

CSC211 Laboratory8

Background Information

See the Laboratory 7 handout if you need a refresher on working with the AVR's or STK500 boards. In particular, it contains step-by-step instructions for programming the microcontroller in a Linux environment. Remember that you will need to modify the Makefile, so that it specifies the correct source file name, for each program that you write.

Remember also that the LEDs on the STK500 light when given a logical 0, not when given a logical 1. Note that the STK500 switches work the same way: when the button is pressed it sends a 0 to the AVR, when not pressed it sends a 1.

Laboratory Exercises

1. Write an assembly program that mirrors the input received from the bank of switches on the STK500 onto the bank of LEDs. In other words, if a switch is pressed, the corresponding LED should light up; otherwise, it should not.
2. Return to your program from the last laboratory (blink.asm) and either modify it or produce a second version as follows. This time, instead of controlling the delay between your two LED patterns with a triply-nested loop, implement a 3-byte number (using 3 registers in tandem) that you can use as a single loop counter.
3. Write a program that blinks the LEDs at two different speeds in response to input from the switches. If any of the switches are pressed, the LEDs should blink at one speed; otherwise, they should blink at the other speed. This program should also use a 3-byte counter to control the length of the delay.
4. Yearn for the ability to implement your delay as a function and pass in the delay time as an argument.
5. **For those with extra time:**

Write a program that again blinks lights, but this time the program should rotate through more than two LED patterns. (For instance, you might start with just the leftmost light on, then add the next leftmost, etc, until all lights are on, and then remove the lights one by one until all are off again.) To do this, load the sequence of light patterns you will need into sequential memory locations in SRAM. Then use a loop that repeatedly reads a pattern from memory, displays it, and delays.

Hint: Remember that you can use registers X, Y, and Z to store memory addresses, and you can use an instruction similar to the following to both read a value from the address pointed to by X, and increment the value in X.

```
LD r16, X+
```