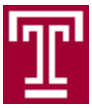


Electrical and Computer Engineering
ECE3613 Processor Systems Laboratory

Atmel Studio 7 Tutorial and AVR assembly Language

Fall 2019

In this lab, you will use an *Integrated Development Environment (IDE)* called Atmel Studio 7. The purpose of this lab is to expose you to: a) the basic operations of the Atmel Studio 7 IDE, b) basic assembly language constructs, c) simulation of a commercial microcontroller, and d) disassembling program to view the lowest level structure of microcontroller and opcodes. Finally, we will work on numbering systems and conversions.

OBJECTIVE

To practice converting data from decimal to binary and hexadecimal systems
To examine and use an AVR assembler
To examine and use an AVR simulator
To examine the flag bits of the SREG (Status Register)

REFERENCE

Mazidi and Naimi "The AVR Microcontroller and Embedded Systems," - Chapter 0.

MATERIAL

Microsoft Windows Calculator Application
AVR assembler and simulator from AVR Studio 7 Software Inc., Atmel Corporation.

MAP OF THIS LAB

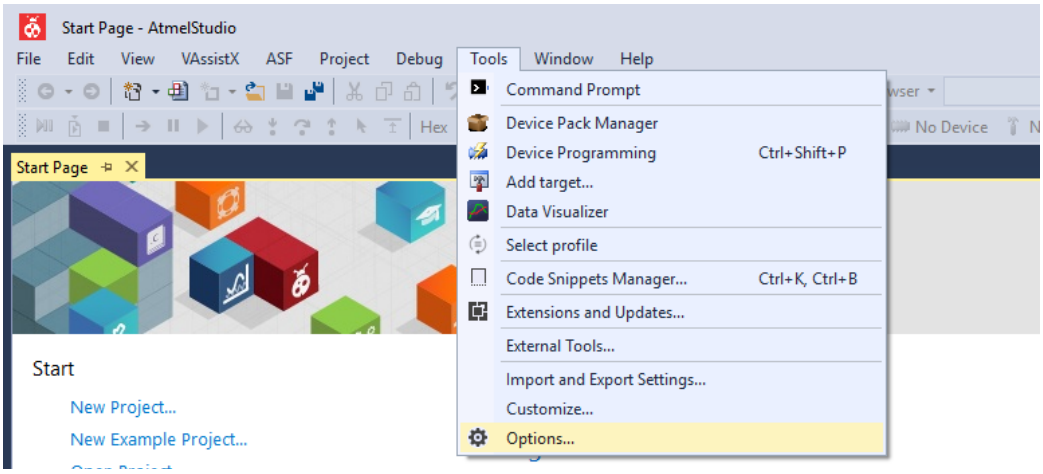
- PRE-ACTIVITY:
  1) Atmel Studio 7 tutorial
  2) Opcode

PRE-ACTIVITY

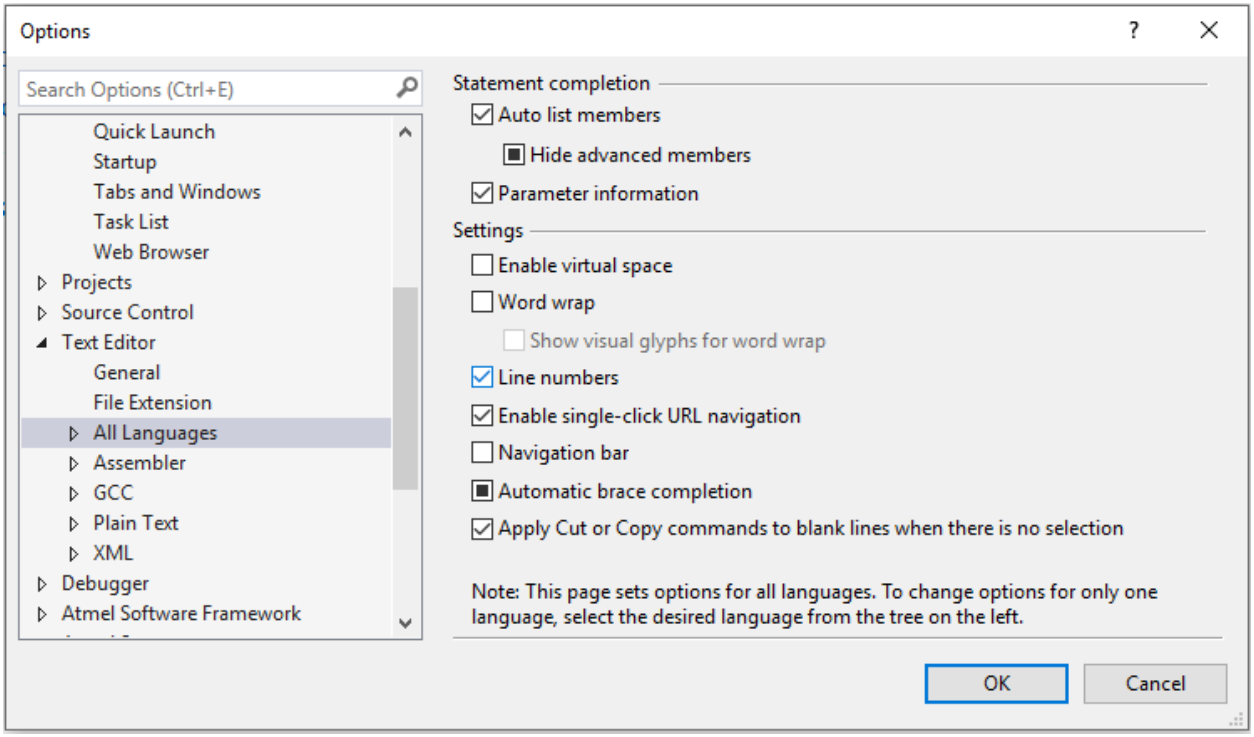
Open Atmel Studio 7.0 and follow the tutorial instructions below to get an understanding of how the Atmel IDE works and how to code, assemble and run an assembly language code. This starts on the next page.

Before we start writing codes, please enable **line number** in your text editor (if not already enabled).

- Start Atmel Studio, go to Tools → Options

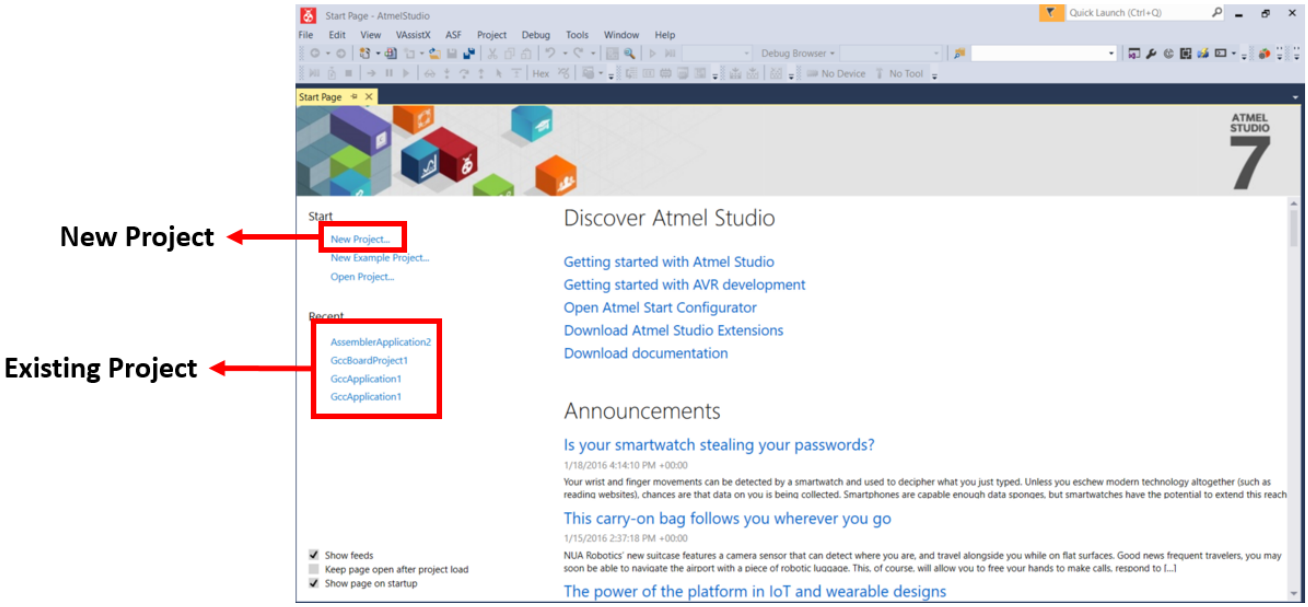


Click on the arrow beside Text Editor, select All Languages, check Line numbers box.



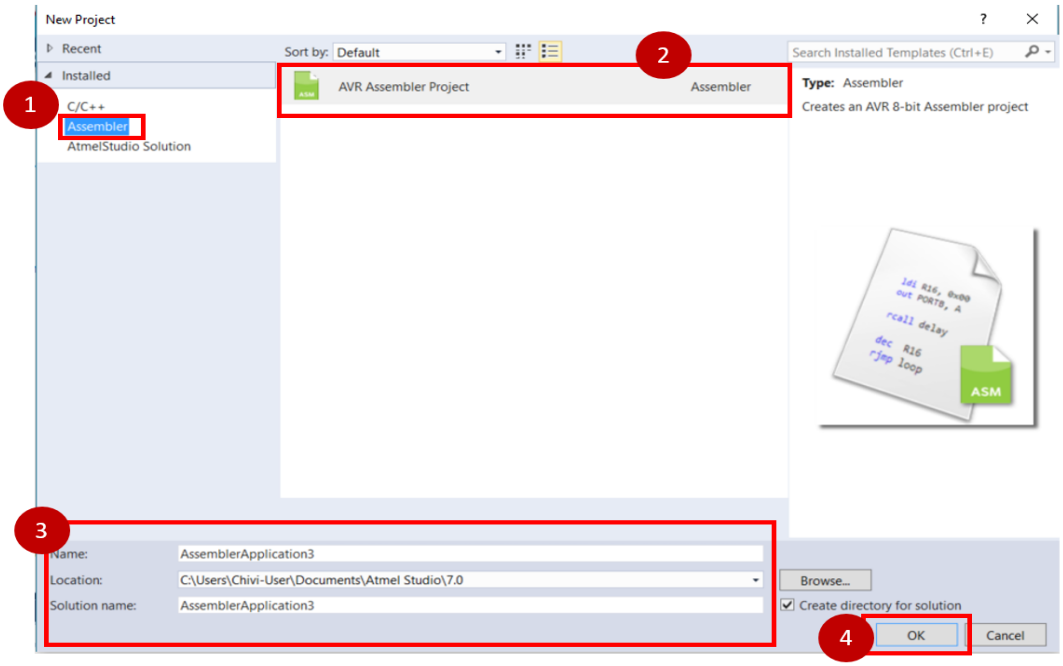
1) ATMEL 7.0 TUTORIAL

- Figure 1 shows the Studio 7 start-up page. You will have the option to either open a new project or an existing one.



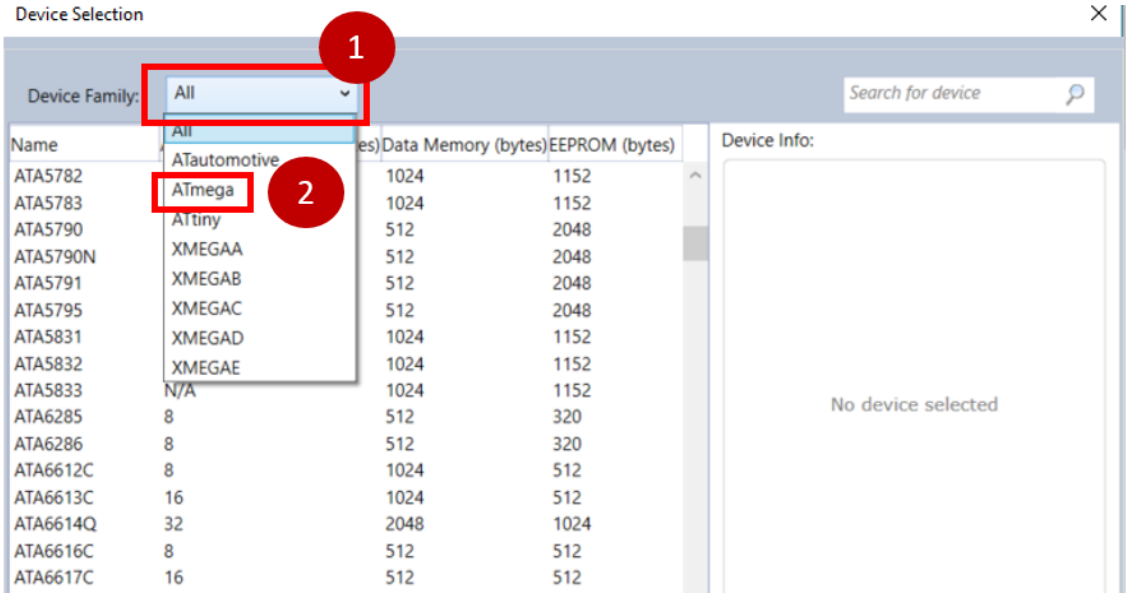
<Figure 1. Atmel Studio Start-up page>

- Pick a new project. When you start a new project, you will need to identify the characteristics of the project. For the initial exercises, we will be doing assembly coding.
  1. Select the ***Assembler*** template. [We will be working with C Executable Projects later in this course]
  2. The assembler template only has one project type: Select ***AVR Assembler Project***.
  3. Type the ***name*** of your project (or lab). You can keep the default project location.
  4. Click ***OK***.



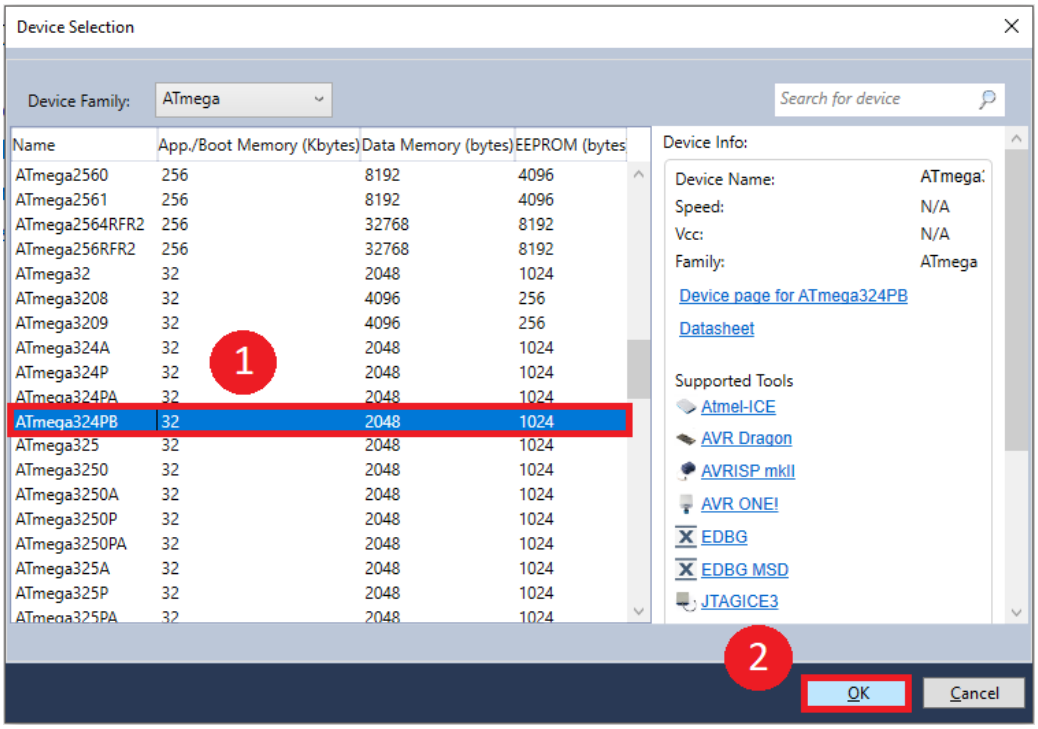
<Figure 2. Steps to identify the location of your Project>

- A ***Device Selection*** window will now appear. From the ***Device Family*** dropdown, select ***ATmega***. This will shorten the list. Or, you can type and search for ***ATmega324PB*** in the search bar. See Figure 3 for a graphical demonstration.



<Figure 3. Device Family Selection>

- You will now have to find the device that you will be utilizing.
  1. Scroll down until you find the device ***ATmega324PB***
  2. Click ***OK***.



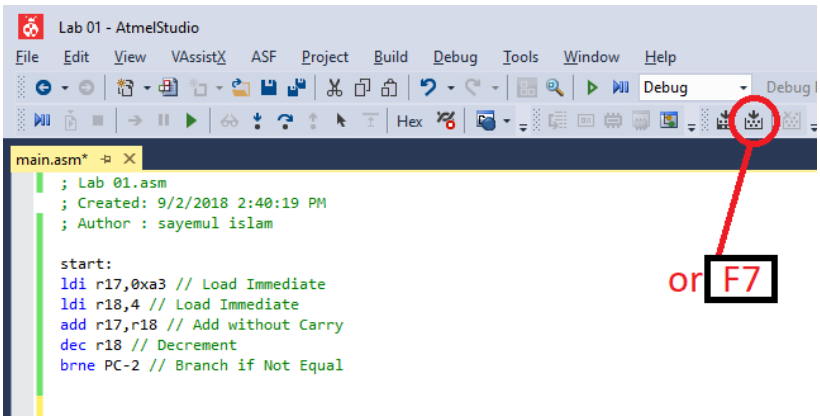
<Figure 4. Device Selection>

- At this point you have a standard Integrated Development Environment. You can enter your assembler code in the editor window. We will enter the following code below and analyze the screens and opcodes for this simple program.

- **Copy the sample program below into your text editor:**

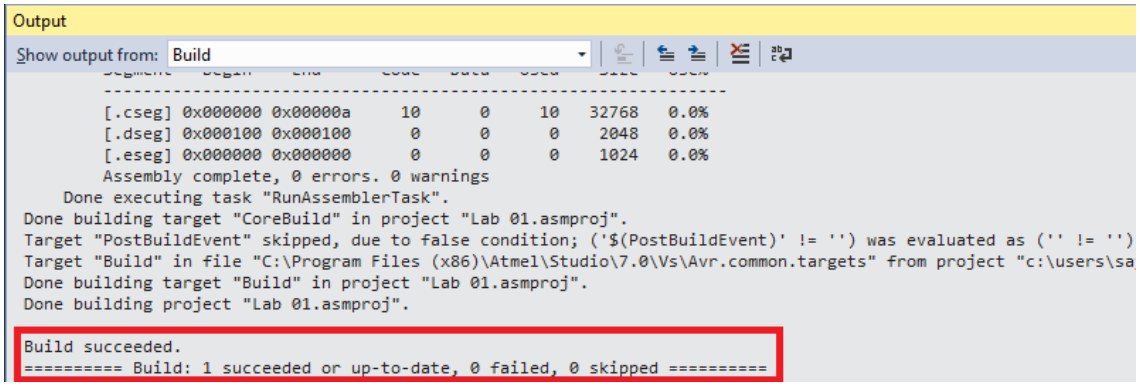
```
ldi r17, 0xa3
ldi r18, 4
add r17, r18
dec r18
brne PC-2
```

- Here is the updated window:



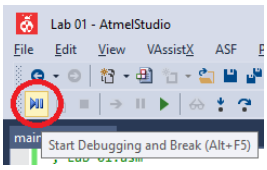
<Figure 5. New Atmel Window>

- When you are finished editing, do the following
  1. Press the ***Build Solution*** button or ***F7*** on your keyboard to build the program.
  2. Notice that your code assembled without errors in the bottom panel (Build Succeeded).



<Figure 6. Building a Solution in ATMEL Studio 7.0>

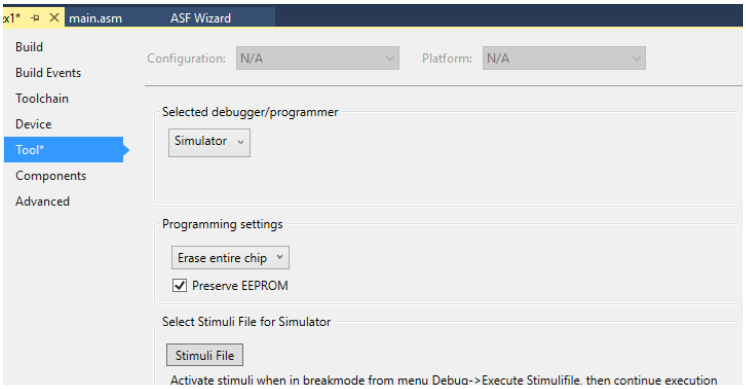
- If the build is successful, you can go to the simulation mode.
  1. Press the ***Start Debugging and Break*** (Alt-F5) button. In Studio 7.0 you will get a message to choose a simulator.



  2. The following screen will open, and you need to select **Simulator** as seen below. The first time you simulate, you will get the following window, to select which simulator

you want to use. This software also supports hardware In Circuit Emulation (ICE) tool (or EDBG) which will be discussed later in the course. If an ICE was available and connected to an actual ATmega324PB device, you could select the ICE at this time.

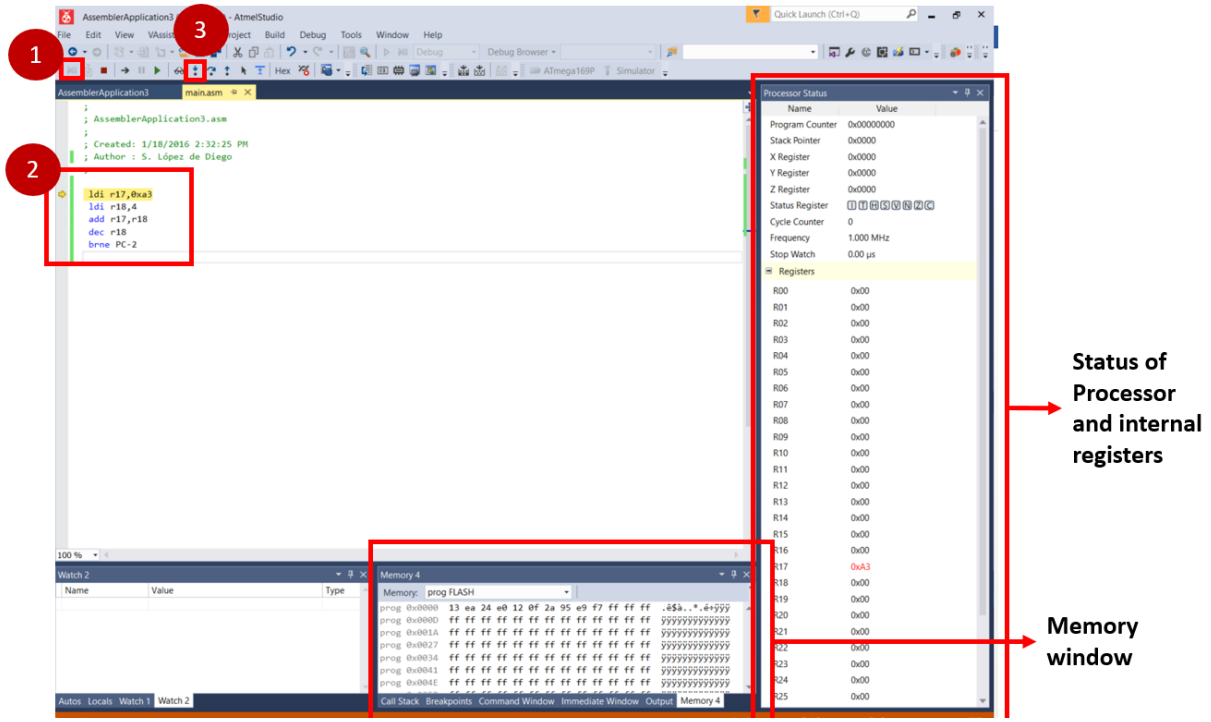
3. Click on the **main.asm** tab to go back to your main program.



<Figure 7. Tool Selection>

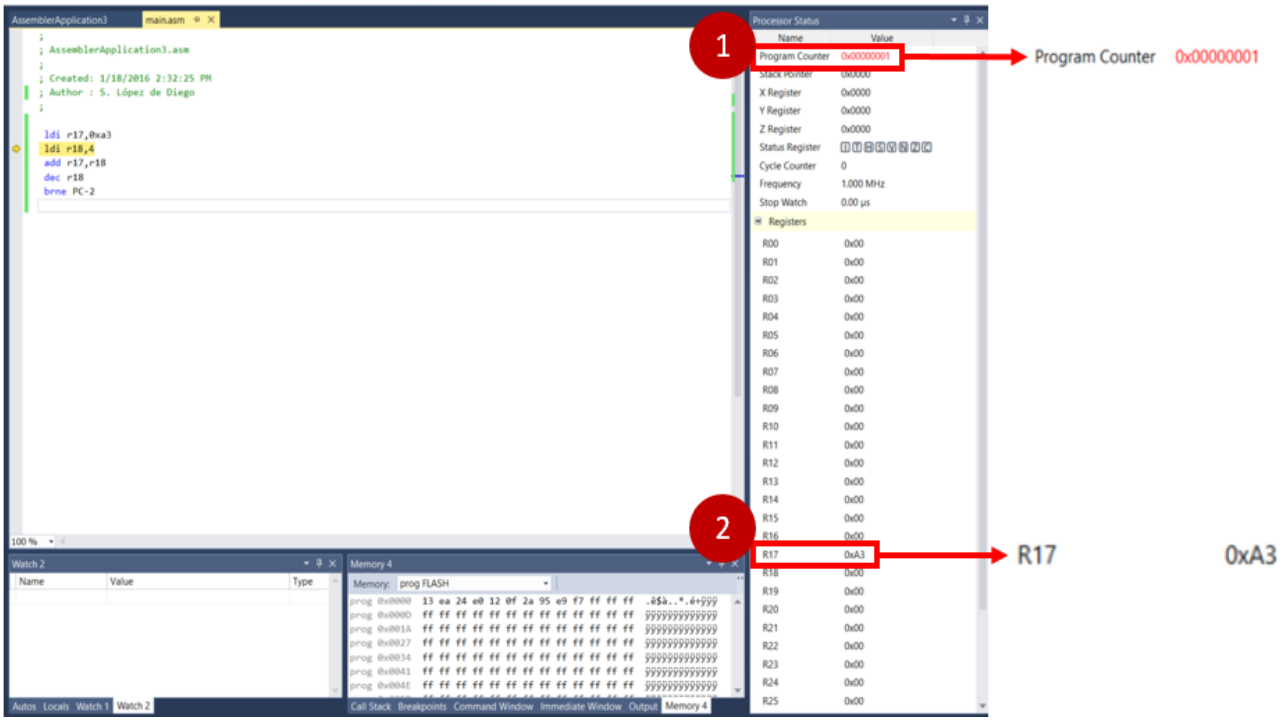
- **Now you can start debugging.**

1. Press the ***Start Debugging and Break*** (or Alt+F5) – (1) button.
2. This will move the simulator to the first instruction and break (stop) as highlighted below (2). The internal registers and states of the processor are in the right pane. The memory window is in the lower right pane. Watch and other information can be put in the lower left pane. The details of these windows will be discussed in the lectures.
3. To single step through the program, use the ***Step Into*** (or F11) - (3) button. As you step through the program, any registers, flags, memory, etc. that are modified by that instruction are highlighted in red. The other information provided is self-explanatory.



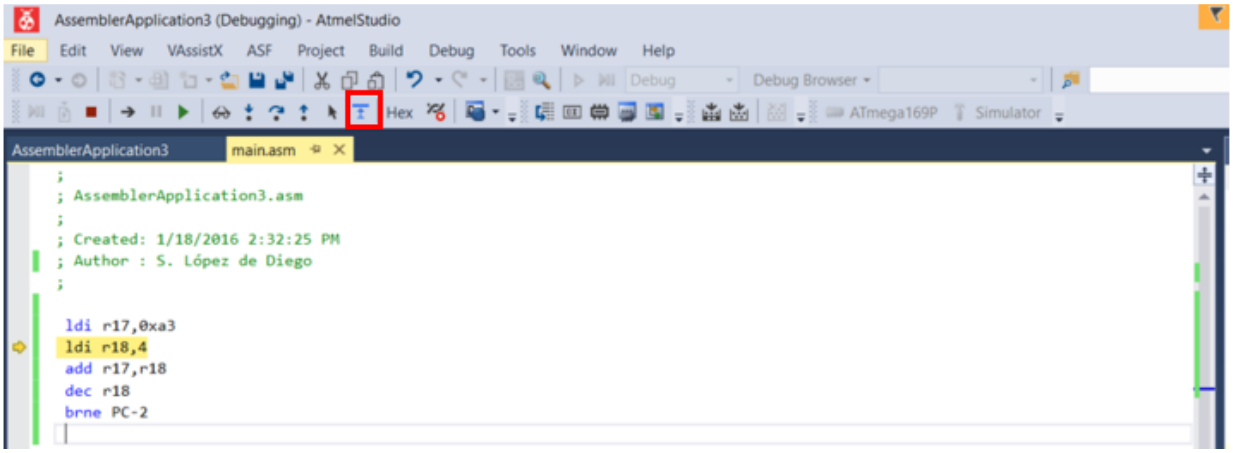
<Figure 8. Starting to Debug>

- After you press the step into button (3), you will see the following screen, where it is possible to see the following (as shown in Fig. 9):
  1. The program counter is incremented by one as seen in the upper right corner
  2. Register R17 is loaded with value 0xA3.



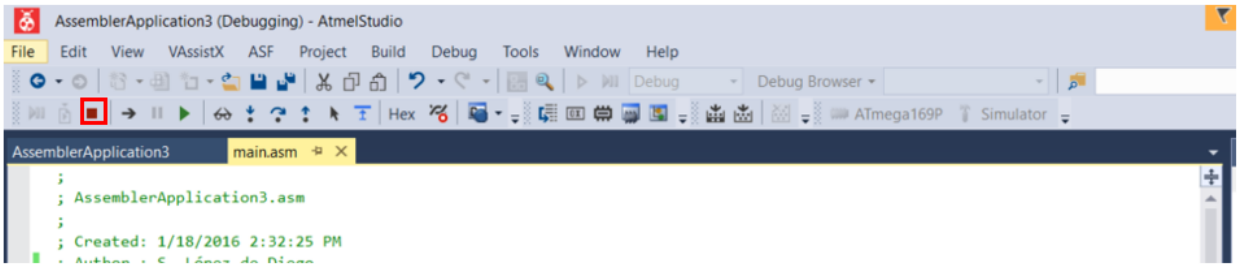
<Figure 9. Execution of the First Line of Code>

- If for some reason you need to reset the program counter to zero and restart the simulation, press the ***Reset*** (Shift + F5) button shown below and then everything will reset to initial condition.



<Figure 10. Resetting the Program>

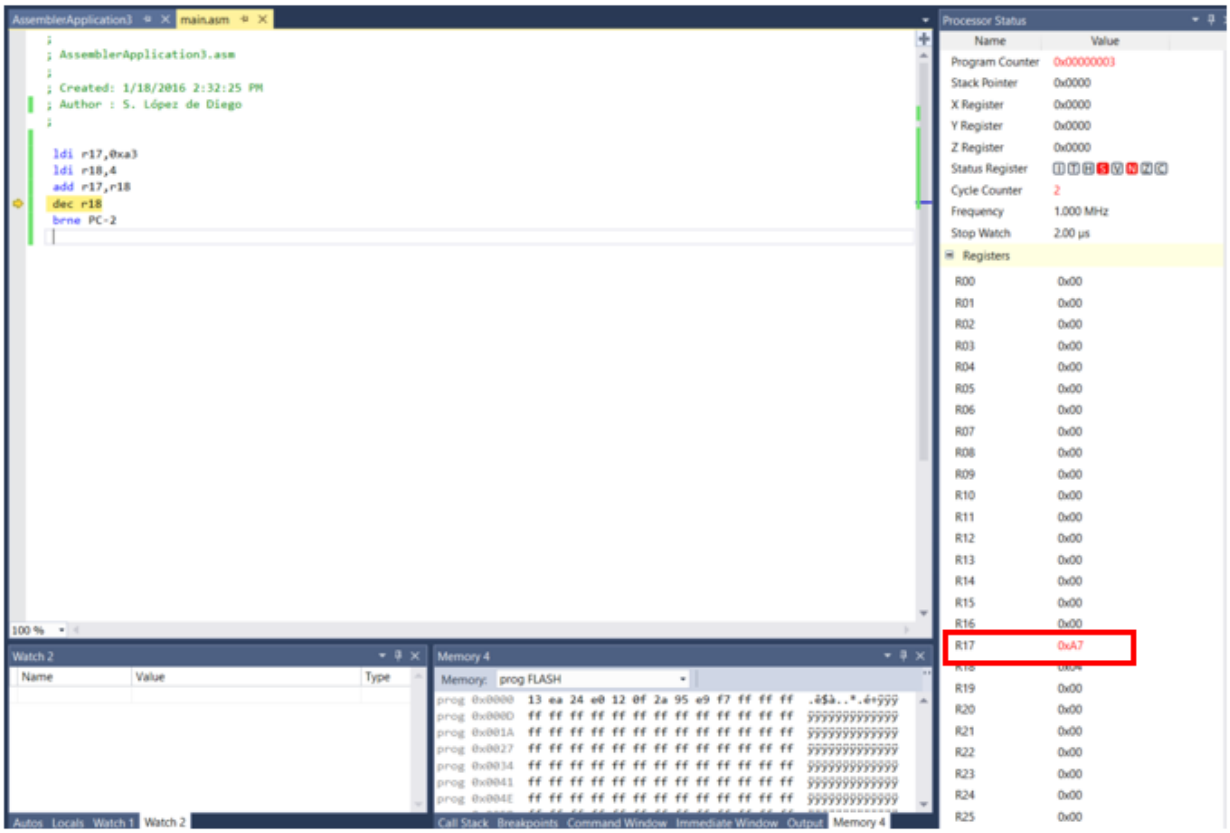
- **Note** that by resetting the program this way, the Program Counter will be reset, but the registers will not. To reset the registers as well, press ***Stop Debugging*** (Ctrl + Shift + F5) and then ***Start Debugging and Break*** (Alt-F5) to restart the program.



<Figure 11. How to Stop Debugging>

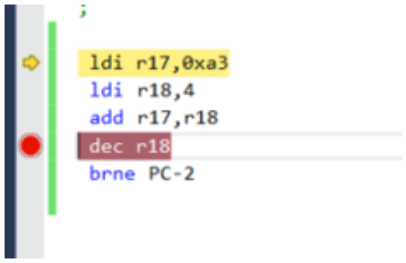
- When you want to exit the simulator, press the ***Stop Debugging*** button. This will take you back to the edit mode. **You cannot edit your code when you are in the simulation (debugging) mode.**

- After executing next few commands, we will see the program counter is incrementing and the contents of the registers r17 and r18 have been added and the result is stored in r17 (A3+4=A7).



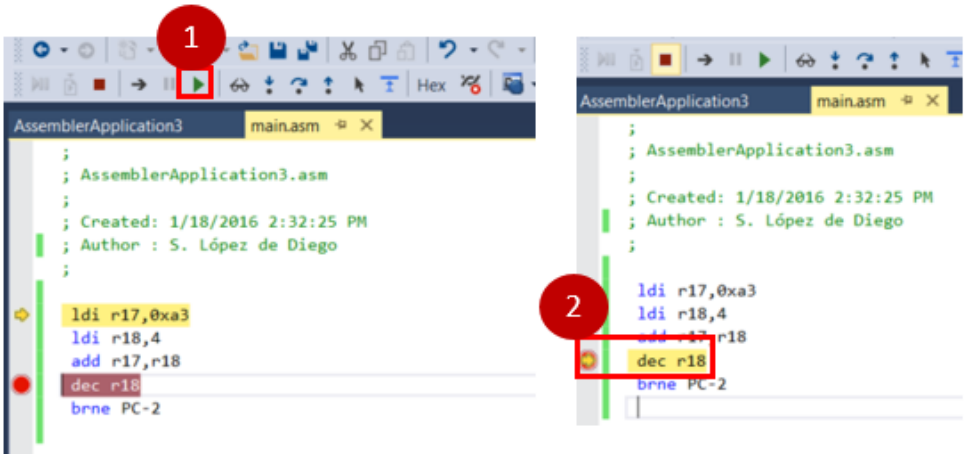
<Figure 12. Execution of the First Three Lines of Code>

- In the simulation mode, to continue running without single stepping, you can insert ***breakpoints*** and the program will run (press the ***Continue*** button) until it reaches a breakpoint.
  1. Breakpoints can be inserted/cleared by pressing the desired instruction's left margin area. These can be set and cleared in both edit and simulation modes. The image below shows a breakpoint inserted at the *dec* instruction.



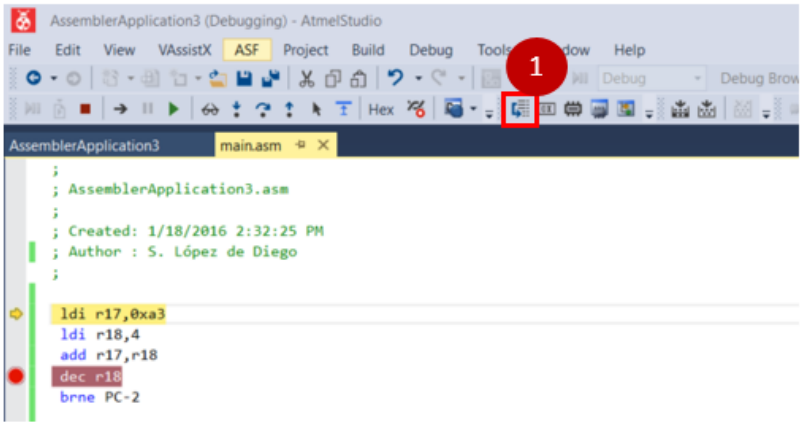
<Figure 13. Inserting a Breakpoint>

  2. Click on ***Run***
  3. The program will always stop at the breakpoint.

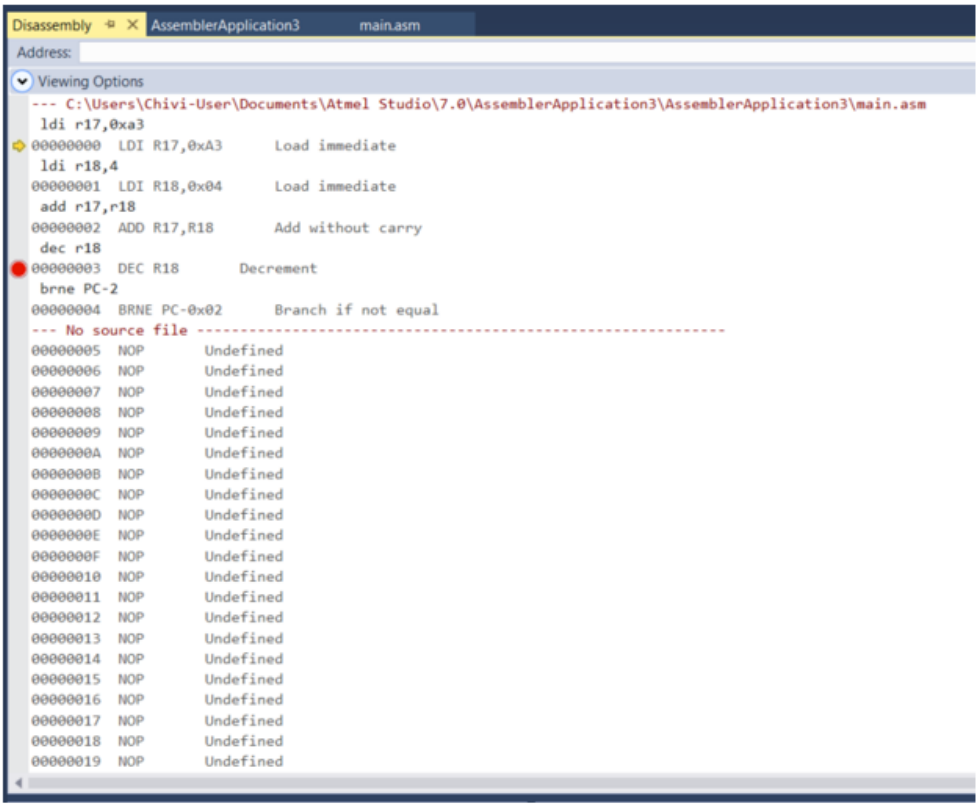


<Figure 14. Running and Stopping at Breakpoint>

- Next, we can view the disassembled code using the Disassembly view:
  1. Click on the ***Disassembly*** (or Alt+8) button (while the debugging is running)
  2. Observe the Disassembly window carefully.



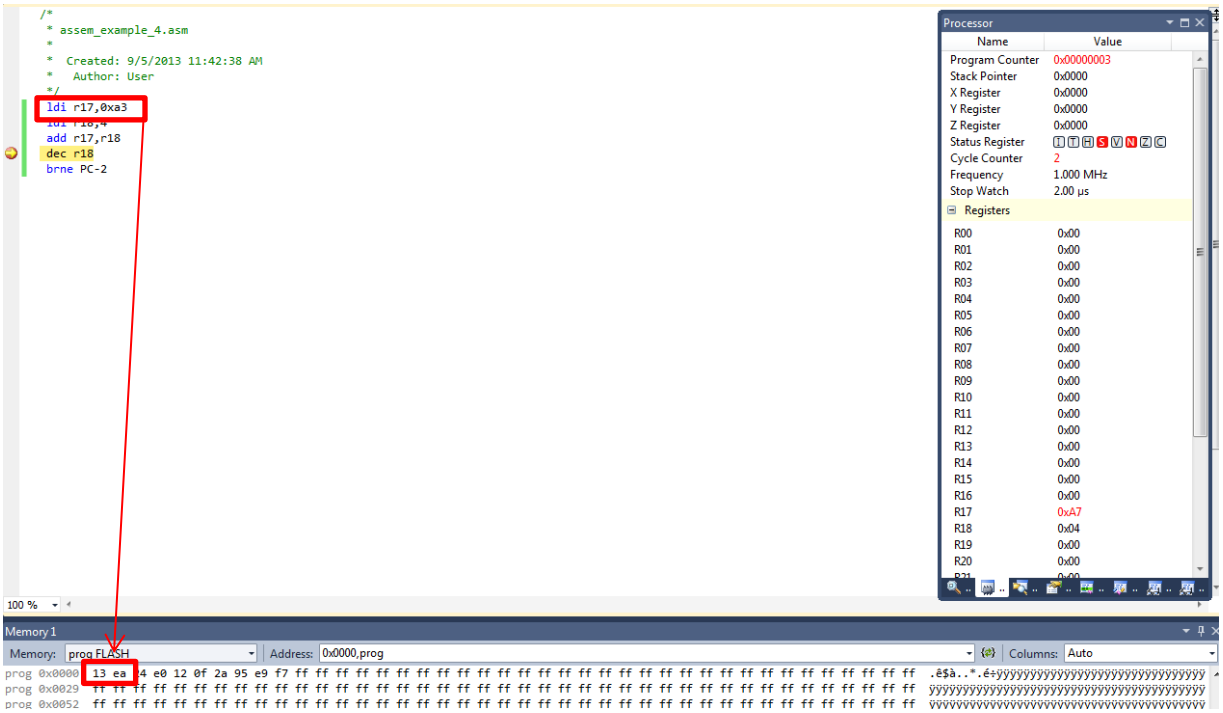
<Figure 15. Disassembly Button>



<Figure 16. Disassembly View>

2) OPCODES

Next, we will look at the opcodes. Memory is shown below at the bottom of the screen. The instruction ldi r17,0xa3 can be found in the first two memory locations.



<Figure 1. Opcode for the First Instruction>

Using AVR opcode table:

Go to: http://lyons42.com/AVR/Opcodes/AVRAllOpcodes.html#Block224 (copy and paste)

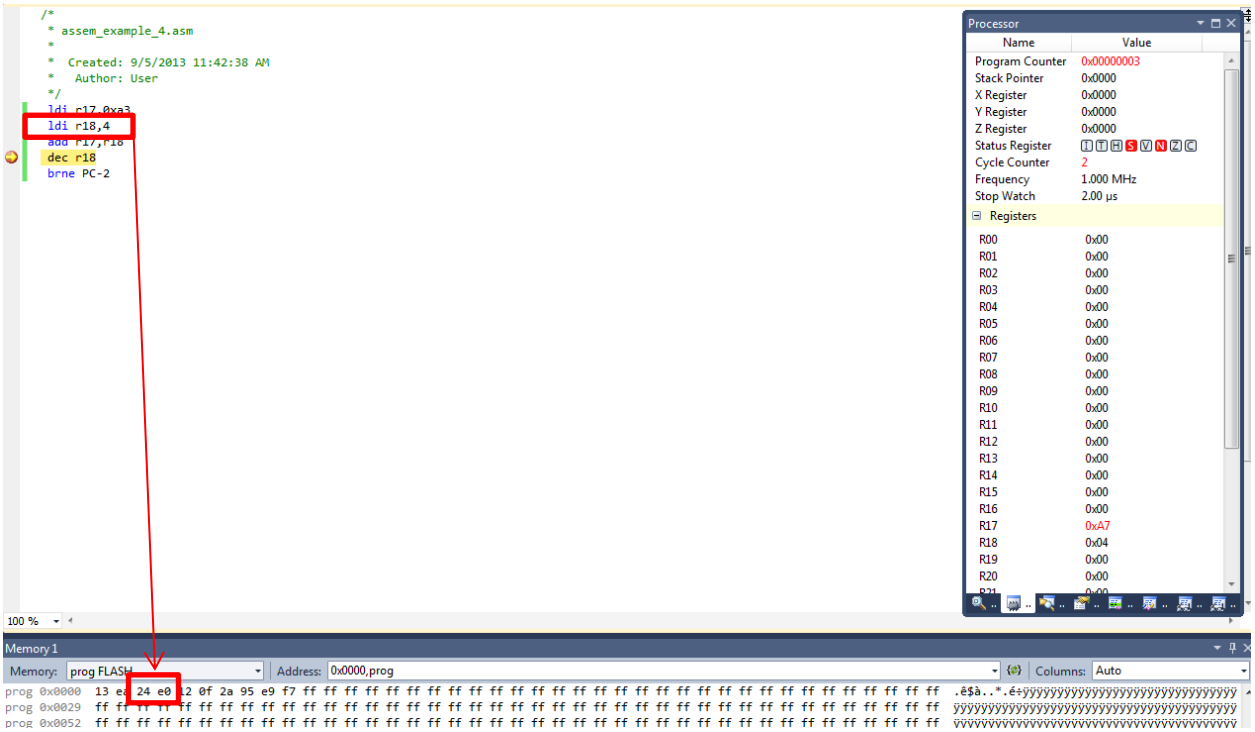
- Scroll down to

Opcodes EAxx (0xEA00 - 0xEAFF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
EA0x	ldi r16, 0xA0	ldi r16, 0xA1	ldi r16, 0xA2	ldi r16, 0xA3	ldi r16, 0xA4	ldi r16, 0xA5	ldi r16, 0xA6	ldi r16, 0xA7	ldi r16, 0xA8	ldi r16, 0xA9	ldi r16, 0xAA	ldi r16, 0xAB	ldi r16, 0xAC	ldi r16, 0xAD	ldi r16, 0xAE	ldi r16, 0xAF
EA1x	ldi r17, 0xA0	ldi r17, 0xA1	ldi r17, 0xA2	ldi r17, 0xA3	ldi r17, 0xA4	ldi r17, 0xA5	ldi r17, 0xA6	ldi r17, 0xA7	ldi r17, 0xA8	ldi r17, 0xA9	ldi r17, 0xAA	ldi r17, 0xAB	ldi r17, 0xAC	ldi r17, 0xAD	ldi r17, 0xAE	ldi r17, 0xAF
EA2x	ldi r18, 0xA0	ldi r18, 0xA1	ldi r18, 0xA2	ldi r18, 0xA3	ldi r18, 0xA4	ldi r18, 0xA5	ldi r18, 0xA6	ldi r18, 0xA7	ldi r18, 0xA8	ldi r18, 0xA9	ldi r18, 0xAA	ldi r18, 0xAB	ldi r18, 0xAC	ldi r18, 0xAD	ldi r18, 0xAE	ldi r18, 0xAF
EA3x	ldi r19, 0xA0	ldi r19, 0xA1	ldi r19, 0xA2	ldi r19, 0xA3	ldi r19, 0xA4	ldi r19, 0xA5	ldi r19, 0xA6	ldi r19, 0xA7	ldi r19, 0xA8	ldi r19, 0xA9	ldi r19, 0xAA	ldi r19, 0xAB	ldi r19, 0xAC	ldi r19, 0xAD	ldi r19, 0xAE	ldi r19, 0xAF

- Note that ea13 is the opcode for ldi r17, 0xA3

- Next, we look at ldi r18, 4



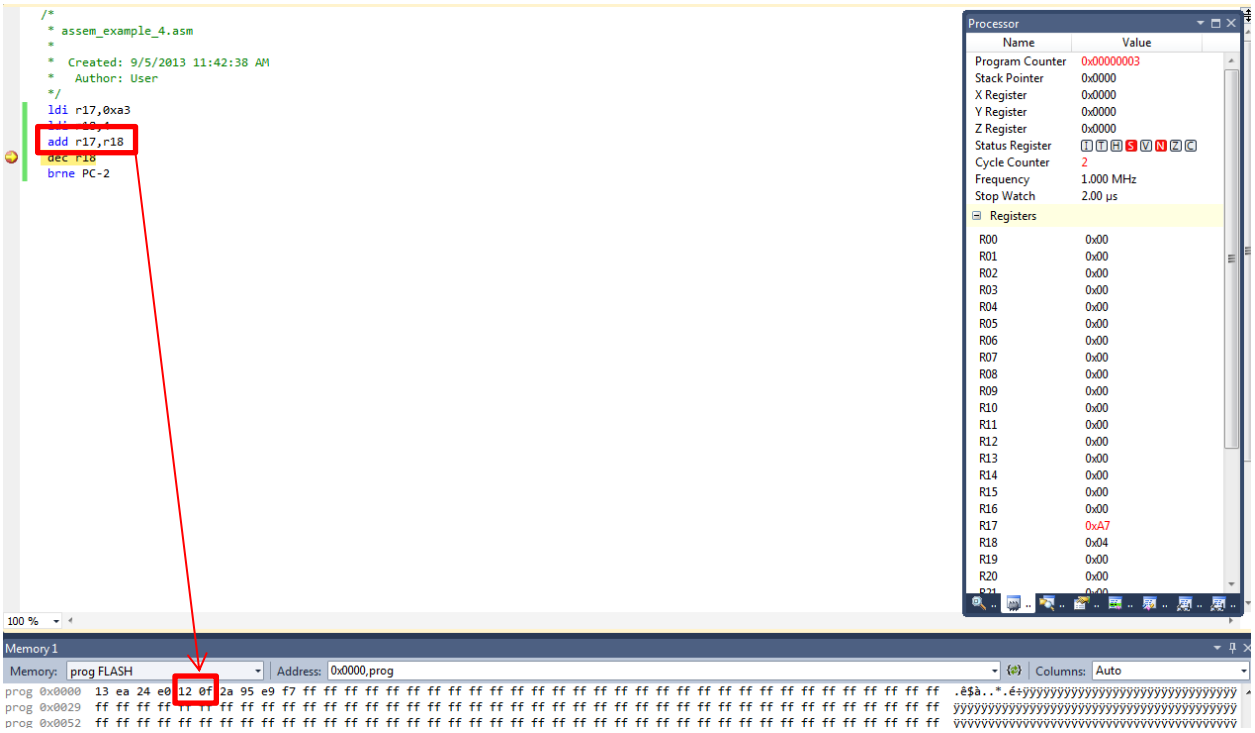
<Figure 2. Opcode for the Second Instruction>

From the opcode table

Opcodes E0xx (0xE000 - 0xE0FF)

	0	1	2	3	4	5	6	7	8	9
E00x	ldi r16, 0x00	ldi r16, 0x01	ldi r16, 0x02	ldi r16, 0x03	ldi r16, 0x04	ldi r16, 0x05	ldi r16, 0x06	ldi r16, 0x07	ldi r16, 0x08	ldi r1 0x0
E01x	ldi r17, 0x00	ldi r17, 0x01	ldi r17, 0x02	ldi r17, 0x03	ldi r17, 0x04	ldi r17, 0x05	ldi r17, 0x06	ldi r17, 0x07	ldi r17, 0x08	ldi r1 0x0
E02x	ldi r18, 0x00	ldi r18, 0x01	ldi r18, 0x02	ldi r18, 0x03	ldi r18, 0x04	ldi r18, 0x05	ldi r18, 0x06	ldi r18, 0x07	ldi r18, 0x08	ldi r1 0x0
E03x	ldi r19, 0x00	ldi r19, 0x01	ldi r19, 0x02	ldi r19, 0x03	ldi r19, 0x04	ldi r19, 0x05	ldi r19, 0x06	ldi r19, 0x07	ldi r19, 0x08	ldi r1 0x0

- Note that e024 is the opcode for ldi r18, 0x04
- Next, we have, add r17, r18
- Let's look at the instruction add r17, r18 which has opcode 0f12



<Figure 3. Opcode for the Third Instruction>

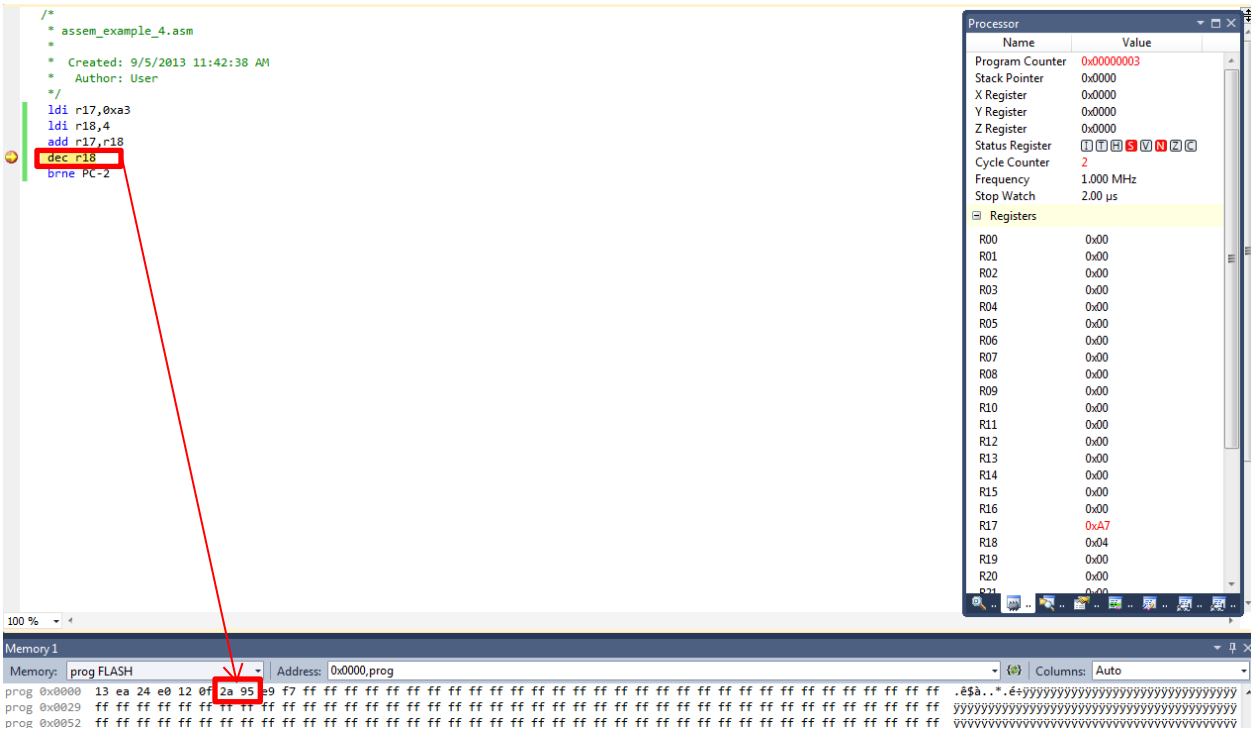
- And from the opcode table

Opcodes 0Fxx (0x0F00 - 0x0FFF)

	0	1	2	3	4	5
0F0x	add r16, r16	add r16, r17	add r16, r18	add r16, r19	add r16, r20	add r16, r21
0F1x	add r17, r16	add r17, r17	add r17, r18	add r17, r19	add r17, r20	add r17, r21
0F2x	add r18, r16	add r18, r17	add r18, r18	add r18, r19	add r18, r20	add r18, r21

- Note that 0f12 is the opcode for add r17, r18

- Now, look at the next instruction dec r18 which has opcode 952a



<Figure 4. Opcode for the Fourth Instruction>

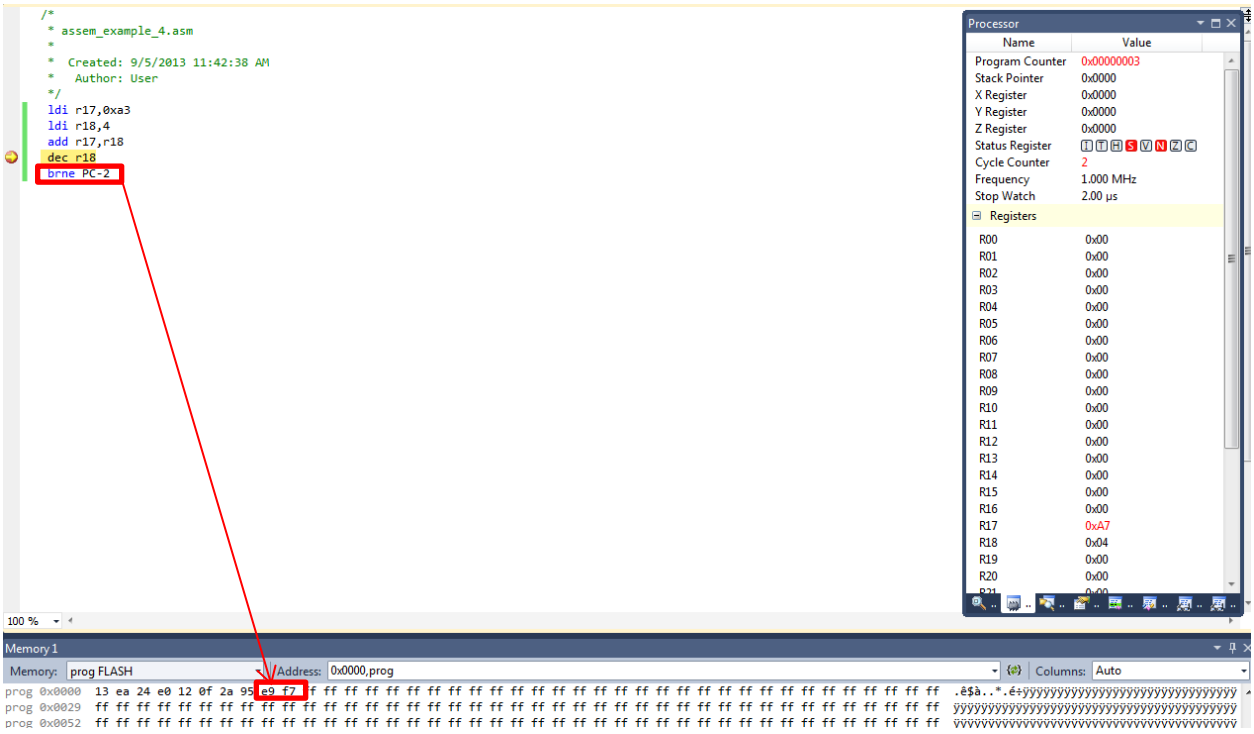
- And from the opcode table

Opcodes 95xx (0x9500 - 0x95FF)

	0	1	2	3	4	5	6	7	8	9	A	B	C
950x	com r16	neg r16	swap r16	inc r16	[R]	asr r16	lsr r16	ror r16	ret	icall	dec r16	[R]	jmp 0x412a18
951x	com r17	neg r17	swap r17	inc r17	[R]	asr r17	lsr r17	ror r17	reti	eicall	dec r17	[R]	jmp 0x452a38
952x	com r18	neg r18	swap r18	inc r18	[R]	asr r18	lsr r18	ror r18	[R]		dec r18	[R]	jmp 0x492a58
953x	com r19	neg r19	swap r19	inc r19	[R]	asr r19	lsr r19	ror r19	[R]		dec r19	[R]	jmp 0x4d2a78
954x	com r20	neg r20	swap r20	inc r20	[R]	asr r20	lsr r20	ror r20	[R]		dec r20	[R]	jmp 0x512a98

- Note that dec r18 which has opcode 952a

- Let's look at the next instruction brne PC-2 which has opcode F7e9



<Figure 5. Opcode for the Last Instruction>

- From the opcode table:

Opcodes F7xx (0xF700 - 0xF7FF)

	0	1	2	3	4	5	6	7	8	9	A	B
F70x	brcc .-64	brne .-64	brpl .-64	brvc .-64	brge .-64	brhc .-64	brtc .-64	brid .-64	brcc .-62	brne .-62	brpl .-62	brvc .-62
F71x	brcc .-60	brne .-60	brpl .-60	brvc .-60	brge .-60	brhc .-60	brtc .-60	brid .-60	brcc .-58	brne .-58	brpl .-58	brvc .-58
F72x	brcc .-56	brne .-56	brpl .-56	brvc .-56	brge .-56	brhc .-56	brtc .-56	brid .-56	brcc .-54	brne .-54	brpl .-54	brvc .-54
F73x	brcc .-52	brne .-52	brpl .-52	brvc .-52	brge .-52	brhc .-52	brtc .-52	brid .-52	brcc .-50	brne .-50	brpl .-50	brvc .-50
F74x	brcc .-48	brne .-48	brpl .-48	brvc .-48	brge .-48	brhc .-48	brtc .-48	brid .-48	brcc .-46	brne .-46	brpl .-46	brvc .-46
F75x	brcc .-44	brne .-44	brpl .-44	brvc .-44	brge .-44	brhc .-44	brtc .-44	brid .-44	brcc .-42	brne .-42	brpl .-42	brvc .-42
F76x	brcc .-40	brne .-40	brpl .-40	brvc .-40	brge .-40	brhc .-40	brtc .-40	brid .-40	brcc .-38	brne .-38	brpl .-38	brvc .-38
F77x	brcc .-36	brne .-36	brpl .-36	brvc .-36	brge .-36	brhc .-36	brtc .-36	brid .-36	brcc .-34	brne .-34	brpl .-34	brvc .-34
F78x	brcc .-32	brne .-32	brpl .-32	brvc .-32	brge .-32	brhc .-32	brtc .-32	brid .-32	brcc .-30	brne .-30	brpl .-30	brvc .-30
F79x	brcc .-28	brne .-28	brpl .-28	brvc .-28	brge .-28	brhc .-28	brtc .-28	brid .-28	brcc .-26	brne .-26	brpl .-26	brvc .-26
F7Ax	brcc .-24	brne .-24	brpl .-24	brvc .-24	brge .-24	brhc .-24	brtc .-24	brid .-24	brcc .-22	brne .-22	brpl .-22	brvc .-22
F7Bx	brcc .-20	brne .-20	brpl .-20	brvc .-20	brge .-20	brhc .-20	brtc .-20	brid .-20	brcc .-18	brne .-18	brpl .-18	brvc .-18
F7Cx	brcc .-16	brne .-16	brpl .-16	brvc .-16	brge .-16	brhc .-16	brtc .-16	brid .-16	brcc .-14	brne .-14	brpl .-14	brvc .-14
F7Dx	brcc .-12	brne .-12	brpl .-12	brvc .-12	brge .-12	brhc .-12	brtc .-12	brid .-12	brcc .-10	brne .-10	brpl .-10	brvc .-10
F7Ex	brcc .-8	brne .-8	brpl .-8	brvc .-8	brge .-8	brhc .-8	brtc .-8	brid .-8	brcc .-6	brne .-6	brpl .-6	brvc .-6

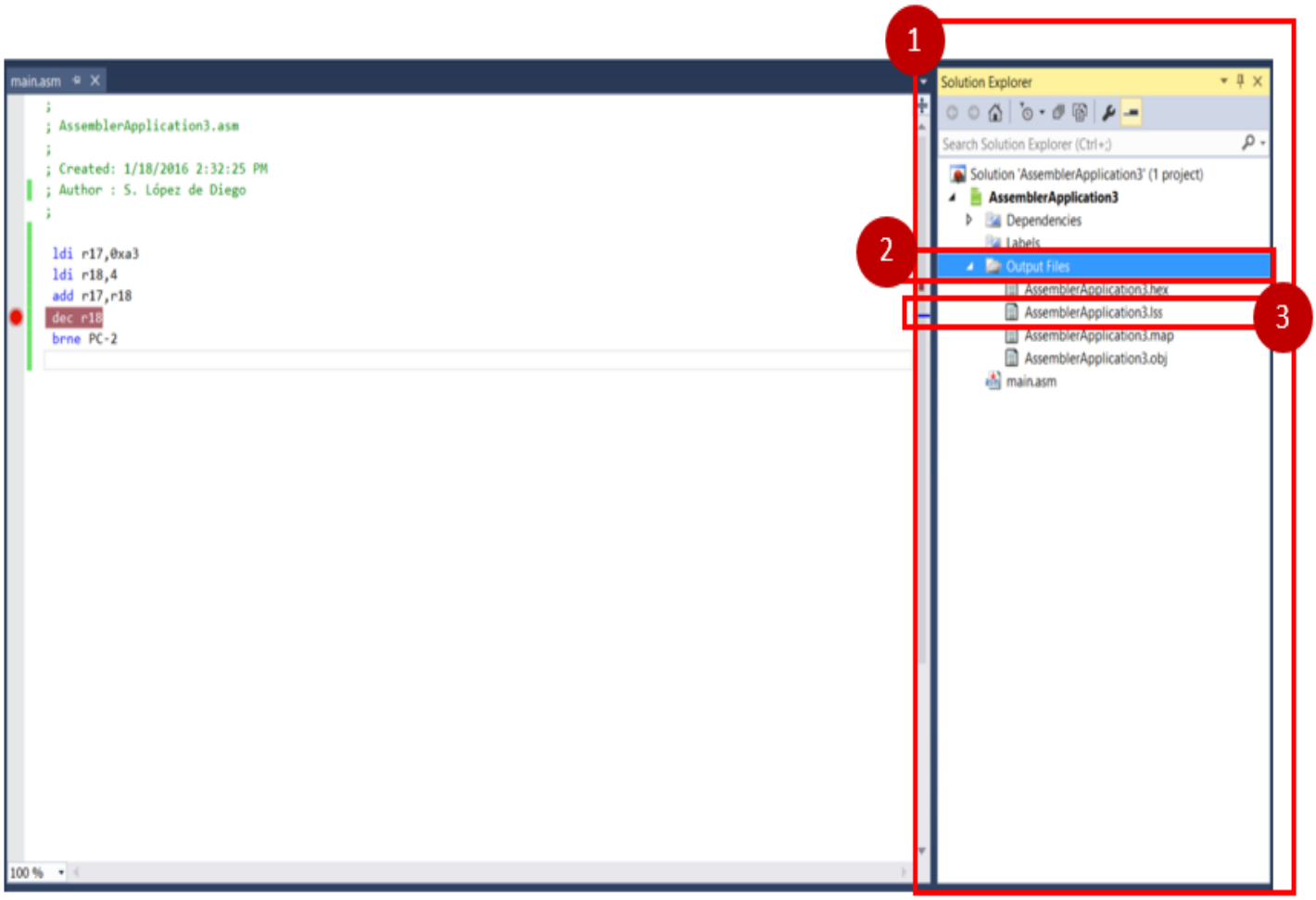
Note that brne PC-2 has opcode F7e9. The PC-2 requires the program counter to move back two instructions. **Each instruction takes two memory locations** and from the sequence of instructions

```
add r17,r18
dec r18
brne PC-2
```

which takes up *six memory locations* then from the opcode table which has **brne.-6** moves the actual instruction pointer to *six memory locations back* to where the add r17,r18 is located.

There are a few items that make this a little tricky. One is that, since it is a Harvard architecture and the data memory is different than the program memory; the program memory is then normally addressed on 16-bit words and the data memory on 8-bit. So, this can get a little confusing when looking at the instructions in a memory dump. Once assembled, there is something called a **lss** file. You should open that, and it will provide you a more traditional view of the machine code.

- To open the Your_*Filename.**lss*** file do the following (Fig. 6):
  1. Open the ***Solution Explorer*** (1) (View > ***Solution Explorer***).
  2. Expand the ***Output Files*** (2).
  3. Double click on the ***lss*** file (3).



<Figure 6. Opening the *lss* file>

- Scroll down near the end of the opened file and see the following:

```
                                ; ***** END OF FILE ****************************************
                                ; Created: 9/2/2018 2:40:19 PM
                                ; Author : sayemul islam

                                start:
000000 ea13                     ldi R17,0xA3
000001 e024                     ldi R18,4
000002 0f12                     add R17,R18
000003 952a                     dec R18
000004 f7e9                     brne PC-2
```

<Figure 7. *lss* File>

- Scroll down some more and see the registers and instructions that are used and how many times they are used:

```
"ATmega324PB" register use summary:
x  :   0 y  :   0 z  :   0 r0 :   0 r1 :   0 r2 :   0 r3 :   0 r4 :   0
r5 :   0 r6 :   0 r7 :   0 r8 :   0 r9 :   0 r10:   0 r11:   0 r12:   0
r13:   0 r14:   0 r15:   0 r16:   0 r17:   2 r18:   3 r19:   0 r20:   0
r21:   0 r22:   0 r23:   0 r24:   0 r25:   0 r26:   0 r27:   0 r28:   0
r29:   0 r30:   0 r31:   0
Registers used: 2 out of 35 (5.7%)

"ATmega324PB" instruction use summary:
.lds  :   0 .sts  :   0 adc   :   0 add   :   1 adiw  :   0 and   :   0
andi  :   0 asr   :   0 bclr  :   0 bld   :   0 brbc  :   0 brbs  :   0
brcc  :   0 brcs  :   0 break :   0 breq  :   0 brge  :   0 brhc  :   0
brhs  :   0 brid  :   0 brie  :   0 brlo  :   0 brlt  :   0 brmi  :   0
brne  :   1 brpl  :   0 brsh  :   0 brtc  :   0 brts  :   0 brvc  :   0
brvs  :   0 bset  :   0 bst   :   0 call  :   0 cbi   :   0 cbr   :   0
clc   :   0 clh   :   0 cli   :   0 cln   :   0 clr   :   0 cls   :   0
clt   :   0 clv   :   0 clz   :   0 com   :   0 cp    :   0 cpc   :   0
cpi   :   0 cpse  :   0 dec   :   1 eor   :   0 fmul  :   0 fmuls :   0
fmulsu:   0 icall :   0 ijmp  :   0 in    :   0 inc   :   0 jmp   :   0
ld    :   0 ldd   :   0 ldi   :   2 lds   :   0 lpm   :   0 lsl   :   0
lsr   :   0 mov   :   0 movw  :   0 mul   :   0 muls  :   0 mulsu :   0
neg   :   0 nop   :   0 or    :   0 ori   :   0 out   :   0 pop   :   0
push  :   0 rcall :   0 ret   :   0 reti  :   0 rjmp  :   0 rol   :   0
ror   :   0 sbc   :   0 sbci  :   0 sbi   :   0 sbic  :   0 sbis  :   0
sbiw  :   0 sbr   :   0 sbrc  :   0 sbrs  :   0 sec   :   0 seh   :   0
sei   :   0 sen   :   0 ser   :   0 ses   :   0 set   :   0 sev   :   0
sez   :   0 sleep :   0 spm   :   0 st    :   0 std   :   0 sts   :   0
sub   :   0 subi  :   0 swap  :   0 tst   :   0 wdr   :   0
Instructions used: 4 out of 113 (3.5%)
```

<Figure 8. Summary of Instructions and Registers Used>

Lab Report

- **Report:** No report is required for this lab. Install appropriately Atmel Studio 7 and get familiar to the tools of Atmel Studio 7.