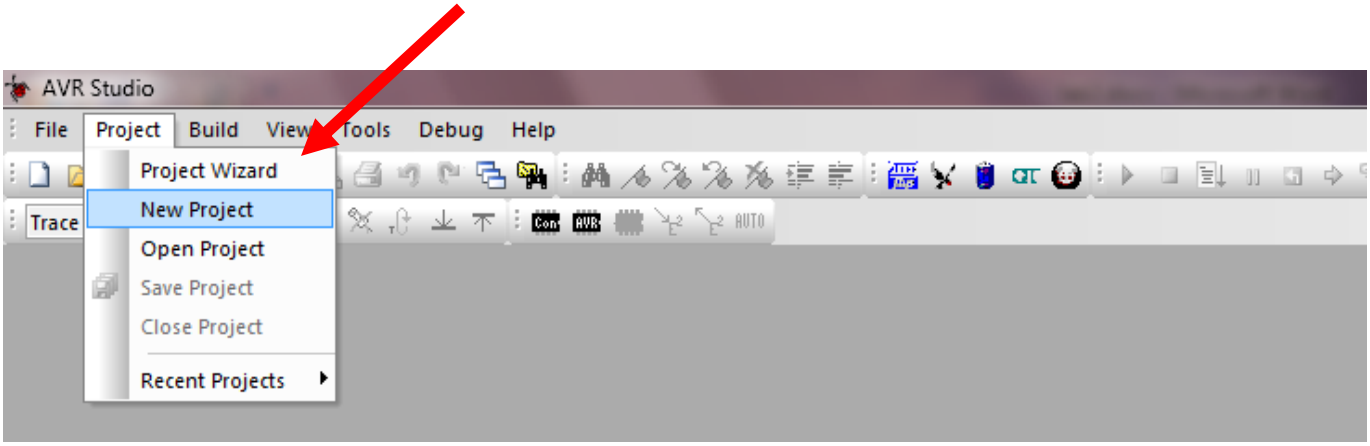
Lab 2 Introduction to Assembly Language

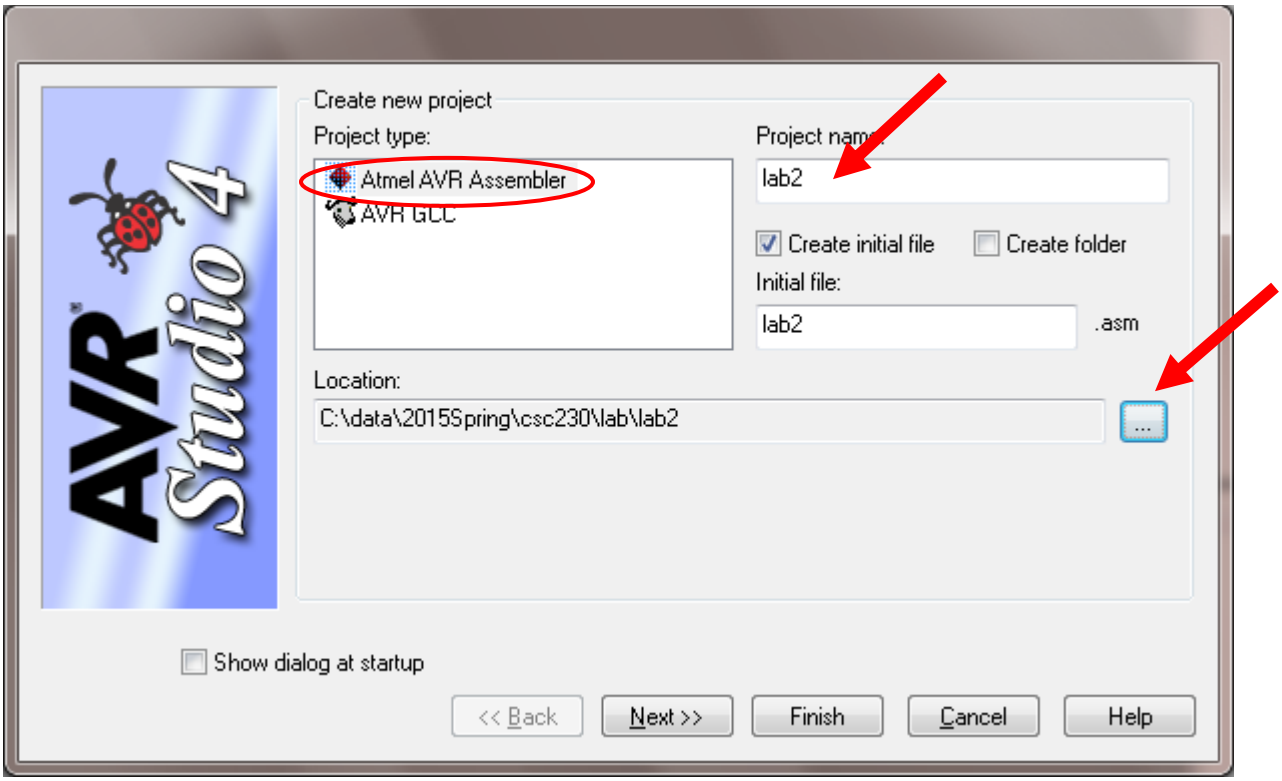
Submit Lab2_checkEven.asm at the end of your lab, not your project. This lab introduces you to the programming environment that we use in CSc 230 lab.

I. Introduction to AVR Studio 4

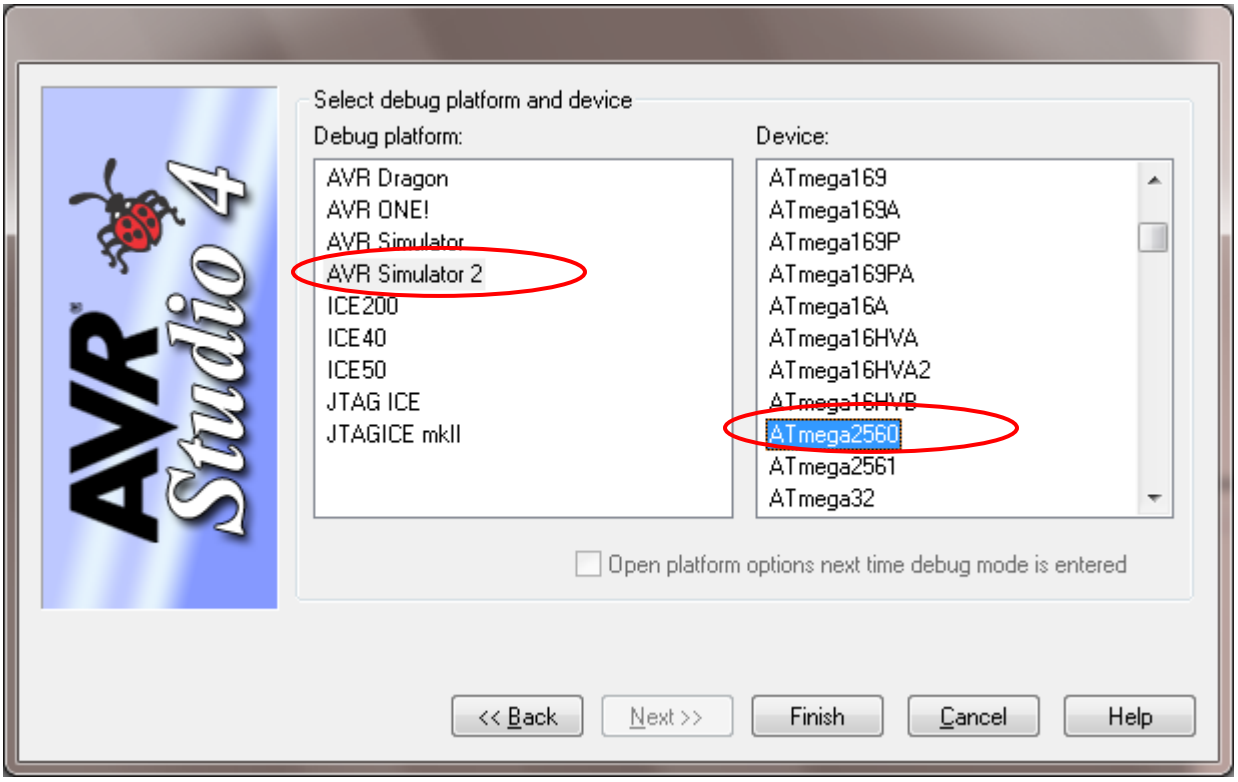
Launch AVR Studio 4 and create a new project named "lab2"



Select "Atmel AVR Assembler" for "Project Type", "lab2" for "Project name" and click on the "…" button to select a directory (suggest H drive).



Click the “Next>>” button and get to this page:
Click on “AVR Simulator 2” for “Debug Platform”
Click on “ATmega2560” for “Device”
Click on “Finish”

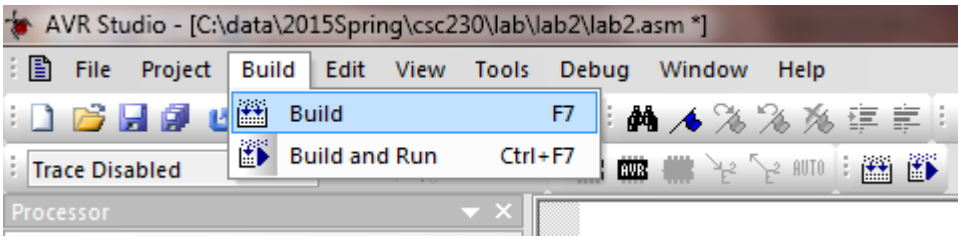


Type the following code:

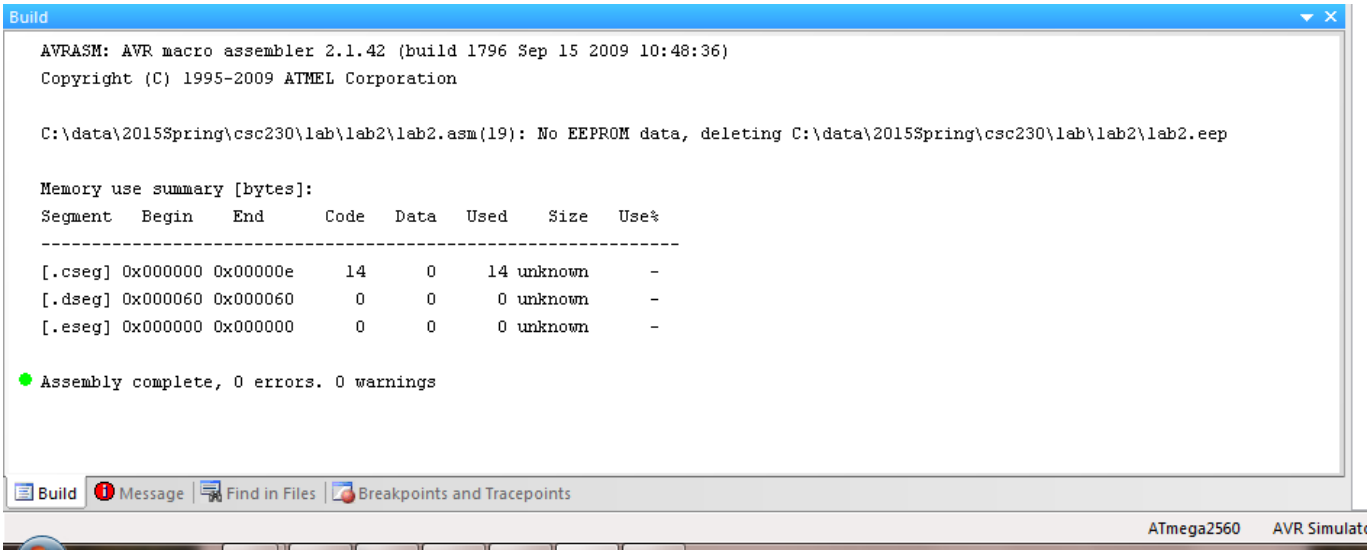
```
 .cseg       ;select current segment as code
 .org 0      ;begin assembling at address 0

 ;Define symbolic names for resources used
 .def count =r16         ;Reg 16 holds counter value

      ldi    count, 0x00 ;Initialize count at 0

lp:
      inc count          ;increment counter

      cpi count, 0xff
      breq done
      rjmp lp

done: jmp done
```

Save the code.

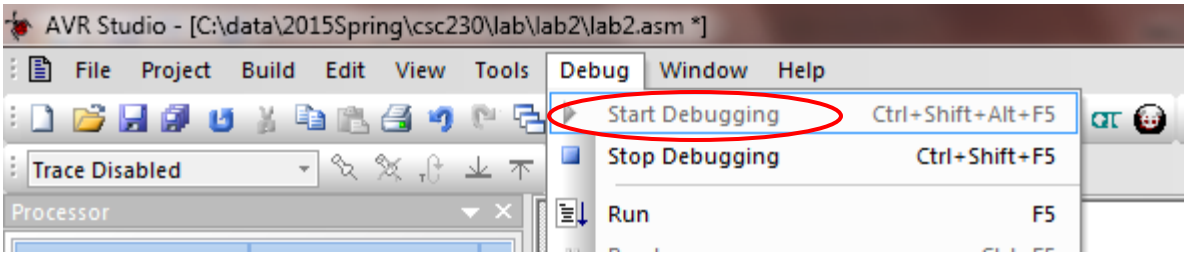
Assemble it (choose Build under the Build menu)



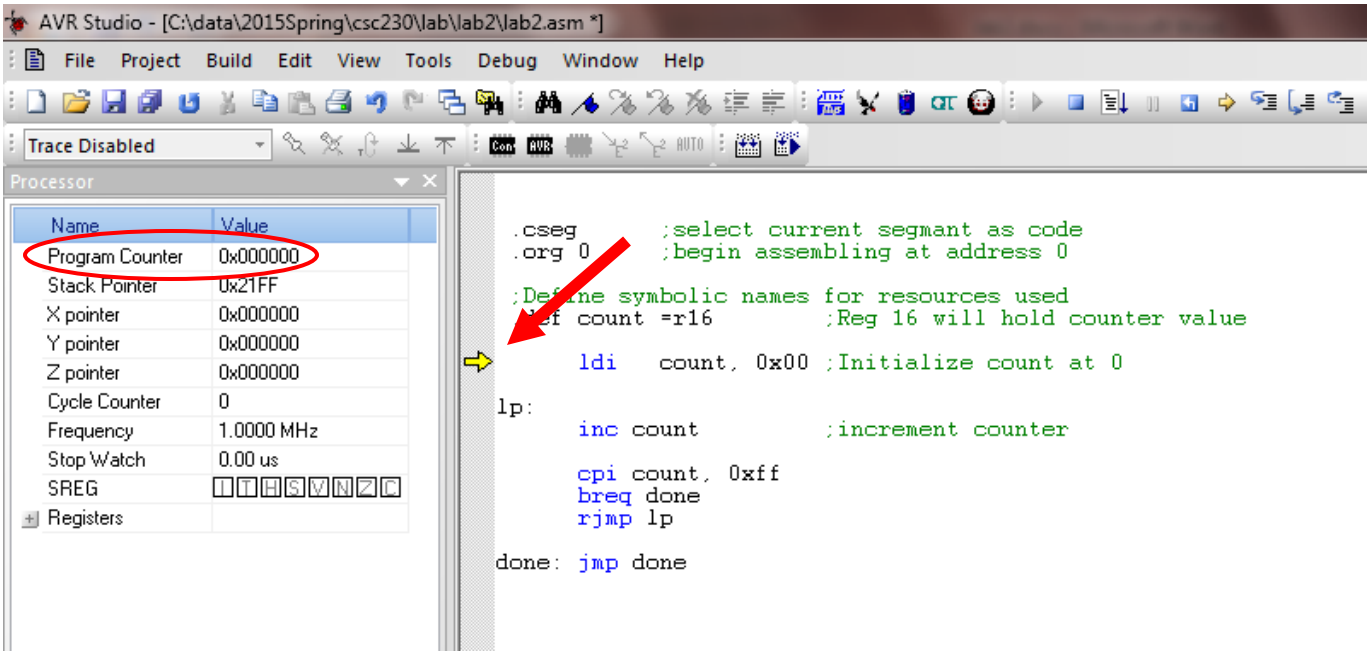
Output from the build at the bottom of the screen:



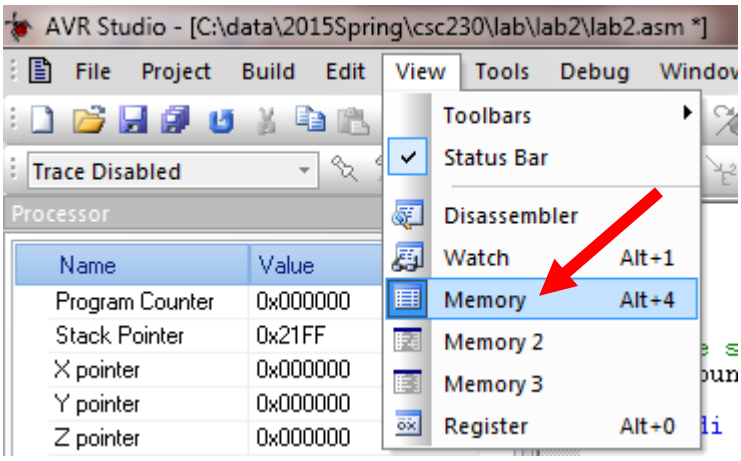
If no errors are detected, start the simulator (debugger). Select "Start Debugging command" under the "Debug" menu.



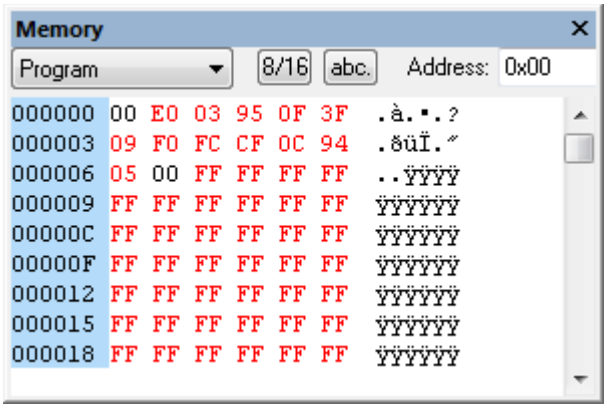
The editor should show a yellow arrow indicating the instruction about to be fetched. The left panel should show the "Processor panel, where you can examine register and processor values"



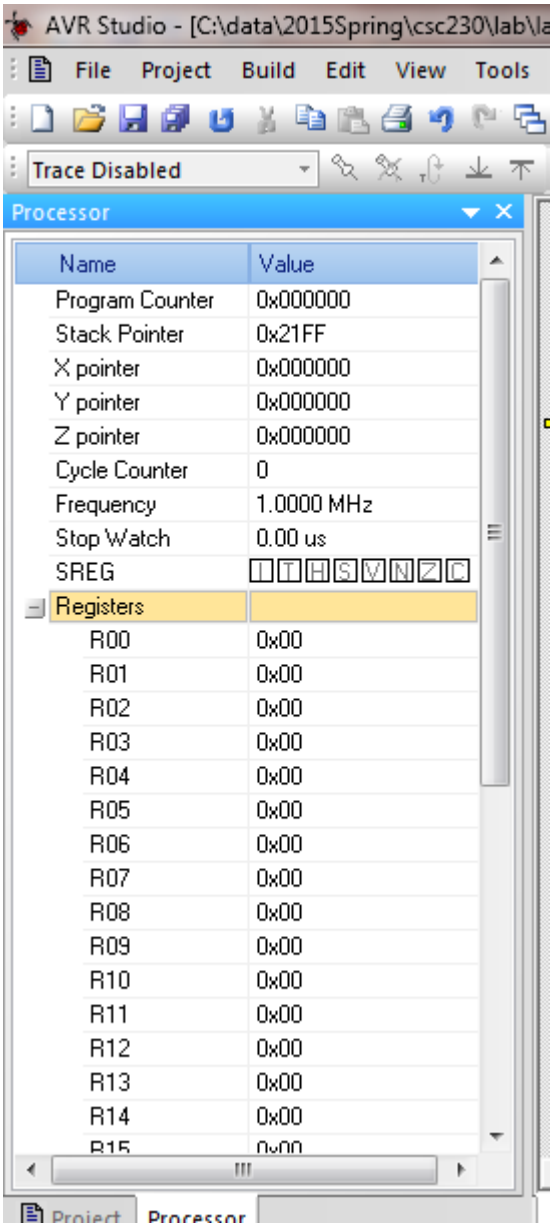
Before executing any instructions, open the "Memory" window (Under the "View" menu):



The memory looks like this: (verify the opcode)



Expand the tree node labeled “Registers”, or choose “View/Register” from the “View” menu to open a popup window.



Select the “Step Into” option from the “Debug” menu (or press F11) to allow the program to fetch and execute the first instruction. Observe the changes in PC (program counter) and register 16 (r16).

Stop the debussing session by using the command under the “Debug” menu.

II. Write some opcodes and verify them using the opcodes in the memory. Are they big-endian or little-endian? Choose one line of code and figure out its opcode.

III. Write a program called lab2CheckEven.asm. The program checks if an 8-bit unsigned number is an even number. If the number is even, it outputs 0x00, otherwise 0xFF. Let's use R18 for input and R19 for output. To test the program, initially set the input to 0x09 and test it. Then change the input to 0x08 and manually check if the output is correct. Here is the pseudo code for the algorithm:

- Initialize R19 (set it to 0x00): R19 <= 0x00
- Set R18 to 0x09 as input to test: R18 <= 0x09
- Set all bits except the last (least significant bit) of R18 to 0 (hint: use ANDI with a mask)
- If the value of R18 is equal to 0x01, set R19 to 0xFF (hint: you can use zero flag to test this)
- Done

Submit lab2CheckEven.asm at the end of your lab. You can submit this by using the Dropbox tool on conneX.

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