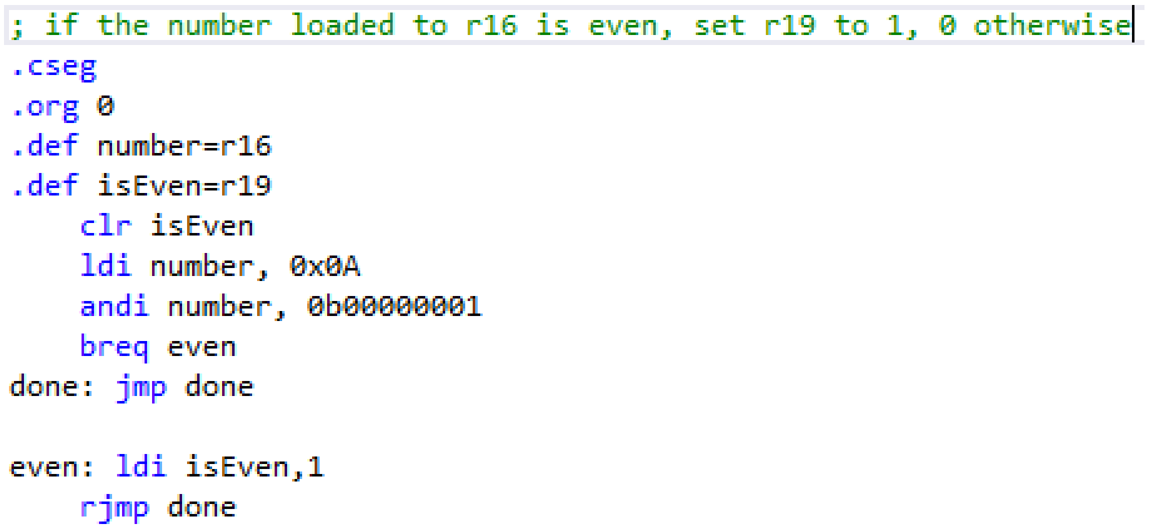
Lab 3 Control Structures & Introduction to ATmega2560 Board and I/O Instructions

I. Control Instructions

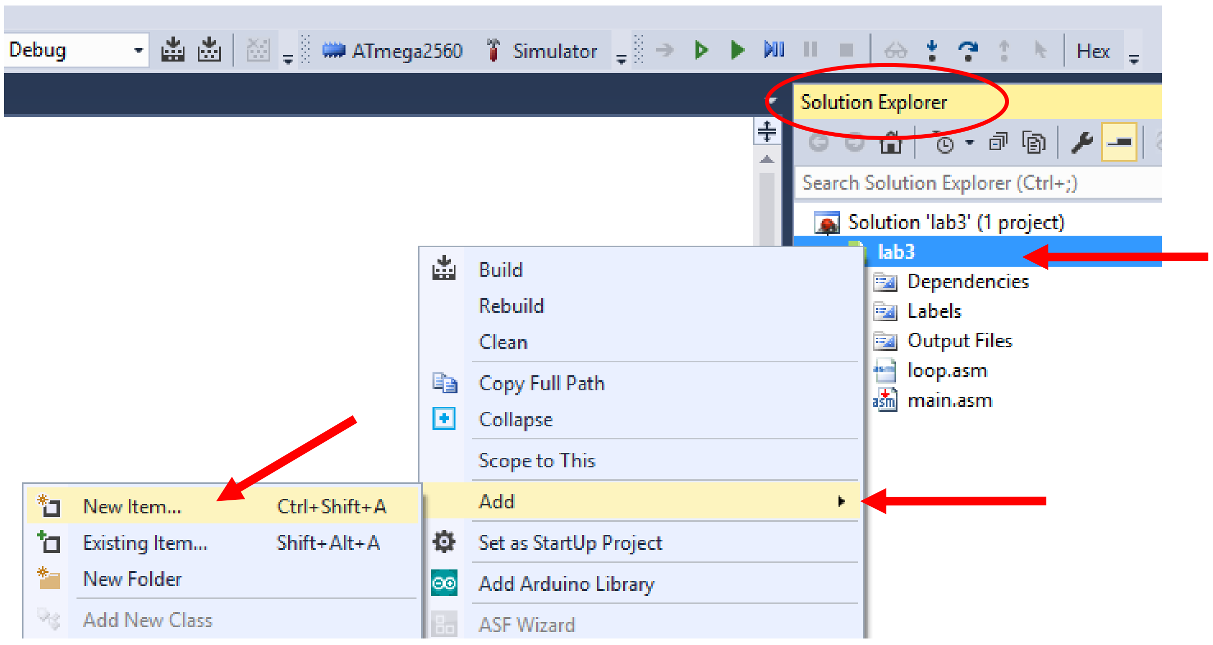
Learn to write branches: recall the code we did in lab 2. Now, modify the code: if "number" (r16) is even, set register r19 to 1, 0 if "number" is odd. Suggest name r19 as "isEven".

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

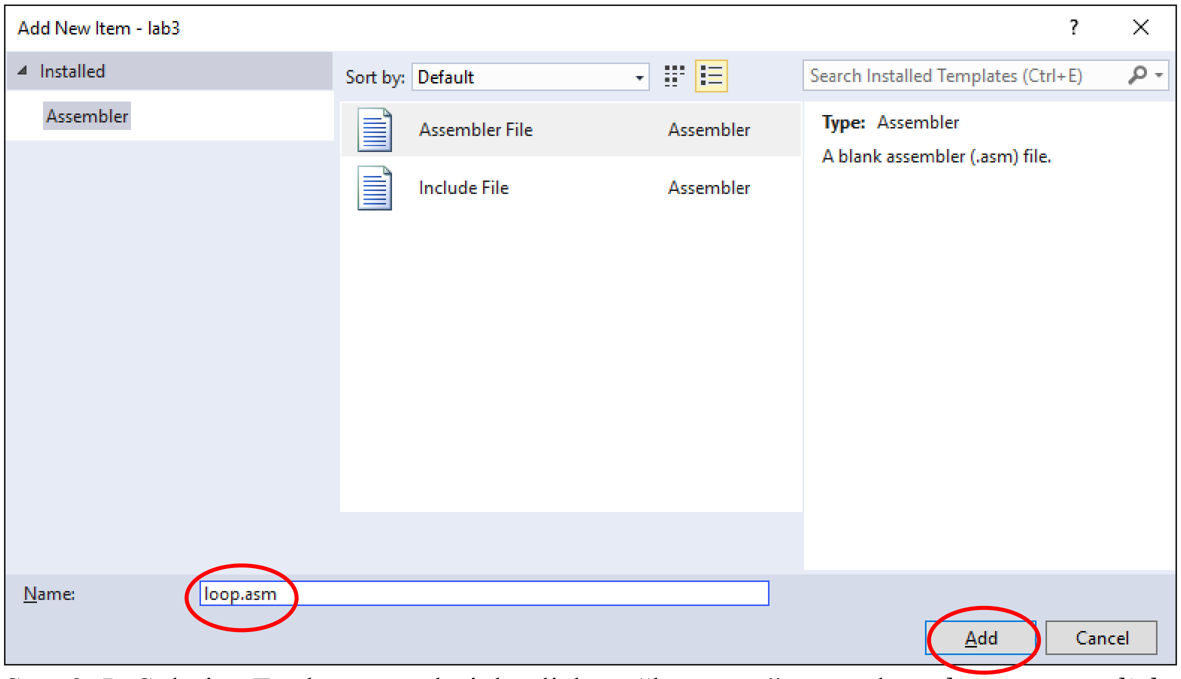


```
; if the number loaded to r16 is even, set r19 to 1, 0 otherwise
.cseg
.org 0
.def number=r16
.def isEven=r19
    clr isEven
    ldi number, 0x0A
    andi number, 0b00000001
    breq even
done: jmp done

even: ldi isEven,1
    rjmp done
```

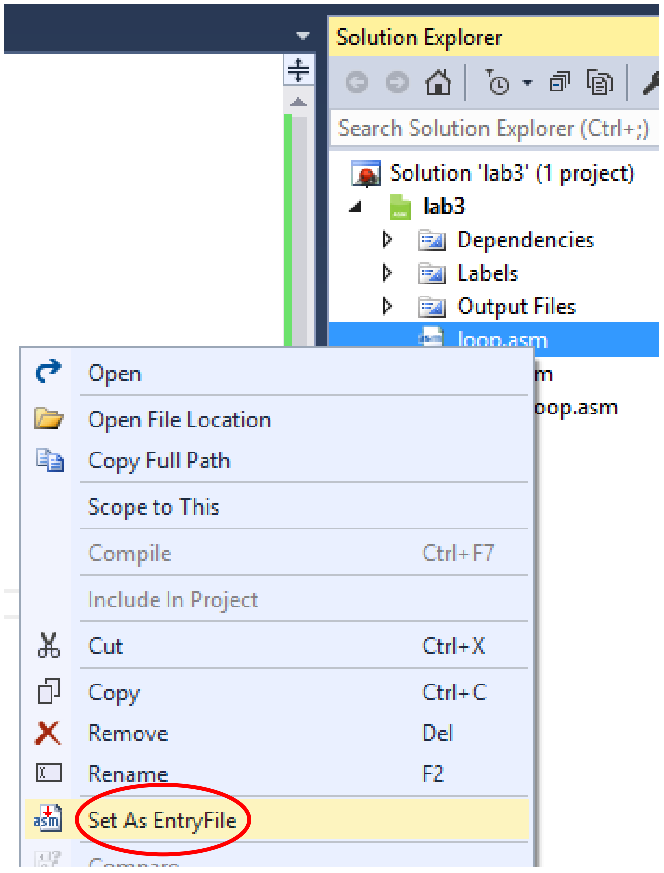
Add a new file to project lab3, and name it loop.asm:
Step 1: in Solution Explorer panel, right click on the project name "lab3", choose "Add" -> "New item…" in the pull down menus:

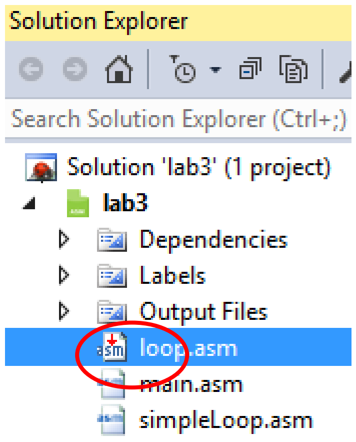


Step2: Type the file name “loop.asm” and click on the “Add” button.

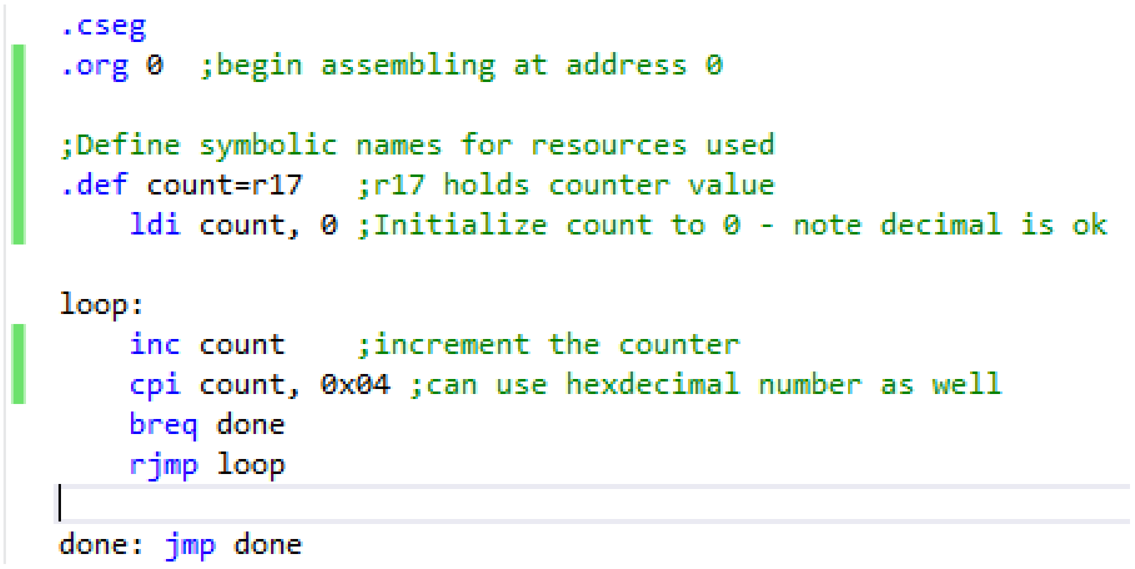


Step 3: In Solution Explorer panel, right click on “loop.asm”, see a drop-down menu, click on “Set As EntryFile” (left screenshot). After choosing “loop.asm” as entry file, there is a red down arrow for “loop.asm” (right screenshot).





<u>Learn to write a loop</u>: 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 hexdecimal number as well
    breq done
    rjmp loop

done: jmp done
```

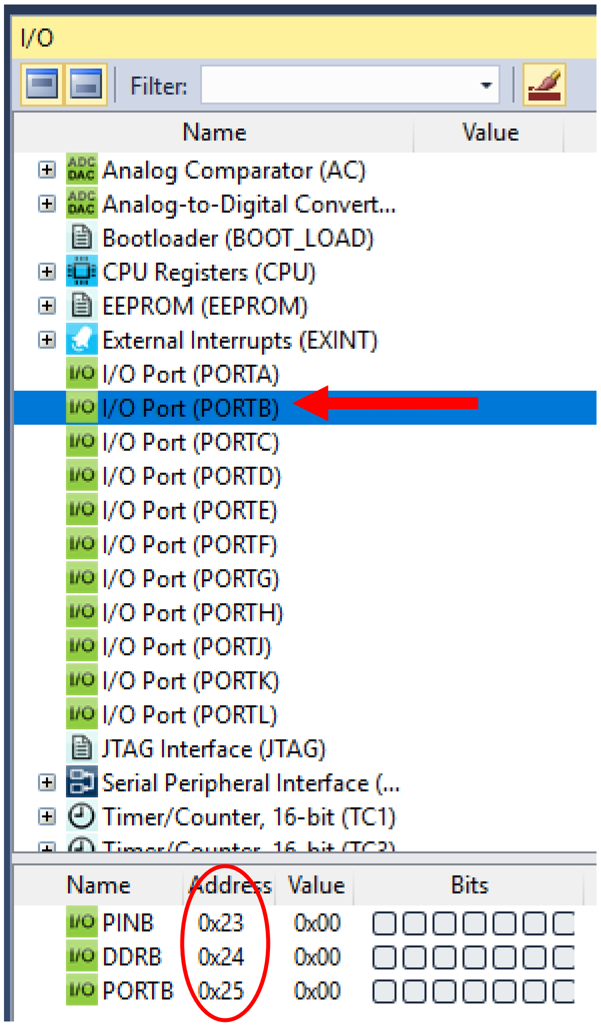
Build and run the program.
Optional: Load a big number to "count", decrement it to 0. What is the biggest number that can be loaded to count? What if you want to repeat it 1024 times, or 10K times? Hint: Use ADIW or SBIW. Refer to p113 of Some Assembly Required, p16 and p126 of the AVR_Instruction_Set.pdf on course website.

II. ATmega2560 Board and I/O Instructions
Create a new file named led.asm by repeating the steps 1-3 above, then type the code on page 6. The explanation of the code is:

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:



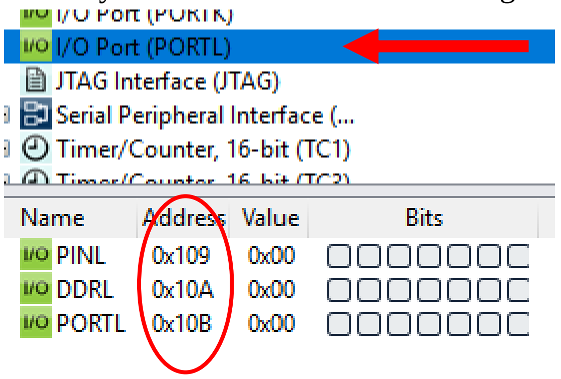
I/O

Filter:

Name	Value
Analog Comparator (AC)	
Analog-to-Digital Convert...	
Bootloader (BOOT_LOAD)	
CPU Registers (CPU)	
EEPROM (EEPROM)	
External Interrupts (EXINT)	
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
I/O Port (PORTE)	
I/O Port (PORTF)	
I/O Port (PORTG)	
I/O Port (PORTH)	
I/O Port (PORTJ)	
I/O Port (PORTK)	
I/O Port (PORTL)	
JTAG Interface (JTAG)	
Serial Peripheral Interface (...	
Timer/Counter, 16-bit (TC1)	

Name	Address	Value	Bits
PINB	0x23	0x00	☐☐☐☐☐☐☐
DDRB	0x24	0x00	☐☐☐☐☐☐☐
PORTB	0x25	0x00	☐☐☐☐☐☐☐

Memory addresses of the associated registers for PortL:



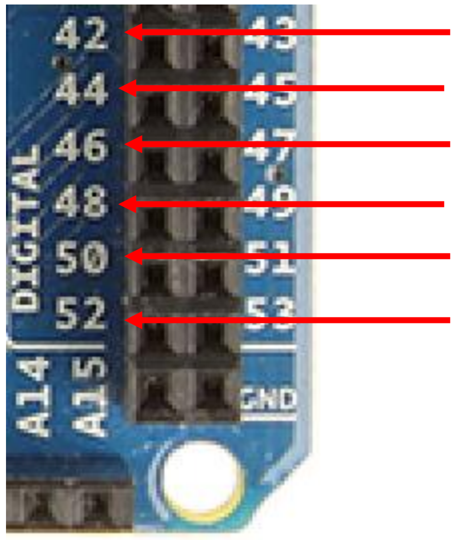
I/O Port (PORTL)
JTAG Interface (JTAG)
Serial Peripheral Interface (...
Timer/Counter, 16-bit (TC1)

Name	Address	Value	Bits
PINL	0x109	0x00	☐☐☐☐☐☐☐
DDRL	0x10A	0x00	☐☐☐☐☐☐☐
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:



PL7 - bit 7, that is 2^7 of PORTL

PL5 - bit 5, that is 2^5 of PORTL

PL3 - bit 3, that is 2^3 of PORTL

PL1 - bit 1, that is 2^1 of PORTL

PB3 - bit 3, that is 2^3 of PORTB

PB1 - bit 1, that is 2^1 of PORTB

For exemple, to turn the top two lights on, set bit 2^7 and 2^5 of PORTL to 1 :

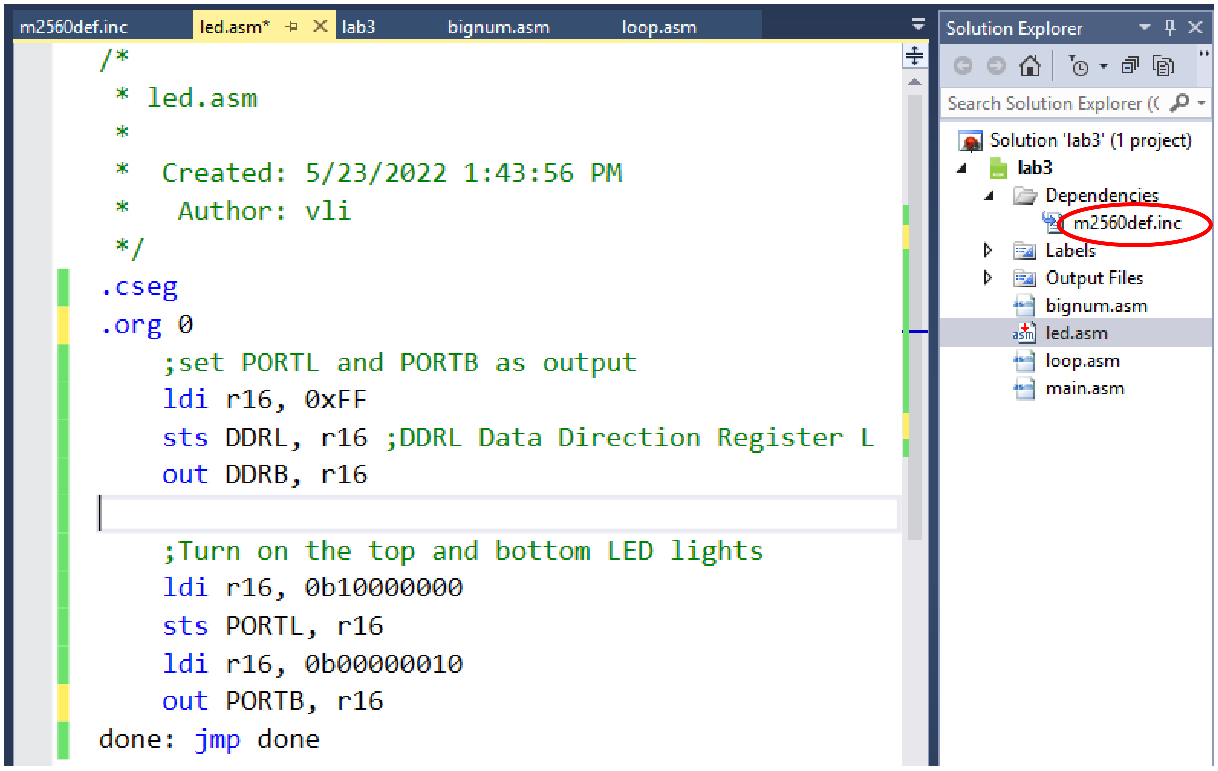
$$\begin{array}{cc} 2^7 & 2^5 \\ | & | \end{array}$$

```
ldi r16, 0b10100000
sts PORTL, r16
```

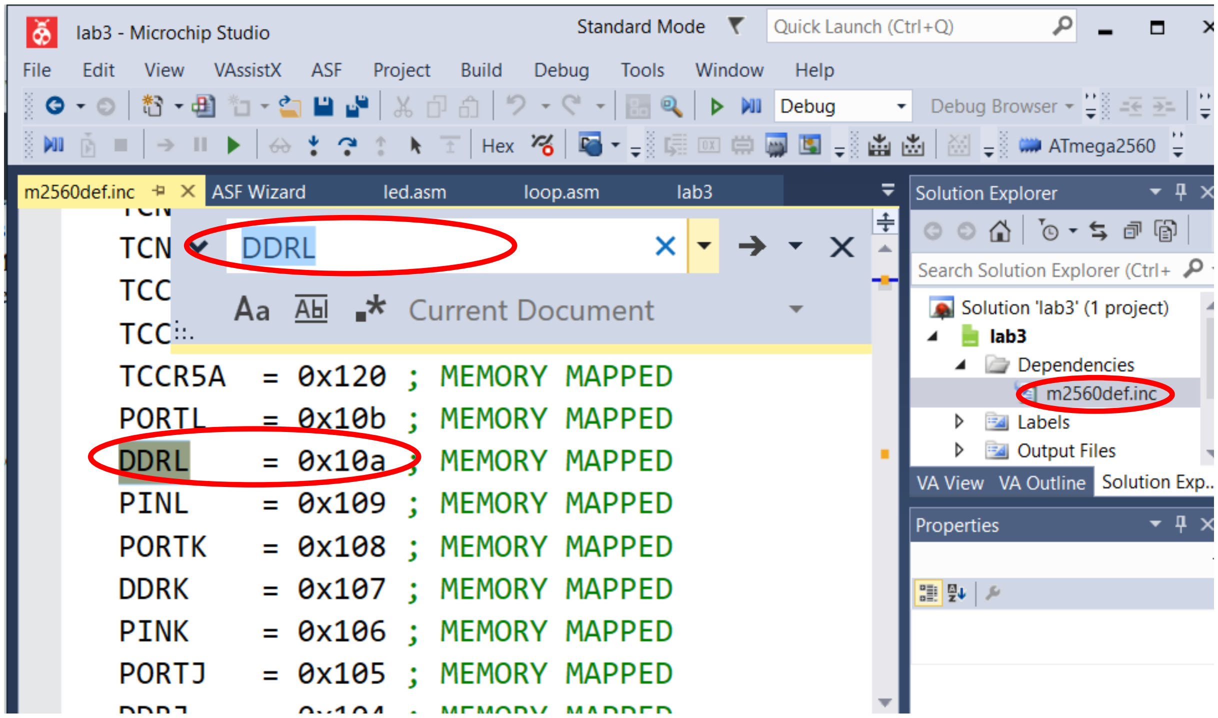
Write a program to turn on a specific LED light: In the same project, add a new file called led.asm.

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 directive 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 on the next page:



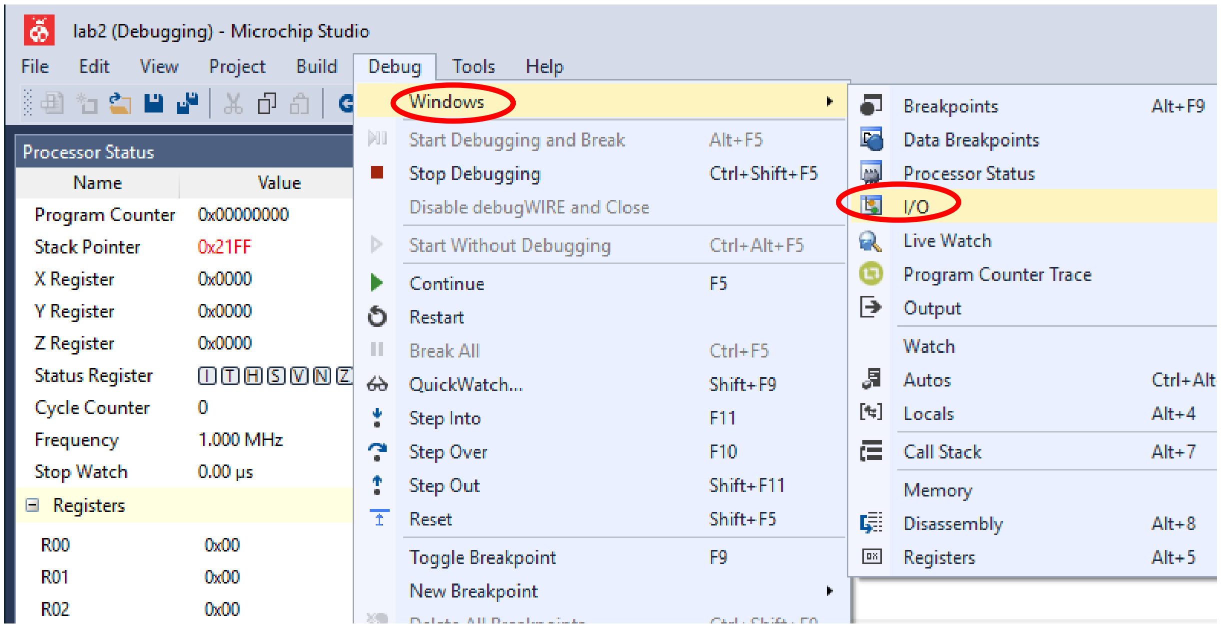
```
/*
 * led.asm
 *
 *  Created: 5/23/2022 1:43:56 PM
 *   Author: vli
 */
.cseg
.org 0
    ;set PORTL and PORTB as output
    ldi r16, 0xFF
    sts DDRL, r16 ;DDRL Data Direction Register L
    out DDRB, r16

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



```
TCN
TCC
TCC
TCCR5A  = 0x120 ; MEMORY MAPPED
PORTL   = 0x10b ; MEMORY MAPPED
DDRL    = 0x10a ; MEMORY MAPPED
PINL    = 0x109 ; MEMORY MAPPED
PORTK   = 0x108 ; MEMORY MAPPED
DDRK    = 0x107 ; MEMORY MAPPED
PINK    = 0x106 ; MEMORY MAPPED
PORTJ   = 0x105 ; MEMORY MAPPED
```

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

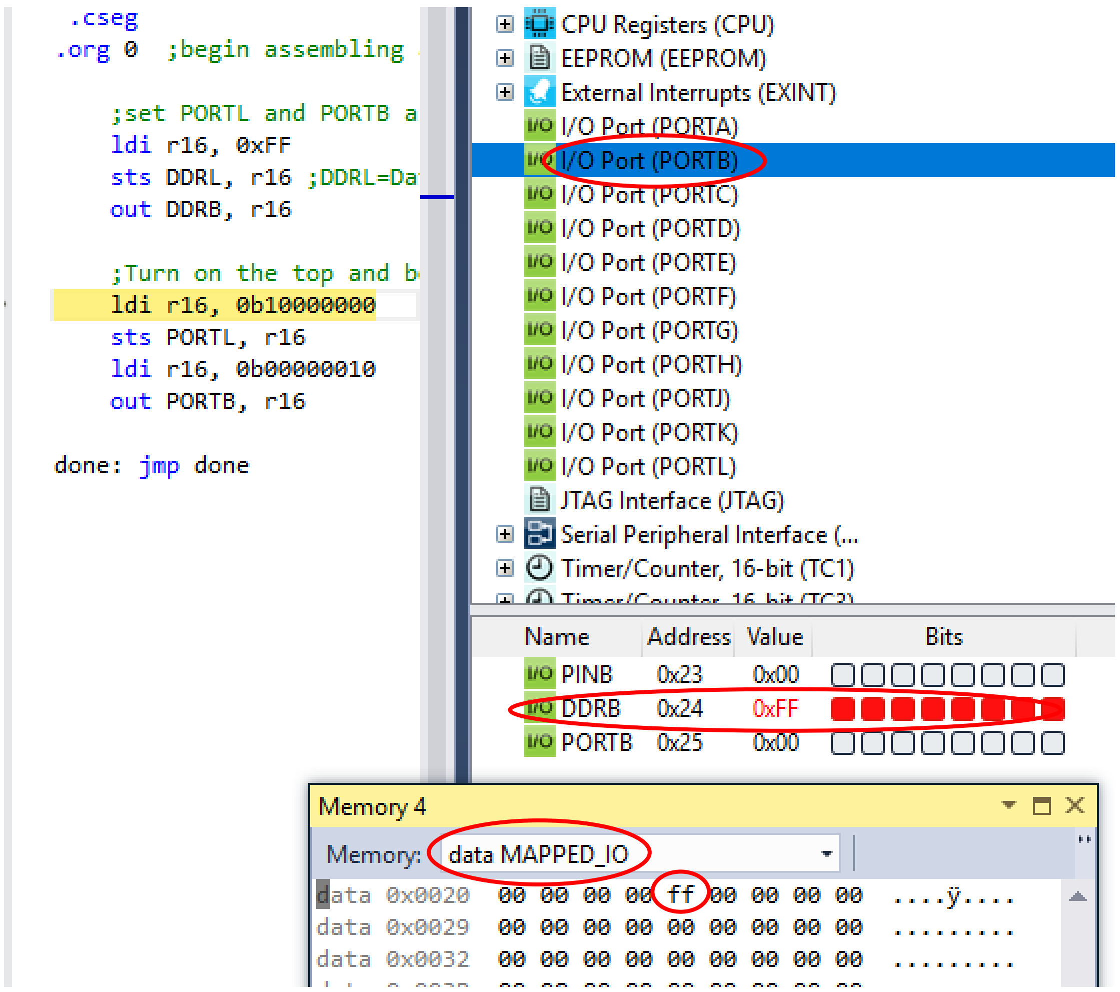
- Right click on *led.asm* and choose “Set As EntryFile” (Go to step 3 of page 2 of this lab note for details.)
- If the I/O pane is not visible, go to the menu and click on “Debug” -> “Start Debugging and Break”, then go to menu, click on “Debug” -> “Windows” -> “I/O”:



Processor Status

Name	Value
Program Counter	0x00000000
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z
Cycle Counter	0
Frequency	1.000 MHz
Stop Watch	0.00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00

- Build -> Build Solution, then run the program above (Debug -> Start Debugging and Break). Observe the changes of the registers and the I/O registers.



```
 .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
```

Name	Address	Value	Bits
PINB	0x23	0x00	☐☐☐☐☐☐☐☐
DDRB	0x24	0xFF	🟥🟥🟥🟥🟥🟥🟥🟥
PORTB	0x25	0x00	☐☐☐☐☐☐☐☐

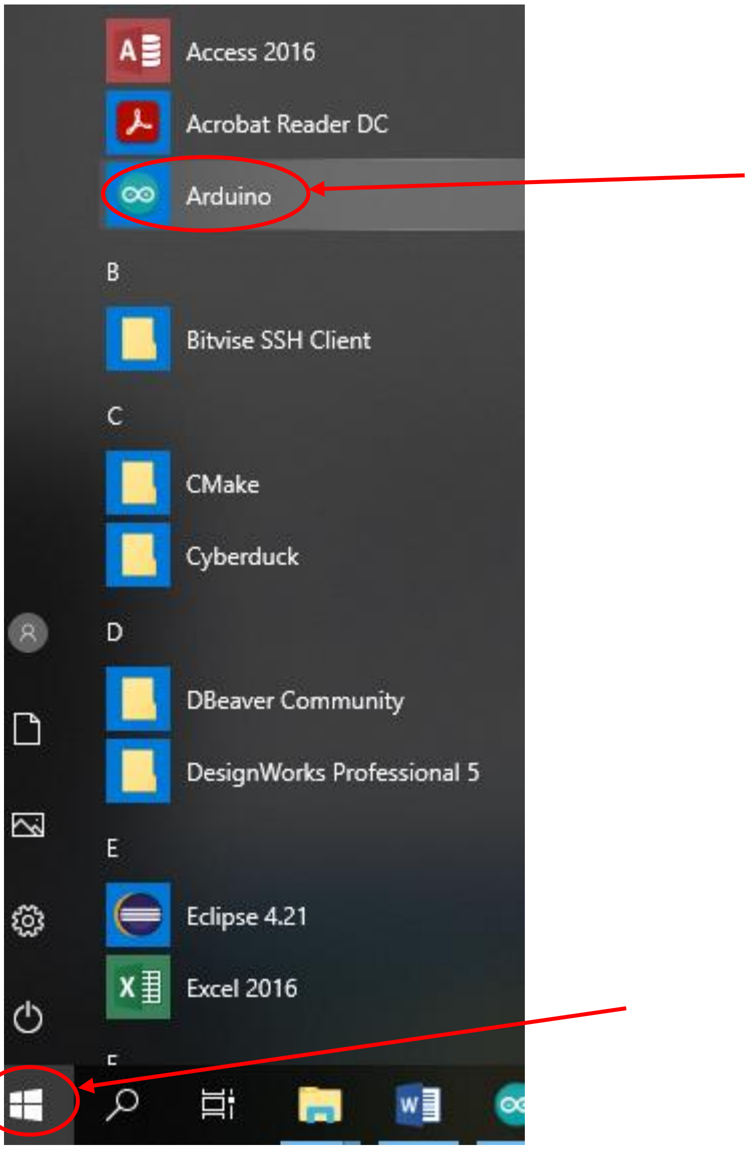
Memory 4

Memory: data MAPPED_IO

```
data 0x0020  00 00 00 00 ff 00 00 00 00  ....ÿ....
data 0x0029  00 00 00 00 00 00 00 00 00  .........
data 0x0032  00 00 00 00 00 00 00 00 00  .........
```

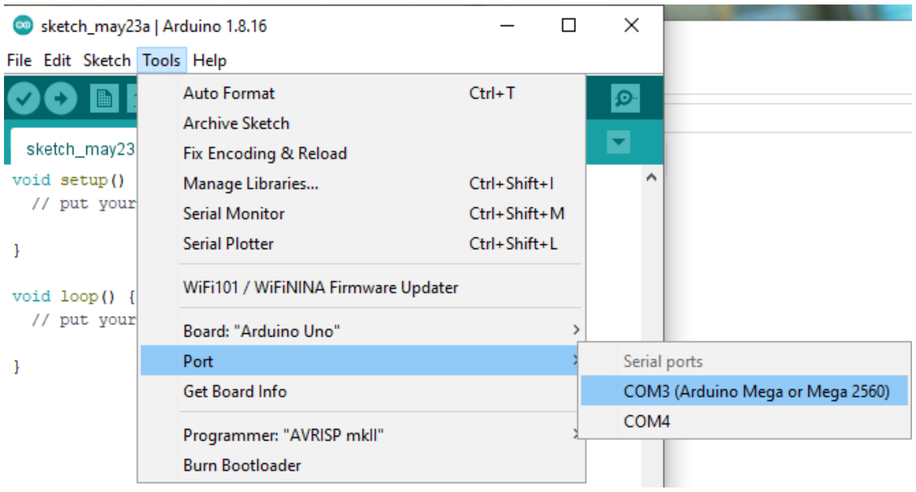
- **Find to which COM port the lab AVR board is connected to. This is different on each machine. You can check this by launch the Arduino IDE. The step by step instructions are:**

- 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.

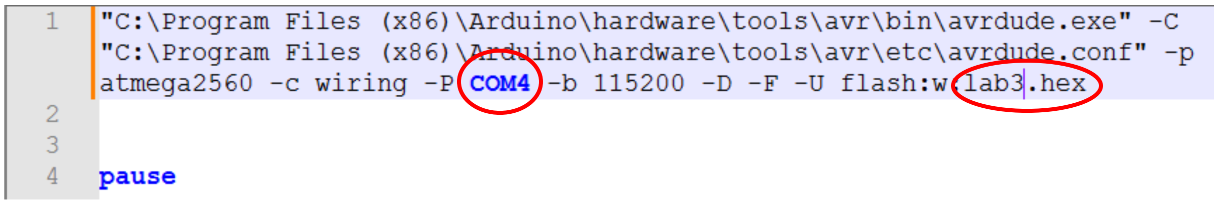

sketch_may23a | Arduino 1.8.16
File Edit Sketch Tools Help
sketch_may23
void setup()
// put your
}
void loop() {
// put your
}
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM3 (Arduino Mega or Mega 2560)
COM4

- Upload the lab3.hex file to the board:

Method I:

Follow the instructions below:

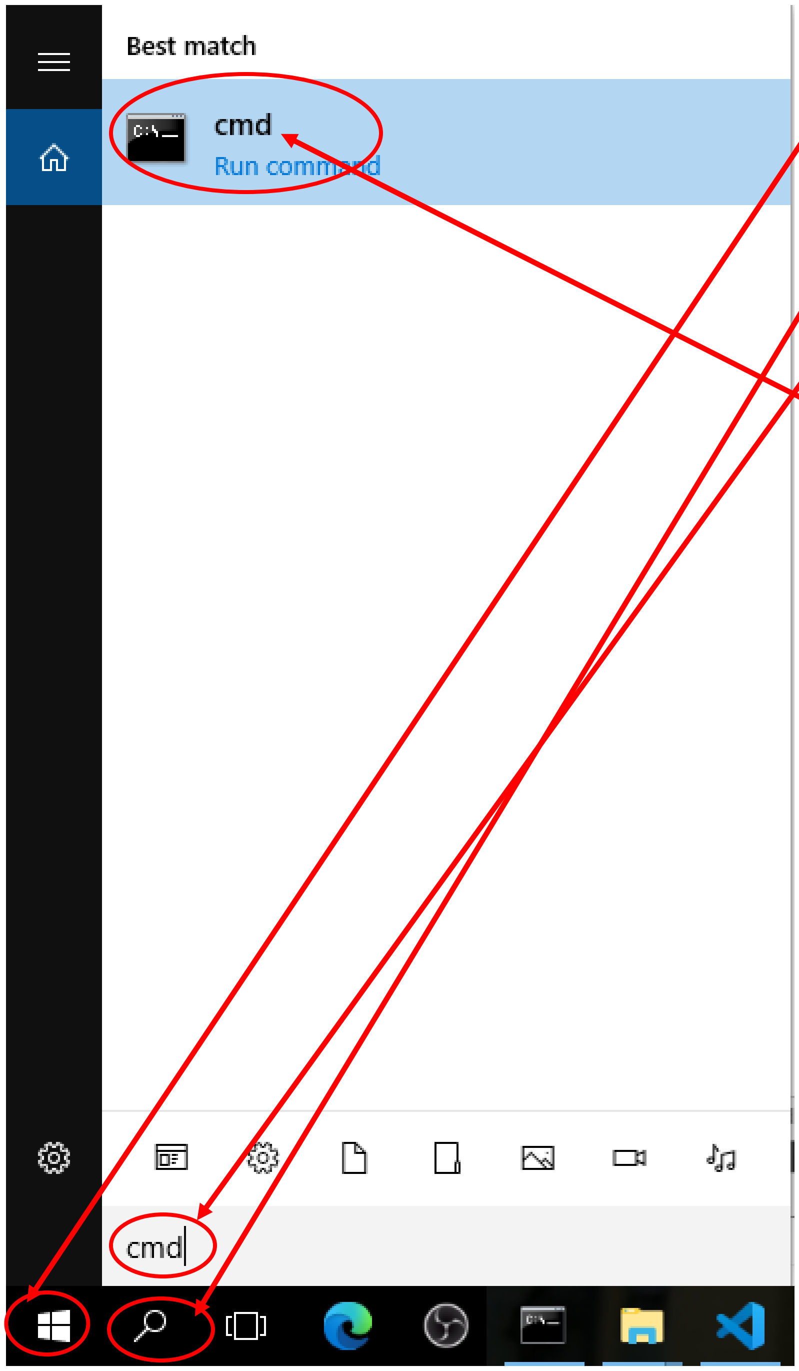
- download upload.bat.txt from the lab 3 folder on the course website
- Save the upload.bat.txt file into the folder where lab3.hex is stored. For example, if the project is called lab3 and saved on H:\230\, then the lab3.hex file is stored at H:\230\lab3\lab3\Debug\. Save the upload.bat.txt file to H:\230\\lab3\lab3\Debug\
- Change the file name from ***upload.bat.txt*** to ***upload.bat***. Right click on upload.bat, and open it with Notepad++, check the port number and .hex file name. If the port is not the same port shown in step 2 on page 9, change it. If the .hex file name is not the same as the one in the Debug directory, change the file name.



```
"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 COM4 -b 115200 -D -F -U flash:w:lab3.hex


pause
```

- Double click on the upload.bat file from the Window's Explorer to upload the .hex file to the Arduino Board.

Method II:



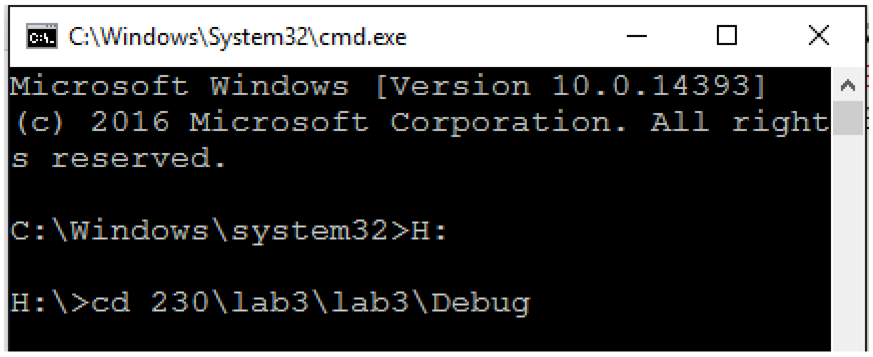
Step a: open the DOS command window, by clicking on the windows icon

Step b: click on the amplifying glass icon

Step c: type cmd

Step d: click on the cmd above

Step e: navigate to the directory containing lab3.hex. Suppose the lab3.hex is stored at H drive in 230\lab3\lab3\Debug:
Go to H drive: type ***H:*** and press the enter key:
Once in H drive, type ***cd 230\lab3\lab3\Debug*** and press enter.
Step h: type ***dir*** to see if ***lab3.hex*** and ***upload.bat*** are in the ***Debug*** directory, then type ***upload.bat*** and press enter. The lab3.hex file should be uploaded from the PC to the board. The screenshot is on the next page.



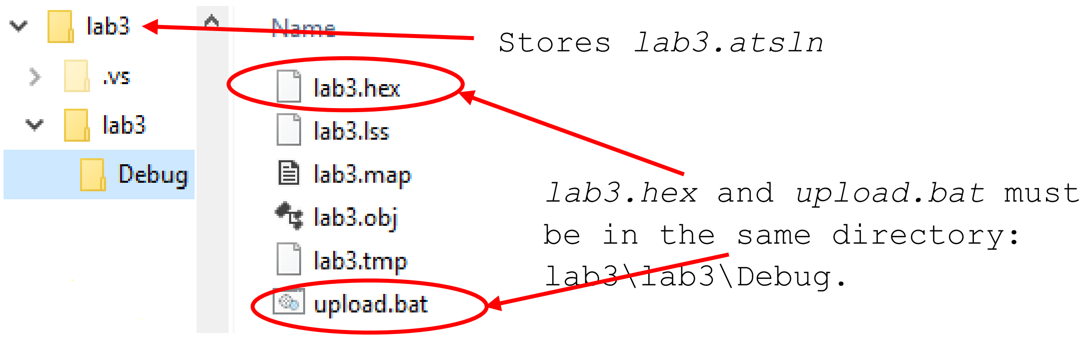
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All right
s reserved.

C:\Windows\system32>H:

H:\>cd 230\lab3\lab3\Debug
```

Method III:
Repeat steps a-e as in "Method II", instead of typing ***upload.bat***, typing 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
```

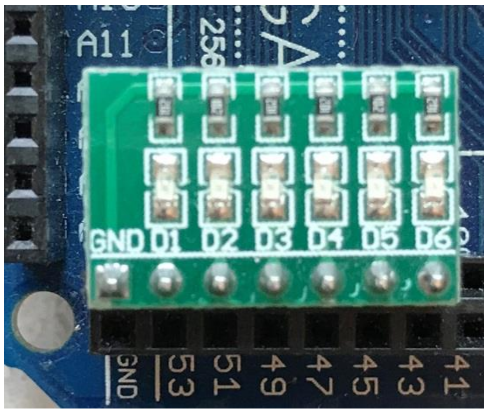
You may need to change the path (e.g. `C:\Program Files (x86)`) and filename (e.g. **lab3.hex**) if the settings are different from this. Instead of typing such a long command, download "upload.bat.txt" file to the same directory where your lab3.hex file is stored. Modify the communication port (-P COM4) 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.

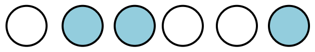
To find the *lab3.hex* file, refer to the following screen shot:



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. (Hint: go to page 5 to see with bit should be 1 if the led controlled by that bit should be on).

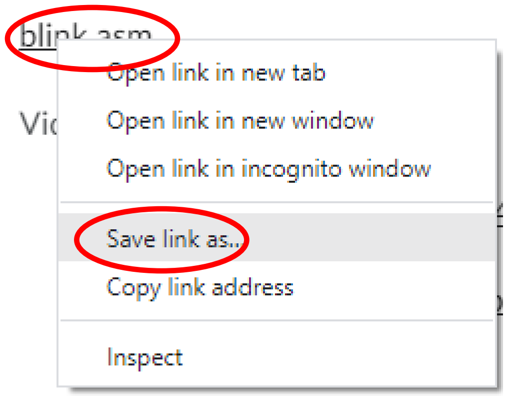




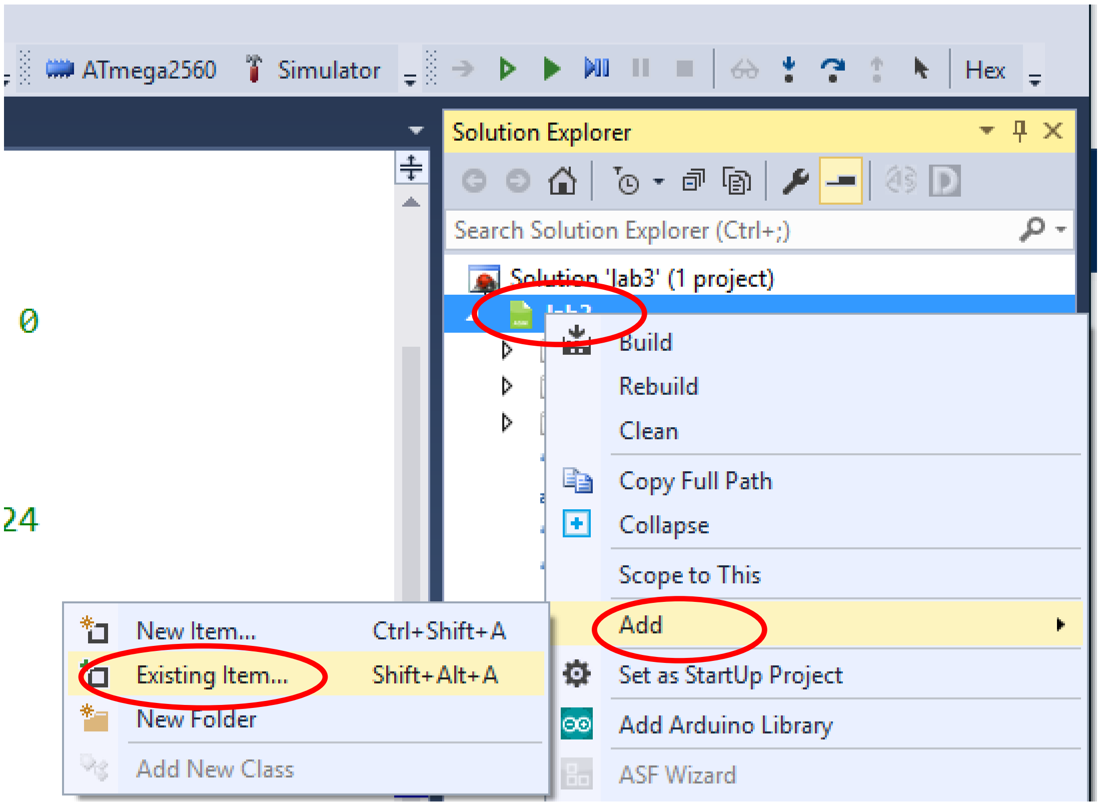
0 1 1 0 0 1

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?). Take a look at the blink.asm to learn more.

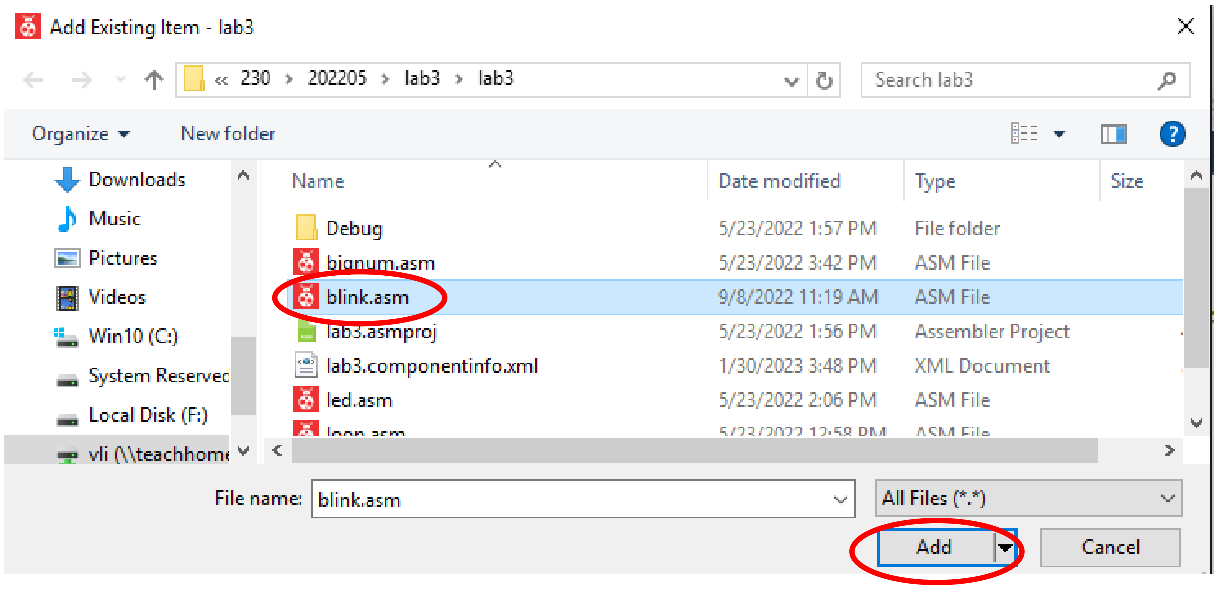
To add blink.asm to your lab3 project, follow the instructions below:
Step 1: Right click on the blink.asm on the lab website, then choose “Save link as …”



Step 2: On the dialogue window, choose the directory where the main.asm of the lab3 project is stored and save the file.
Step 3: Make sure to stop debugging. At the Solution Explorer pane, right click on the project name “lab3” -> “Add” -> “Existing Item” (refer to the screenshot on the next page):



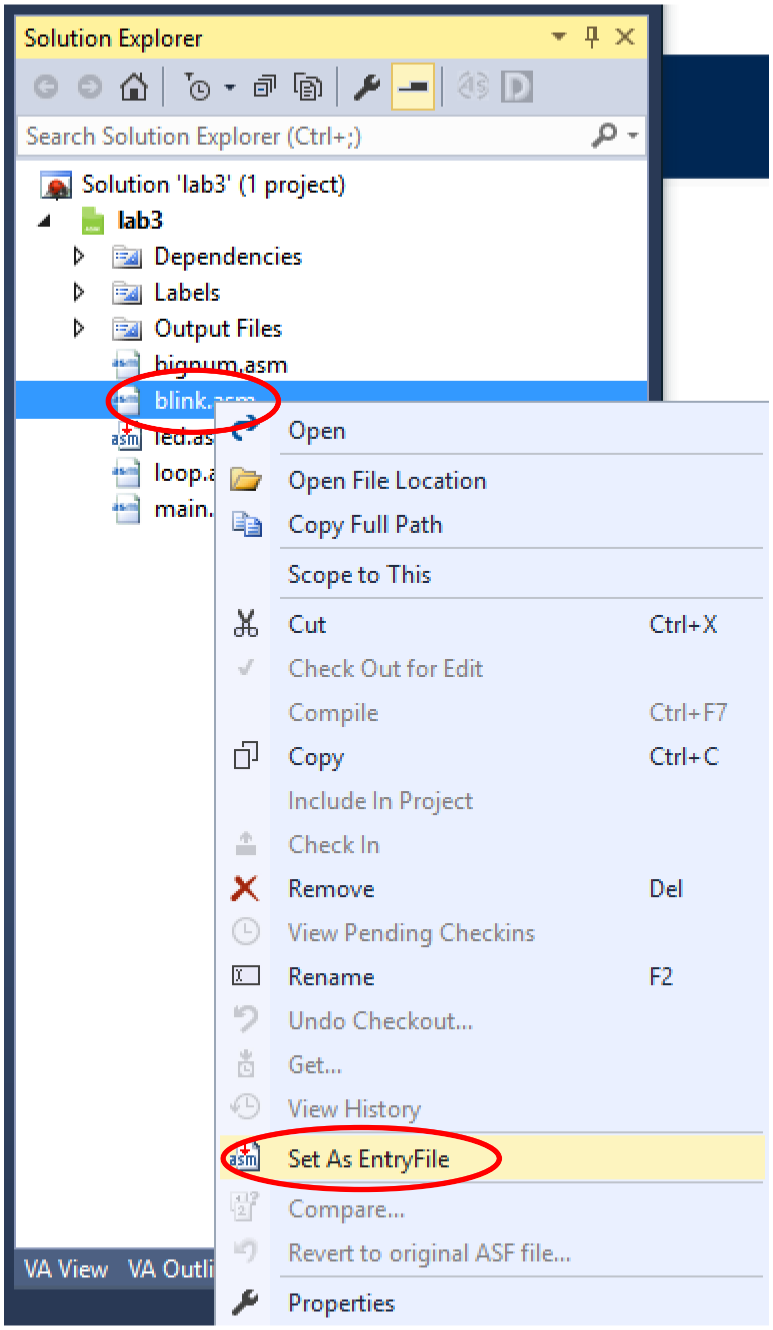
Step 4: on the next dialogue window, click on "blink.asm" and click on the "Add" button:



Name	Date modified	Type	Size
Debug	5/23/2022 1:57 PM	File folder	
bignum.asm	5/23/2022 3:42 PM	ASM File	
blink.asm	9/8/2022 11:19 AM	ASM File	
lab3.asmproj	5/23/2022 1:56 PM	Assembler Project	
lab3.componentinfo.xml	1/30/2023 3:48 PM	XML Document	
led.asm	5/23/2022 2:06 PM	ASM File	

File name: blink.asm

Step 5: set blink.asm as entry file:



Step 6: build and upload.