ECE 376 - Homework #4

C Programming and LCD Displays. Due Monday, September 27th

Please make the subject "ECE 376 HW#4" if submitting homework electronically to Jacob_Glower@yahoo.com (or on blackboard)

1) Determine how many clocks the following C code takes to execute

- Compile and download the code (modify working code and replace the main loop)
- Measure the frequency you see on RC0 (toggles every loop).
  - Use an osiclloscope - or -
  - Connect a speaker to RC0 with a 200 Ohm resistor and measure the frequency with a cell phone app like Piano Tuner
  - RC1 is 1/2 the frequency of RC0, RC2 is 1/4th, RC3 = 1/8th, etc
- The number of clocks it takes to execute each loop is

$$N = \left(\frac{10{,}000{,}000}{2 \cdot Hz} \right)$$

1a) Counting mod 256

- note: if using your cell phone to measure the frequency, you might have to try different pins on PORTC until you get one in the audio range. Each pin is 1/2 the frequency of the previous pin

```
unsigned char i
while(1) {
      i = (i + 1)% 256;
      if(i == 0) PORTC += 1;
      }
```

f = 1302.8Hz

- N = 3837.89 clocks
- N/ 256 = 14.992 (15)

It takes 15 clocks to count mod 256

1b) Counting mod 255

```
unsigned char i
while(1) {
      i = (i + 1)% 255;
      if(i == 0) PORTC += 1;
      }
```

f = 41.1 Hz

- N = 121,654 clocks
- N / 255 = 477.07

It takes 477 clocks to count mod 255

1c) Integer Multiply

```
unsigned int A, B, C;
unsigned char i;
A = 0x1234;
B = 0x5678;
while(1) {
     i = (i + 1)% 256;
     if(i == 0) PORTC += 1;
     C = A*B;
     }
```

f = 42.3Hz

- N = 118,203
- N / 256 = 461.7
  - 15 clocks to count mod 256
  - plus 447 clocks to do an integer multiply

It taekes 467 (ish) clocks to do an integer multiply

1d) Floating point multiply

```
float A, B;
A = 1.0002;
B = 0.02;
while(1) {
    i = (i + 1)% 256;
    if(i == 0) PORTC += 1;
    B = B * A;
    }
```

f = 85.3Hz

- N = 58,616.6
- N / 256 = 228.97 (229 clocks)
  - 15 clocks to count mod 256
  - plus 214 clocks to do a floating point multiply

$65 Cat Nap Alarm

2) Write a C program which turns your PIC into an alarm clock with a resolution of 100ms

- On reset, TIME is set to 0 (0.0 seconds)
- RB0: When you press RB0, TIME is reset to 150 (15.0 seconds)
- Every 100ms, TIME is decremented by one, stopping at 0.0 seconds
- When TIME reaches zero, PORTA turns on for 1 second (approx)

```
// Global Variables

const unsigned char MSG0[20] = "HW4 Catnap Alarm    ";

// Subroutine Declarations
#include <pic18.h>

// Subroutines
#include        "lcd_portd.c"

// Main Routine

void main(void)
{
   unsigned int SEC;
   unsigned int i, FLAG;

   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0;
   TRISD = 0;
   TRISE = 0;
   ADCON1 = 0x0F;

   LCD_Init();                  // initialize the LCD

   SEC = 0;
   FLAG = 0;

   LCD_Move(0,0);  for (i=0; i<20; i++) LCD_Write(MSG0[i]);
   Wait_ms(70);

   while(1) {
      RC0 = !RC0;

      if(RB0) SEC = 150;

      LCD_Move(1,0);  LCD_Out(SEC, 3, 1);

      if(SEC) {
         SEC -= 1;
         if(SEC == 0) {
            PORTA = 0xFF;
            Wait_ms(1000);
            PORTA = 0;
            }
         }
      RC0 = 1;
      Wait_ms(86);
      RC0 = 0;
      }
   }
```

3) How many lines of assembler does your code compile into?

```
HI-TECH C PRO for the PIC18 MCU Family (Lite)

Summary:
    Program space        used   916h (  2326) of 10000h bytes   (  3.5%)
    Data space           used    29h (    41) of   F80h bytes   (  1.0%)
    EEPROM space         used     0h (     0) of   400h bytes   (  0.0%)
    ID Location space    used     0h (     0) of     8h nibbles (  0.0%)
    Configuration bits   used     0h (     0) of     7h words   (  0.0%)

Running this compiler in PRO mode, with Omniscient Code Generation enabled,often
produces code which is 60% smaller and at least 400% faster than inLite mode. The
HI-TECH C PRO compiler output for this code could be1382 bytes smaller and run 4
times faster.See http://microchip.htsoft.com/portal/pic18_profor more
information.
```

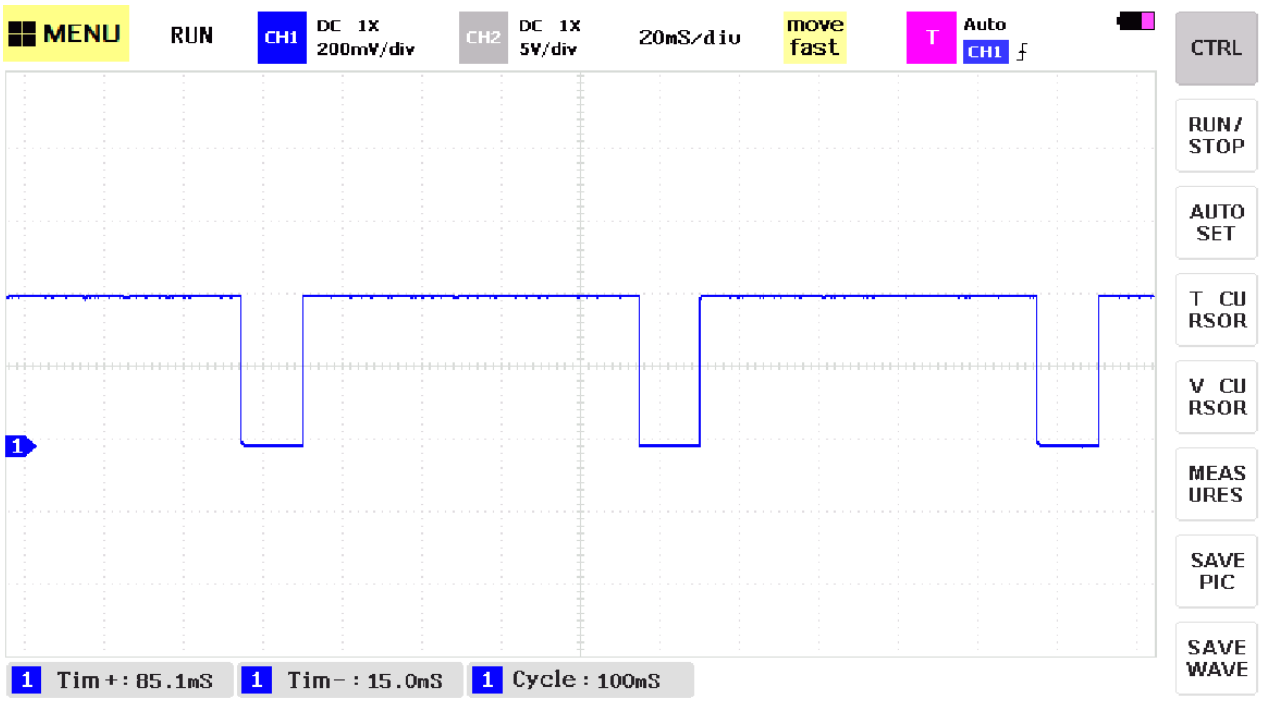
Each intruction takes 2 bytes, meaning 2326/2 assembler instrctions

ans: 1163 lines of assebler

4) Collect data to determine how accurate your program is (one count = 100ms ideally)

- RC0 = 1 for 85.1ms
  - time spent in Wait_ms(86)
- RC0 = 0 for 14.9ms
  - time for the rest of the routine
- Period = 100ms
  - each count is 100ms

Wait_ms(86) actually takes 85.1ms (



PIC Banjo

5) Requirements: Specify the inputs / outputs / how they relate.

Inputs: Buttone RB0 .. RB3

Outputs: RC0

Relationship

Play the following notes when a button is pressed

- RB0: C4 (261.63Hz)
- RB1: G4 (392.00Hz)
- RB2: B3 (246.94Hz)
- RB3: D4 (293.66Hz)

Tolerance: +/- 1%

6) C code, flow chart, and resulting number of lines of assembler

To generate a note, the following test code was used

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

   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0;
   TRISD = 0;
   TRISE = 0;
   ADCON1 = 0x0F;

   while(1) {
      if(RB0) {
         RC0 = !RC0;
         for(i=0; i<1000; i++);
         }
       }
   }
```

The results was a 312.2Hz square wave.