

Lab 3 Control Structures & Introduction to ATmega2560 Board and I/O Instructions

I. Control Instructions

Learn to write 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”.

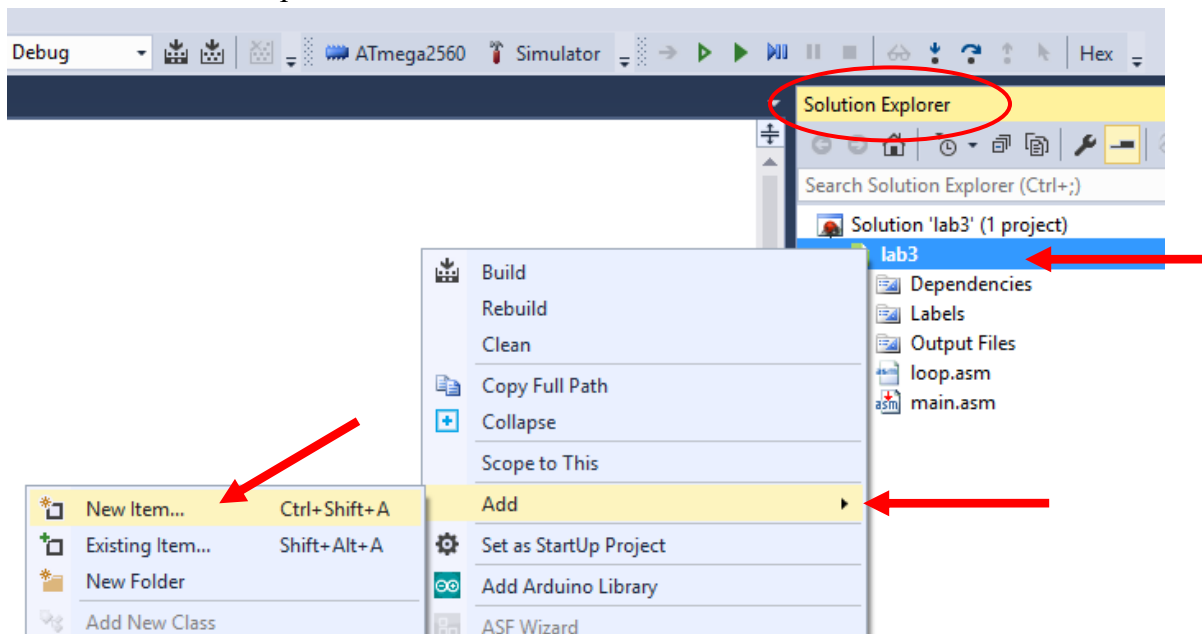
Create a project named lab3 (see the instructions in lab2 on how to create a project). In the main, type the following code:

```
; if the number loaded to r16 is even, set r19 to 1, 0 otherwise
.cseg
.org 0
.def number=r16
.def isEven=r19
    clr isEven
    ldi number, 0x0A
    andi number, 0b00000001
    breq even
done: jmp done

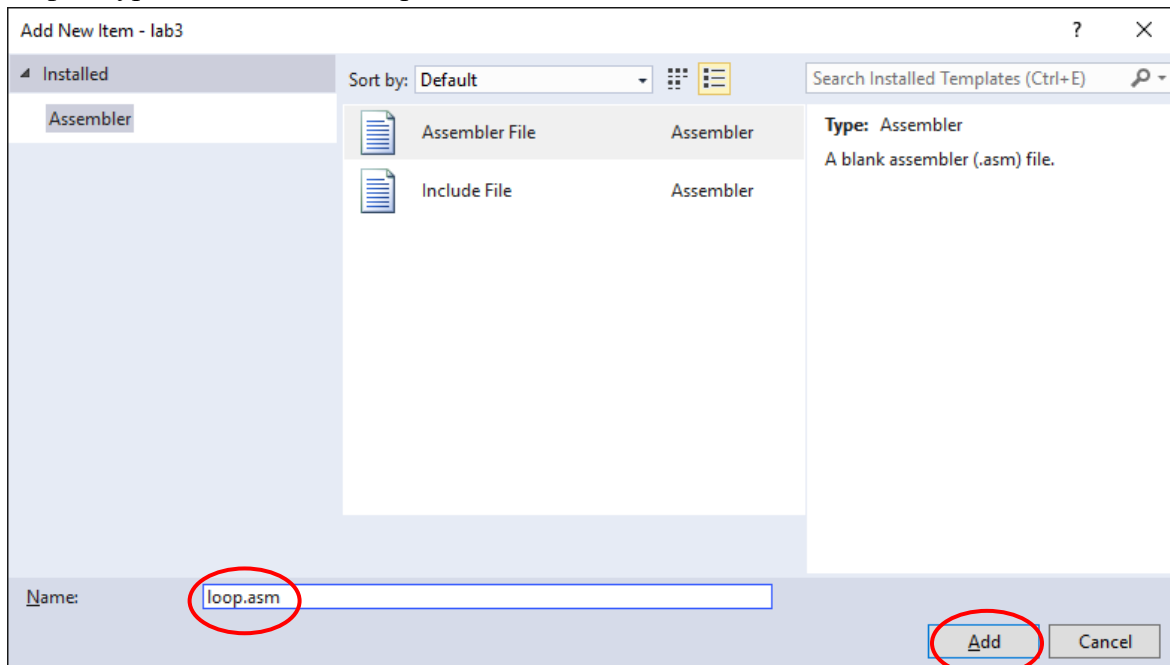
even: ldi isEven,1
    rjmp done
```

Add a new file to project lab3, and name it loop.asm:

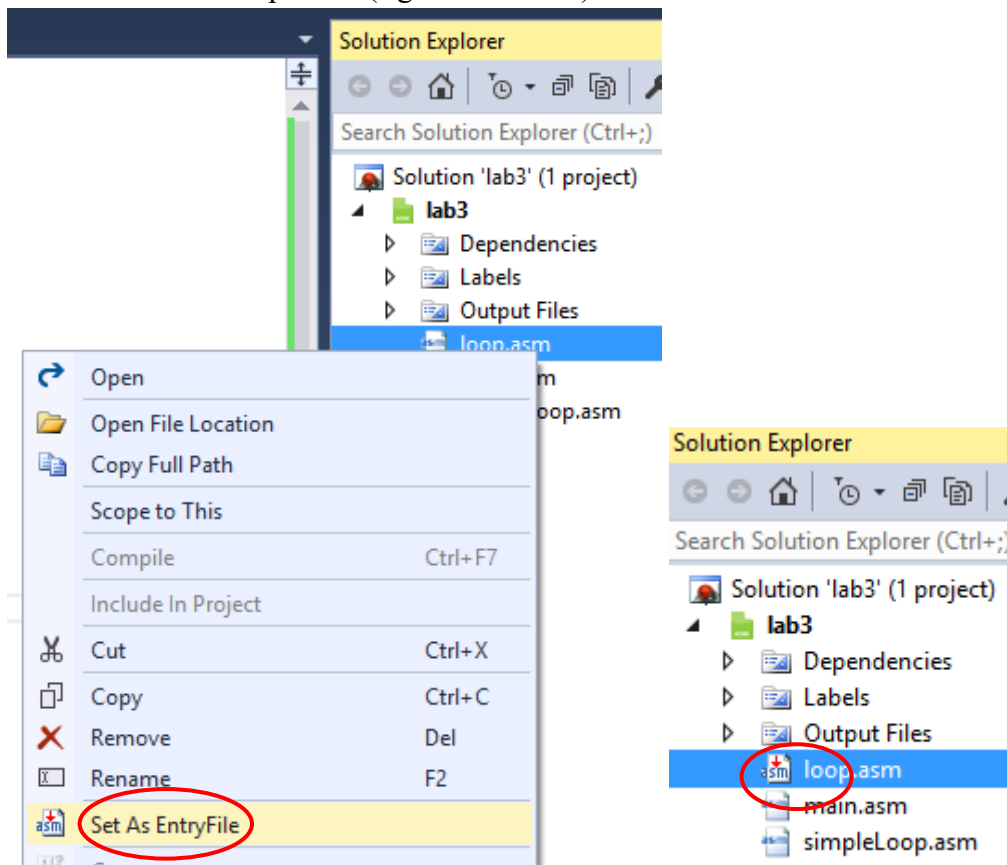
Step 1: in Solution Explorer panel, right click on the project name “lab3”, choose “Add” -> “New item...” in the pull down menus:



Step2: Type the file name “loop.asm” and click on the “Add” button.



Step 3: In Solution Explorer panel, right click on “loop.asm”, see a drop-down menu, click on “Set As EntryFile” (left screenshot). After choosing “loop.asm” as entry file, there is a red down arrow for “loop.asm” (right screenshot).



Learn to write a loop: count a number from 0 to 4. In the newly created loop.asm file, type the following code:

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

;Define symbolic names for resources used
.def count=r17 ;r17 holds 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
```

Build and run the program.

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? Hint: Use ADIW or SBIW. Refer to p113 of Some Assembly Required, p16 and p126 of the AVE_Instruction_Set.pdf on course website.

II. ATmega2560 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).

Observe the memory address of PortB and PortL: choose the current .asm file in part I, build the program. From the menu, click on “Debug” -> “Start Debugging and Break”. On the right hand side, observe the I/O View. To see the memory address of PORTB and PORTL, click on PORTB or PORTL:

Memory addresses of the associated registers for PortB:

The screenshot shows a list of I/O registers. The entry 'I/O Port (PORTB)' is highlighted with a red arrow. Below the list, a table provides the memory addresses for the registers associated with PortB:

Name	Address	Value	Bits
I/O PINB	0x23	0x00	□□□□□□□□
I/O DDRB	0x24	0x00	□□□□□□□□
I/O PORTB	0x25	0x00	□□□□□□□□

Memory addresses of the associated registers for PortL:

The screenshot shows a list of I/O registers. The entry 'I/O Port (PORTL)' is highlighted with a red arrow. Below the list, a table provides the memory addresses for the registers associated with PortL:

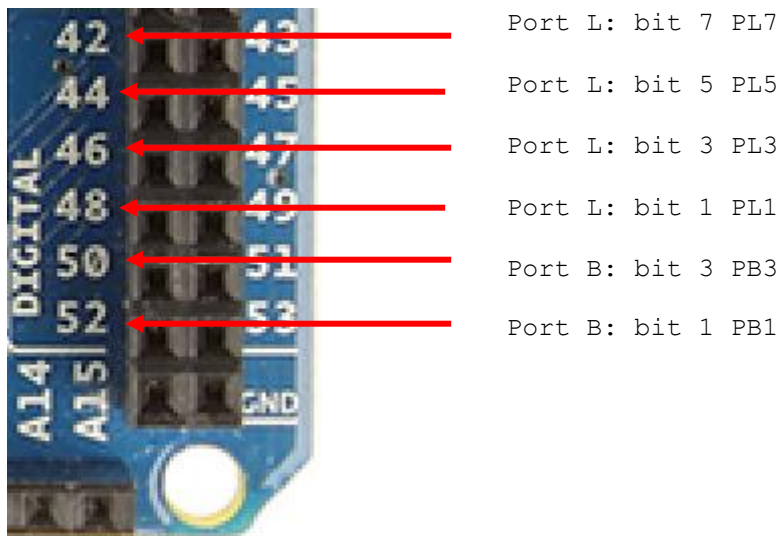
Name	Address	Value	Bits
I/O PINL	0x109	0x00	□□□□□□□□
I/O DDRL	0x10A	0x00	□□□□□□□□
I/O PORTL	0x10B	0x00	□□□□□□□□

Some ports are mapped to addresses smaller than or equal to 0x3F, for example, PORTB is mapped to 0x25. Use IN/OUT instructions to transfer data between registers and SRAM (data memory) for addresses 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, in this case, use LDS and STS instructions. 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

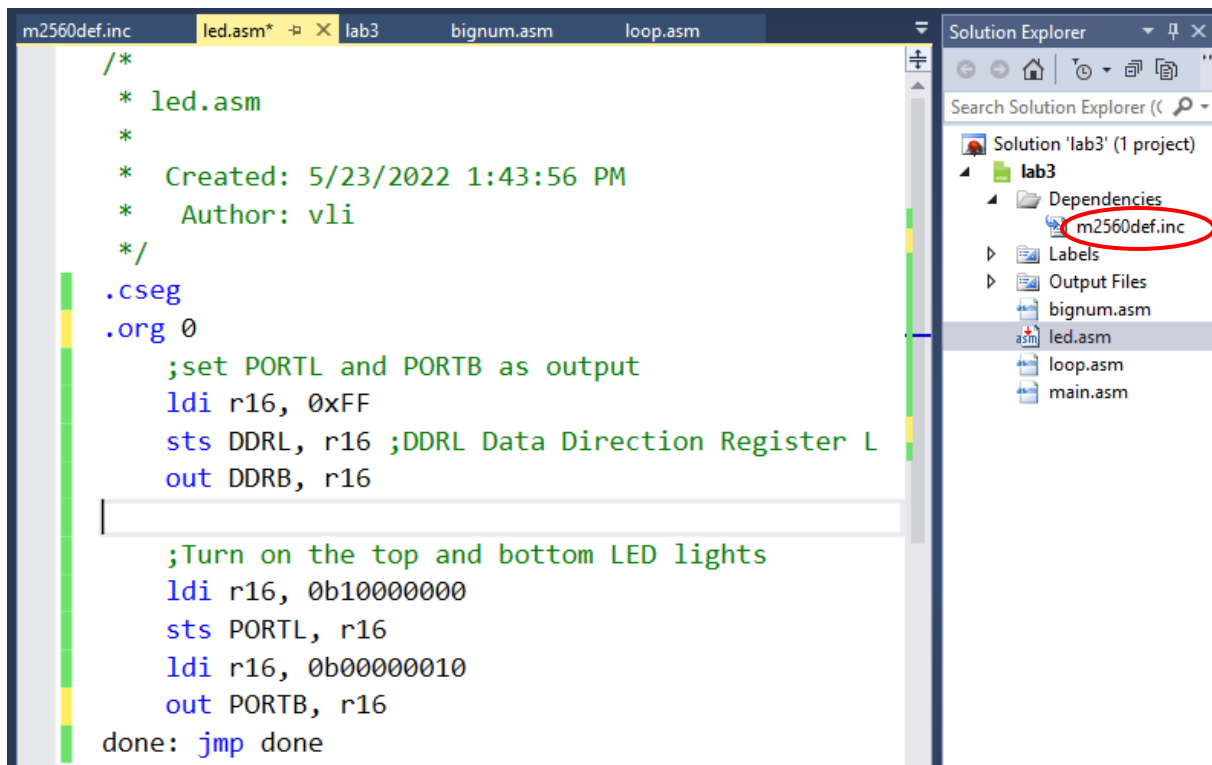
The six LEDs are associated with six pins and the pins are mapped to some bits of two ports: PORTB and PORTL:



Write a program to turn on a specific LED light: In the same project, add a new file called led.asm.

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.

The memory addresses of DDRL, DDRB, PORTL, and PORTB are defined in file named "m2560def.inc". To read the file, in the "Solution Explorer", click on "Dependencies" -> "m2560def.inc". Refer to the screenshot below:



The screenshot shows an IDE with a main editor window and a Solution Explorer on the right. The main editor displays the contents of `m2560def.inc`, which is an assembly file for an LED project. The code includes comments about the file's creation and author, and defines constants for the PORTL and PORTB registers. It then sets up the DDRL and DDRB registers and turns on the top and bottom LED lights. The Solution Explorer on the right shows the project structure for 'lab3', including dependencies, labels, output files, and source files. The file `m2560def.inc` is highlighted in the Dependencies folder.

```
/*
 * led.asm
 *
 * Created: 5/23/2022 1:43:56 PM
 * Author: vli
 */
.cseg
.org 0
;set PORTL and PORTB as output
ldi r16, 0xFF
sts DDRL, r16 ;DDRL Data Direction Register L
out DDRB, r16

;Turn on the top and bottom LED lights
ldi r16, 0b10000000
sts PORTL, r16
ldi r16, 0b00000010
out PORTB, r16
done: jmp done
```

Solution Explorer

Search Solution Explorer (C)

Solution 'lab3' (1 project)

- lab3
 - Dependencies
 - m2560def.inc
 - Labels
 - Output Files
 - bignum.asm
 - led.asm
 - loop.asm
 - main.asm

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

- Right click on led.asm and choose “Set As EntryFile” (Go to step 3 of page 2 of this lab note for details.)
- Build -> Build Solution, then run the program above (Debug -> Start Debugging and Break). Observe the changes of the registers and the I/O registers.

The screenshot displays an IDE with three main panels. The left panel shows assembly code for setting up PORTB. The middle panel shows a list of I/O registers, with PORTB highlighted. The bottom panel shows a memory dump where the value 'ff' is circled, indicating the state of PORTB.

Assembly Code:

```
.cseg
.org 0 ;begin assembling

;set PORTL and PORTB a
ldi r16, 0xFF
sts DDRL, r16 ;DDRL=Da
out DDRB, r16

;Turn on the top and b
ldi r16, 0b10000000
sts PORTL, r16
ldi r16, 0b00000010
out PORTB, r16

done: jmp done
```

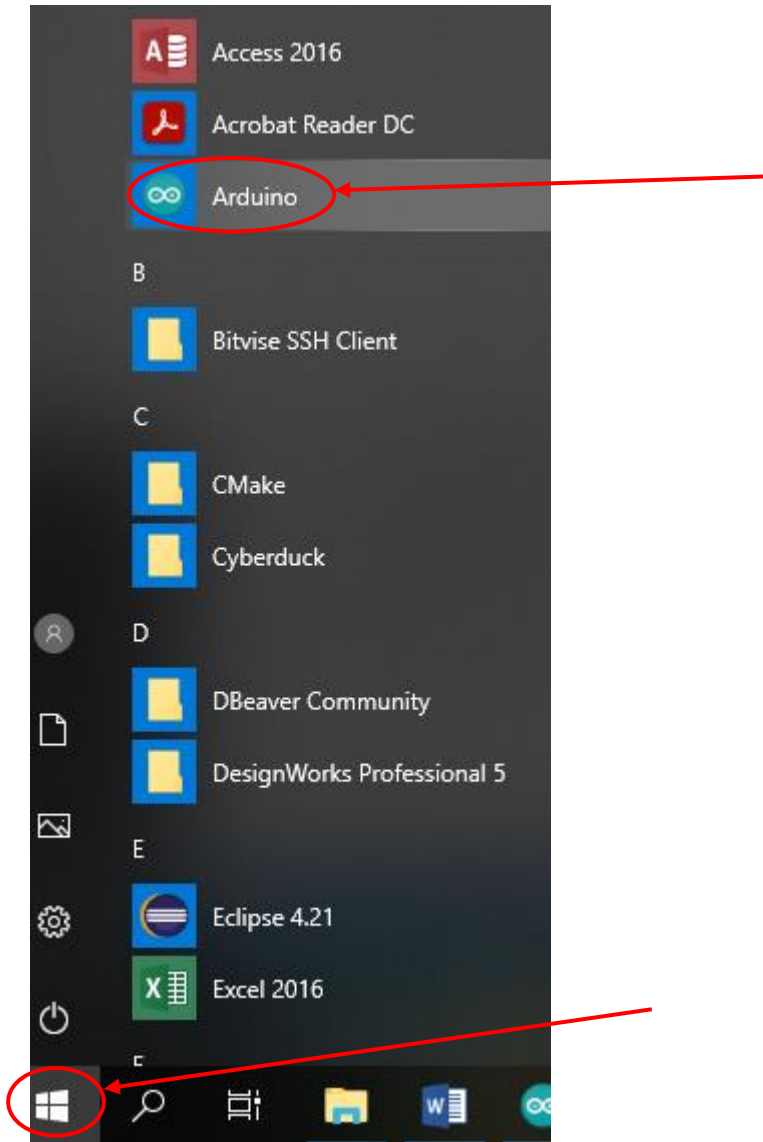
I/O Registers:

Name	Address	Value	Bits
PINB	0x23	0x00	00000000
DDRB	0x24	0xFF	11111111
PORTB	0x25	0x00	00000000

Memory Dump (Memory 4):

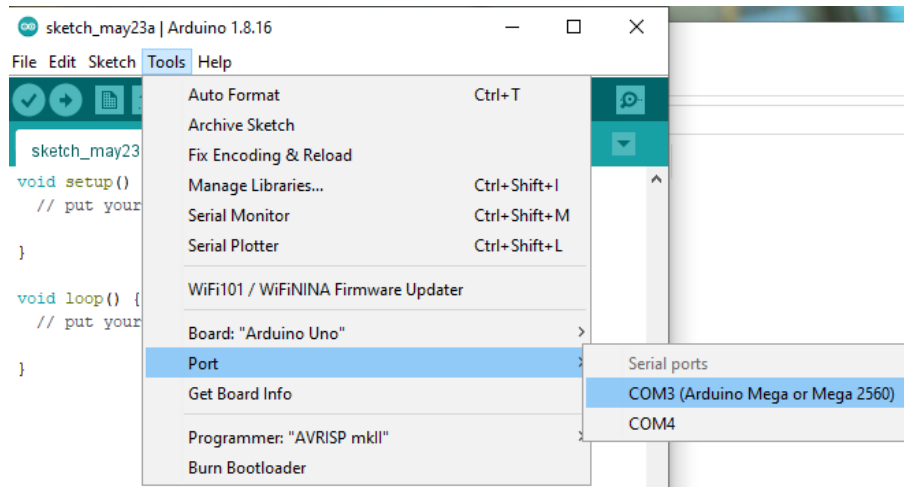
Address	Value	Hex	ASCII
0x0020	00 00 00 00 ff 00 00 00	000000ff00000000ÿ....
0x0029	00 00 00 00 00 00 00 00	0000000000000000
0x0032	00 00 00 00 00 00 00 00	0000000000000000

- Find to which COM port the lab AVR board is connected to. **This is different on each machine.** You can check this by launch the Arduino IDE. The step by step instructions are:
 - Step 1: Launch Arduino IDE: click on the *Start*  button, then click on the *Arduino* program.



- Step 2: In the Arduino IDE program, from the menu, click on *Tools* → *Port:* → *"COM3"* (Arduino Mega or Mega 2560) *COM4*.

The machine shown in the screenshot below shows that the Arduino board is connected to the computer through port COM3.



- Upload the lab3.hex file to the board:

Step a: open the DOS command window, navigate to the directory containing lab3.hex.

Step b: type the following command at the command window:

```
"C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe" -C
"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf" -p
atmega2560 -c wiring -P COM3 -b 115200 -D -F -U flash:w:lab3.hex
```

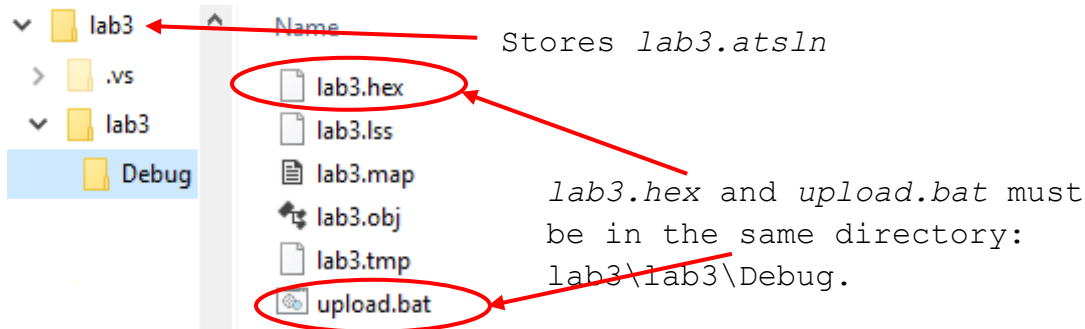
Alternative Step b, follow the instructions below:

- download upload.bat.txt from the lab 3 folder on the course website
- Save the upload.bat.txt file into the folder where lab3.hex is stored. For example, if the project is called lab3 and saved on H:\230\, then the lab3.hex file is stored at H:\230\lab3\lab3\Debug\. Save the upload.bat.txt file to H:\230\lab3\lab3\Debug\
- **Change the file name from upload.bat.txt to upload.bat.** Open upload.bat with Notepad++, check the port number and .hex file name. If the port is not the same port shown in step 2 on page 8, change it. If the .hex file name is not the same as the one in the Debug directory, change the file name.
- Double click on the upload.bat file from the Window's Explorer to upload the .hex file to the Arduino Board.

You may need to change the path (e.g. `C:\Program Files (x86)`) and filename (e.g. `lab3.hex`) if the settings are different from this. Instead of typing such a long command, download "upload.bat" file to the same directory where your lab3.hex file is stored. Modify the communication 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

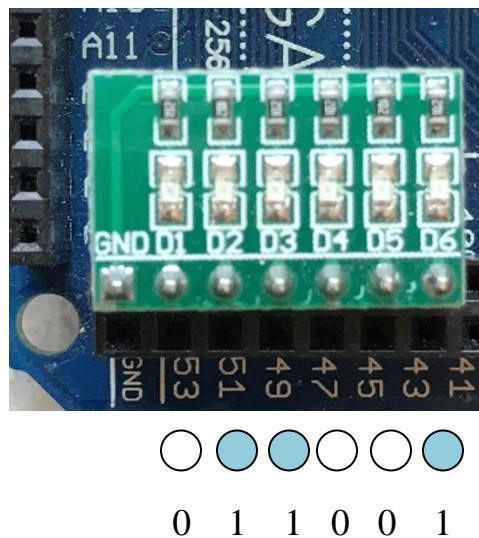
name *lab3.hex* if your project is not called *lab3*. Learn some DOS commands, such as “cd” – change directory.

To find the *lab3.hex* file, refer to the following screen shot:



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 represents binary 1, off represents 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?).