

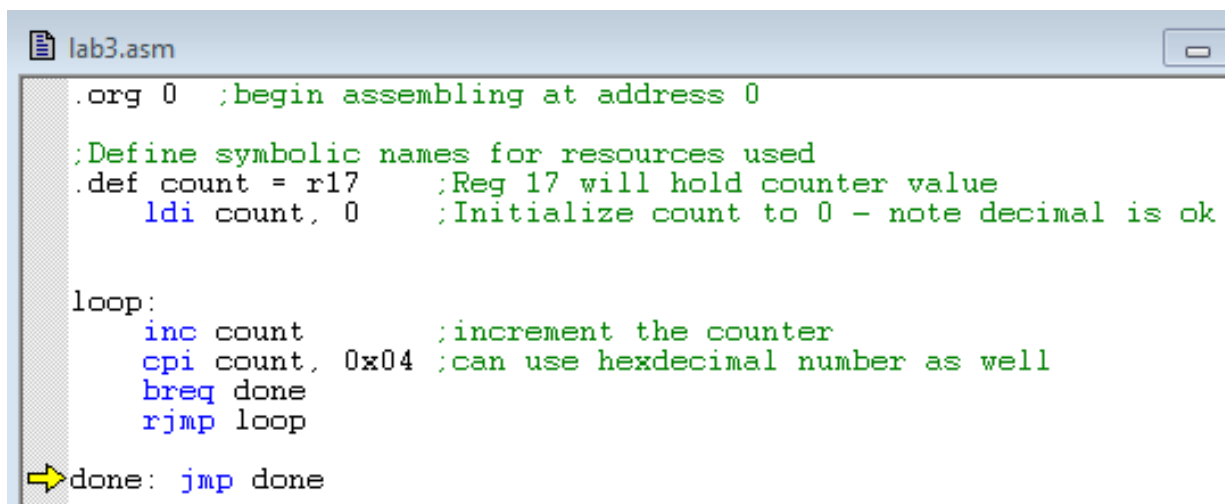
Lab 3 Control Structures & Introduction to MEGA 2560 Board and I/O Instructions

Submit led.asm by 5:00pm on Jan. 26th, 2018.

I. Control Instructions

Learn to use branches: recall the code we did in lab 2. Now, modify the code: if “number” (r16) is even, set register r19 to 1, 0 if “number” is odd. Suggest name r19 as “isEven”.

Learn to use a loop: count a number from 0 to 4.



```
.org 0 ;begin assembling at address 0

;Define symbolic names for resources used
.def count = r17 ;Reg 17 will hold counter value
    ldi count, 0 ;Initialize count to 0 - note decimal is ok

loop:
    inc count ;increment the counter
    cpi count, 0x04 ;can use hexadecimal number as well
    breq done
    rjmp loop

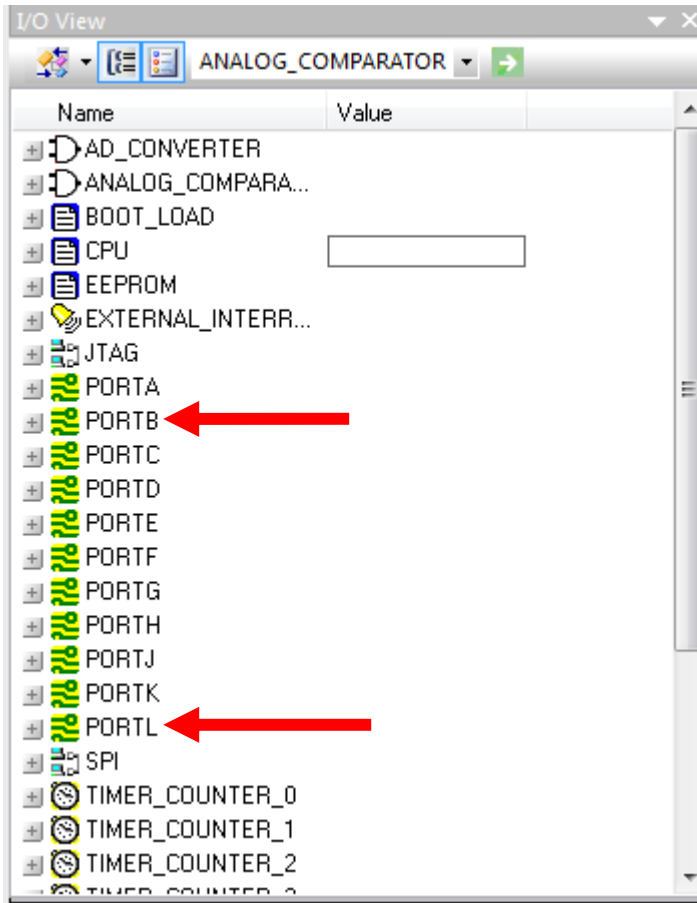
done: jmp done
```

Optional: Load a big number to “count”, decrement it to 0. What is the biggest number that can be loaded to count? What if you want to repeat it 1024 times, or 10K times?

II. MEGA 2560 Board and I/O Instructions

In the AVR MEGA 2560 microcontroller, each Input/Output (I/O) port has three associated registers, the data direction register (DDRx), output register (PORTx), and input register (PINx). These registers correspond to addresses in the data space accessible by the processor (the first 0x200 bytes in SRAM).

Launch AVR Studio 4 and observe the I/O View. Expand PORTB and PORTL



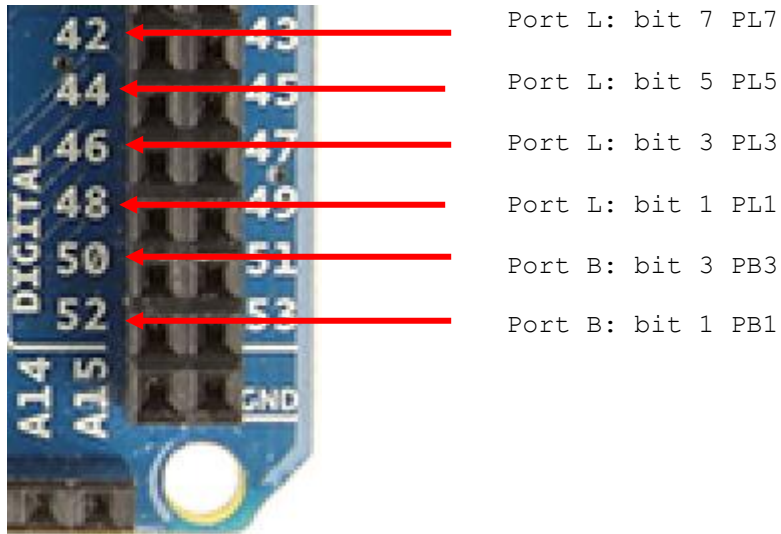
Name	Address	Value	Bits
DDRB	0x04 (0x24)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
PINB	0x03 (0x23)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
PORTB	0x05 (0x25)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Name	Address	Value	Bits
DDRL	na (0x10A)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
PINL	na (0x109)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
PORTL	na (0x10B)	0x00	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Some ports are mapped to two addresses, for example, PORTB is mapped to 0x05 and 0x25, you may use IN/OUT instructions to transfer data between registers and SRAM (data memory) for address between 0x00 and 0x3F, but some ports such as PORTL are mapped to addresses which are bigger than 0x3F, they can't be accessed using the IN/OUT instructions, what are we going to do? We use LDS and STS instructions instead. In AVR, Ports A to G use Port Mapped I/O (separate addresses from memory) and Ports H to K use Memory Mapped I/O (usage is similar to any memory location). Port Mapped addresses have to use separate In/Out instructions while Memory mapped use LDS and STS.

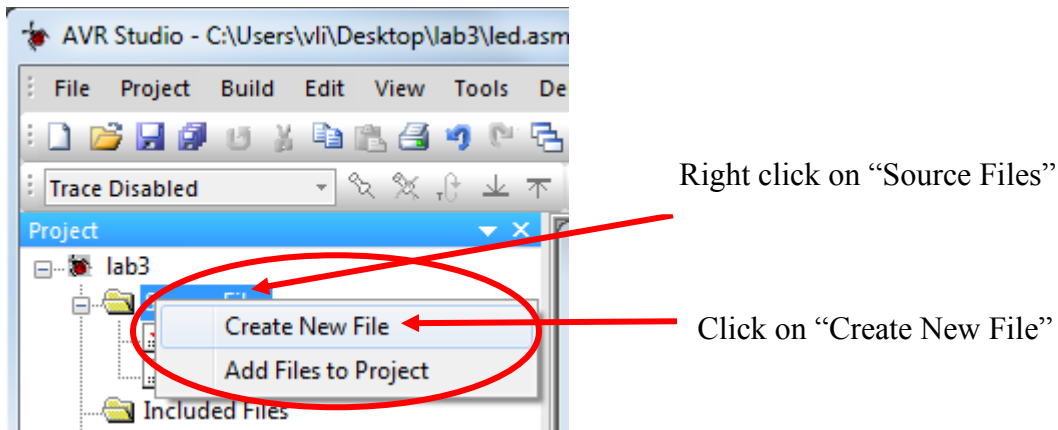
III. Pins and Ports

The six LEDs are associated with six pins and the pins are mapped to some bits of two ports: PORTB and PORTL:

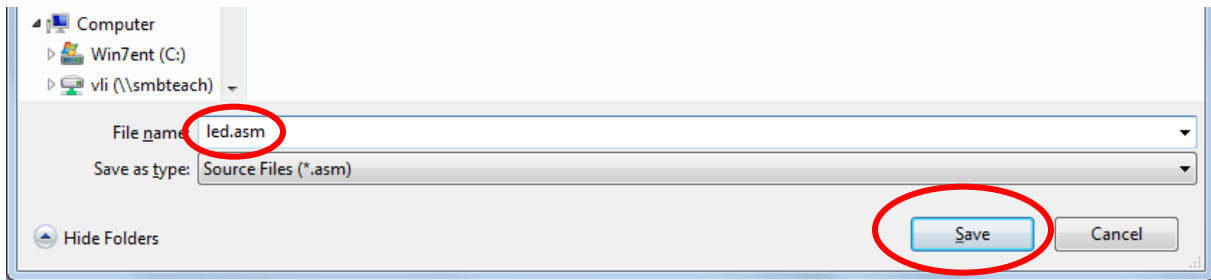


Write a small program to turn on a specific LED light: In the same project, add a new file called led.asm. The following is the step by step instruction:

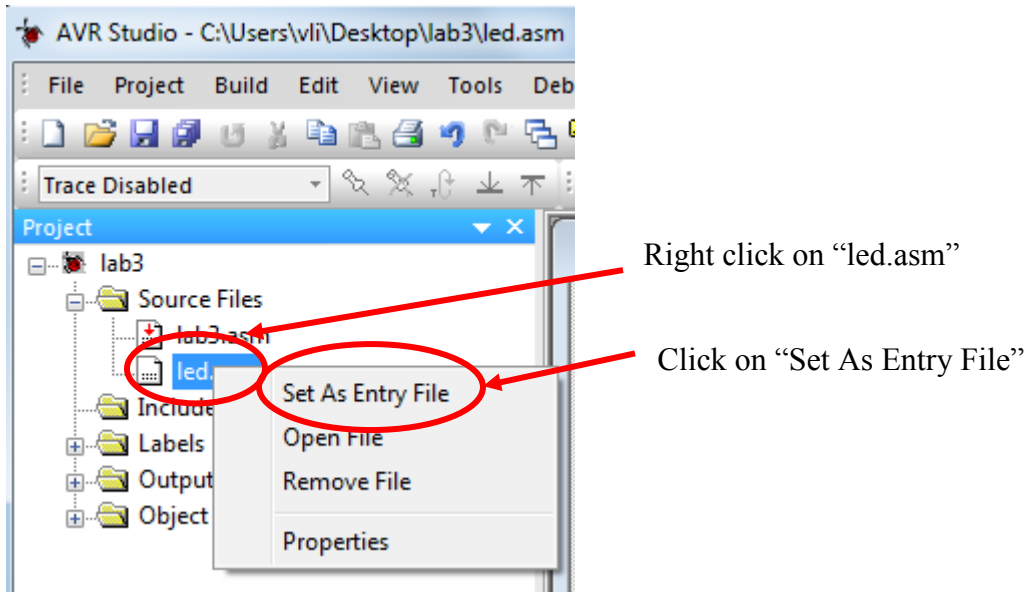
Step 1: On the “Project” pane, right click on “Source Files”, click on “Create New File”.



Step 2: Type “led.asm” as file name. Click on the “Save” button.



Step 3: Set “led.asm” as entry file so that the when you build and run the project, it will use this file. Right click on “led.asm”, see a drop-down menu, click on “Set As Entry File” in the drop-down menu.



Once led.asm is added to the project, type the following code. Note that the I/O registers - 0x10B and 0x10A - are specified in hexadecimal numbers. The two I/O registers are given names, PORTL and DDRL, by using the .equ directives to represent the port numbers (the memory addresses in data memory). Once equated, the I/O registers are referred to by names instead of numbers in the program.

```
;pins.asm
;learn which port is mapped to which pin and how to turn it on/off
.equ PORTL = 0x10B
.equ DDRL = 0x10A

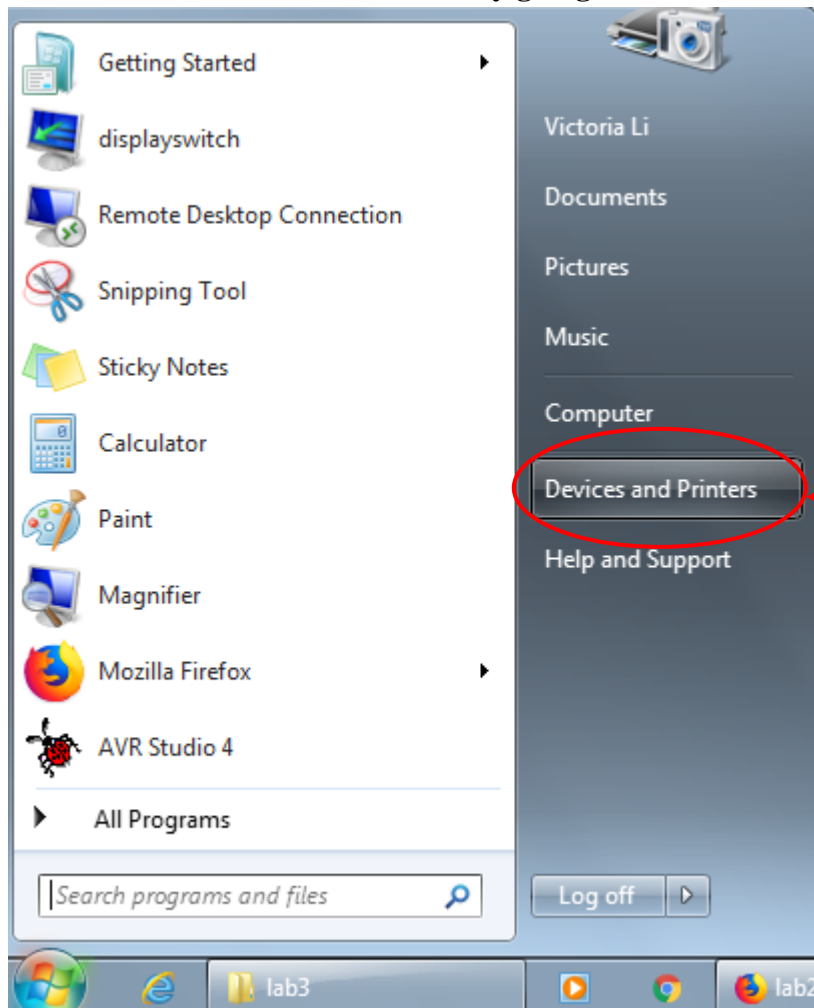
;configure PORTL as output by setting DDRL to 1
ldi r16, 0xff
sts DDRL, r16

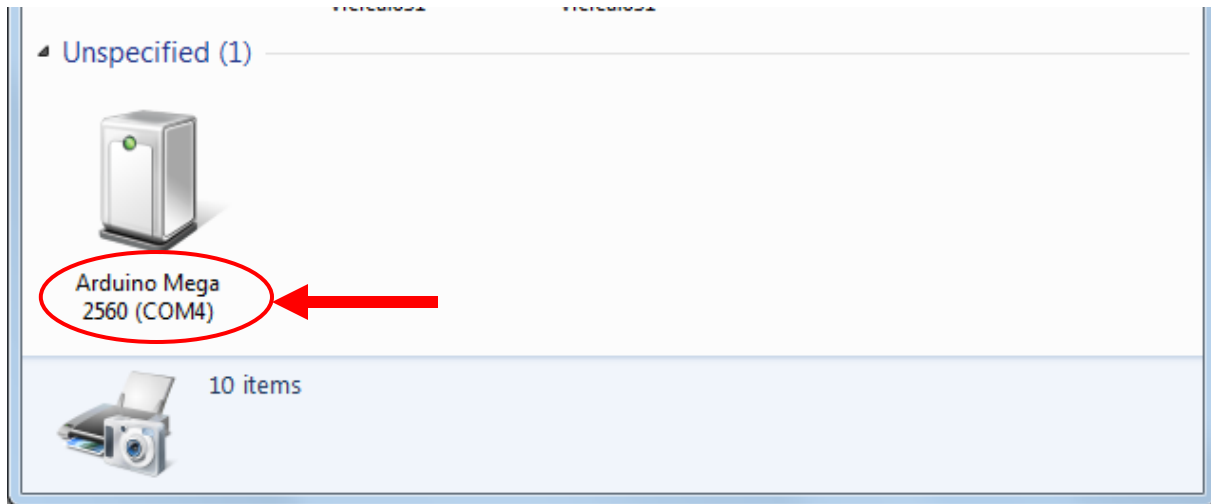
;set LED at pin 48 on
ldi r19, 0x02
sts PORTL, r19

done: jmp done
```

IV. Build and upload the .hex file to the board:

- Build the program above. Step through the program, observe the changes of the registers and the I/O registers.
- Note to which COM port the lab AVR board is connected to. **This is different on each machine.** You can check this by going to Start -> Devices and Printers.





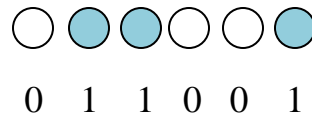
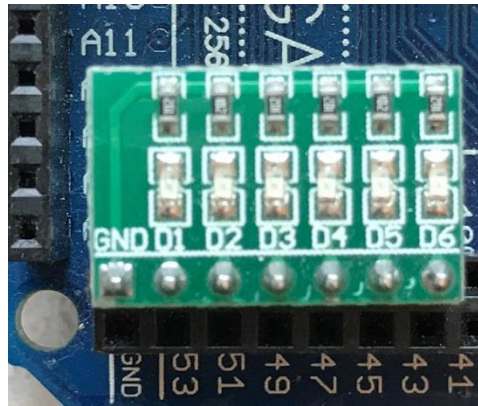
- In the example above, the Arduino Mega is on COM4. Check the port number of Arduino Mega on the machine you are currently working on.
- Upload the lab3.hex file to the board by typing the following command at the command window:

```
"C:\winAVR\bin\avrdude" -C "C:\WinAVR\bin\avrdude.conf" -p atmega2560 -c  
wiring -P COM6 -b 115200 -D -F -U flash:w:lab3.hex
```

You may need to change the path and filename “lab3.hex” (in blue color) if the settings are different from this. Or, download “upload.bat” file to the same directory where your lab3.hex is stored. Modify the com port (-P COM4) to reflect the one on your machine. Open a command window get to the same directory where upload.bat is stored, and type upload.bat. “upload.bat” is a batch file. It is another way to type a command. You need to change the file name “lab3.hex” if your project is not called lab3. Learn some DOS commands, such as “cd” – change directory.

V. Exercises:

Tilt the LED light by 90 degrees. If the 6 LED lights represent 6 bit binary number, with pin 52 represent the most significant bit and pin 42 represent the least significant bit, modify your code in led.asm, let the lights represent any 6 bit binary number. Let LED light on represent binary 1, off represent 0. The blue circle represents light on in the following diagram, therefore, the pattern represents 0b011001.



Optional: You can make the LED lights blink by letting the lights on for a while and off for a while. Hint: use nested loops to run “nop” (no operation) for a million times, then turn the lights on or off (XOR gate?).

Submit led.asm by 5:00pm on Jan. 26th, 2018.

This lab is derived from the Chapters 2 and 3 of your textbook (Some Assembly Required by Timothy S. Margush).