Lab 11: Interrupt Driven Serial I/O

Introduction
In this lab, you will modify the lab from Lab 6, so that you can display output on a terminal. Unlike Lab 7, you will use interrupt driven serial I/O, all output will go the terminal as well as the LCD, and you can control states from the terminal.

Terminal Display
Use the hyperterminal program on Windows to connect to the board. You can start hyperterminal, by going to Start->Run and typing hypertrm. You can create a new connection with a 19200 baud rate and 8-bit data, no parity, and 1 stop bit.

To enable display of data on the terminal, you will need to provide complete the three routines: SerialChar(char c), SerialString(const char *s), and SerialDecimal(unsigned short d). The template for these files are provided in the serial.c file which is on the website. The file also provides routines to set the cursor position and clear the display. You will need to call these functions when needed.

Terminal Input
In previous labs, you transitioned states by checking for the ENTER button press. In this lab, you will also add the ability to change states by typing a number at the terminal. For example, a ‘1’ would put you in state 1 – display of name.

Interrupt driven serial I/O
The serial port is controlled by a device called the USART (Universal Synchronous Asynchronous Receiver Transmitter). This sometimes known as a UART for non-synchronous serial communication or SCI (Serial Communication Interface). In most computer systems the USART is a separate module on the motherboard, but most microcontrollers will include at least one and sometimes several USARTs on the microcontroller itself. The PIC16F874 that we are using has a single built-in USART with the transmit and receive ports connected to pins RC6 and RC7 respectively.

Set up the USART as you did in Lab 7. You will also need to enable the UART transmit and receive interrupts.

One of the reasons to use interrupts, is that transmitting or receiving characters takes a very long time. At 19200 baud, each character takes roughly 0.521 ms to transmit. In that time, the PIC microcontroller could execute almost a 1000 instructions. So waiting around for the character wastes a lot of processing power.
When using interrupts, we would like to use what is called a circular buffer. A circular buffer has an array with two indexes – the start and the end. When the start and end indices are the same, the buffer is empty. If they are different, there is data in the buffer. When we want to transmit data, we put data into the transmit circular buffer and advance the end index appropriately. When we get a TXIF interrupt, indicating that the USART is ready to transmit, the interrupt code will take the character from the start of the circular buffer and move the start index forward one. If there are no characters, to transmit turn off TX interrupts until we put a new character in the circular buffer. When we get an interrupt do to a receive, the interrupt code will put the data into the receive circular buffer.

New variables

```c
char rcbuf[RCBUFSZ], rcrent, rced;
char txbuf[TXBUFSZ], txstart, txend;
```

Discussion and Conclusions

Keep all other functions operational. Thus BarChart, BarIntensity, DisplayV, DisplayTemp should continue to operate. If your code exhibits difficulties, set up a watch window and use the ICD to debug.

Demonstrate your project and hand in your well-commented C code by the end of class.