Task 5 **Due: See Web Page**

Task Purpose: **Timers and Interrupts – Part 1**

The purpose of the next two tasks is to develop familiarity with the use of timers and interrupts. The programming problem set up here is completely artificial.

In previous tasks, we have displayed the increment of a variable in the LED's connected to port D, and we have rotated a message in the LCD display. In tasks 5 and 6 we are going to use timers and interrupts (rather than the delays that were used in the previous tasks) to control the speed of the shifting of the LCD display and the speed of the increment of the LED display. In the first task, we will use a single interrupt and shift a message in the LCD display. In task 6, we will add a second interrupt and display a count in the LED's.

Timers:

The 18F4620 has 4 timers. These devices can be used either as timers, where the counting is based on the system clock (specifically $f_{osc}/4$), or as counters, where the count is based on the occurrence of an external event. The devices generate an interrupt when the count rolls over, going from all 1's to all 0's.

A timer typically has an 8 or 16 bit register that holds the count, and usually the signal that is being counted can be pre-scaled (divided by some power of 2). To generate an interrupt after a certain period of time, it is necessary to set up the counter with the desired pre-scale and other setup parameters, and then load a value into the count register such that the timer count will roll over in the desired amount of time.

Interrupts:

Interrupts involve both hardware issues and software issues. In hardware, we need to set all the appropriate bits to enable the desired interrupt to happen. In software, we need to write an interrupt service routine.

The compiler provides two built in functions for handling interrupts. They are:

```
      void interrupt(void)       // high priority interrupts
      void interrupt_low(void)  // low priority interrupts
```

All these function do is place the code you write at the proper location in memory (that location being the location where the microcontroller begins execution on an interrupt.) Since there are no arguments sent or received by these functions, all variable using in them must be global. (All the register names, defined in the header file, are global variables.) It is sufficient to only use high priority interrupts (the default.)

Semaphores:

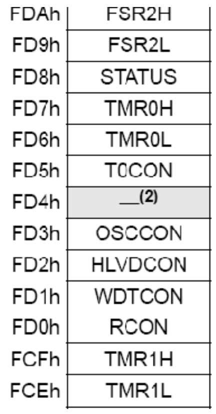
In general, you want to, as much as possible, the amount of code that is part of your interrupt service routine. There is an additional limitation that comes from our compiler, in that you cannot have calls to the same function from two different execution threads.

Simply put, this means that any functions that are called in the non-interrupt service code cannot be called in the interrupt service routine.

A simple solution to minimizing the code required in the interrupt service routine is to use a signal called a **semaphore** that is set in the interrupt service routine to signal the code in the main program to do something. Once the main program does the desired actions, the semaphore is cleared and thus the condition won't occur until the next time there is an interrupt.

Programming Trick:

We will be using timer 0 in this task, which can be configured as a 16 bit counter. The two 8 bit variables that are declared in the header file are **`tmr0h`** and **`tmr0l`**, holding the high and low byte of the 16 bit count respectively. It would be nice if we could deal with these two registers as a single variable. In some circumstances, including this one, we can. Looking at a portion of the memory map of the 4620, we see that these two values are located in sequential locations in memory, and that the lower byte is at a lower address in memory. When 16 bit values are stored in our compiler, the low byte is stored at the lower memory location, so we can overlay a 16 bit variable over these two 8 bit variables and deal with this as a single variable. Either of the two statements that follow will create a 16 bit variable called tmr0_reg that overlays the two 8 bit registers of timer 0 in the correct way.



FDAh	FSR2H
FD9h	FSR2L
FD8h	STATUS
FD7h	TMR0H
FD6h	TMR0L
FD5h	T0CON
FD4h	—(2)
FD3h	OSCCON
FD2h	HLVDCON
FD1h	WDTCON
FD0h	RCON
FCFh	TMR1H
FCEh	TMR1L

```
unsigned short tmr0_reg@0xFD6;
unsigned short tmr0_reg@TMR0L;
```

Program Details:

In this task, you will be writing an interrupt driven program that will shift the contents of the LCD display once per second.

1. Using the 18F4620 data sheet, *list the appropriate settings (registers and bits within those registers) so that* **`timer0`** *will provide an interrupt every second.* Note that this involves **`timer0`** settings, as well as the value that is loaded into the **`timer0`** count registers.

2. Using the 18F4620 data sheet, *list the appropriate settings so that* **`timer0`** *is able to interrupt the processor.*

3. Write a program that will write a message to the LCD display, and then shift the LCD display once per second, using **`timer0`** and interrupts.

4. Verify that you program works correctly. Note that if you are having trouble, you might consider using the toggle of a bit and the logic analyzer to see, for example, whether you are getting an interrupt.

Report:

In addition to the answers to the questions posed in this task, please include a listing of the software you have written as part of this task in your task report.

Please remember to include the team name and the task number in the file name of the report.