Lab Introduction to Assembly Language

I. Bitwise Manipulation Continues

For more information about bitwise operations in the AVR instruction-set architecture, refer to the file named table_01_bit_operators.pdf (provided by Dr. Michael Zastre) in the same folder.

Common group sizes and names:

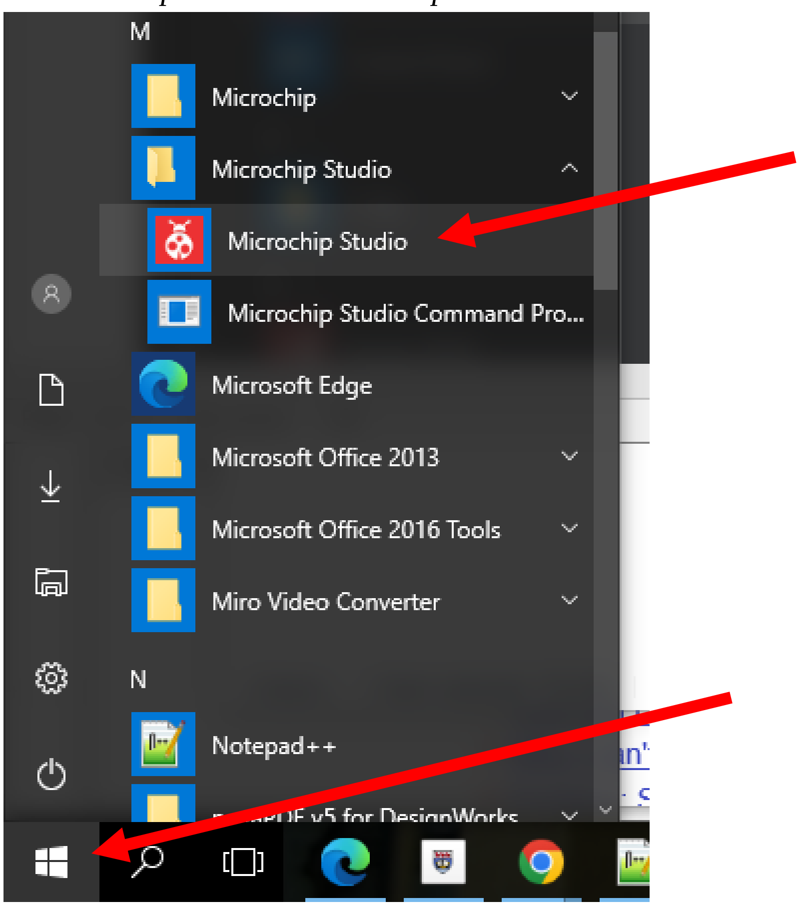
nybble (also spelled nibble):	4 bits
byte:	8 bits
word:	16 bits (for the AVR architecture used in this course)

Exercise 1: Determine the result of performing a bitwise OR operation on this pair of nybbles: 0b1001 and 0b 1100.

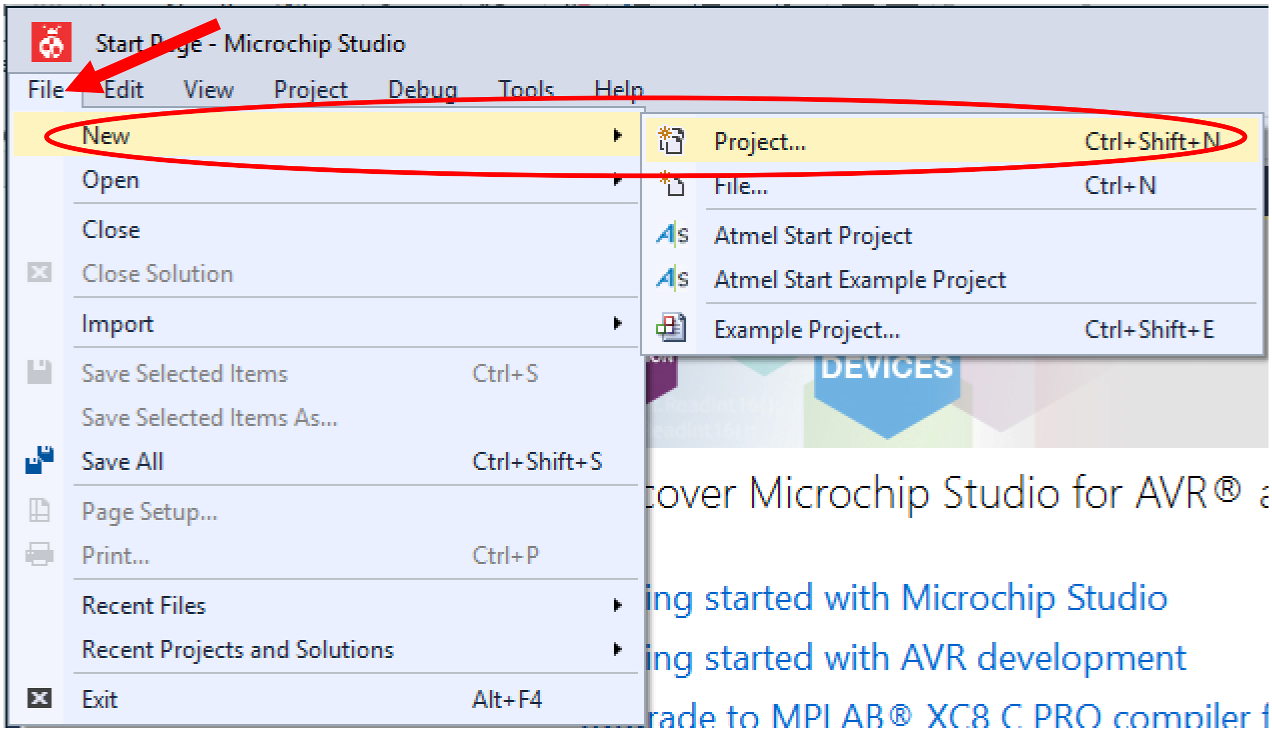
Exercise 2: What mask and operation would be used to zero the upper nibble in a byte? To clear the lower nybble of a byte?

II. Introduction to Microchip Studio

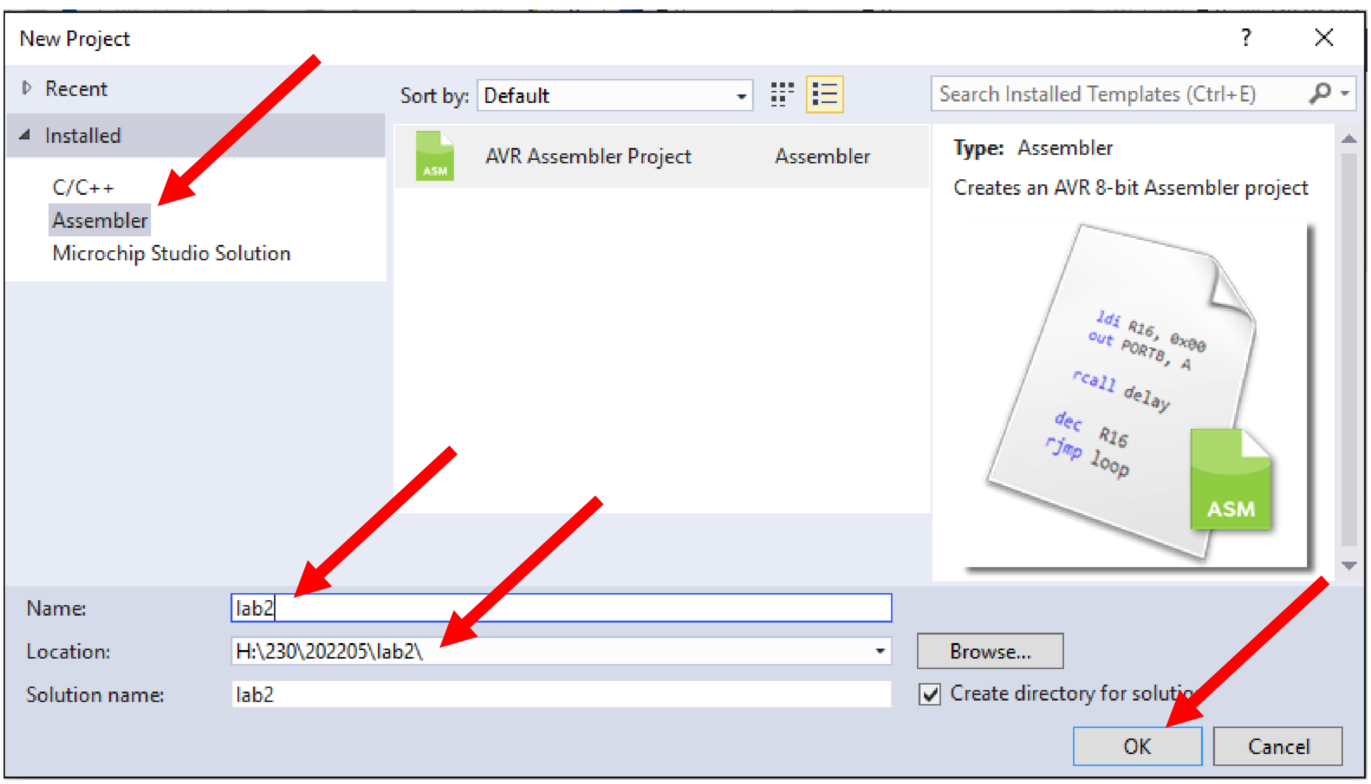
Launch *Microchip Studio* and create a new project named "lab2": click on the *Start* button, then click on *Microchip Studio* -> *Microchip Studio:*



Create a new project named *lab2*: on the menu, click on File -> New -> Project:



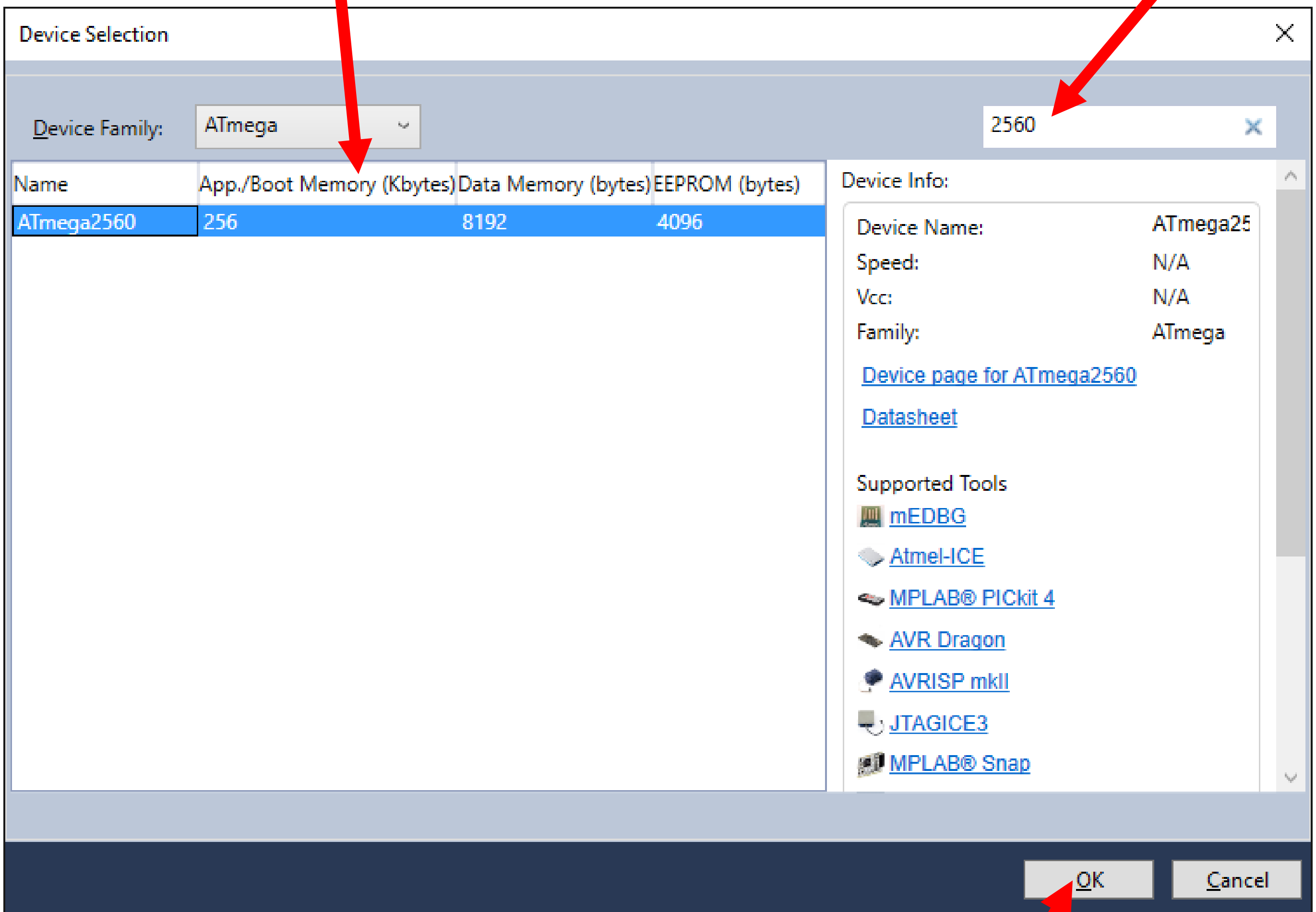
In the new dialog box, on the left pane, under *Installed*, select *Assembler*. Type the project Name *lab2* and select the Location (suggest you use H drive). Click on the *OK* button.



Then a new dialog box appears: click on *ATmega2560* as the *Device* for this project. Click on the *OK* button.

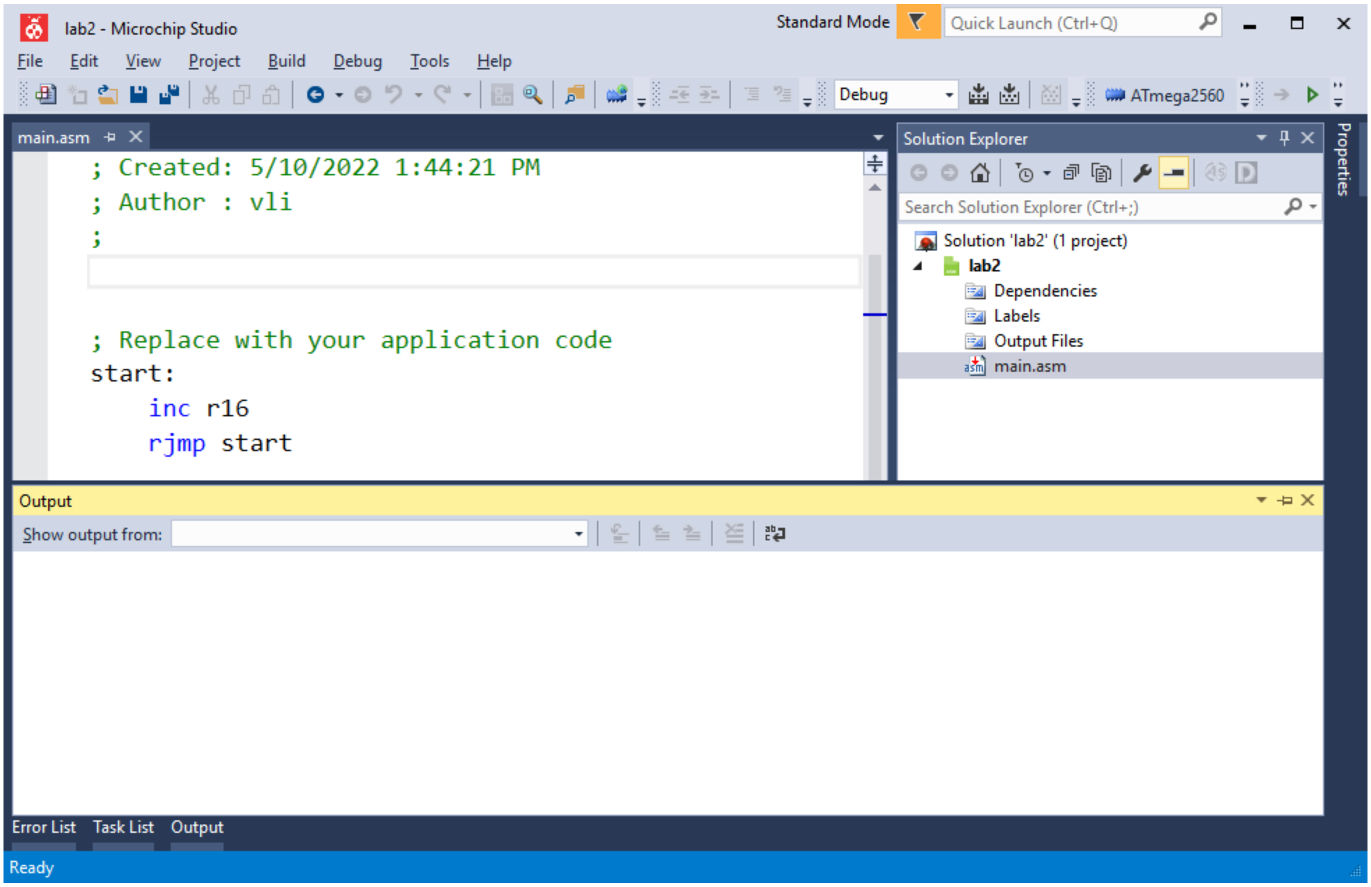
Step 2: click on *ATmega2560*.

Step 1: Type *2560* to list only the device families containing *2560*.



Step 3: Click on the *OK* button.

The new project looks like this:

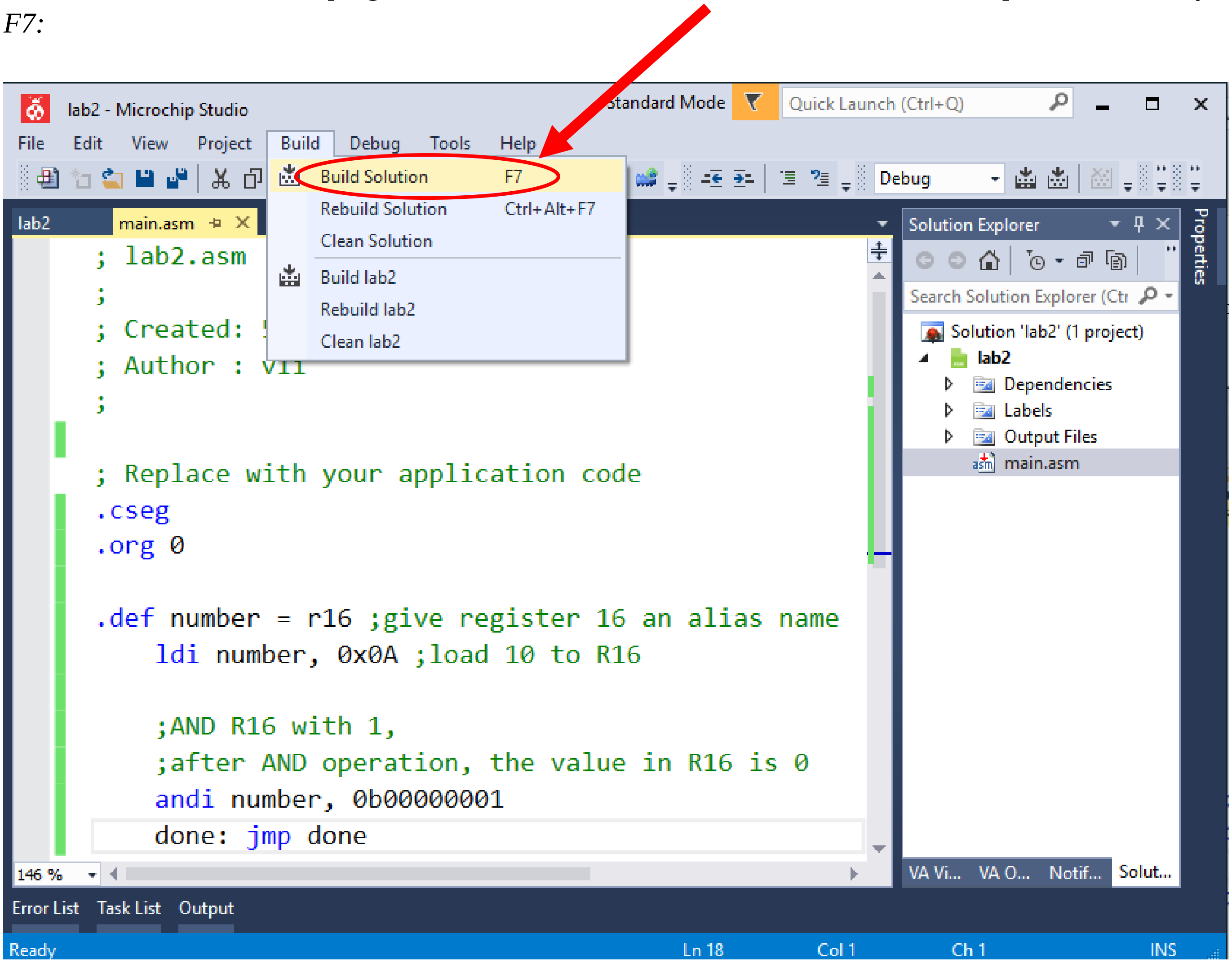


Delete the default code and type the following code:

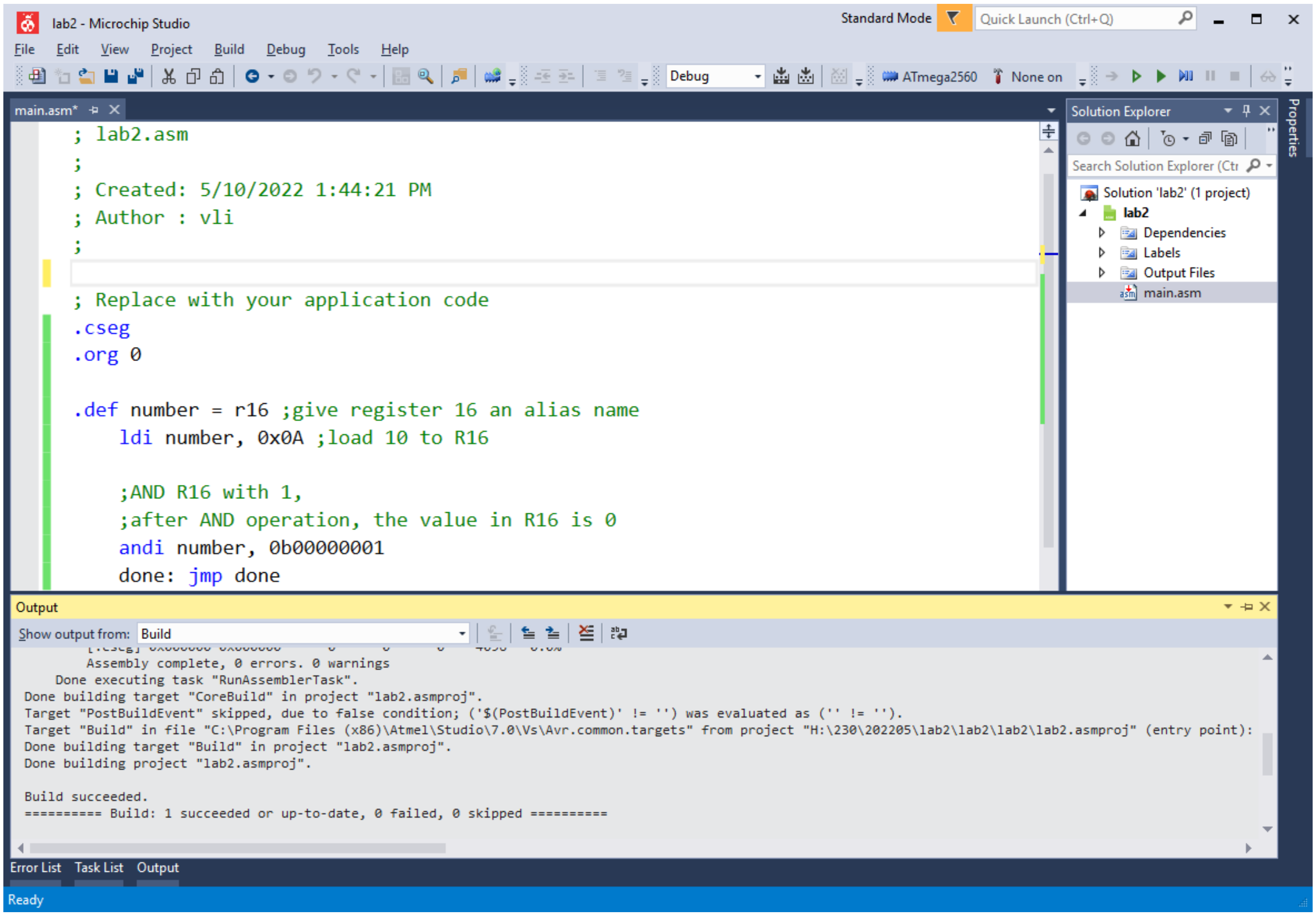
main.asm

```
; Created: 5/10/2022 1:44:21 PM
; Author : vli
;

; Replace with your application code
.cseg
.org 0

.def number = r16 ;give register 16 an alias name
    ldi number, 0x0A ;load 10 to R16

    ;AND R16 with 1,
    ;after AND operation, the value in R16 is 0
    andi number, 0b00000001
    done: jmp done ;infinite loop
```

Save the code and build the program: on the menu, click on *Build -> Build Solution* or press shortcut key *F7:*

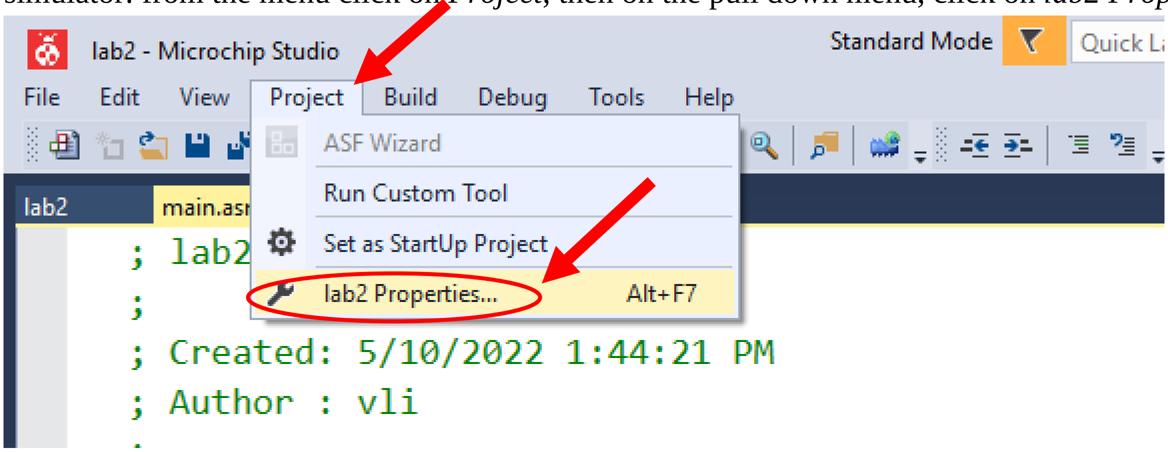


After building the program, the screen looks like this:

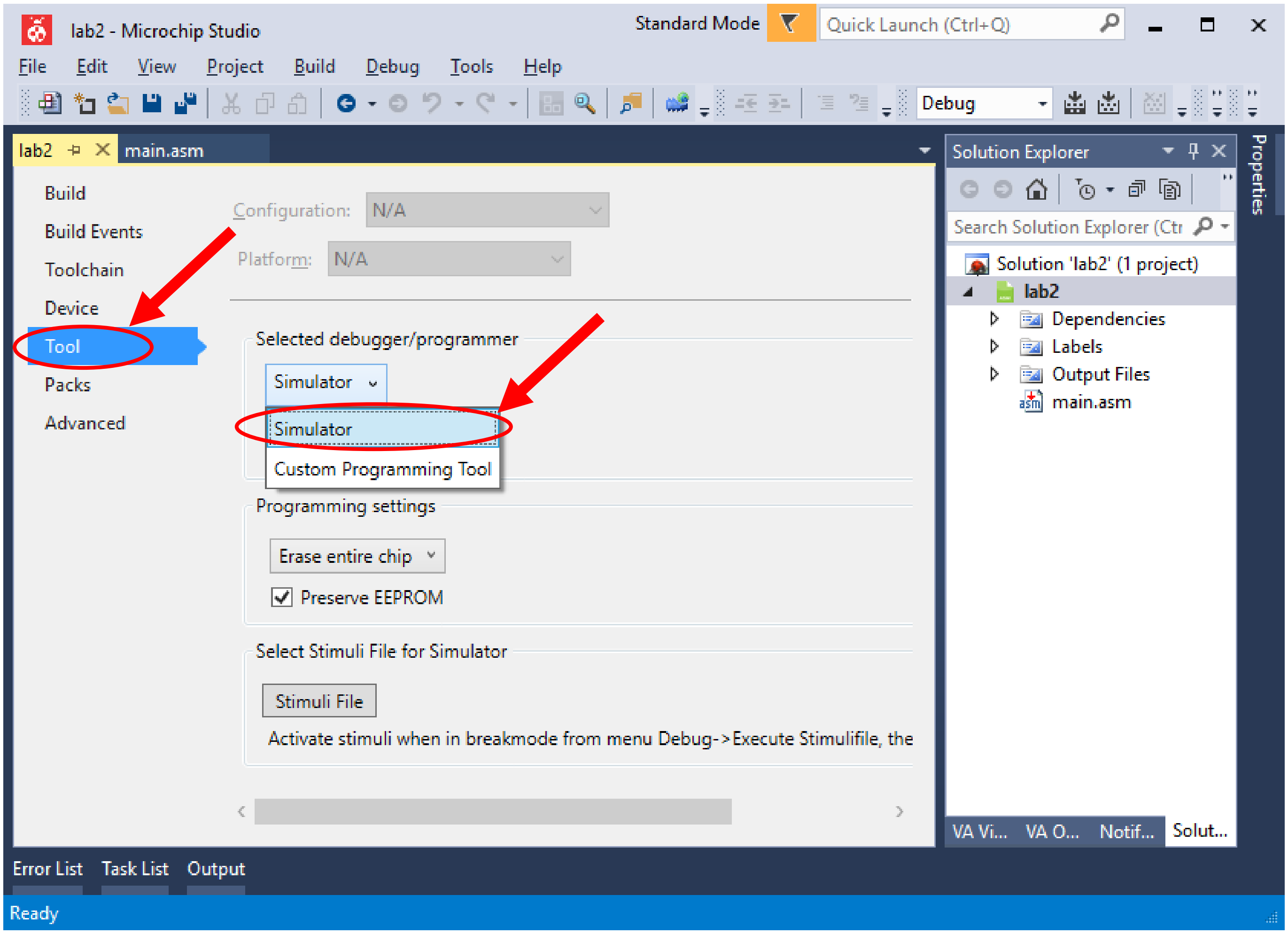


If there are any errors, you fix them and rebuild your program until there are no build errors.

Now, you are ready to run your program using the simulator. Set up the proper configuration of the simulator: from the menu click on *Project*, then on the pull down menu, click on *lab2 Properties…:*

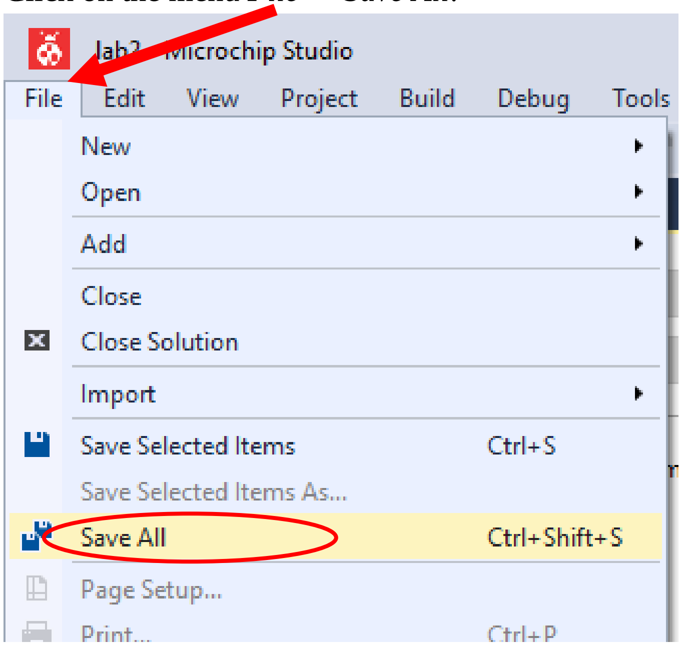


In the *lab2* tab, click on *Tool* and select *Simulator*:

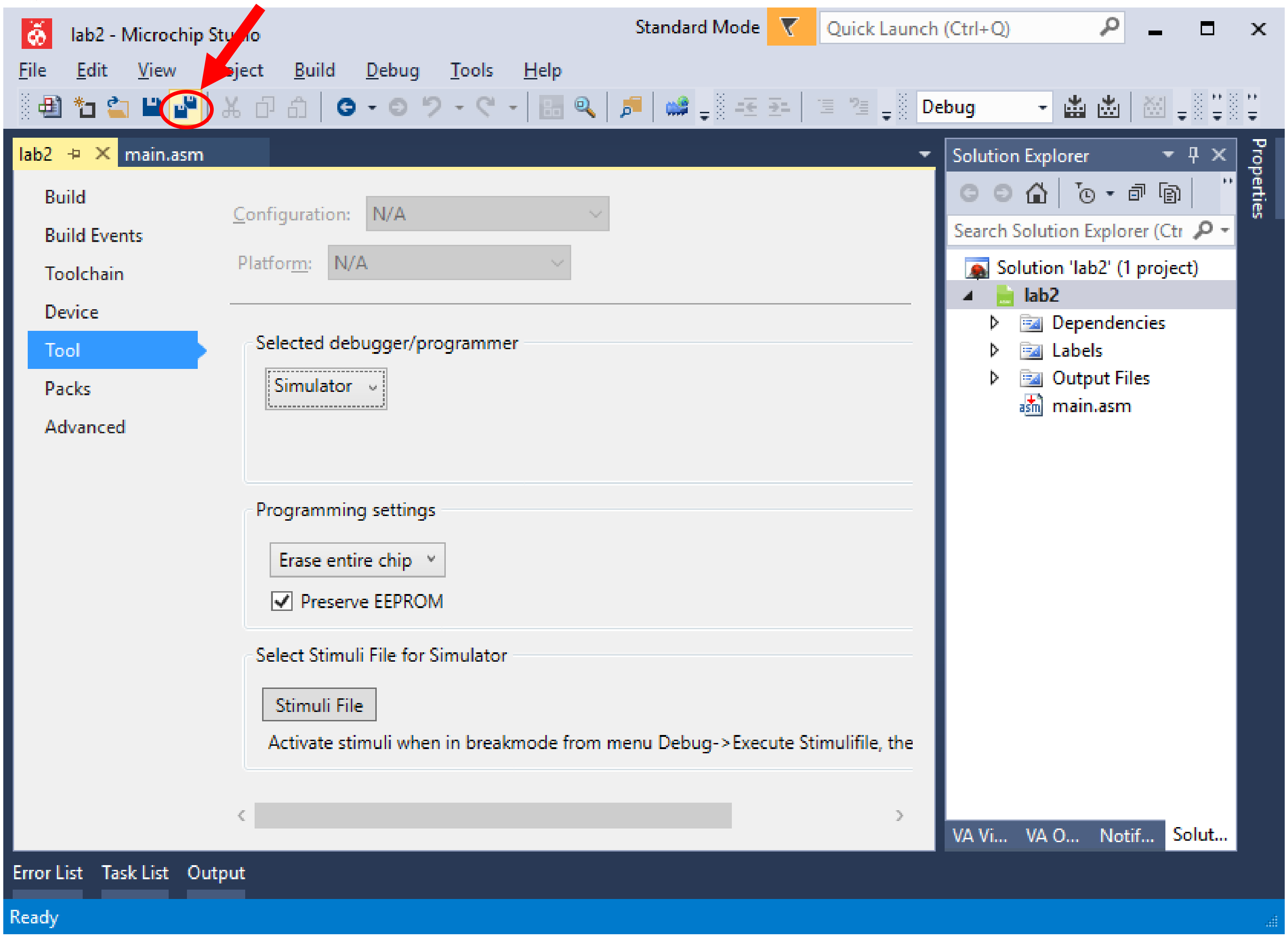


Save the project in two ways:

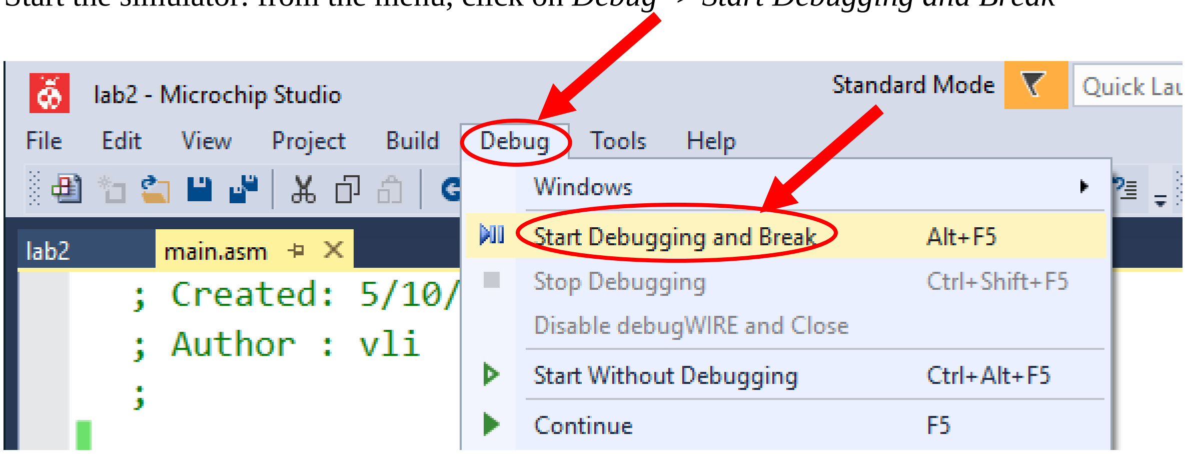
1. Click on the menu *File -> Save All*:



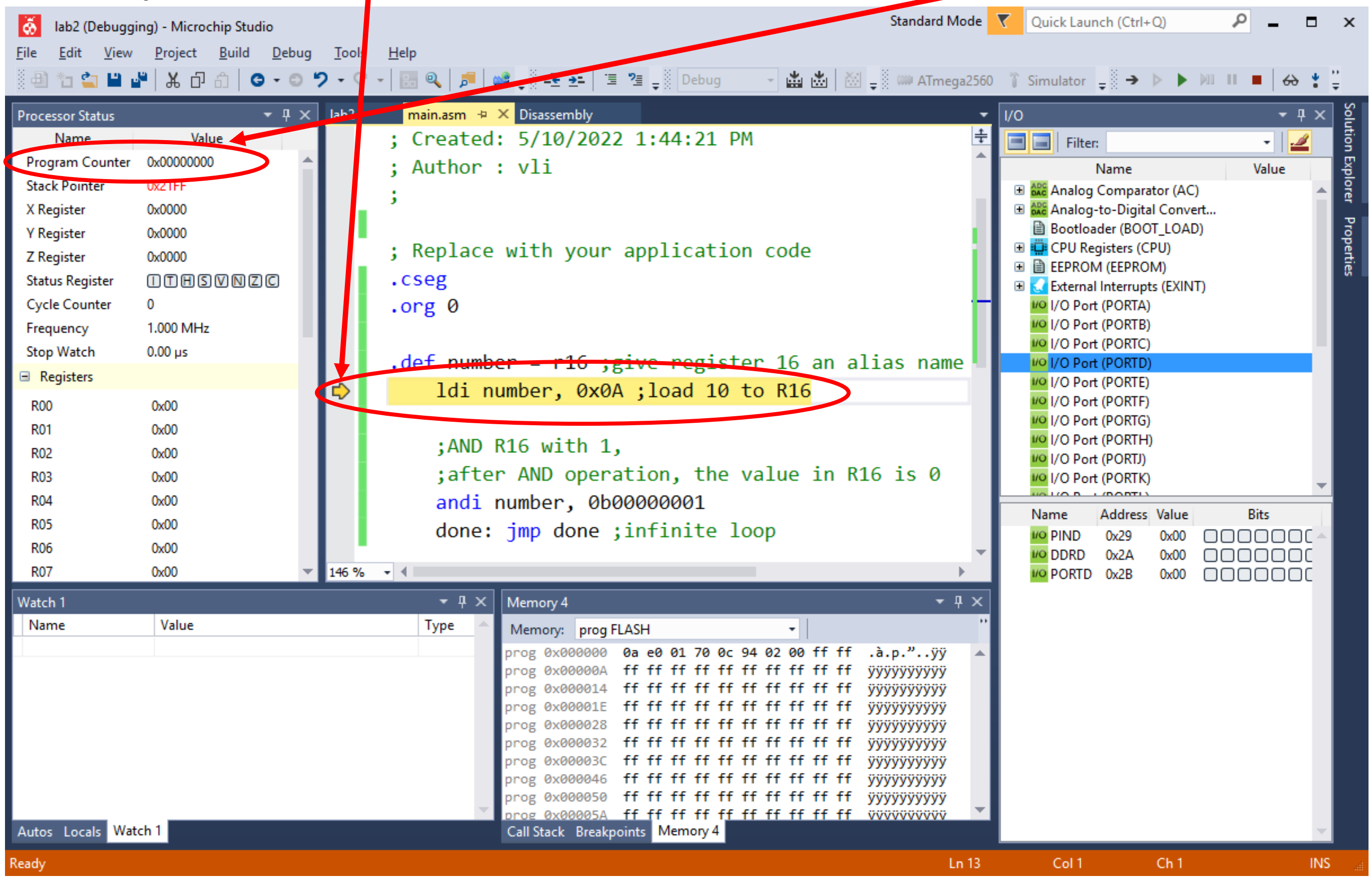
2. Or, click on the icon :



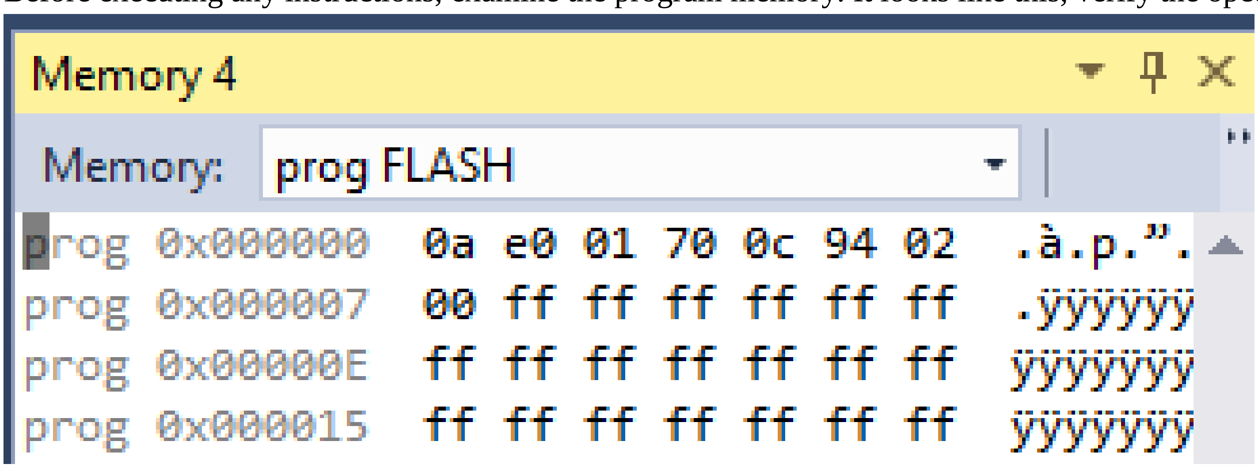
Start the simulator: from the menu, click on *Debug -> Start Debugging and Break*



On the screen, there is a yellow arrow indicating the instruction about to be fetched. The left panel shows the "*Processor Status*", where you can examine the values in registers. The *"Program Counter"* shows the memory address of the next instruction to be fetched.

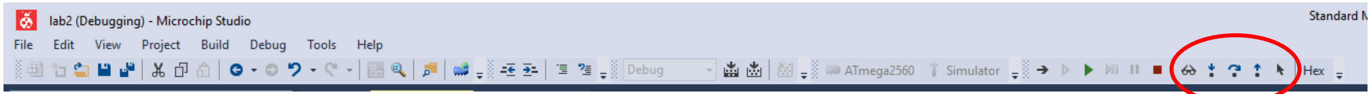


Before executing any instructions, examine the program memory. It looks like this, verify the opcode:

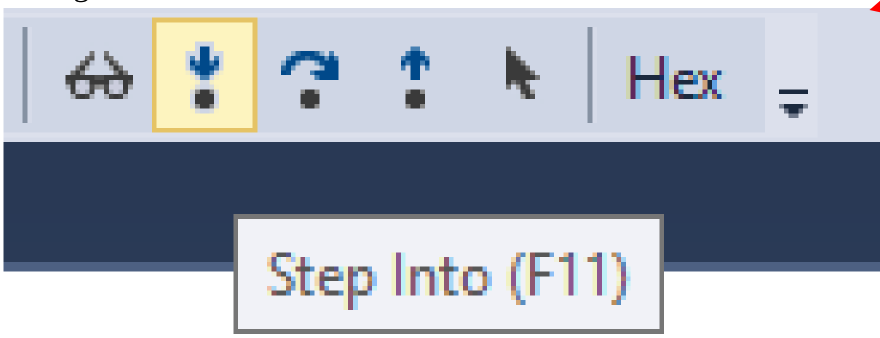


Fetch and execute the first instruction: click on the *Step Into* button (or press the F11 key) in three ways:

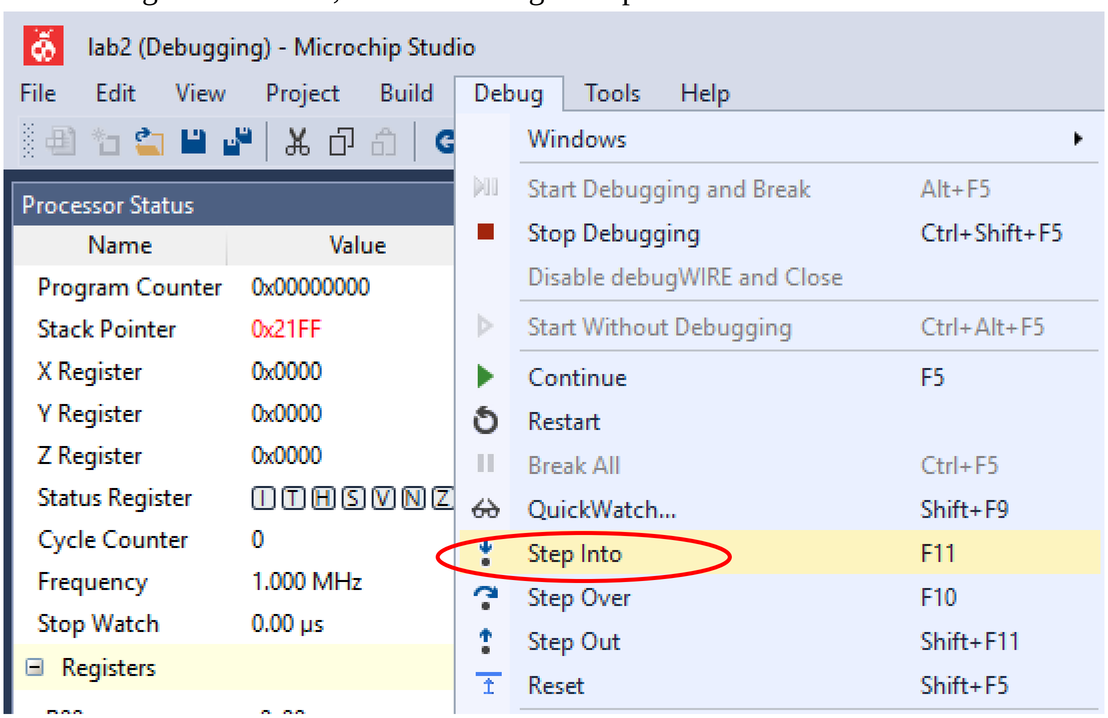
1. Click on the icon:



Enlarge the icons:

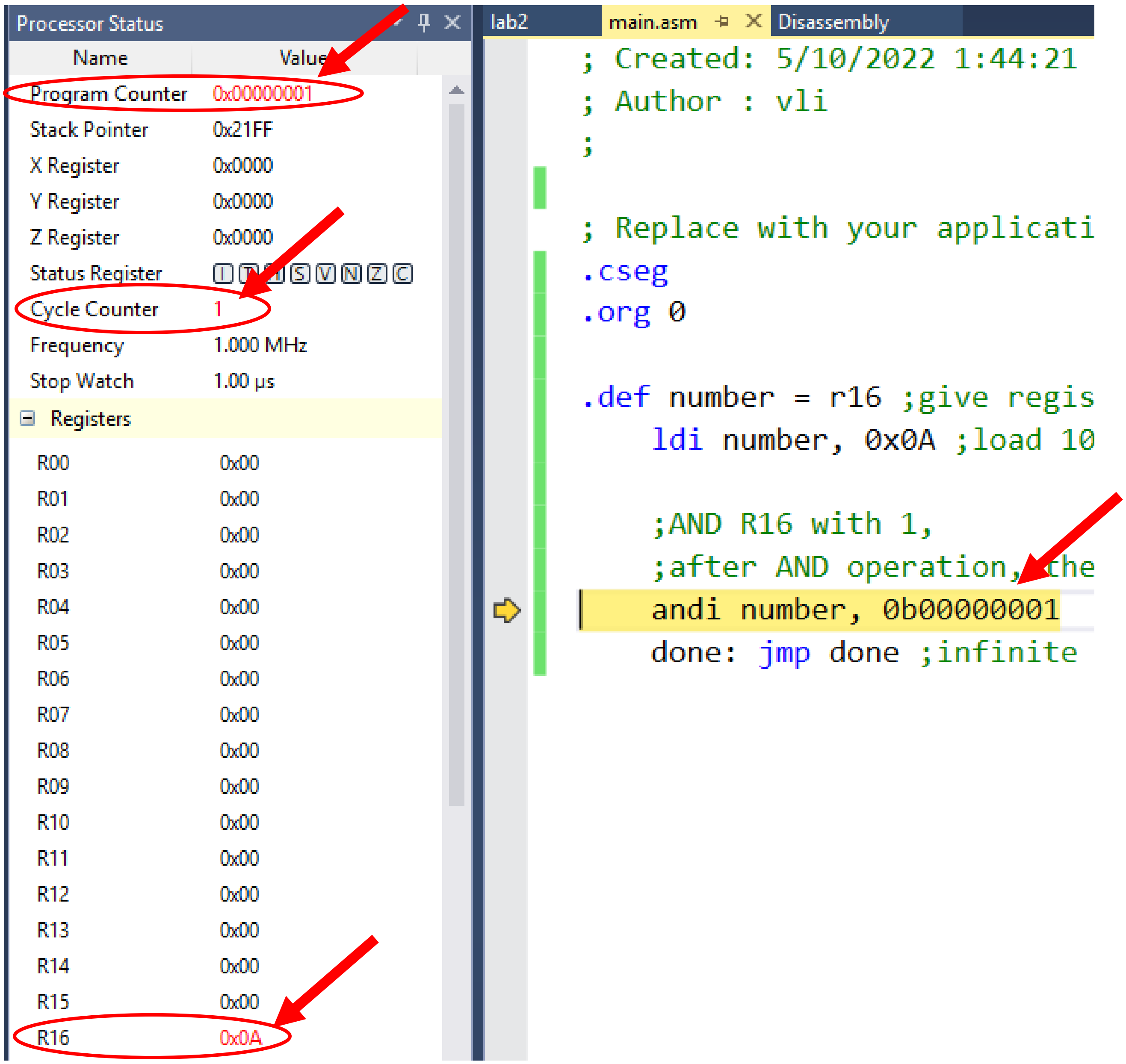


2. Or go to the menu, click on Debug ->Step Into:

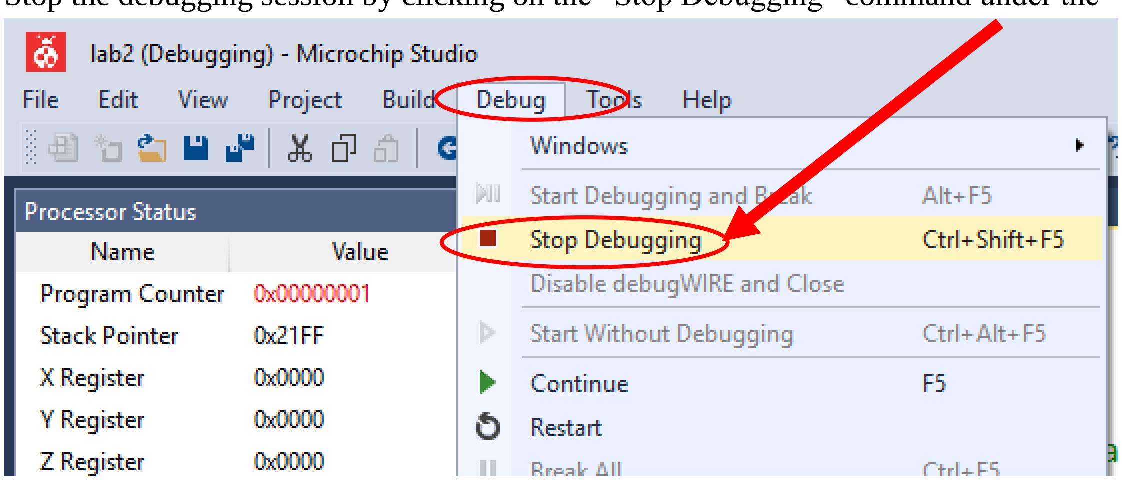


3. Or press shortcut key F11

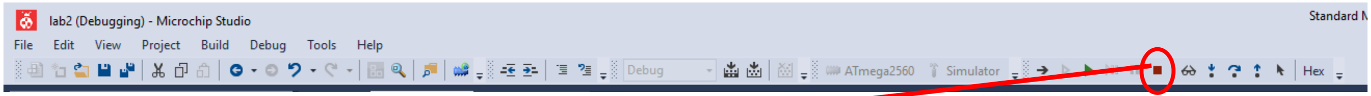
After executing the first instruction, the value in register 16 (R16) is changed from 0x00 to 0x01. The changed values are in red, such as the Program Counter, Cycle Counter, in addition to R16:



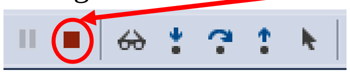
Stop the debugging session by clicking on the “Stop Debugging” command under the “Debug” menu:



Or click on the stop icon:



Enlarge the icon

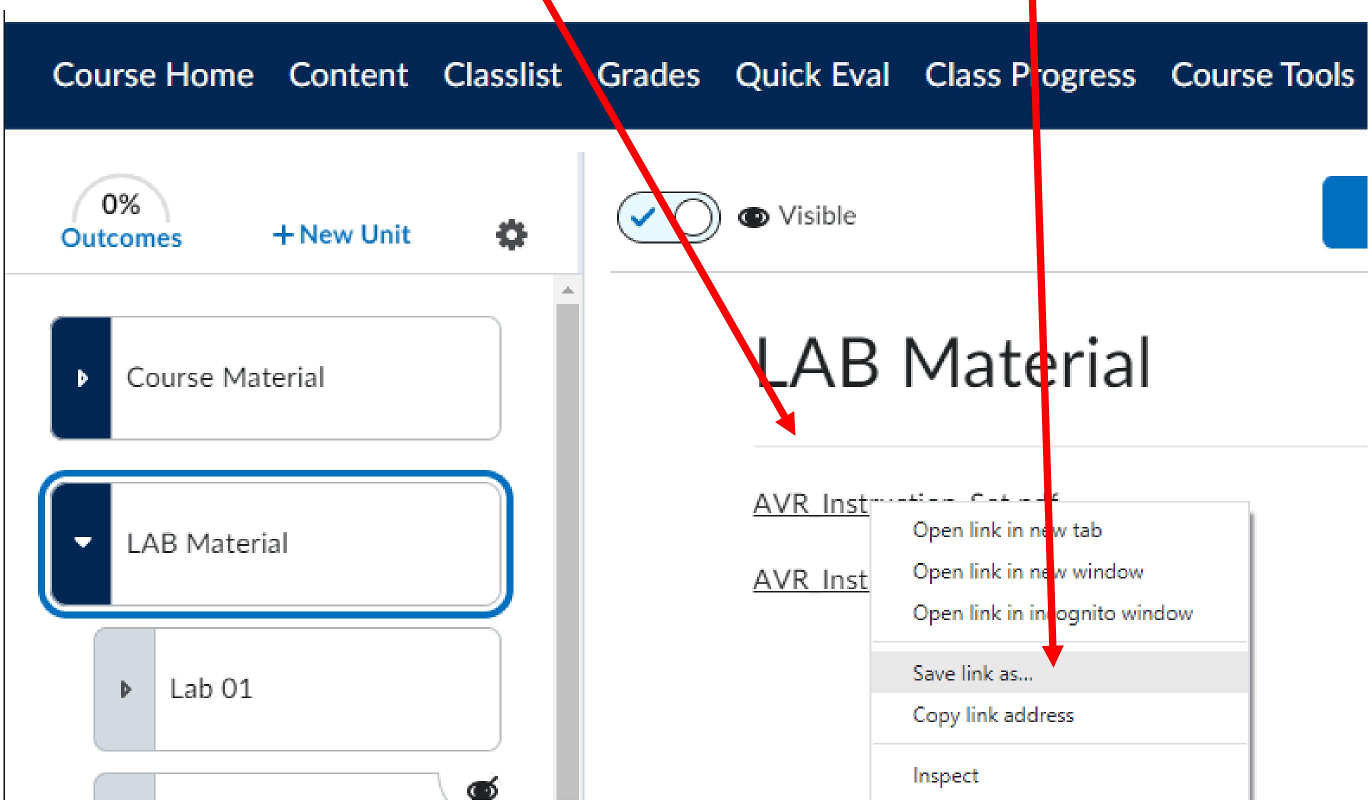


III. Write some assembly language code (mnemonics) and convert it to machine instructions. Verify the result using the machine code (hexadecimal numbers) in the memory. Are they big-endian or little-endian? Choose one line of code and figure out its machine instruction.

How to convert mnemonics to machine instruction?
First, download the AVR Instruction Set Manual:

Note: The location of the Manual is different from the Fall, 2022 semester. The video clip is not updated since there is no time allocated for that. The text for the Fall, 2022 is kept on page 14 in order to help you understand how to download files when you watch the video. You just need to find instruction below for this semester.

1. Go to the BrightSpace course website -> Content -> LAB Material, find the file named AVR_Instruction_Set.pdf
2. Right click on the file name, from the pull down menu, click on "*Save link as…*", then follow the instructions on the next dialogue box and save it to your H: drive. We are going to use the Manual for the rest of the semester.



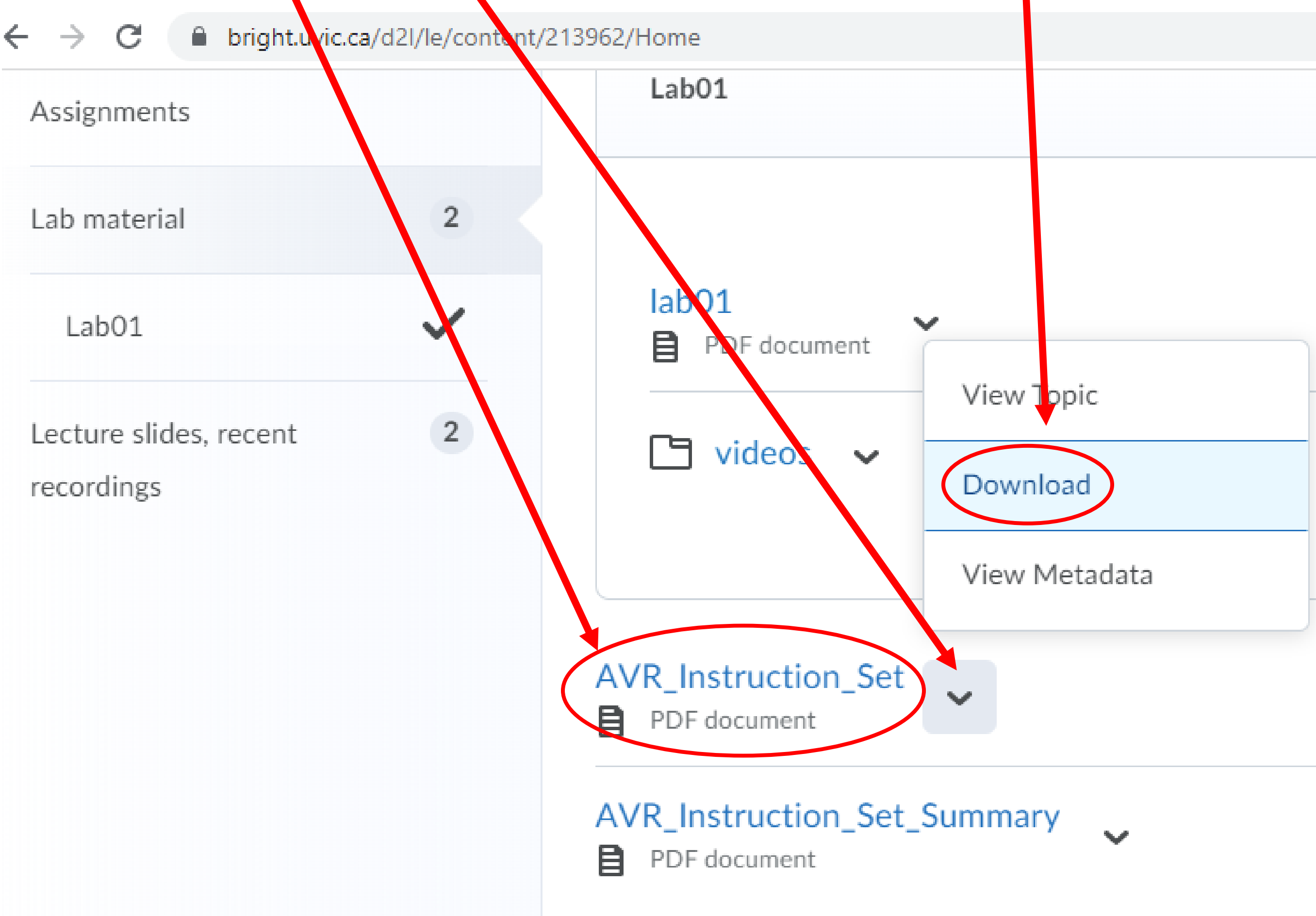
The examples on page 16 show you how mnemonics are converted to opcode:
On page 94 of the AVR Instruction Set Manual:

You may skip the next page (page 15) and move on to page 16.

(This page is specific for the Fall, 2022 semester, it is kept for the video clip since there is no time allocated to updated it.)

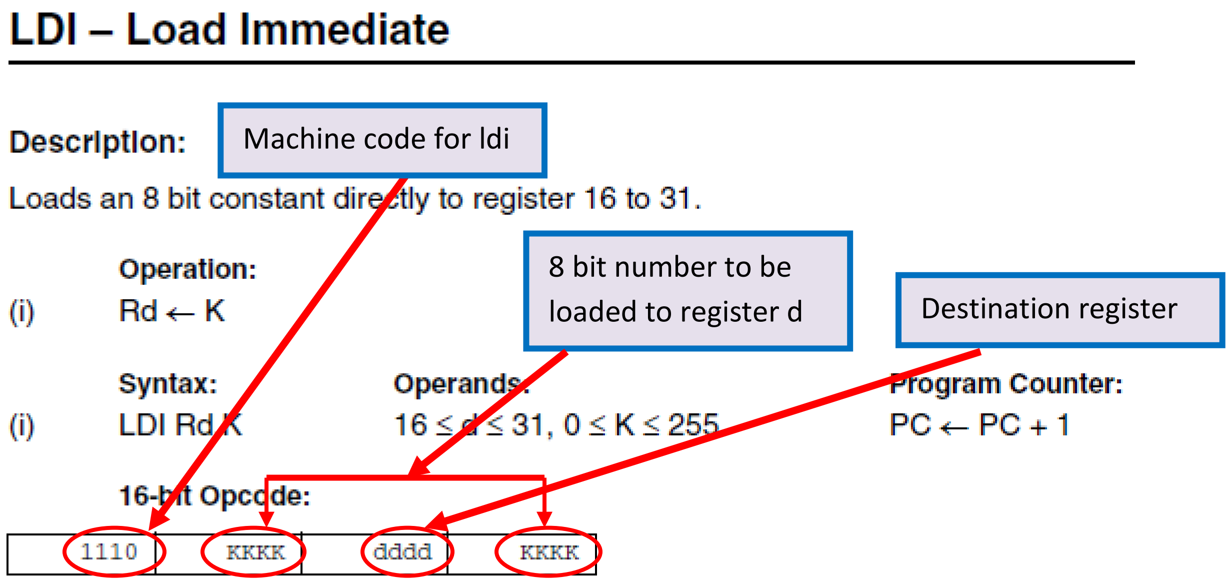
First, download the AVR Instruction Set Manual:

3. go to the BrightSpace course website -> Content -> Lab material, find the AVR_Instruction_Set.pdf file.
4. Click on the down arrow ∨ then from the pull down menu, click on *Download*. We are going to use the manual for the rest of the semester.



The following example shows you how mnemonics are converted to opcode:

On page 94 of the AVR Instruction Set Manual:



LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

Syntax: (i) LDI Rd,K

Operands: $16 \le d \le 31$, $0 \le K \le 255$

Program Counter: PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

So the opcode (operation code) of ldi Rd, K is **1110 KKKK dddd KKKK**

For example:

Mnemonics: ldi r16, 0x0A ;load 10 (hexadecimal A) to register 16

In this example, the number to be loaded to R16 is 0x0A. 8 bits are used to represent the number in the opcode.

K = 0x0A, convert 0x0A to binary 0b**00001010**

d = 16, in binary, it is 1**0000** (Is this *ldi r0, 4* correct and why? It is not correct because $16 \le d \le 31$)

The 16-bit Opcode of ldi is: **1110 KKKK dddd KKKK**

Step 1: The **opcode** for command ldi is: **1110**

Step 2, add the high nibble of K, which is **0000, we have 11100000**

Step 3, add the low nibble of K, which is **1010, we have 11100000 1010**

Step 4, add the last four bits of destination register (why? Because there are only 4 bits reserved for register numbers in opcode.), which is **0000, we have 1110000000001010**

Step 5, convert the 16 bit machine code to hexadecimal: 0xE00A

But in the memory, you see 0x0A 0xE0, why? Because it is little endian.

In summary, the machine code for ldi r16, 0x0A is:

machine instruction in binary form: 0b**1110000000001010**

machine instruction in hexadecimal form: 0xE00A (big endian)

machine instruction in hexadecimal form in AVR memory: 0x0AE0 (little endian)

Lab exercise:

Convert machine code of the following mnemonics:

Andi r16, 0b00000001 (verify it by checking the values in memory)
ldi r17, -5 ;hint, two's complement of -5
lsl r18 ;LSL (logical shift left)
rol r19 ; ROL (rotate left)
lsr r0 ;LSR (logical shift right)
lsr r19
ror r18 ; ROR (rotate right)
ori r16, 1 ; ORI , set bit 0 of reister 16
eor r1, r17 ;EOR (exclusive OR)

To figure out the meaning of the instructions and how to use them, read the AVR Instruction Set Manual. Verify their behavior using values store in registers. It is very important to do the exercises. You may find them helpful for the assignment.