CCPS310 Lab 4 – AVR I

Preamble

This lab is intended to introduce you to the Arduino environment and get you started writing AVR assembly. We'll be using a mix of inline assembly and Processing code in the Arduino IDE. For this lab, and all upcoming AVR labs, you may work in groups of ***1-3 people***.

Lab Description

First thing's first, download and install the Arduino IDE:

https://www.arduino.cc/en/main/software

As of this writing, the latest version is 1.8.12. Just get the latest version for your platform from the above website.

0) Start by trying out some of the sample programs that come with the Arduino software. They are all written in the *Processing* language. We'll be using Processing for some utilities like **`delay()`** and **`Serial.print()`** in the spirit of keeping things fun instead of tedious.

The sample programs can be found under File->Examples. Explore, try out some of the simpler ones, make sure your board is connected and working properly. The Arduino software should work out of the box without any trouble.

Examples->Basics->Blink is a nice one to start with. It simply blinks an LED periodically on the board. If this works, you're good to go.

1) Now for the actual lab assignment. Starting with the Blink example as a template, we're going to convert the Processing statements into inline AVR assembly. Inline assembly allows us to mix Processing code and assembly in the same program while remaining in the Arduino IDE. Below is the same Blink program, but with **pinMode()** and **digitalWrite()** function calls replaced with inline assembly. Try this out on your own board. The LED should blink as before.

Notice we've kept the **`delay()`** function as-is. This is a handy part of inline assembly. Rather than making you implement your own delay loop, we'll just make use of the high level function call.

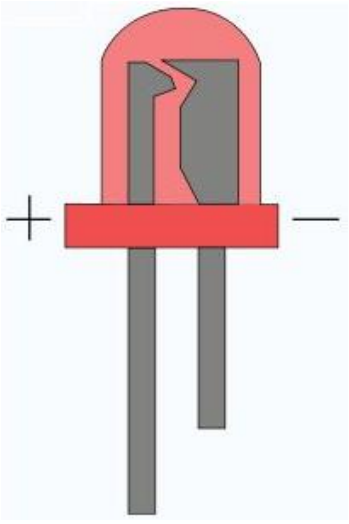
```
void setup() {
  asm("sbi 0x04, 5");
  asm("rjmp start");
}

void loop()
{
  asm(" start:        ");
  asm(" sbi 0x05, 5   ");
  delay(200);
  asm(" cbi 0x05, 5   ");
  delay(200);
  asm(" rjmp start    ");
}
```

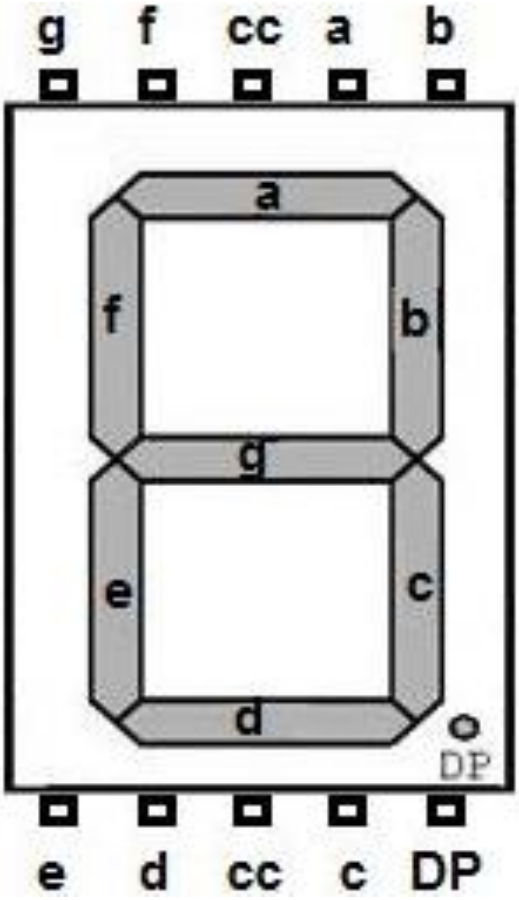
Inline assembly instructions are passed as strings to the **`asm()`** function. This is annoying, but worth it for the added benefit of being able to develop in the Arduino IDE.

Look up the instructions **`sbi`** and **`cbi`** in the instruction set to see what they do. Additionally, look up the registers at address 0x04 and 0x05 to see what bits we're setting and clearing.

2) The above program blinks an LED built into the Arduino board. That LED is connected to PORTB pin 5. Using the data sheet and the Arduino pinout (can also be found in the slides), determine the corresponding output port on the UNO. Connect a separate LED to this pin using your breadboard. Use a 1k-ohm resistor between the positive LED wire and the port on the UNO. Connect the other end of the LED to one of the ground (GND) ports. Without changing any of your code or even turning your Arduino off, the LED should blink just like the one on the board.



3) The 7-segment display that came with your kit is nothing more than a set of LEDs, each of which can be connected to an output pin on the Arduino. Here, CC means *common cathode*. The cathode is connected to GND, and each of the pins **`a-g`** connect to an output port on the Arduino (through a 1k-ohm resistor, as before). On your breadboard, connect each of the LEDs **`a-g`** to ports **`PD0-PD6`** (you can ignore DP).



4) Once the 7-segment display is connected, modify your assembly program to blink the number '3' on the display. Use inline assembly for this! You'll have to use the data sheet to determine the corresponding addresses of pins **`PD0-PD6`**.

5) Finally, modify your program to not just blink the number 3, but rather cycle through the numbers 0-9 and then loop back again at a rate of one increment per second (**`delay(1000)`**). You're not using any sort of addition for this – Just set or clear the appropriate bits for each number. Don't forget to set the appropriate bits in the data direction register in **`setup()`** as well!

Submission

Labs are to be submitted ***in groups of 1-3***! If working in a group, only one person should submit. Clearly indicate in the submission the names of all group members. Submit the following files under Lab #4 on D2L:

- A picture of your UNO and breadboard with the 7-Segment display. You don't have to include your face or anything like that, I just want proof that you physically wired up the 7-segment display.
- Your source code for question 5 as a plain text (.txt) or assembly source (.asm) file.

I will be testing your submission on my own UNO with the display wired up in the same manner. It's critical that you're using the correct pins indicated in the lab, as that's the configuration I'll be testing on.