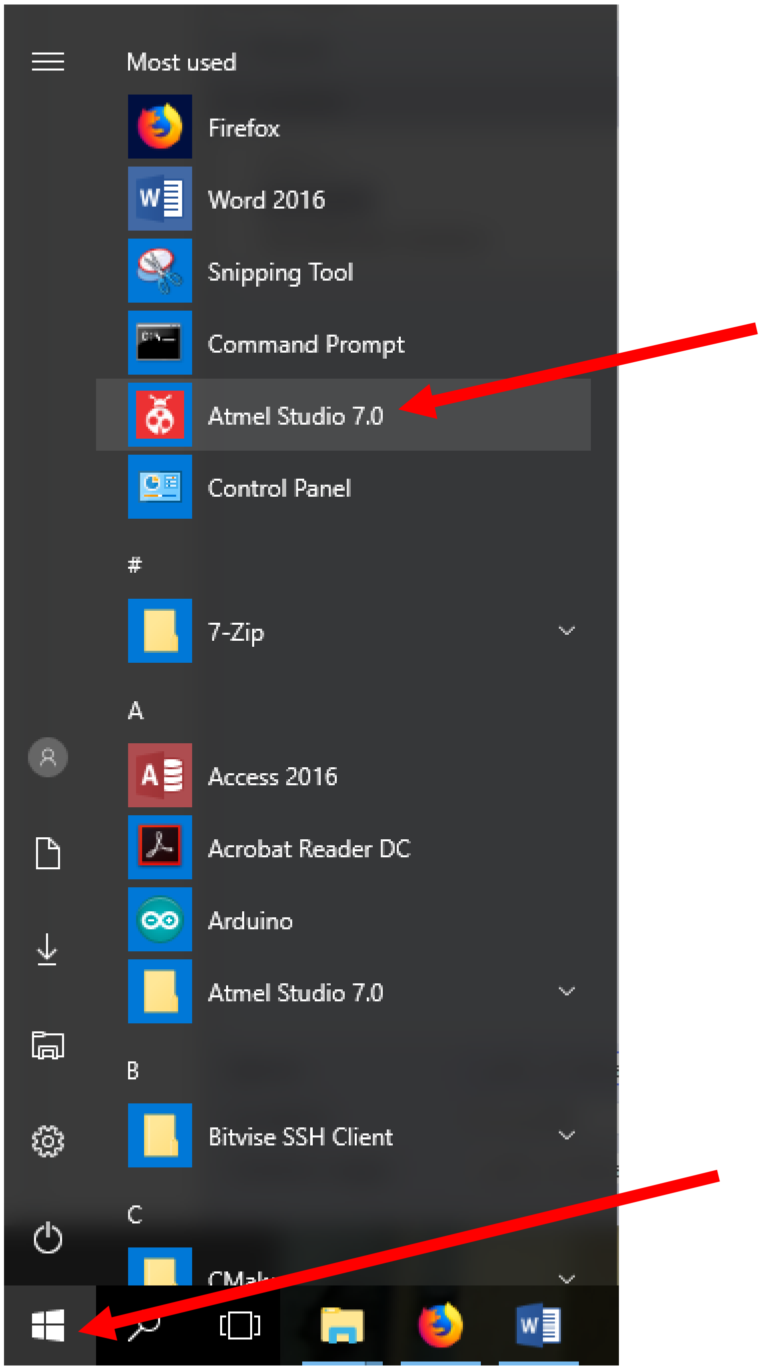
Lab 2 Introduction to Assembly Language

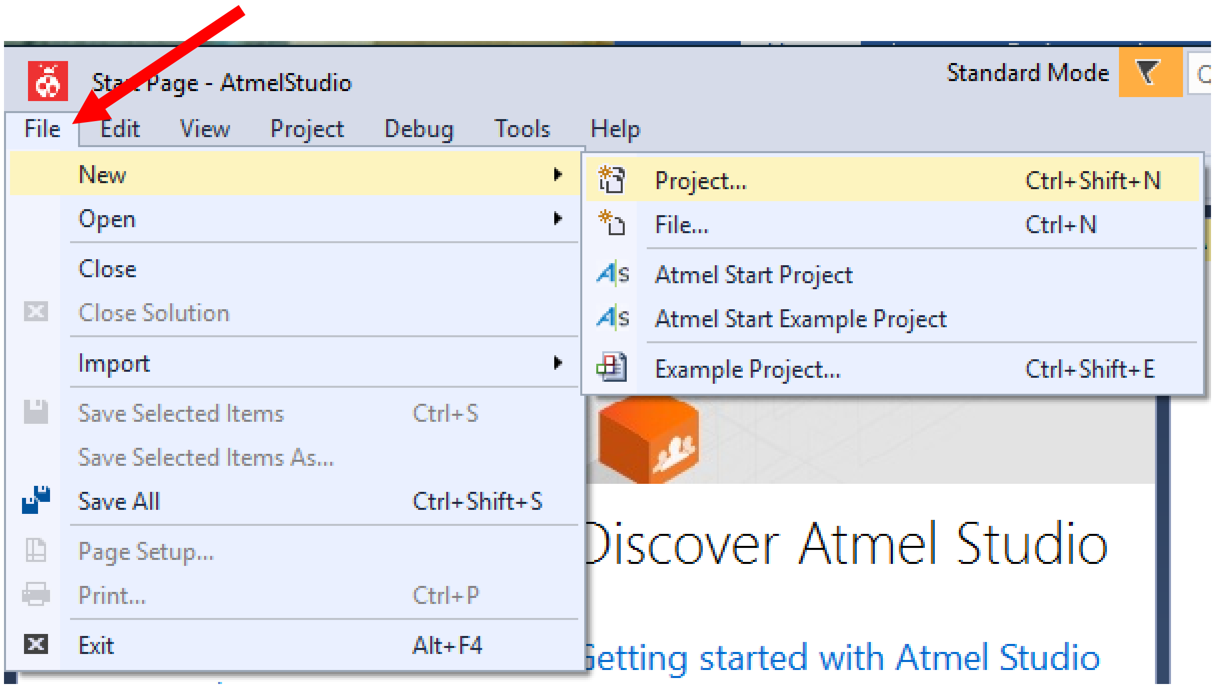
Submit main.asm.

I. Introduction to Atmel Studio 7.0 [1]

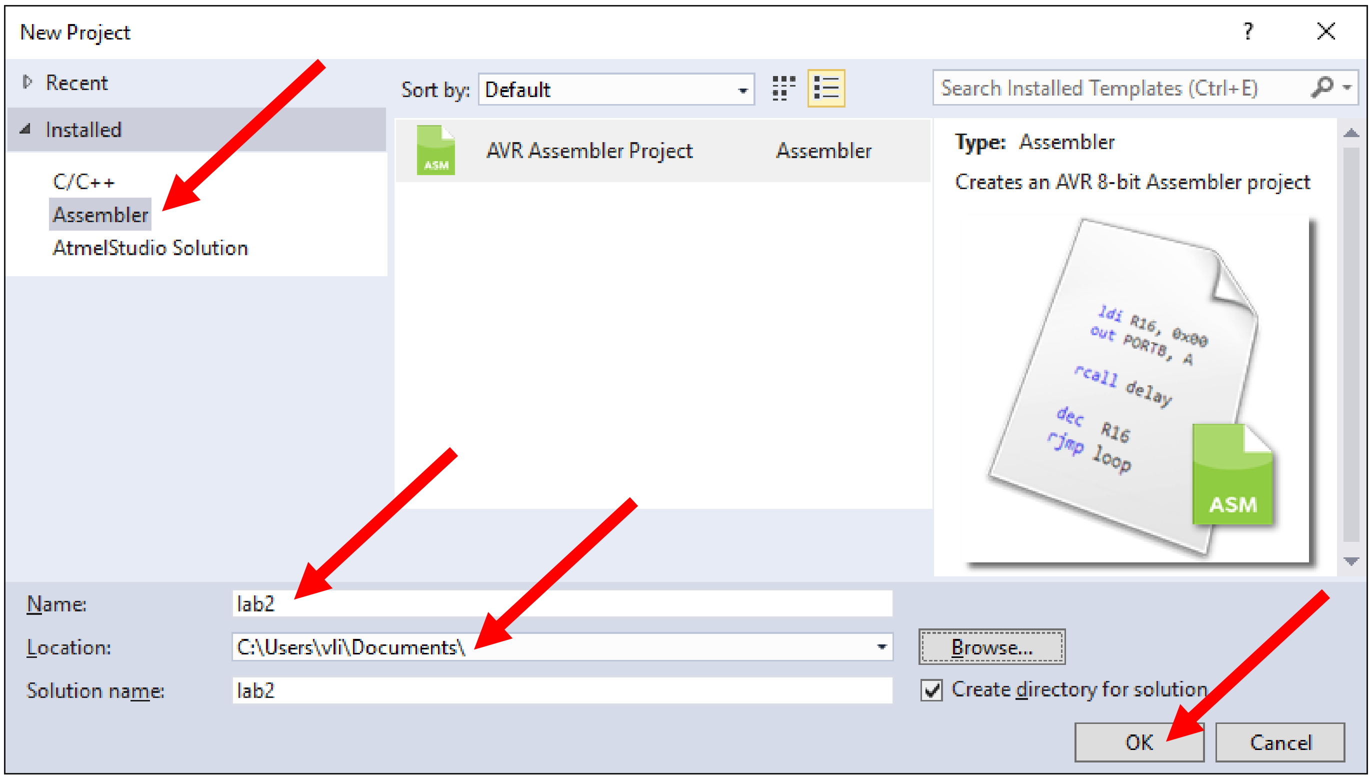
Launch *Atmel Studio 7.0* and create a new project named "lab2": click on the *Start* button, then click on *Atmel Studio 7.0:*



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

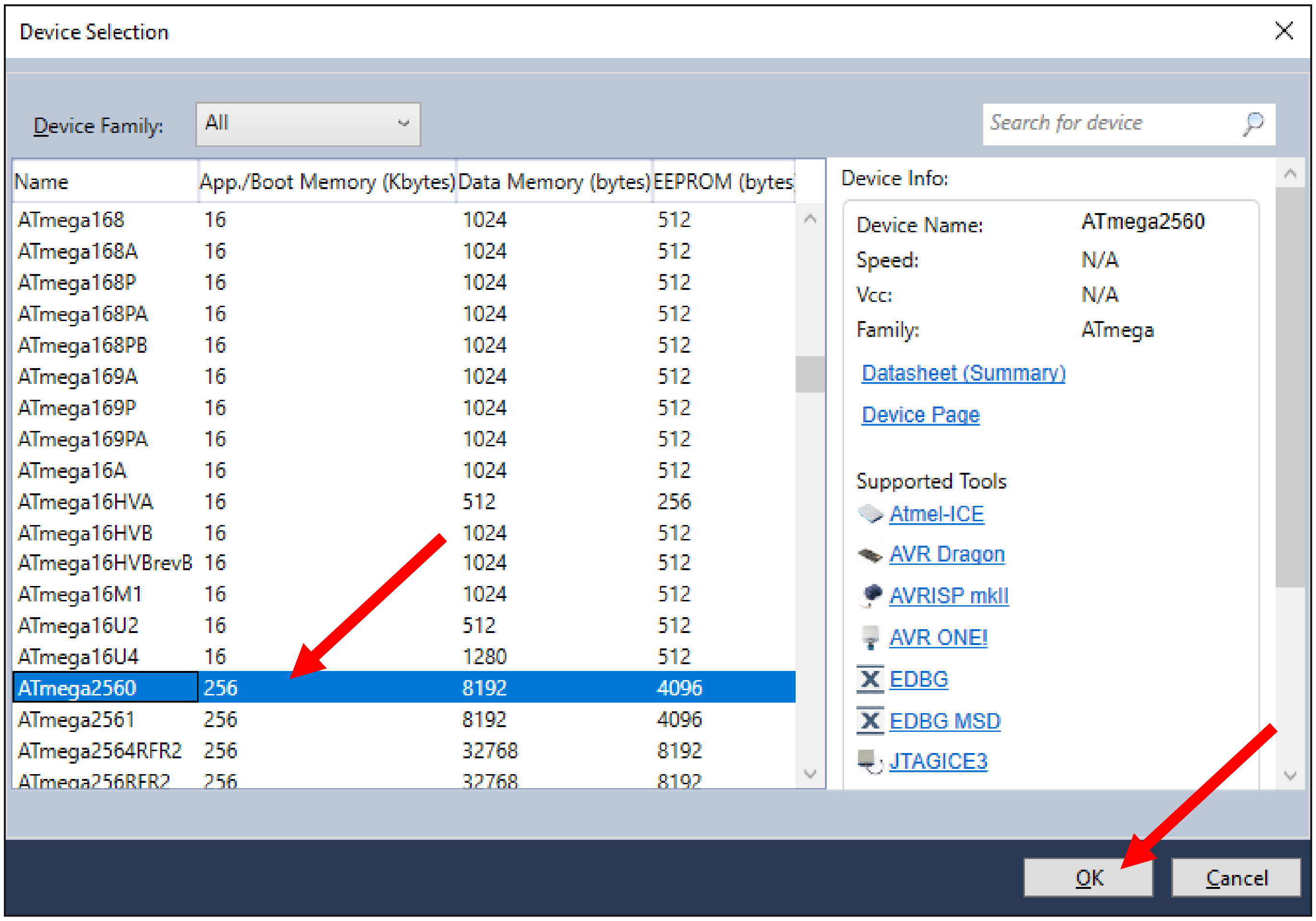


In the new dialog box, on the left pane, under *Installed*, select *Assembler*. Type the project Name *lab2* and select the Location. Click on the *OK* button.



Name: lab2
Location: C:\Users\vli\Documents\
Solution name: lab2

Then a new dialog box appears, choose *ATmega2560* for the *Device Family*. Click on the *OK* button.



Device Selection

Device Family: All

Search for device

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes
ATmega168	16	1024	512
ATmega168A	16	1024	512
ATmega168P	16	1024	512
ATmega168PA	16	1024	512
ATmega168PB	16	1024	512
ATmega169A	16	1024	512
ATmega169P	16	1024	512
ATmega169PA	16	1024	512
ATmega16A	16	1024	512
ATmega16HVA	16	512	256
ATmega16HVB	16	1024	512
ATmega16HVBrevB	16	1024	512
ATmega16M1	16	1024	512
ATmega16U2	16	512	512
ATmega16U4	16	1280	512
ATmega2560	256	8192	4096
ATmega2561	256	8192	4096
ATmega2564RFR2	256	32768	8192
ATmega256RFR2	256	32768	8192

Device Info:

Device Name:	ATmega2560
Speed:	N/A
Vcc:	N/A
Family:	ATmega

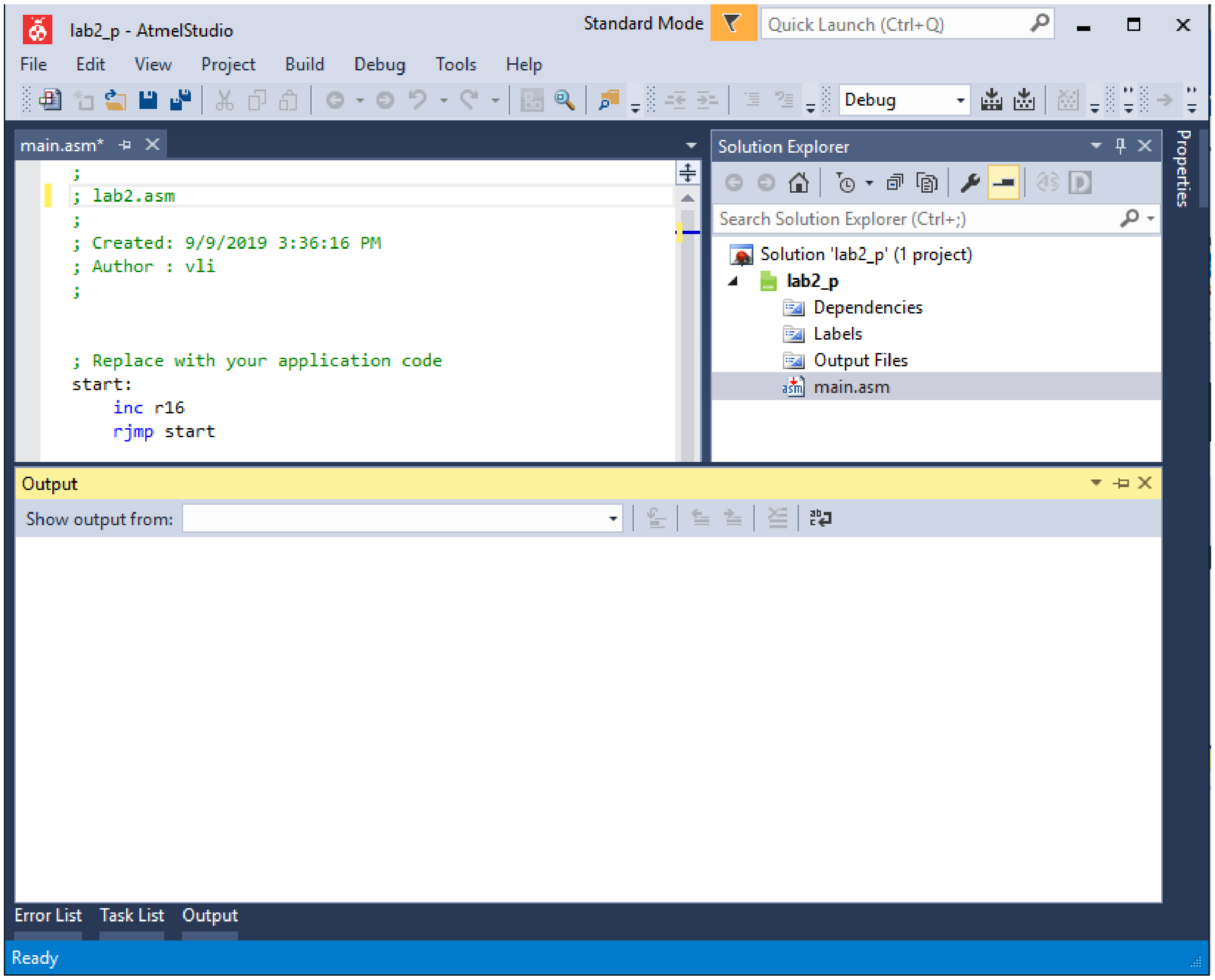
Datasheet (Summary)

Device Page

Supported Tools

- Atmel-ICE
- AVR Dragon
- AVRISP mkII
- AVR ONE!
- EDBG
- EDBG MSD
- JTAGICE3

The new project looks like this:



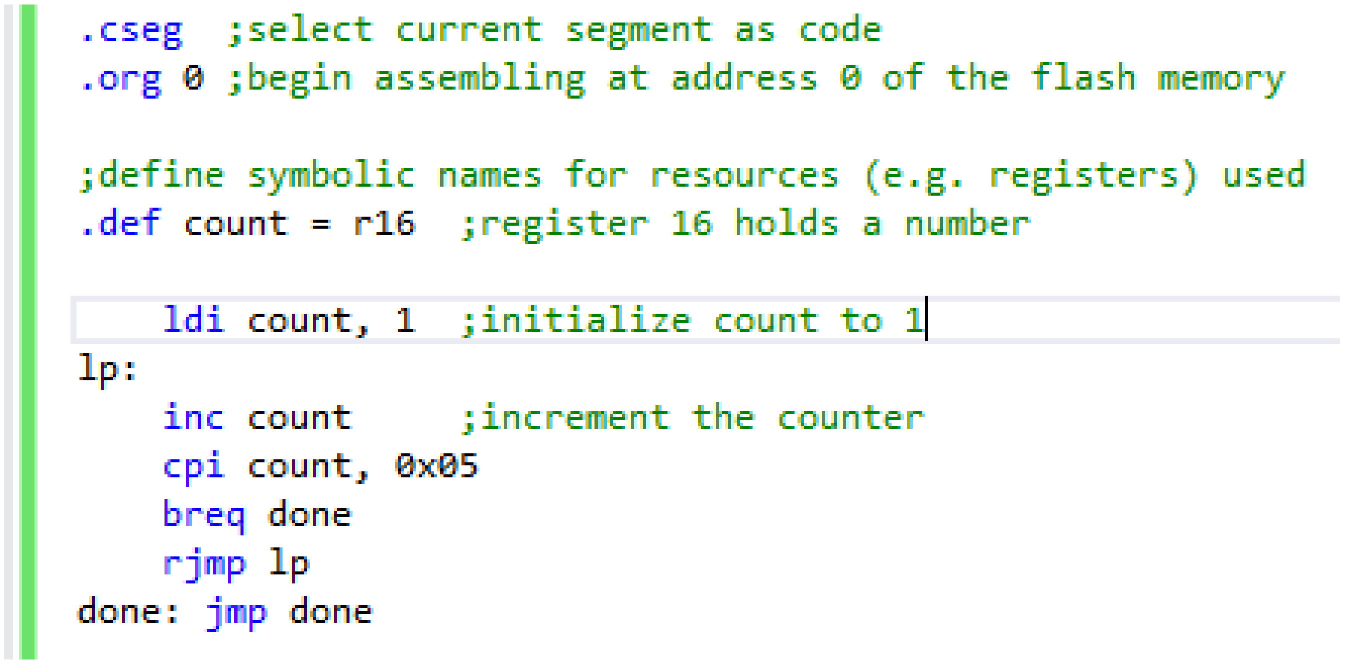
```
;
; lab2.asm
;
; Created: 9/9/2019 3:36:16 PM
; Author : vli
;


; Replace with your application code
start:
    inc r16
    rjmp start
```

```
Solution 'lab2_p' (1 project)
  lab2_p
    Dependencies
    Labels
    Output Files
    main.asm
```

Remove "`start:`

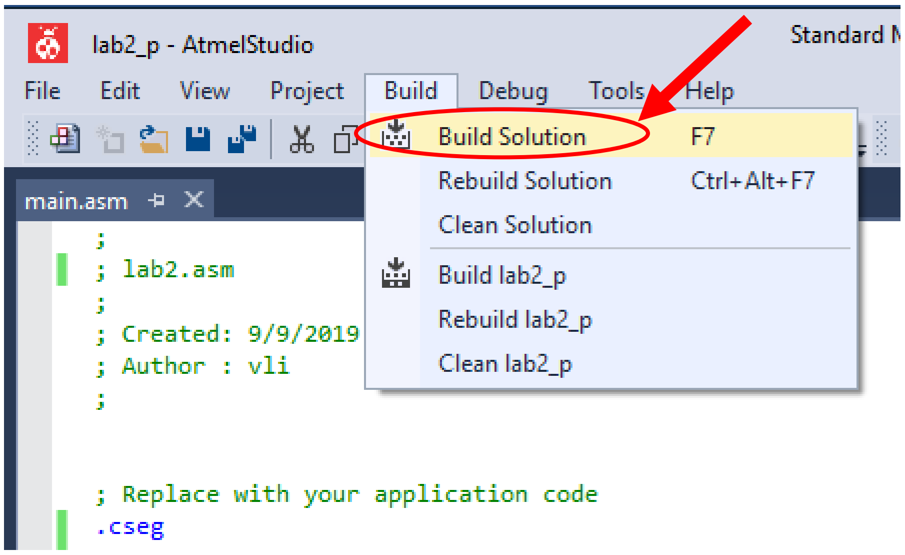
```
        inc r16
        rjmp start
```
"

Type the following code:



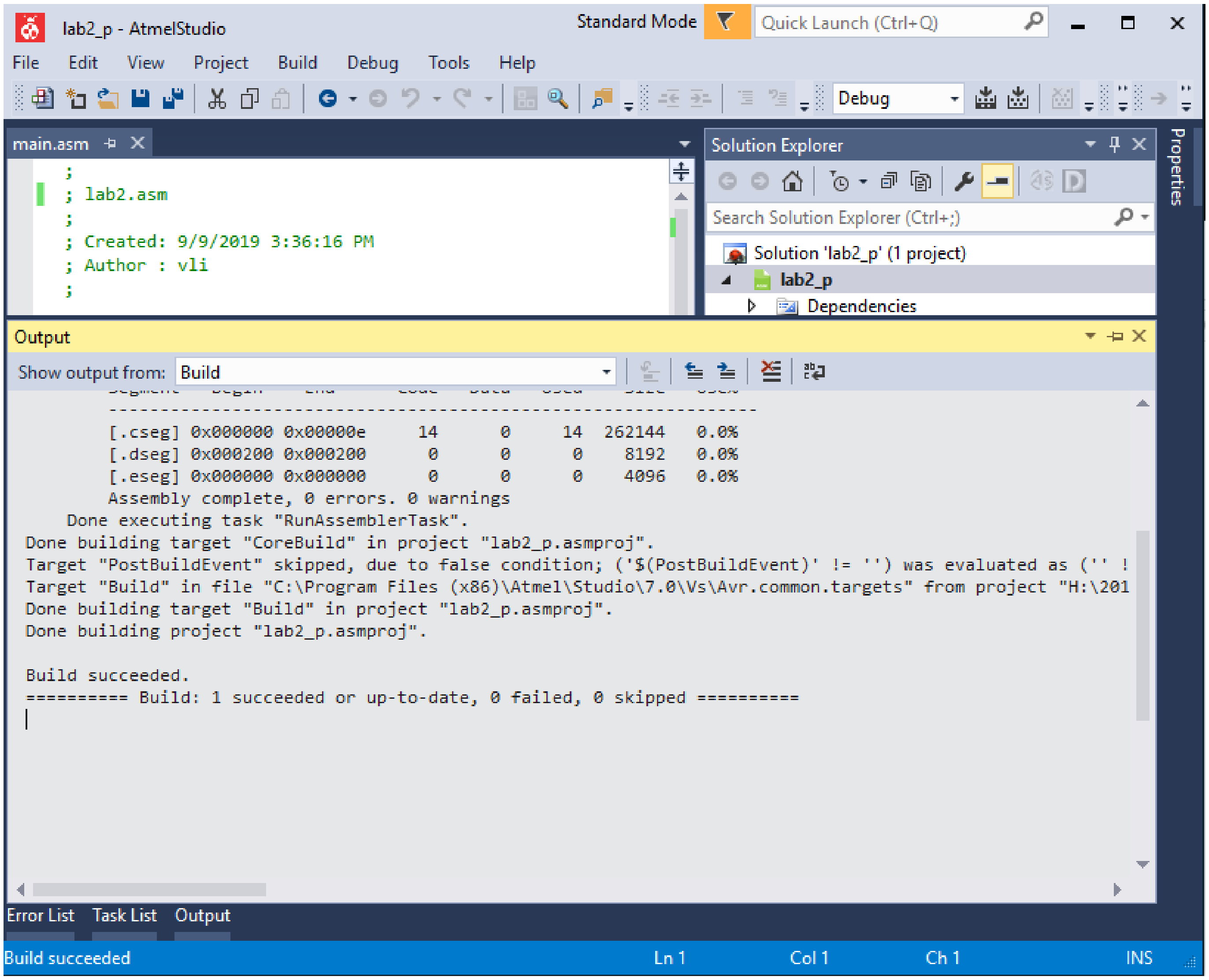
```
.cseg  ;select current segment as code
.org 0 ;begin assembling at address 0 of the flash memory

;define symbolic names for resources (e.g. registers) used
.def count = r16  ;register 16 holds a number

    ldi count, 1  ;initialize count to 1
lp:
    inc count     ;increment the counter
    cpi count, 0x05
    breq done
    rjmp lp
done: jmp done
```

Save the code and build the program: on the menu, click on *Build* -> *Build Solution*:



```
;
; lab2.asm
;
; Created: 9/9/2019
; Author : vli
;


; Replace with your application code
.cseg
```

The screen looks like this:

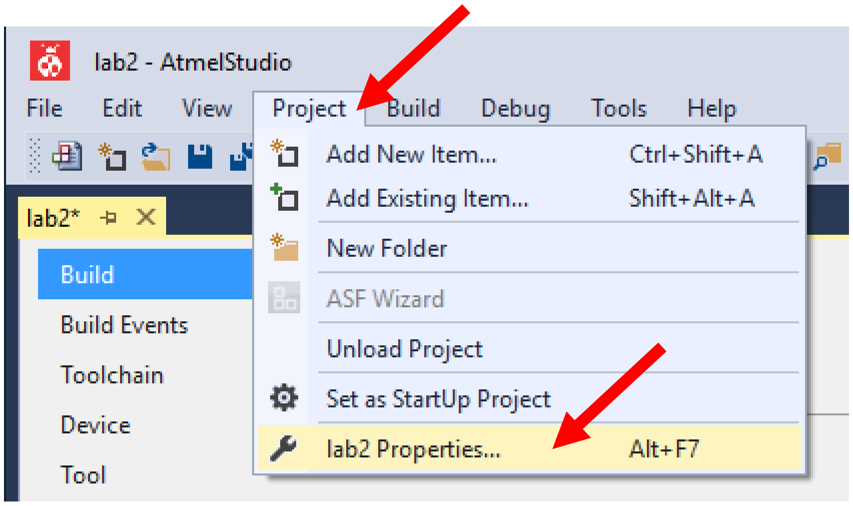

lab2_p - AtmelStudio
Standard Mode
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Tools Help
Debug
main.asm
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'lab2_p' (1 project)
lab2_p
Dependencies
Output
Show output from: Build
Error List Task List Output
Build succeeded Ln 1 Col 1 Ch 1 INS
Properties

```
;
; lab2.asm
;
; Created: 9/9/2019 3:36:16 PM
; Author : vli
;
```

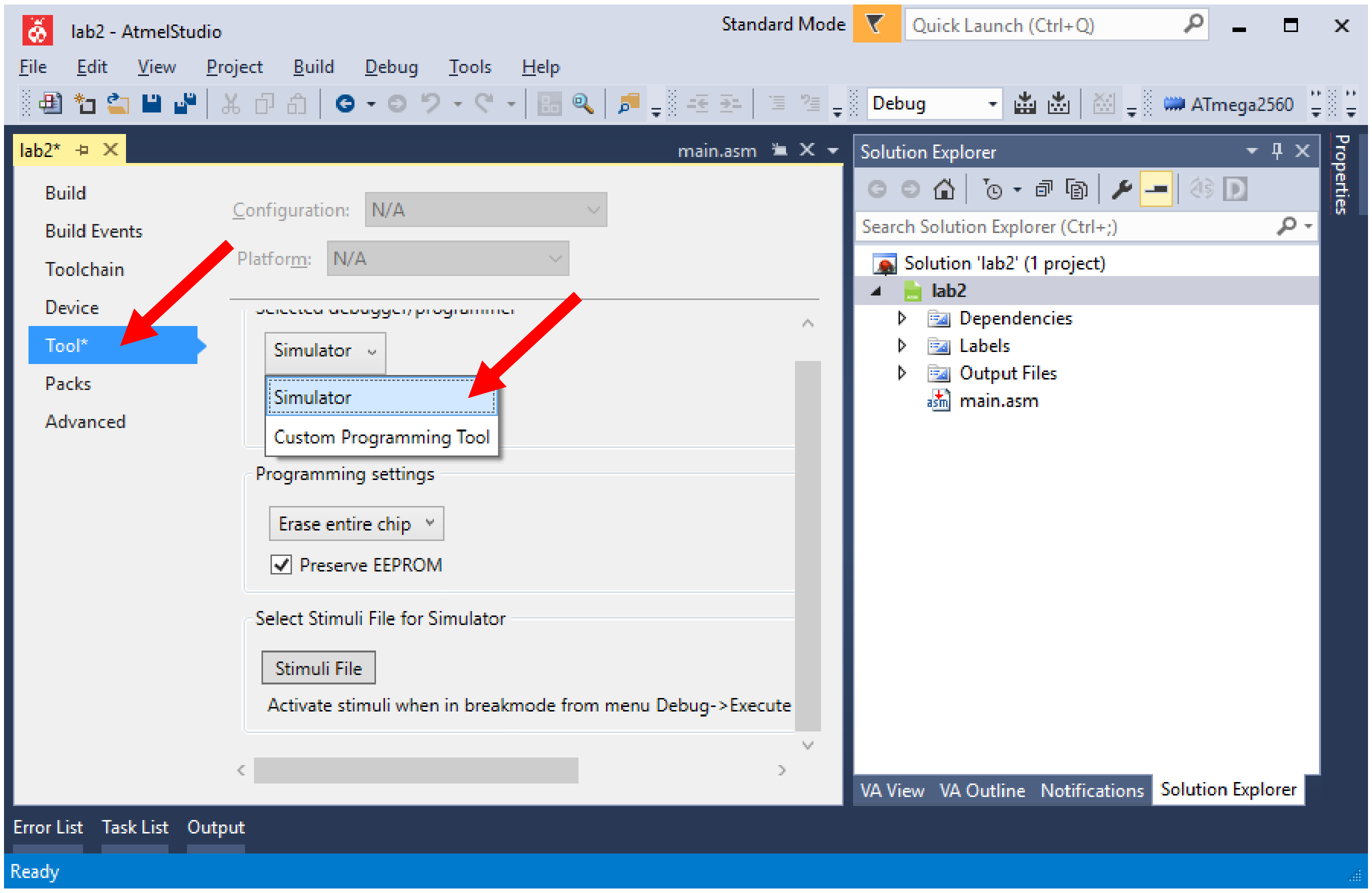
```
        -------------------------------------------------------------
        [.cseg] 0x000000 0x00000e     14      0     14  262144   0.0%
        [.dseg] 0x000200 0x000200      0      0      0    8192   0.0%
        [.eseg] 0x000000 0x000000      0      0      0    4096   0.0%
        Assembly complete, 0 errors. 0 warnings
    Done executing task "RunAssemblerTask".
Done building target "CoreBuild" in project "lab2_p.asmproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' !
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "H:\201
Done building target "Build" in project "lab2_p.asmproj".
Done building project "lab2_p.asmproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

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

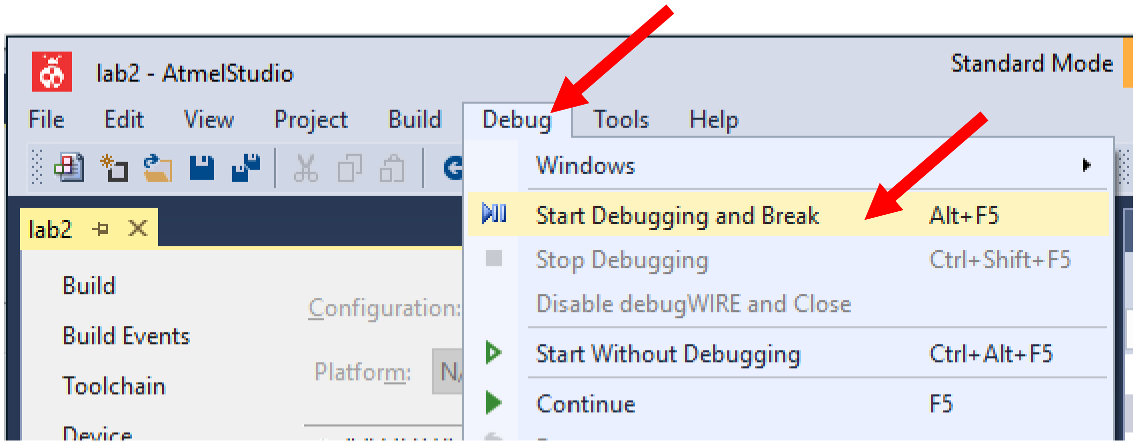
Now, you are ready to run your program using the simulator. Set up the configuration of the simulator: from the *Project* menu -> *lab2 Properties…*:


lab2 - AtmelStudio
File Edit View Project Build Debug Tools Help
lab2*
Build
Build Events
Toolchain
Device
Tool
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
New Folder
ASF Wizard
Unload Project
Set as StartUp Project
lab2 Properties... Alt+F7

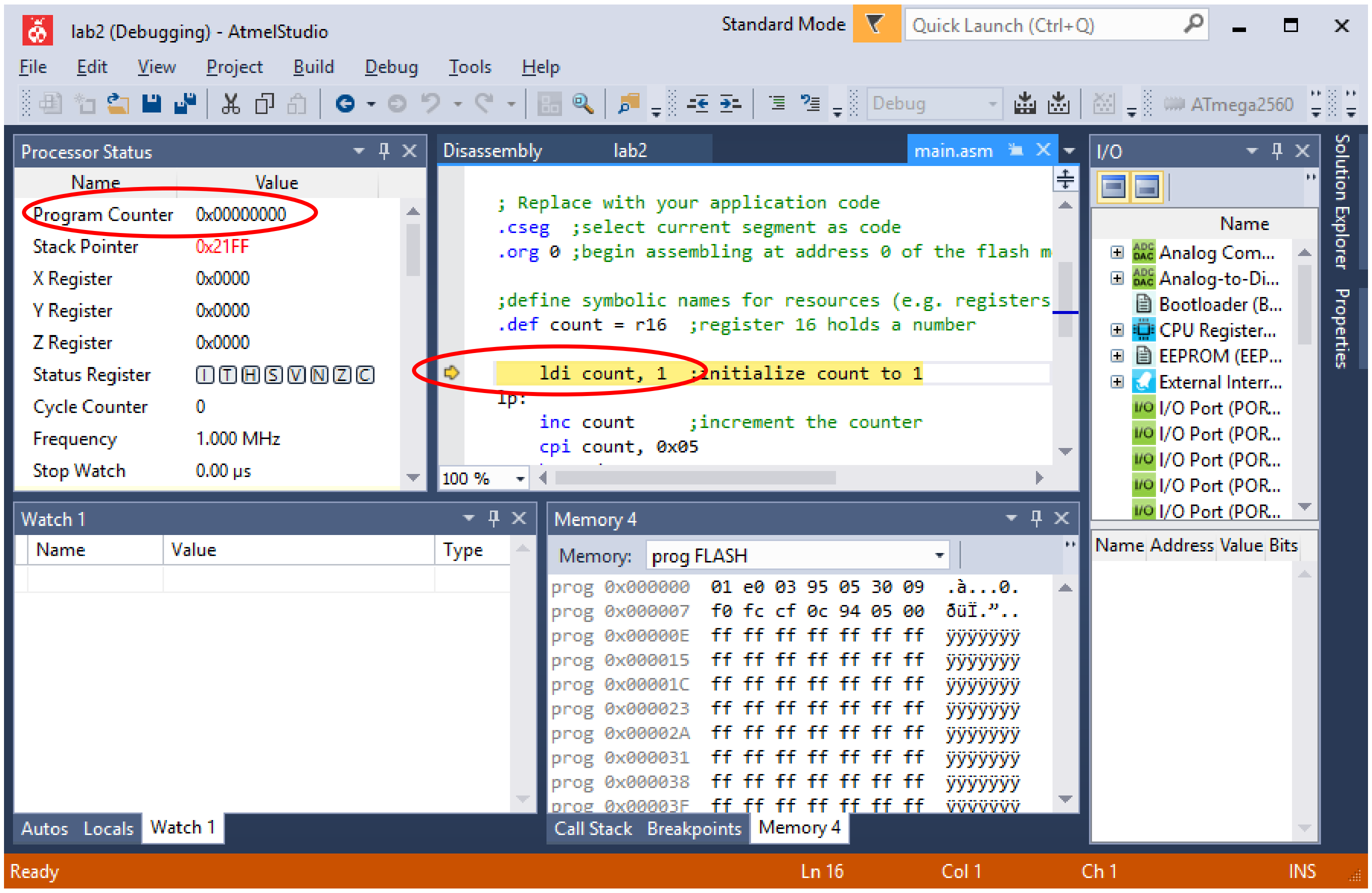
In the settings window, click on Tool and select Simulator:



Save the project by click on “File” menu -> “Save All”.
Start the simulator: from the menu, click on *Debug -> Start Debugging and Break*


lab2 - AtmelStudio
File Edit View Project Build Debug Tools Help
Windows
Start Debugging and Break Alt+F5
Stop Debugging Ctrl+Shift+F5
Disable debugWIRE and Close
Start Without Debugging Ctrl+Alt+F5
Continue F5

The editor should show a yellow arrow indicating the instruction about to be fetched. The left panel shows the "*Processor Status*", where you can examine the register and processor values. The *"Program Counter"* shows the memory address of the next instruction to be fetched.



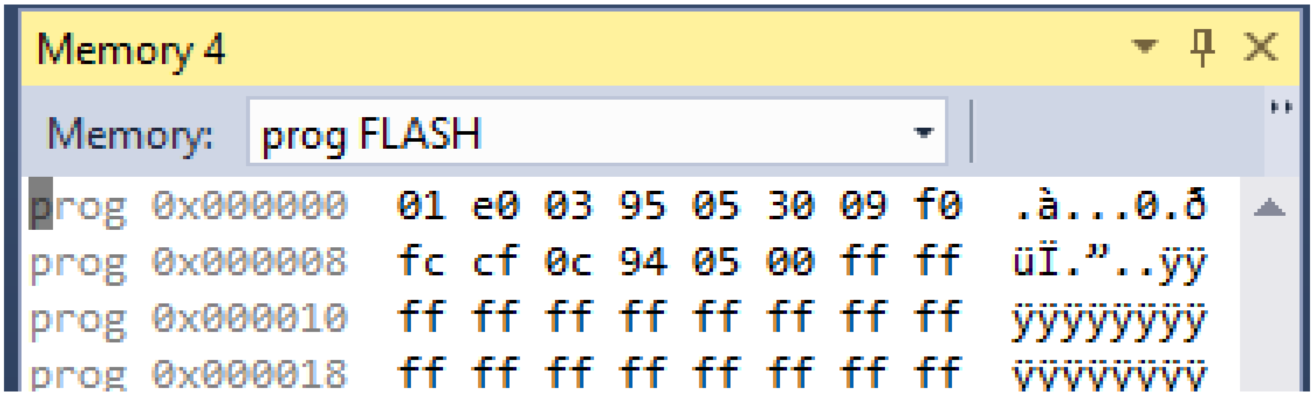
Name	Value
Program Counter	0x00000000
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	0
Frequency	1.000 MHz
Stop Watch	0.00 µs

```
; Replace with your application code
.cseg  ;select current segment as code
.org 0 ;begin assembling at address 0 of the flash m

;define symbolic names for resources (e.g. registers
.def count = r16  ;register 16 holds a number

    ldi count, 1  ;initialize count to 1
lp:
    inc count     ;increment the counter
    cpi count, 0x05
```

Name	Value	Type

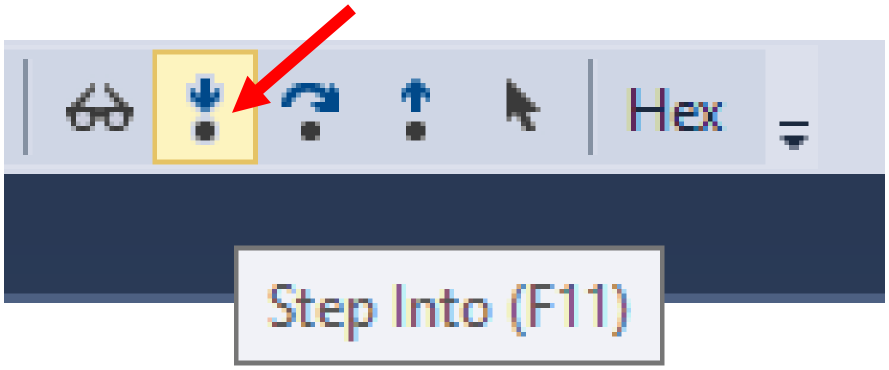
```
Memory: prog FLASH
prog 0x000000  01 e0 03 95 05 30 09  .à...0.
prog 0x000007  f0 fc cf 0c 94 05 00  ðüÏ.”..
prog 0x00000E  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x000015  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x00001C  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x000023  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x00002A  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x000031  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x000038  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
prog 0x00003F  ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿ
```

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

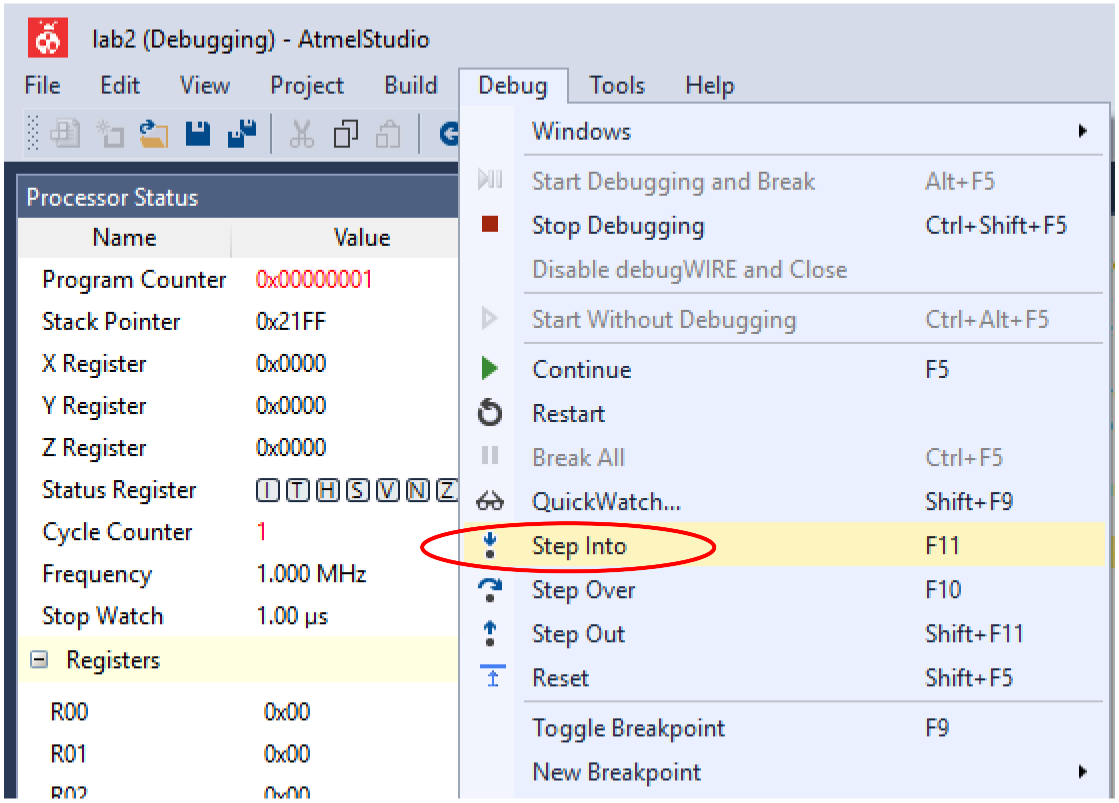


```
Memory: prog FLASH
prog 0x000000  01 e0 03 95 05 30 09 f0  .à...0.ð
prog 0x000008  fc cf 0c 94 05 00 ff ff  üÏ.”..ÿÿ
prog 0x000010  ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿ
prog 0x000018  ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿ
```

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



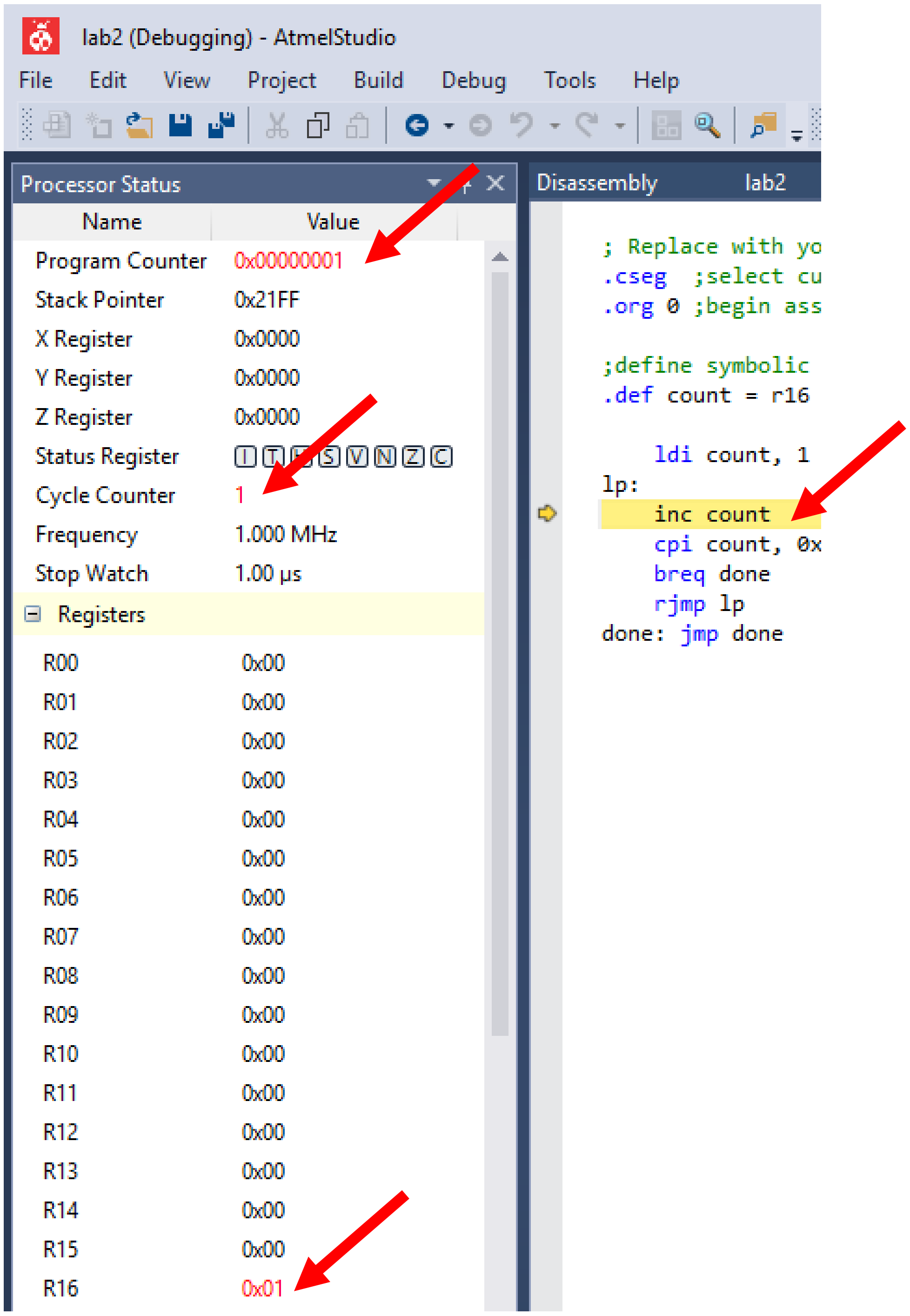
Or go to the menu, click on Debug ->Step Into:



Processor Status

Name	Value
Program Counter	0x00000001
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z
Cycle Counter	1
Frequency	1.000 MHz
Stop Watch	1.00 µs
Registers	
R00	0x00
R01	0x00

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


lab2 (Debugging) - AtmelStudio
File Edit View Project Build Debug Tools Help
Processor Status
Disassembly
lab2

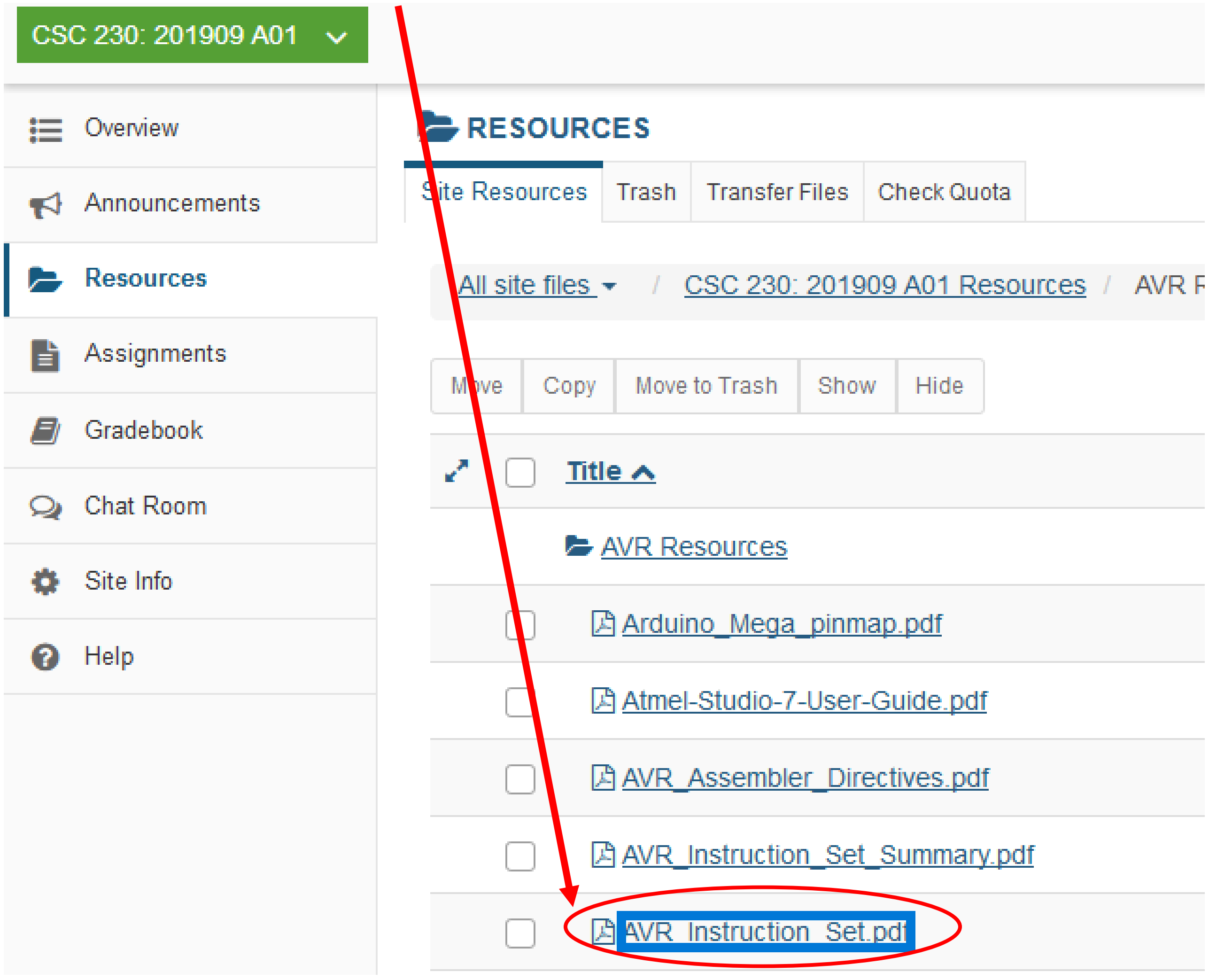
Name	Value
Program Counter	0x00000001
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I T H S V N Z C
Cycle Counter	1
Frequency	1.000 MHz
Stop Watch	1.00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x01

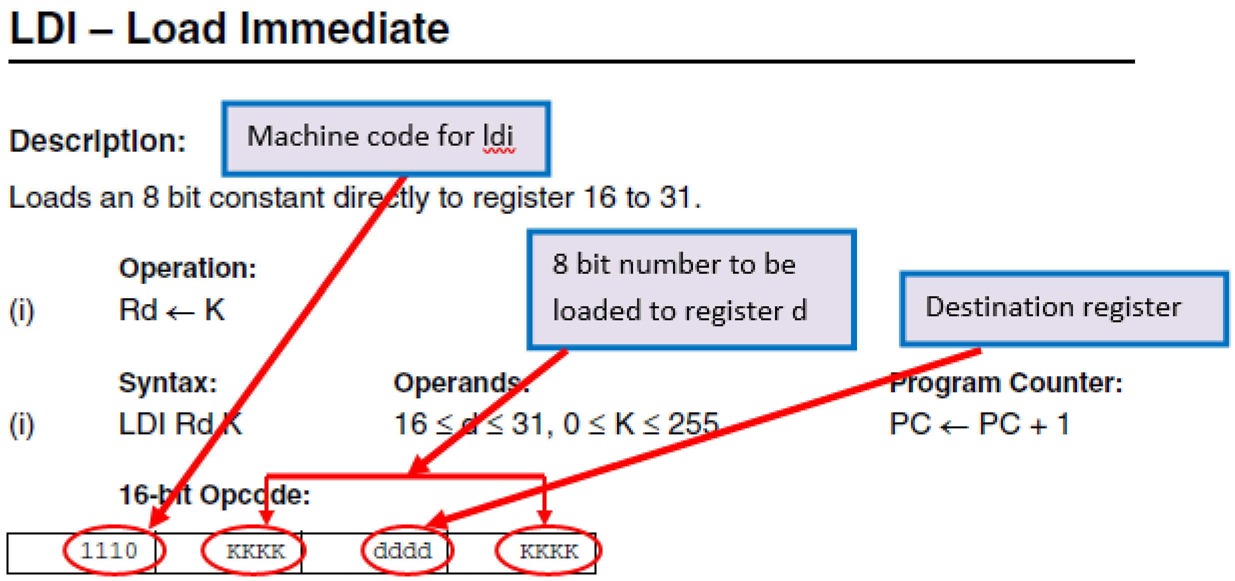
```
; Replace with yo
.cseg  ;select cu
.org 0 ;begin ass

;define symbolic
.def count = r16

    ldi count, 1
lp:
    inc count
    cpi count, 0x
    breq done
    rjmp lp
done: jmp done
```

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

II. Write some assembly language code (mnemonics) and convert it to machine instruction. 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?
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:	**Operands:**	**Program Counter:**
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Machine code for ldi
8 bit number to be loaded to register d
Destination register

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

For example:
Mnemonics: ldi r16, 0x82 ;load 0x82 to register 16
In this example, K = 0x82, in binary, it is 0b**10000010**
d = 16, in binary, it is 1**0000** (Is this "ldi r0, 4" correct? No, because 16 ≤ d ≤ 31. That means only R16 to R31 can be used in the *ldi* instruction.)

The 16-bit Opcode of ldi is: **1110 KKKK dddd KKKK**
Step 1: The **opcode** for ldi is: **1110????????????**
Step 2, fill in the high nibble of K, which is **1000, we have 11101000????????**
Step 3, fill in the low nibble of K, which is **0010, we have 11101000????0010**
Step 4, fill in the last four bits of destination register (why? Because only 4 bits are reserved for registers.), which is **0000, we have 1110100000000010**
Step 5, convert the 16 bit machine code to hexadecimal: 0xE802

But in the memory, you see 0x02E8, why? Because it is little endian. The low byte is written before the high byte.

In summary, the machine code for ldi r16, 0x82 is:
machine instruction in binary form: 0b**1110100000000010**
machine instruction in hexadecimal form: 0xE802
machine instruction in hexadecimal form in AVR memory (little endian): 0x02E8

Lab exercise:
Write machine code of the following mnemonics:

a. Andi r16, 0b00000001 (verify it by checking the values in memory) (page 20)
b. ldi r17, -5 ;hint, two's complement of -5

Do you get 0x0170 for "a" and 0x1BEF for "b"?

III. Comment out the program you have written so far and write a new program. The program calculates the number of students in csc999. There are two lab sections B01 and B02. The number of students registered:
B01: 23
B02: 21
The maximum enrollment for the course is 60. If the course enrollment is bigger than 60, set register 0 to 1, 0 otherwise. Store the course enrollment in register 19. Assume the total number of students is fewer than or equal to 255.

In high-level programming language, it can be done like this:

```
unsigned int x=23
unsigned int y=21
unsigned int sum=x+y
boolean over_enrollment=false
```

```
if (sum>60)
    over_enrollment=true
```

Submit main.asm.

1. Adapted from the lab notes written by Dr. Bill Bird for csc 230 in the summer of 2018.