**Goals:**
The goal of this lab is to gain a better understanding of the port registers in a microcontroller.

**Equipment Usage:**
For this lab the following equipment will be used:

- LEDATtiny2313
- AVR Studio 5
- AVR RISP mkII
- 8pin Dip switch
- 10 bit LED Bar

**Background:**

**Time Delay:**
Creating a delay in code requires proper calculation of each instruction set. For example, each instruction in assembly usually takes one machine cycle to execute. The length of a single machine cycle in the real world is equal to 1/frequency of the clock. For example, a microcontroller running at 1MHz has machine cycles that take 1us to execute in real time.

```
LDI R16, 45 ; 1 machine cycle
ADD R17, R16 ; 1 machine cycle
NOP ; 1 machine cycle
```

The above code takes 3 machine cycles to execute, which means it takes 2us for them to execute once placed on a chip. However, it is crucial to realize that not every instruction executes at the same speed. Consider the branch commands (**BRNE**) which take 2 cycles to execute. It is crucial to be aware of how long each instruction takes to execute; fortunately this can be monitored during most debugging processes by watching the changing value of the clock cycle.

The third command above is called the **NOP**. This command essentially is used to create a delay since it performs **no operation**. By using NOP you can create a simple delay sequence. Another technique you may already know of is using the internal timers of the clock to generate a counting sequence to create the desired delay time. In any even either method is fine. If you desired to create a 1 second delay using a 1MHz clock you could create an infinite loop of NOPs that runs for 1 million cycles or you could create a timing scheme using different modes such as Clear Timer on Compare to achieve the same result. The only real consideration here is preference. Generally NOP is used when you need a few clock cycles in-between instructions because you are waiting for some trigger. Using the different timing schemes becomes useful when doing more complex timing situations.
Port Control
By now you should be familiar with outputting signals through a microcontroller. However, being able to read in signals is just as crucial as being able to send out signals. By manipulating the input and output (I/O) ports of the microcontroller you can achieve a higher level of interaction with external components.

Microcontrollers often have ports that are used reading input and outputs. For proper function the selected ports are labeled (A, B, C, D), and they must be declared as inputs or outputs. **BY DEFAULT MOST MICRONTROLLERS ARE SET FOR INPUT.** This means that if you made no changes you could read in signals connected to the microcontroller ports. Take the given code:

```
LDI R20, 0x24 ; store 0010 0100
OUT DDRB, R20 ; set PB2 and PB5 to output (all other pins set to input)
```

The first line declares which pins will be set for output (1), but it also declares which pins are set for input (0). On the next line we take the stored value in register 20 (R20) and send it to the port map. This sets pins PB2 and PB5 for output meaning we can send a signal out. This line also sets PB7, PB6, PB4, PB3, PB1, and PB0 all to input. Once you have declared which pins are used for input and output you must use the proper commands to read and write to these pins.

```
IN R21, PINB ; read in value from PB pins and store it in R21
OUT PORTB, R22 ; sends the value in R22 to the PORTB pins
```

**IMPORTANT:** When reading (in) or writing (out) to a port the pin status (input or output) will take priority over the command. For example, attempting to read and output pin will fail because the output signal it sends off will take priority. There for if you attempted to place a low signal on an output pin and it was giving off a high signal, reading the pin will report a high signal on the pin regardless of what any external device attempts to send in.

**Prelab:**

**Design 1:** write a small slice of code that will read the value of PB2 and PB5 and store these values into R26.

**Design 2:** Write a small slice of code that will send a high output signal to PB2 wait 5 clock cycles then send a low output signal to PB2.

**HINT:** You can use the I/O view in the simulator and place values on the port by clicking the boxes in the PINB row. This place a simulated value to be read but the IN commands.
Lab Experiments:
For this lab you must read in the inputs that are displayed on an 8 pin Dip and send them to the LED bar. Useful Assembly commands (NEG, INC, DEC, RJMP)

Experiment 1: Write code to read in the values on the 8 pin DIP switch and display the data out to the LED Bar.

Experiment 2: Building off the code from experiment 1, make the LED Bar count to 255 using the input from the DIP switch as the starting value. Make sure there is some delay so that the count is visible.

Post-Lab Deliverables:
1) Submit your working code along with a screenshot of the I/O registers.
2) PCB Layout and Netlist today’s circuit (LED Bar, Dip switch and Microcontroller)
3) Answer the following questions
   a. In previous labs you used the SBI command instead of the OUT command when dealing with enabling ports. Explain the difference between the two and the benefits of using one over the other.
   b. Today's lab focused on reading, writing and précising timing in a microcontroller. Give an example of a real world application where using a microcontroller as you did in today's lab would make sense.
   c. Number conversion: Convert these numbers( 80, 65, 73,78) into
      i. Binary equivalent
      ii. Hex equivalent
      iii. 2’s compliment (negative in binary)