CCPS310 Lab 5 – AVR II

Preamble

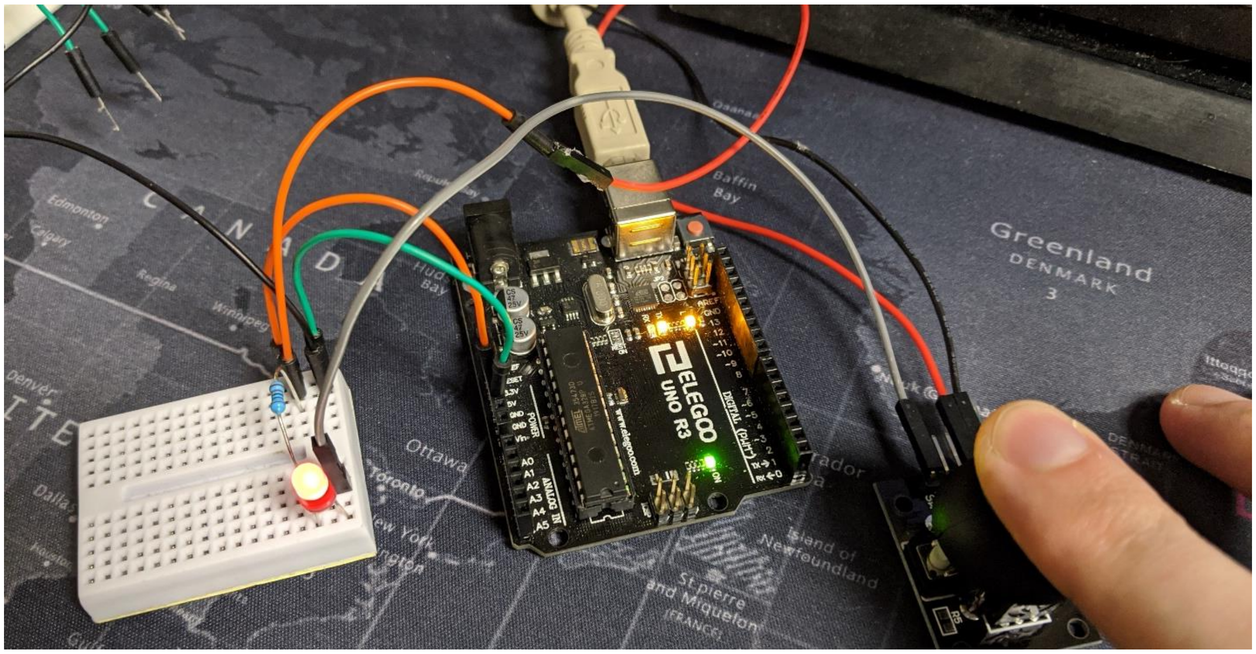
This lab is intended to introduce you to branching, digital input, and analog input in AVR. Once again we'll mix Processing and assembly (and a bit of C). You will connect the joystick to the analog-to-digital converters, and use branching to turn on LEDs based on the joystick's position.

Lab Description

1) First, we want to create a circuit. Start by wiring up an LED of your choosing on your breadboard. Make sure to use a resistor, just like you did in lab 1. Nothing new here. Connect the LED to +5V through the resistor, but don't ground it just yet.

Next, we'll connect the Joystick that came with your kits. The Joystick has 5 pins: GND can be connected to GND on the Arduino, +5V can be connected to 5V, and SW should be connected to the GND end of the LED on the breadboard. What is SW? The joystick can be pressed like a button. When pressed, SW will be LOW. This means SW can act as GND when the joystick is pressed. Don't worry about VRx and VRy just yet.

When the button is pressed, current goes through the resistor, through the LED, and into the SW pin of the joystick. If wired correctly, pressing the button should turn your LED on. We don't even need to upload a program to achieve this.



2) We don’t want to control the LED with the button directly. We want to connect SW to a digital **input**, read that digital input in our assembly program, and turn the LED on or off accordingly.

Start by connecting the LED to pin 6 of PORTD, and setting that pin for output just like we did in lab 1. The joystick stays connected to +5V and GND as before, but now we’re going to connect SW to pin 7 of PORTD and configure that pin for ***intput***. See the sample **`setup()`** function below. For a more complete description of what these assembly statements are doing, see the slides.

```
void setup()
{
   asm("sbi 0x0A, 6"); // Set bit 6 in PORTD for output
   asm("cbi 0x0A, 7"); // Clear bit 7 in PORTD for input
   asm("sbi 0x0B, 7"); // Set pullup resistor for pin 7 in PORTD
}
```

In your **`loop()`**, we need branching logic for turning the LED on or off as a result of the button press. Analyze **`loop()`** below, understand what it does. Once again, for more details, consult the slides.

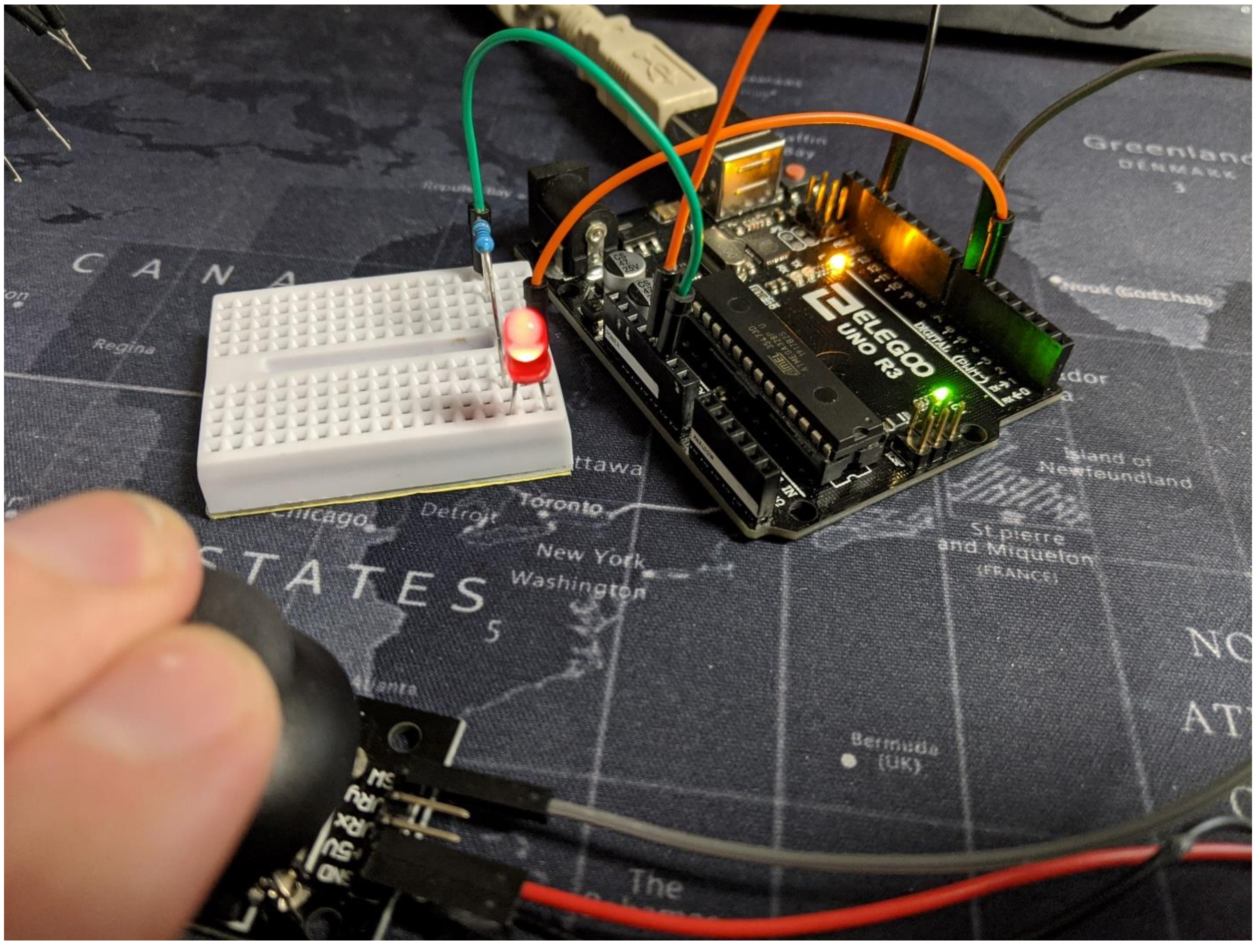
```
void loop()
{
   asm(" in   r25, 0x09  ");  // Load PIND into r25
   asm(" andi r25, 0x80  ");  // Bit-mask the 7th bit
   asm(" cpi  r25, 0x80  ");  // Compare with 0x80
   asm(" brne click_on   ");  // If not zero, branch to click_on

   asm(" cbi  0x0B, 6    ");  // Turn off LED
   asm(" rjmp end        ");

   asm(" click_on:       ");
   asm(" sbi 0x0B, 6     ");  // Turn on LED

   asm(" end:            ");
   delay(2);
}
```

Wire up your circuit and upload the above program. Your joystick should now turn the LED on and off via the button press.



3) So far so good? Good. Time to make use of those VRx and VRy pins. Both of these pins will produce a voltage from 0-5V, depending on the position of the joystick along the X and Y axis respectively. Connect VRx to pin A0 on the Arduino. Connect VRy to pin A1. A0 and A1, when properly configured, will be used to read the voltage coming in and produce a value between 0 and 255.

We'll start with an example using a mix of C and Processing code. This will give you an idea of how the output works. We will use the Serial monitor to print the values on each axis. Once again, we cover this code in greater detail in the slides.

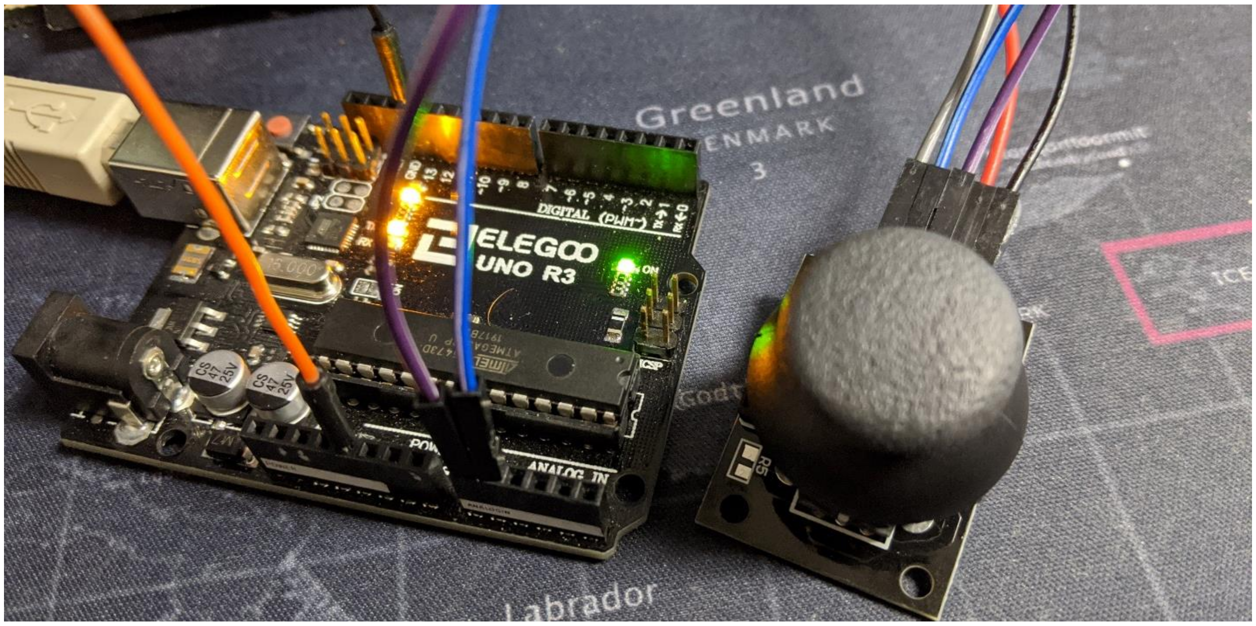
```
void setup()
{
   Serial.begin(9600);
   ADCSRA |= (1 << ADEN);  // enable ADC
   ADMUX  |= (1 << REFS0); // Vref AVcc page 255
   ADMUX  |= (1 << ADLAR); // Left justified output for 8bit mode
   ADCSRA |= (1 << ADSC);  // start conversion
}
```

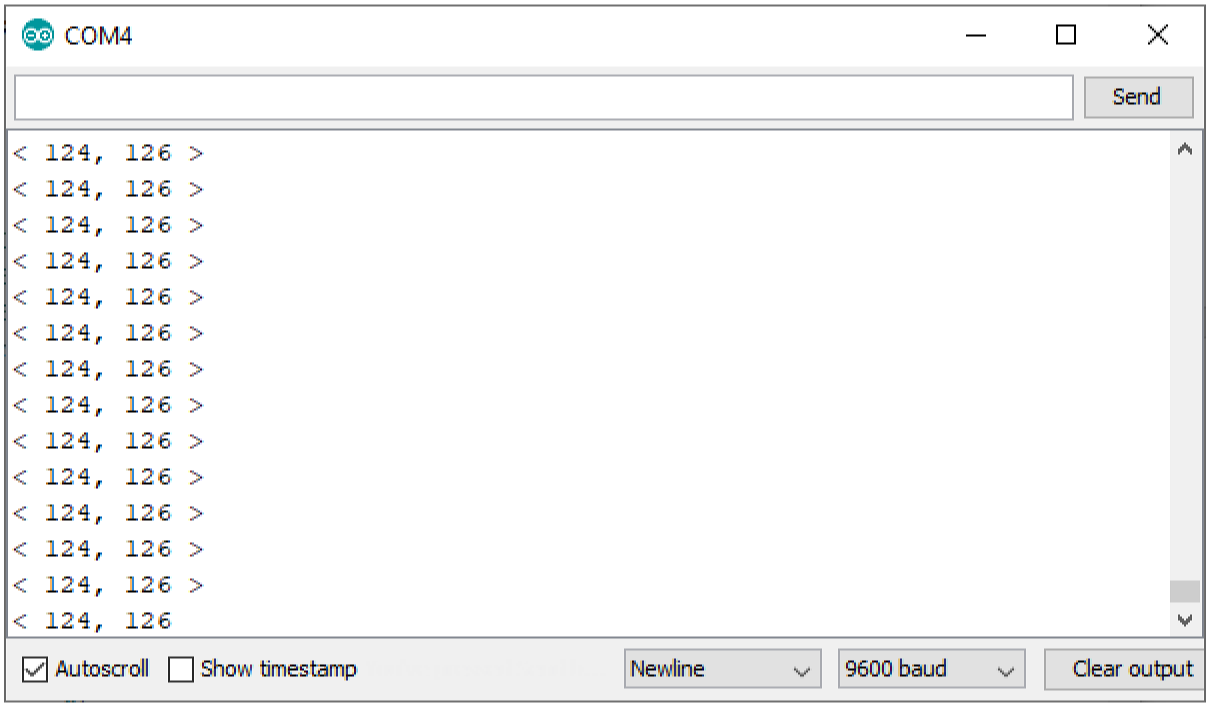
```
uint8_t analog8( uint8_t channel )
{
   ADMUX  &= 0xF0;     //Clearing the last 4 bits of ADMUX
   ADMUX  |= channel;  //Selecting channel
   ADCSRA |= (1 << ADSC);
   while ( ADCSRA & ( 1 << ADSC ) );
   return ADCH;
}
```

This C-style function accepts one argument – The ADC channel to read from. It returns an 8-bit value between 0 and 255. In `loop()`, we print the value to the serial port. Notice we're not including the button press in this circuit, just the X and Y axis output.

```
void loop()
{
   int x_axis = analog8(0);
   int y_axis = analog8(1);

   Serial.print("< ");
   Serial.print(x_axis);
   Serial.print(", ");
   Serial.print(y_axis);
   Serial.println(" >");
   delay(2);
}
```





```
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126 >
< 124, 126
```

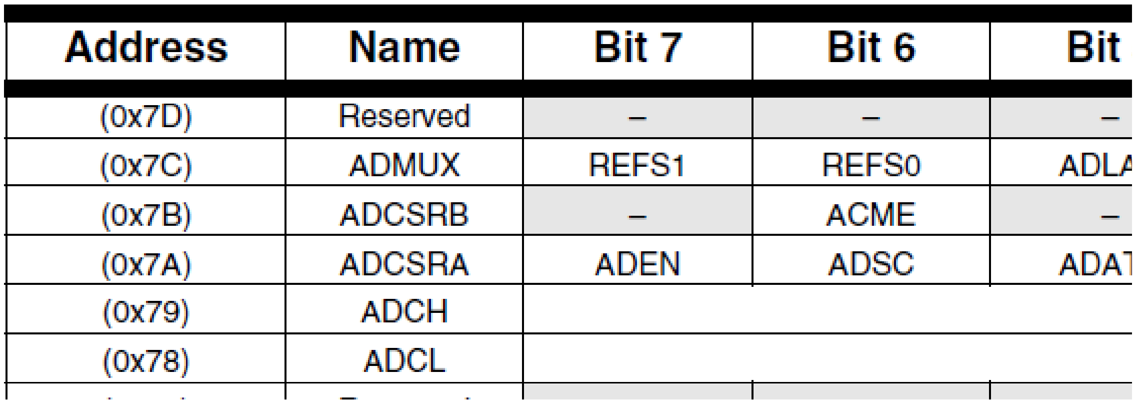
To open the Serial monitor, select Tools->Serial monitor. A window like the above should appear, and it will be constantly spewing numbers to the screen. When the joystick is at its home position, you should have roughly values <128, 128> for X and Y. When you move the joystick around, these values should change to reflect the new position. Play around – See where <0,0> is, see where <255,255> is, etc. We're going to be using these values coming up.

Keep in mind, if we're using serial output, we cannot make use of pins 0 and 1 on the Arduino (PORTD pins 0, 1). These are TX/RX pins and are used for serial communication. We used them safely in lab 1, because in that lab we didn't do any serial communication.

4) We have one last problem. We want to implement branching logic using these X, Y values, but right now they're stored in variables. We can't access variables in our assembly code. No problem! Notice what we're returning in the analog8() function:

```
return ADCH;
```

ADCH is a register we can read from using assembly statements! From the datasheet, page 526, we can see that ADCH is the high byte of the ADC data register. This is where our value from 0-255 is written after each call to **`analog8()`**. We can also see its address – **0x79**. Learn the datasheet. Love the datasheet.



Address	Name	Bit 7	Bit 6	Bit
(0x7D)	Reserved	–	–	–
(0x7C)	ADMUX	REFS1	REFS0	ADLA
(0x7B)	ADCSRB	–	ACME	–
(0x7A)	ADCSRA	ADEN	ADSC	ADAT
(0x79)	ADCH			
(0x78)	ADCL			

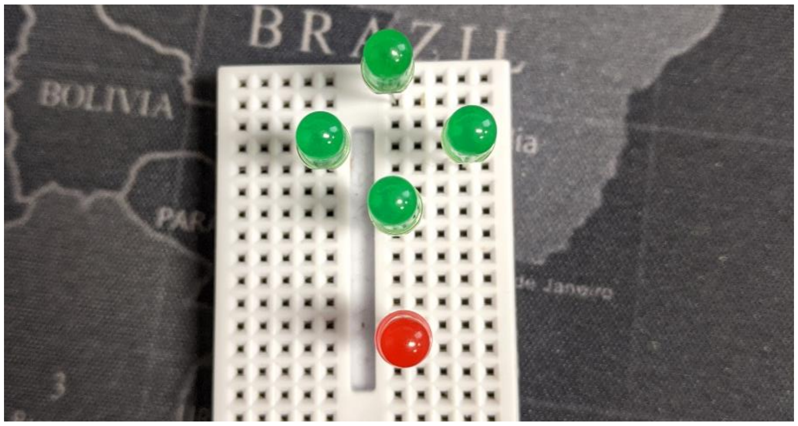
We start by calling **analog8()** with the first ADC channel, then loading ADCH into a general-purpose register. Then we call **analog8()** once more with the second ADC channel, and load ADCH into a second register:

```
int x_axis = analog8(0);
asm(" lds r16, 0x79 ");
int y_axis = analog8(1);
asm(" lds r17, 0x79 ");
```

We don't need **x_axis** and **y_axis** at this point. The value **x_axis** is now found in **r16**, and the value **y_axis** is found in **r17**. From here, we can use these registers in our in our branching logic to turn LEDs on and off, just like we did with **SW**.

We have all the tools we need: digital output (done in lab 1), digital input (done using SW in part 2), and analog input. If you've made it this far, you're ready for the final part of this lab.

5) For your submission, I will give you a program and circuit description and you will have to fulfil the requirements as described using the tools we've learned so far. To start, consider this mini breadboard with LEDs:



Your job is to build a circuit starting with a general LED layout like the one above, and write an Arduino assembly program that does the following:

Behavior:

- The red LED should light up when the joystick button is pressed.
- Each of the green LEDs should light up when the joystick is moved in that direction. The uppermost LED should light up when the joystick is pushed up, and so on.
- It should be possible for two green LEDs to light up simultaneously. If the joystick is pushed to the top-left, both the top and left LEDs will light up.
- “Up” is considered relative to the pinout text on the joystick board. For my joystick, the wires come out of the “Left” direction. Your joystick may differ.

Constraints:

- All LEDs should be correctly wired with resistors (1k, 2k, doesn’t matter much).
- VRx and VRy pins on the joystick should be connected to A0 and A1 respectively.
- Left, Up, Right, Down LEDS should be connected to pins 2, 3, 4, 5 respectively on the Arduino (PORTD bits 2-5).
- The red LED (button-press) should be connected to pin 6 on the Arduino (PORTD bit 6)
- SW (digital input) should be connected to pin 7 on the Arduino (PORTD bit 7)
- Everything must be done in assembly, with the following exceptions:
  - ADC **`analog8()`** and **`analog_init()`**
  - The **`delay()`** function
  - **`Serial.begin()`** and **`Serial.print()`** functions
  - See starter template below

Advice:

On the next page of this lab description is a skeleton program with comments to get you started. Here are some mistakes I made myself while producing this lab:

- Forgot digital input was active-low (multiple times).
- Flipped GND and +5V on joystick.
- Tried controlling LEDs over pins 0 and 1 while doing serial output (for 20 mins).

This stuff is tricky, and mistakes are very easy to make. Don’t get discouraged. Go one step at a time and test, test, test. Don’t wire the entire circuit before testing, just like you wouldn’t write 100 lines of code without compiling once. Go slowly, and don’t be afraid to use the Serial port to print values. This can be handy for debugging.

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 #5 on D2L:

- A picture of your Arduino and breadboard with the circuit you produced for Part 5 clearly visible. Just like the previous lab.
- Your source code for Part 5 as a plain text (.txt) or assembly source (.asm) file.

I will be testing your submission on my own Uno with the joystick and LEDs 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.

Template to get you started:

```
void analog_init()
{
  // Analog functions can be used as-is
  ADCSRA |= (1 << ADEN);
  ADMUX  |= (1 << REFS0);
  ADMUX  |= (1 << ADLAR); // Left justified output for 8bit mode
  ADCSRA |= (1 << ADSC);  // start conversion
}

uint8_t analog8(uint8_t channel)
{
  // Analog functions can be used as-is
  ADMUX  &= 0xF0;
  ADMUX  |= channel;
  ADCSRA |= (1 << ADSC);
  while ( ADCSRA & ( 1 << ADSC ) );
  return ADCH;
}

void setup()
{
  Serial.begin(9600);
  analog_init();

  // Setup your pins and ports here.
}
```

```
void loop()
{
  asm(" start:         ");

  // Get analog values for X and Y
  // Load from ADCH (0x79) into r16/r17
  int x_axis = analog8(0);
  asm(" lds r16, 0x79 ");
  int y_axis = analog8(1);
  asm(" lds r17, 0x79 ");

  // EVERYTHING BELOW THIS LINE SHOULD BE ASSEMBLY

  // Click button logic here. Should the button LED be on or off?


  // Up direction logic here. Should the Up LED be on or off?


  // Left direction logic here.


  // Down direction logic.


  // Right direction logic.


  // EVERYTHING ABOVE THIS LINE SHOULD BE ASSEMBLY

  // You can use this serial code for handy debugging.
  // It will let you observe the values coming out of your ADC.
  Serial.print("< ");
  Serial.print(x_axis);
  Serial.print(", ");
  Serial.print(y_axis);
  Serial.println(" >");
  delay(2);

  asm(" rjmp start  ");
}
```

For those wanting an extra challenge…

1) Convert **`analog_init()`** to assembly.

   - You'll have to explore the data sheet. In particular, the register and ADC pages.
   - Figure out what addresses the names are mapped to (ADCSRA, ADMUX, etc.) and figure out which bits other names are mapped to (ADEN, ADSC, etc.)

2) Convert **`analog8()`** to assembly.

   - You've got looping here, but we did that in ARC.
   - Otherwise similar challenges as converting **`analog_init()`**

3) Implement both **`analog8()`** and **`analog_init()`** as AVR assembly subroutines.

   - No more C-style function calls.
   - **`analog_init()`** can be an assembly subroutine inside **`setup()`**
   - **`analog8()`** can be an assembly subroutine inside **`loop()`**