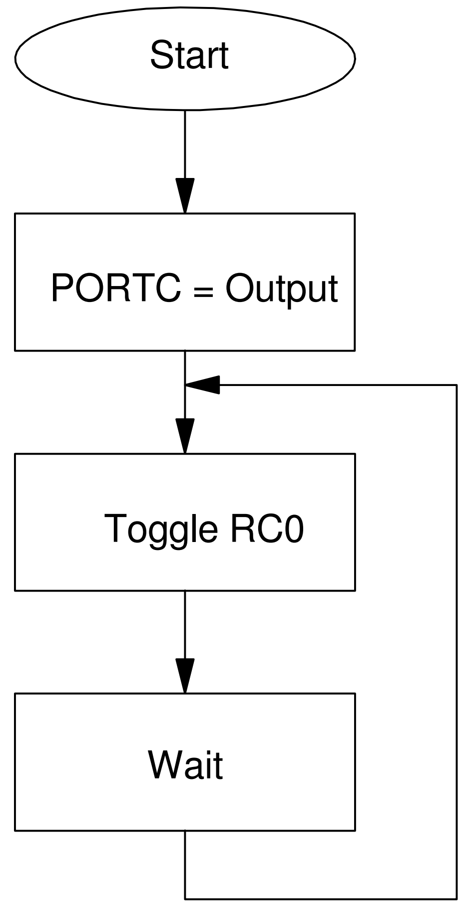
```
 #include <p18f4620.inc>

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

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



Play 261Hz on RC0

$$\# \text{ Clocks} = \frac{10{,}000{,}000}{2xHz} = 19{,}157 = N2 + N1 + N0$$

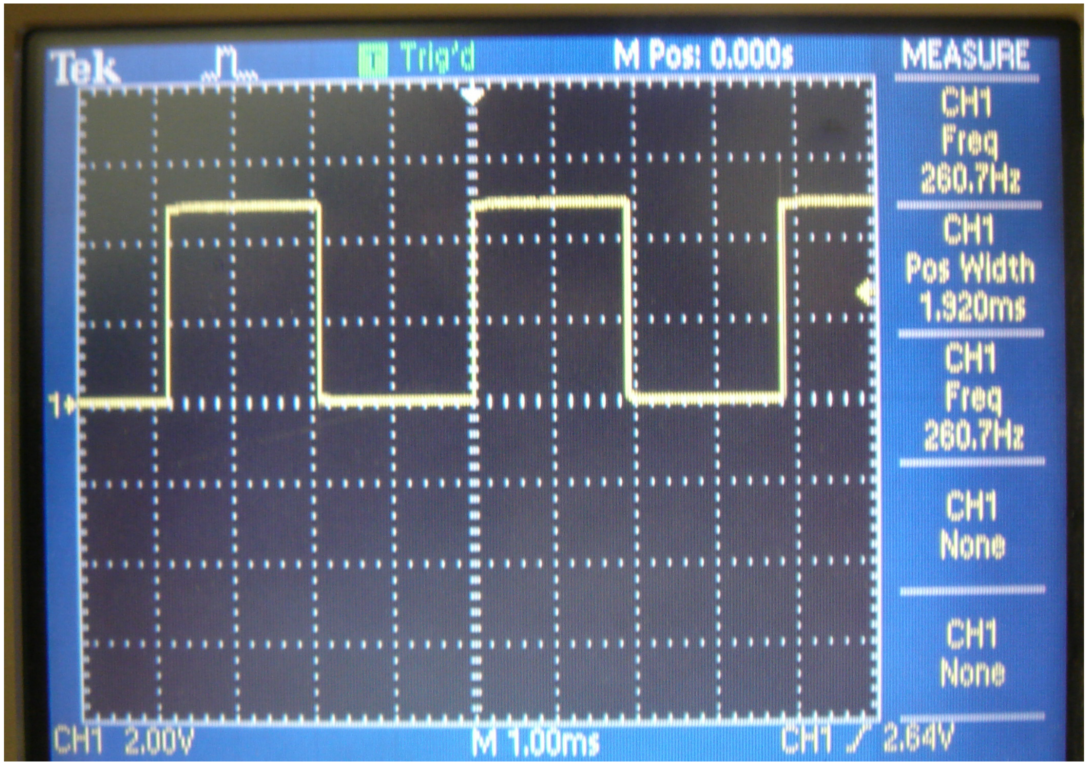
```
Wait:                          N2 = 4
     movlw   19
     movwf   CNT1
Loop1:                         N1 = 5*19
         movlw   100
         movwf   CNT0
Loop0:                         N0 = 10 * 100 * 19
             nop
             nop
             nop
             nop
             nop
             nop
             nop
             decfsz   CNT0,F
             goto     Loop0
         decfsz   CNT1,F
         goto     Loop1
     return
```



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

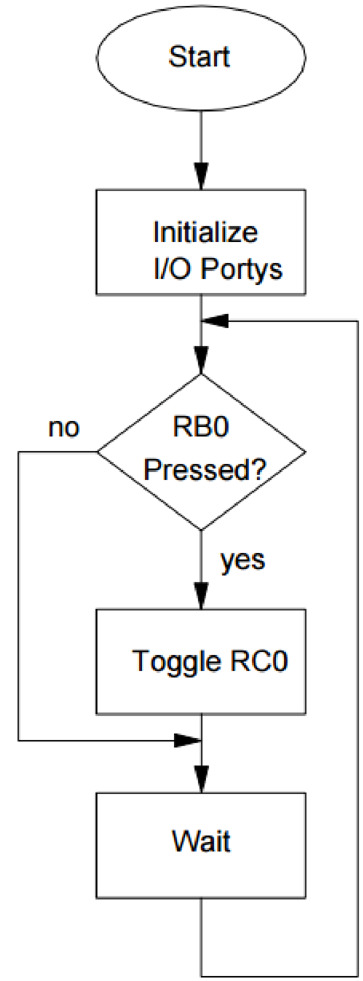
Clocks = (10*100 + 5) * 19 + 5

= 19,100 (0.29% low)



One Key Piano

```
; Play 261 Hz on RC0

#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 )
```



4-Key Piano:

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

Use four wait loops: one for each note

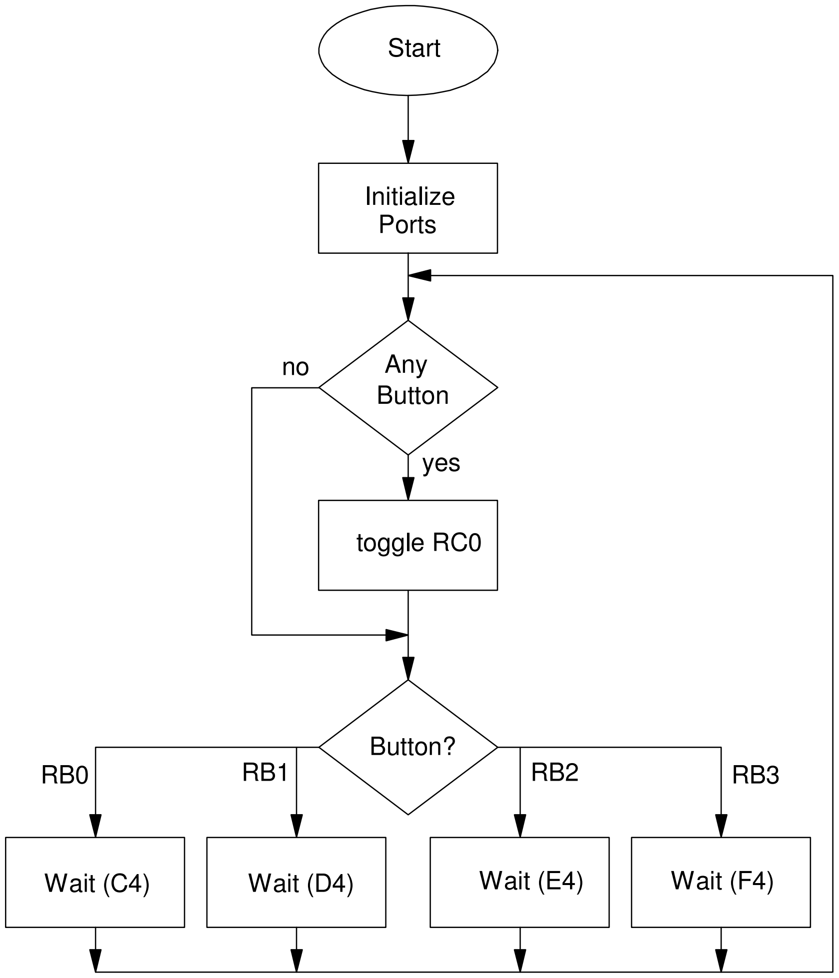
$$\# \text{ Clocks} = \frac{10{,}000{,}000}{2 \text{ x Hz}}$$

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

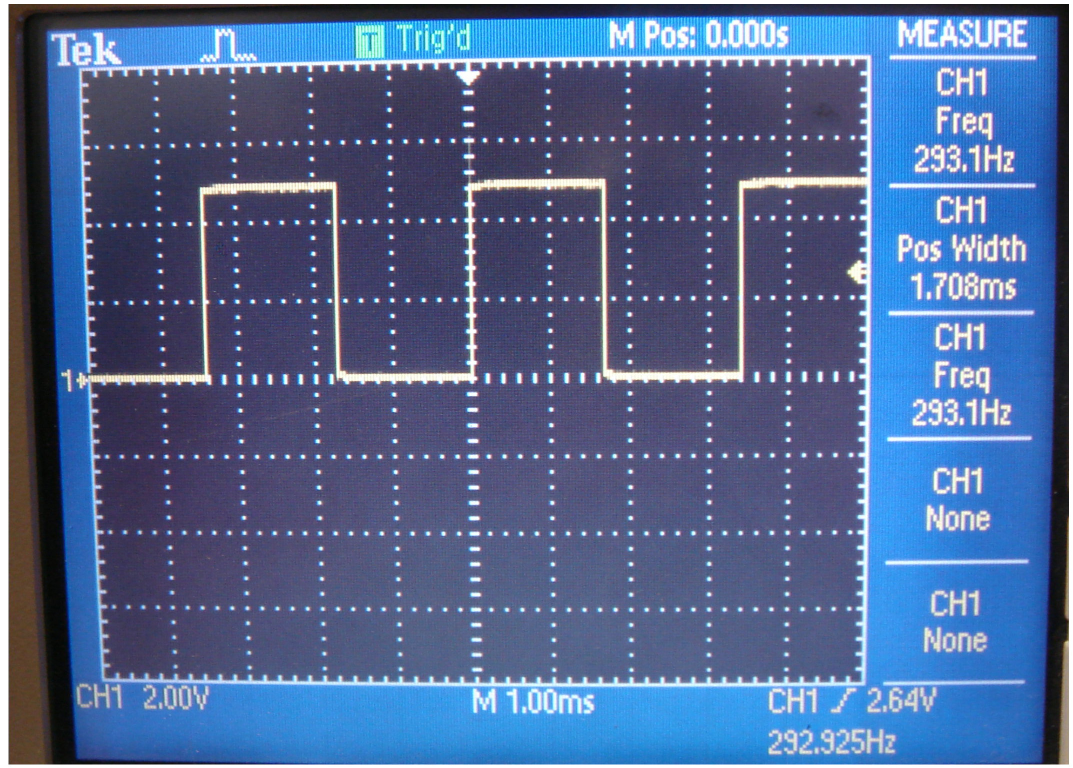
Software

```
     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
```



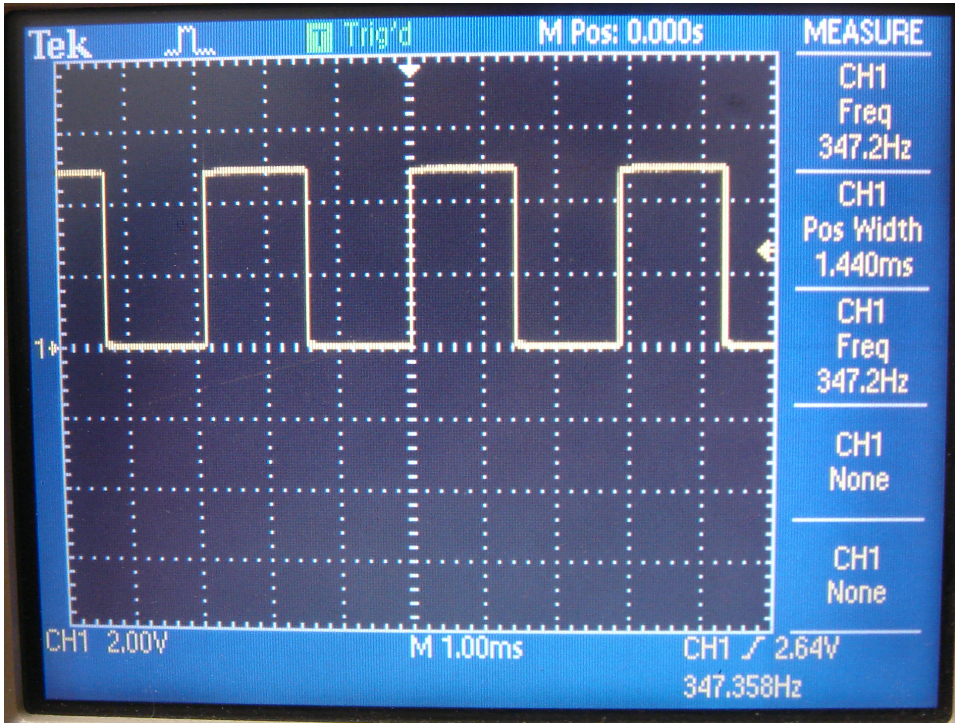
D4 (293Hz)

```
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
```



F4 (349Hz)

```
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
```



Result: 4-Key Piano

Hz	261	293	329	349
Hz (actual)	260.89	293.26	328.62	347.83
Error (%)	-0.04	0.09	-0.12	-0.34

Note:

- With assembler, you know precisely how long a program takes to execute
- If you add a couple NOP statements, you can get the timing accurate to 100ns (one clock)

Summary

Each line of assembler takes one clock

- 100ns

By counting the number of instructions, you can precisely set the time it takes a program to execute

This allows you to

- Set the executing time in seconds, or
- Output a precise frequency

With what we know now, you can only do one thing

- Only one program is running