Getting Started

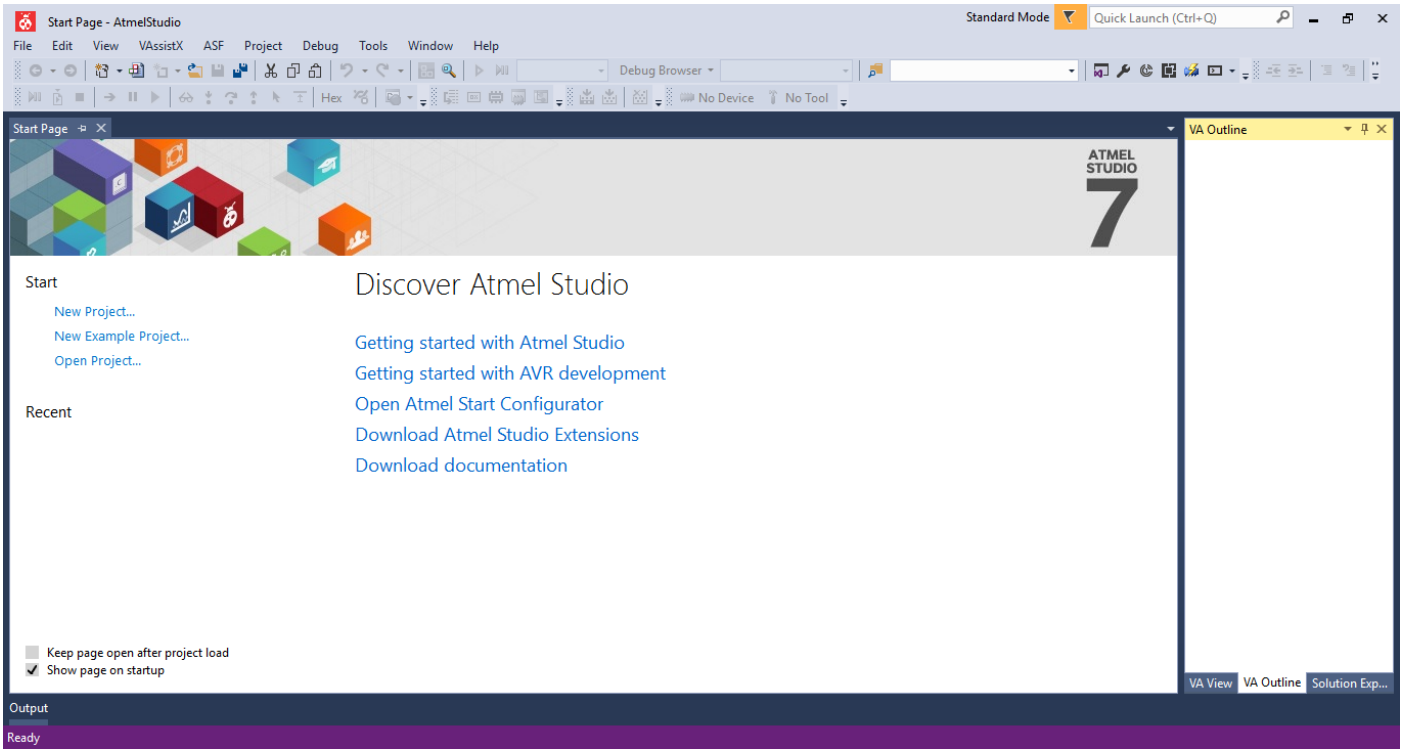
Installing Atmel Studio

Download the Atmel Studio 7 installer from this link http://www.atmel.com/tools/atmelstudio.aspx#download.

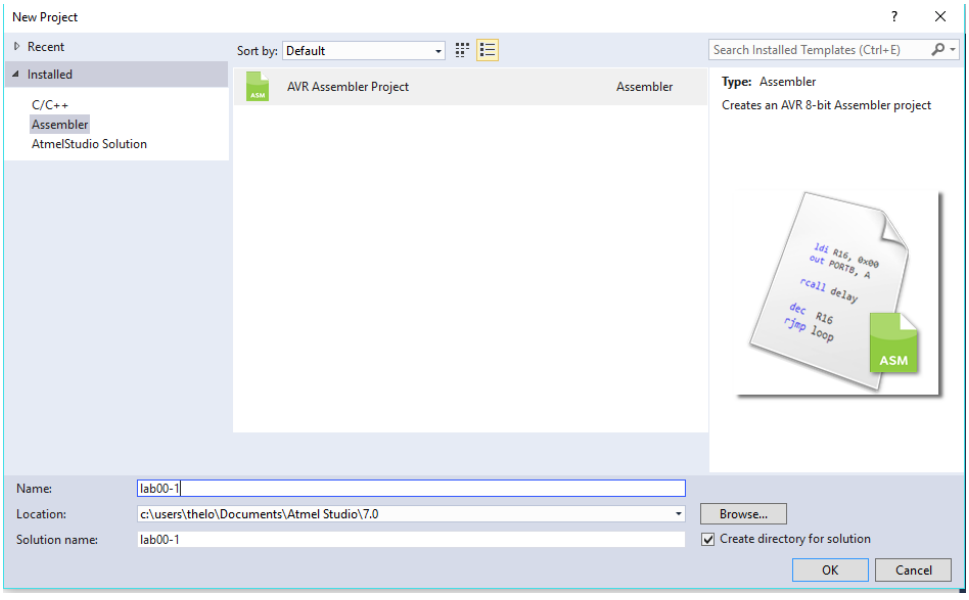
Make sure you select the AVR 8-bit MCU option for the installation.

Building and Debugging

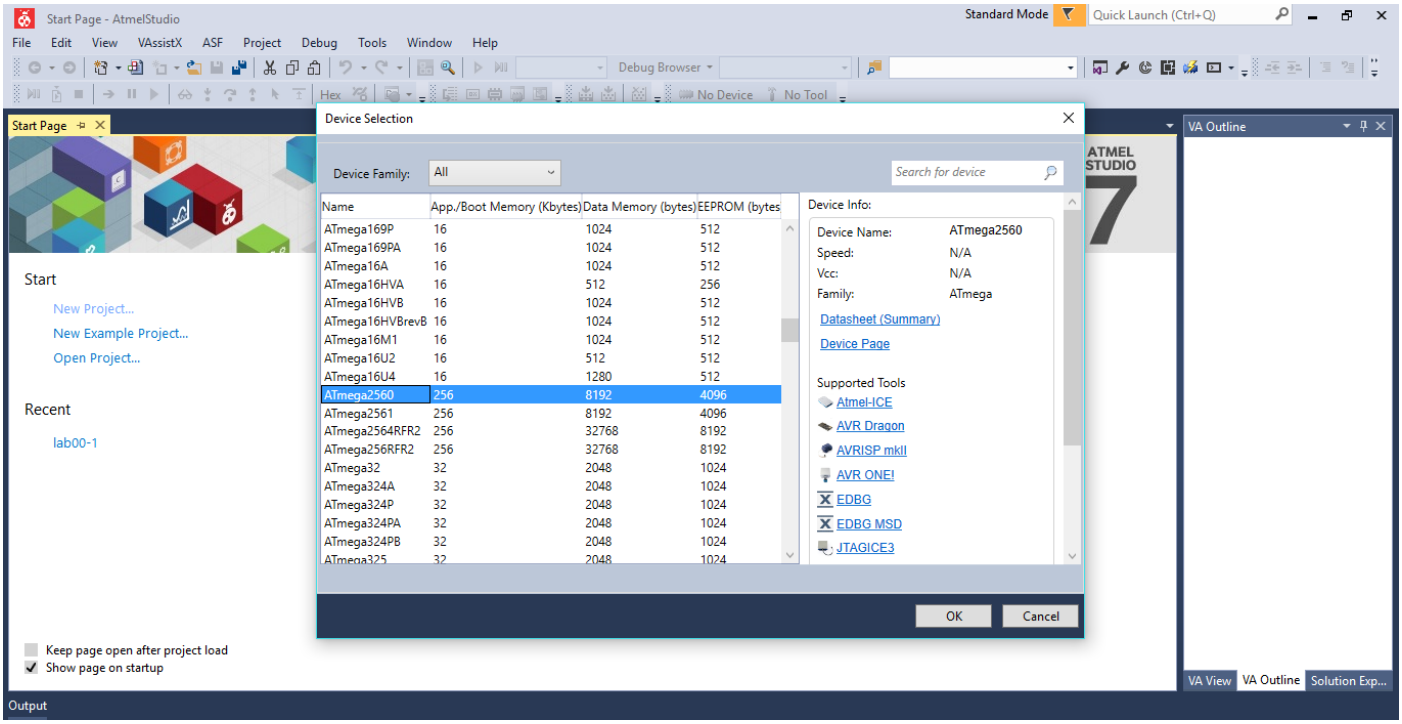
Once Atmel Studio has launched, select New Project on the Start Page.



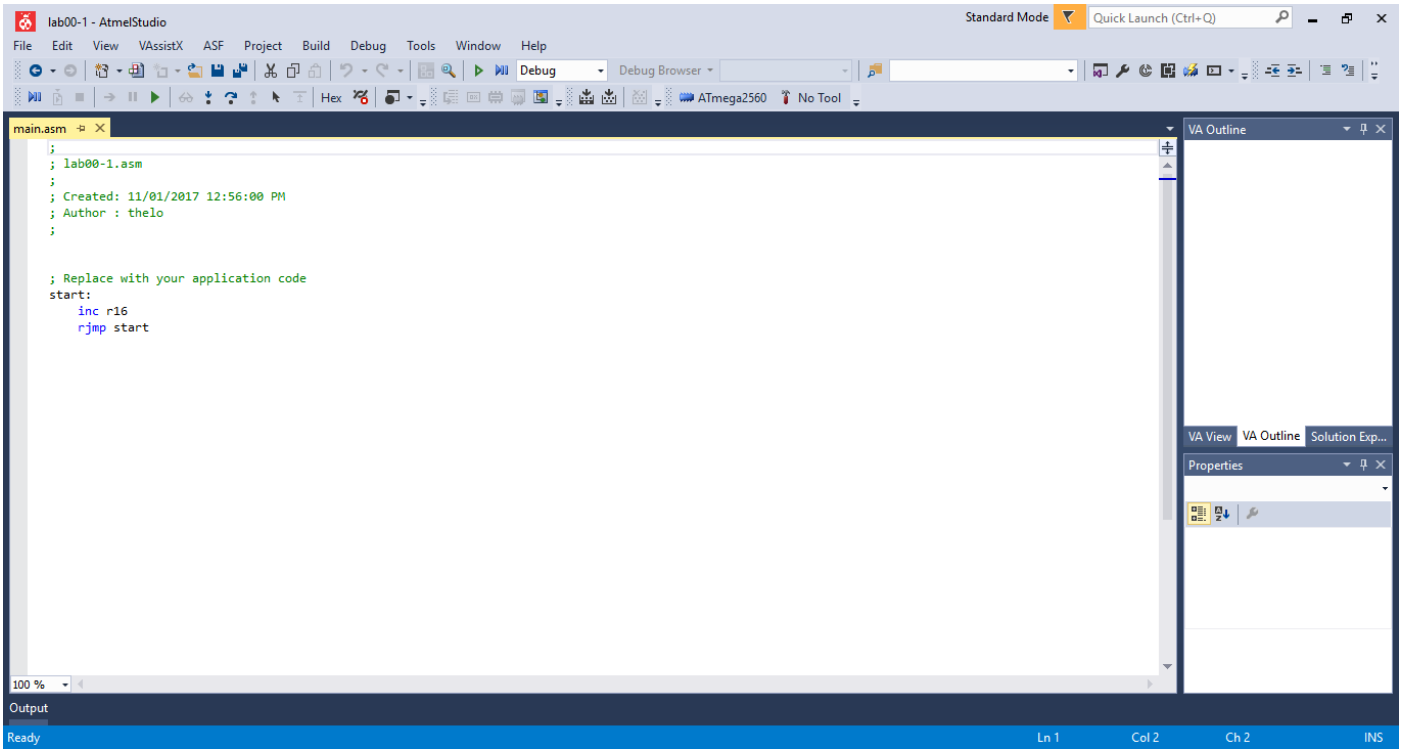
You will then be presented with the New Project window. Underneath the Installed tab, select the Assembler option, select AVR Assembler Project and name the project “lab00-1”.



Once you click OK on this screen, you will be taken to the device selection window. Either scroll or search for “ATmega2560”, select it and click OK.



If the editor has not yet opened “main.asm”, then open that file. The contents of the file should be a simple assembly file scaffold similar to the below screenshot.



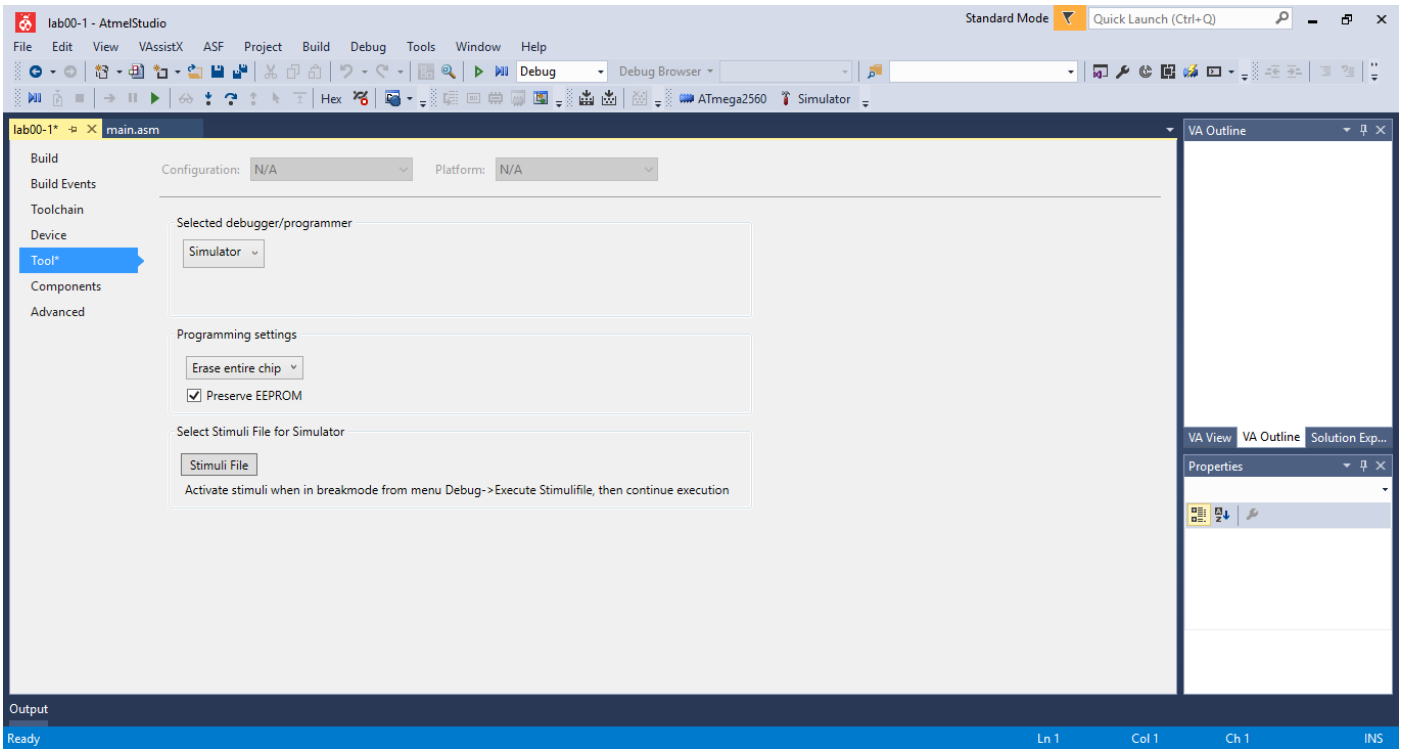
Now replace the entire contents of this file with the following code snippet.

```
.include “m2560def.inc”

start:
        ldi r16, 8
        ldi r17, 9

halt:
        rjmp halt
```

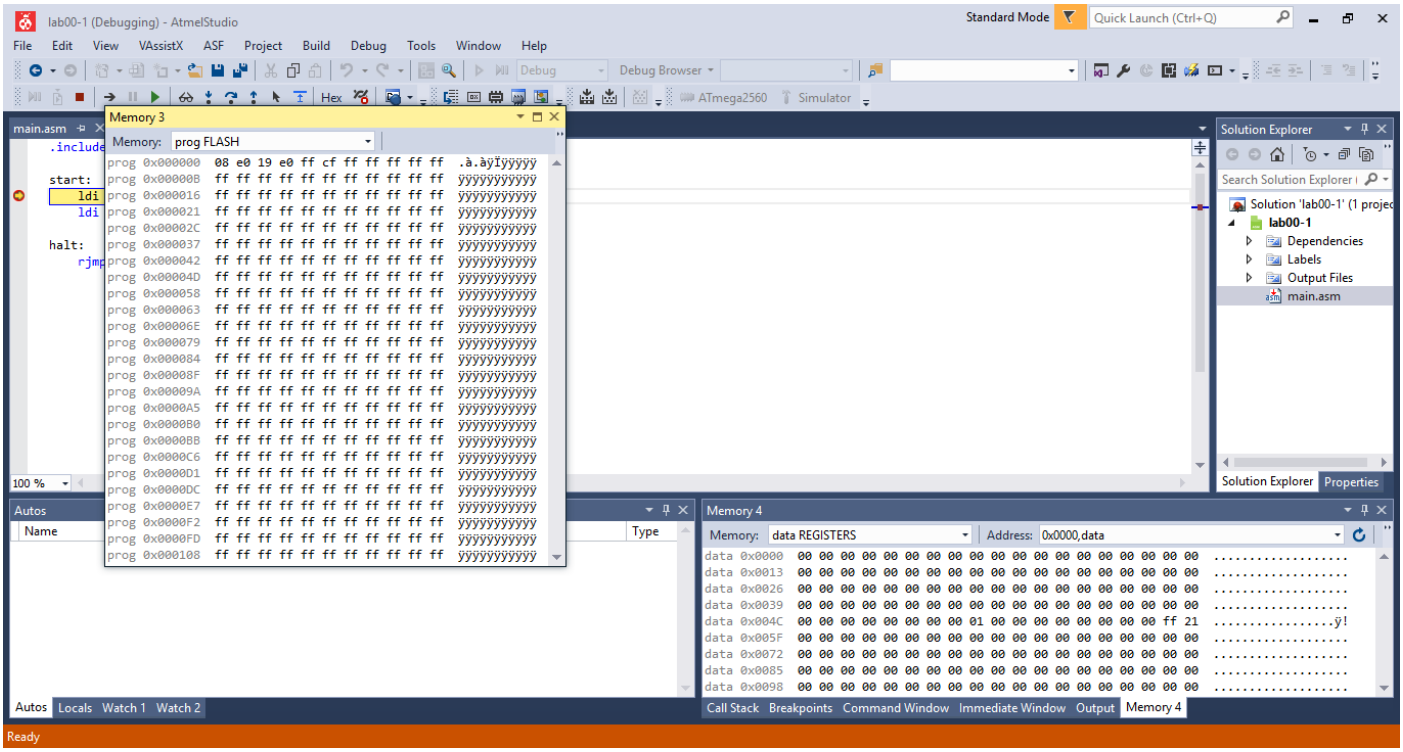
Note: There is an issue with formatting of the double quotes in this code, so simply copying and pasting the code will cause an error. You should type it directly into the editor instead.

The “.include” directive works in the same way as C’s “#include”. “m2560def.inc” contains port and register definitions for the ATmega2560 chip and should be included at the top of every assembler program.

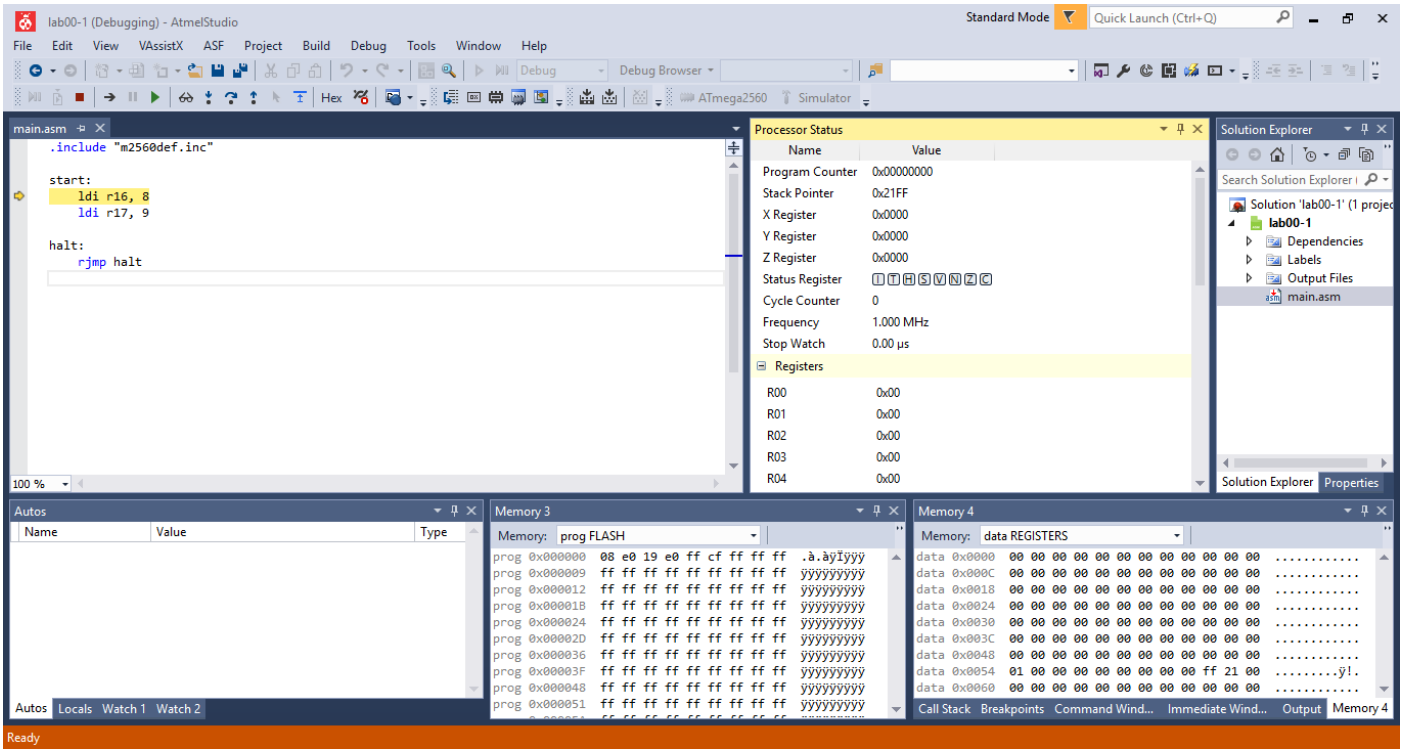
Press Alt+F5 to start executing the program and to halt at the first instruction. If the below screen appears then select the Simulator as the Selected Debugger and then click on the “main.asm” tab to return to your source code. Press Alt+F5 again to begin simulating.



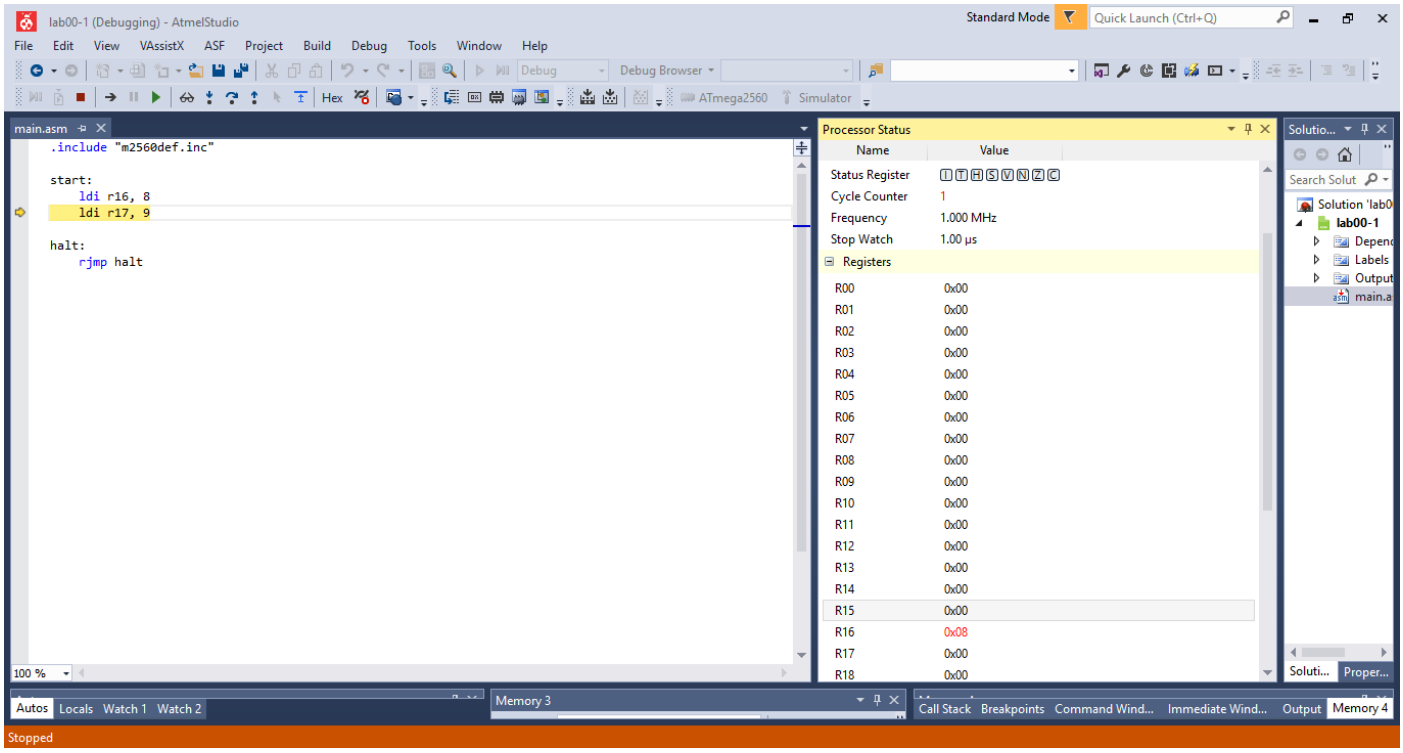
The screen will change and some new windows will appear. If a window appears obstructing the view of your source code, you can drag and drop windows in order to rearrange them.



Now you should go to Debug->Windows->Processor Status and drag the new window to the side. This window contains important views that will help you debug your assembly programs including the registers, status register and stack pointer. The yellow arrow denotes the instruction to be performed next.

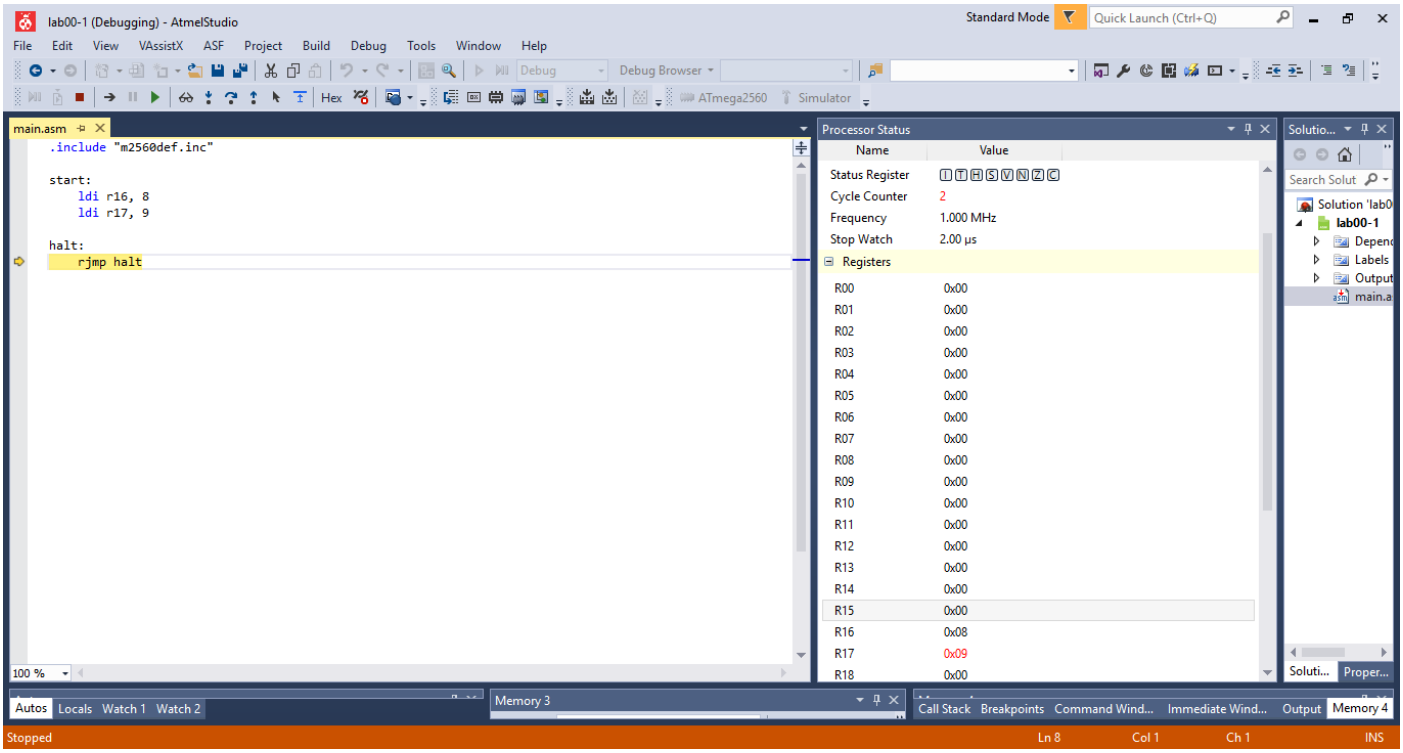


You can now step through the program by individual instruction by pressing F10 (step over). Do this now.



Focusing your attention on the processor status window, you can see all 32 registers and their current values are displayed. The instruction just performed was to load an immediate value of 8 into the 16th register (r16) and you can see the processor status indicates this has happened. Notice that the value is in red denoting that the value has changed from the previous step. Press F10 to step again.

You have now reached the end of the program (continuing to press F10 will have no effect as the instruction jumps to itself effectively making an infinite loop). Note that R17 has changed and the value is highlighted red. Press the stop button to stop the simulator.

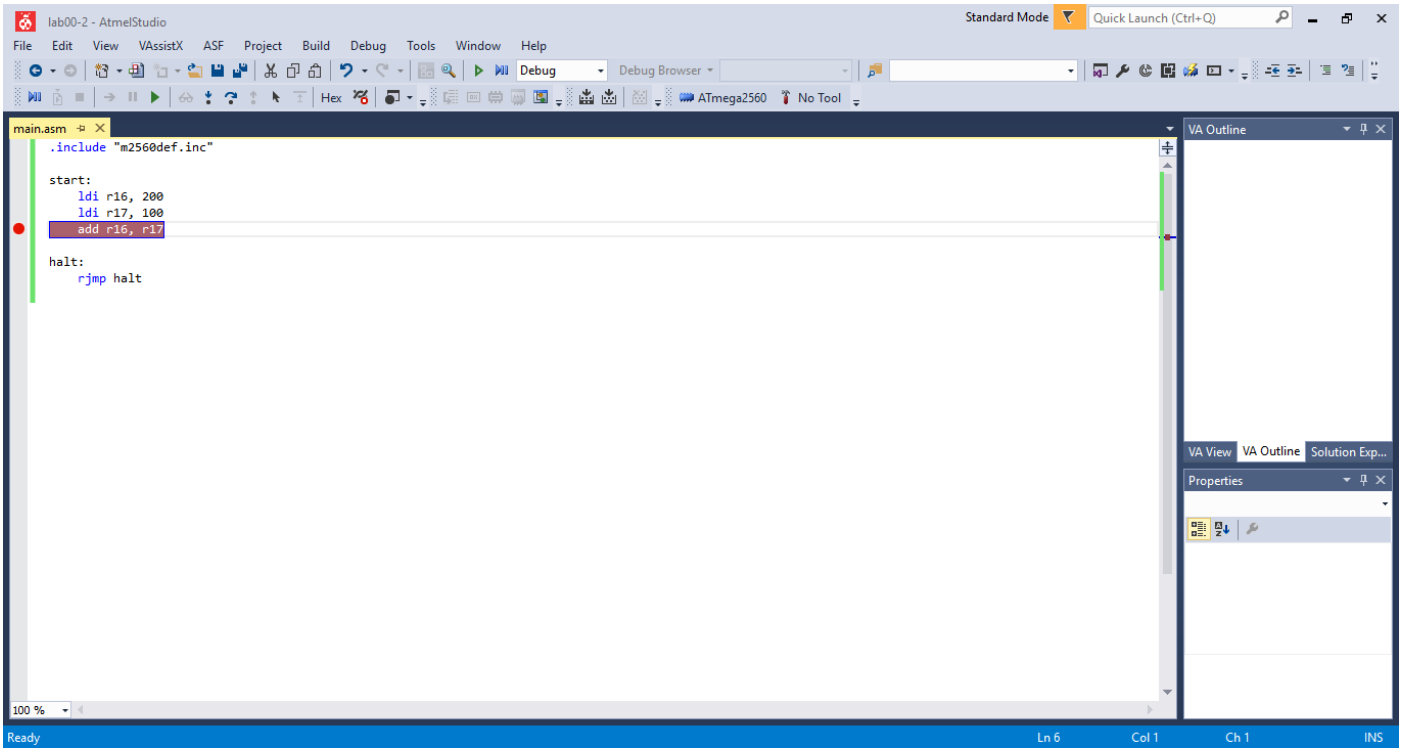


Breakpoints and the Status Register

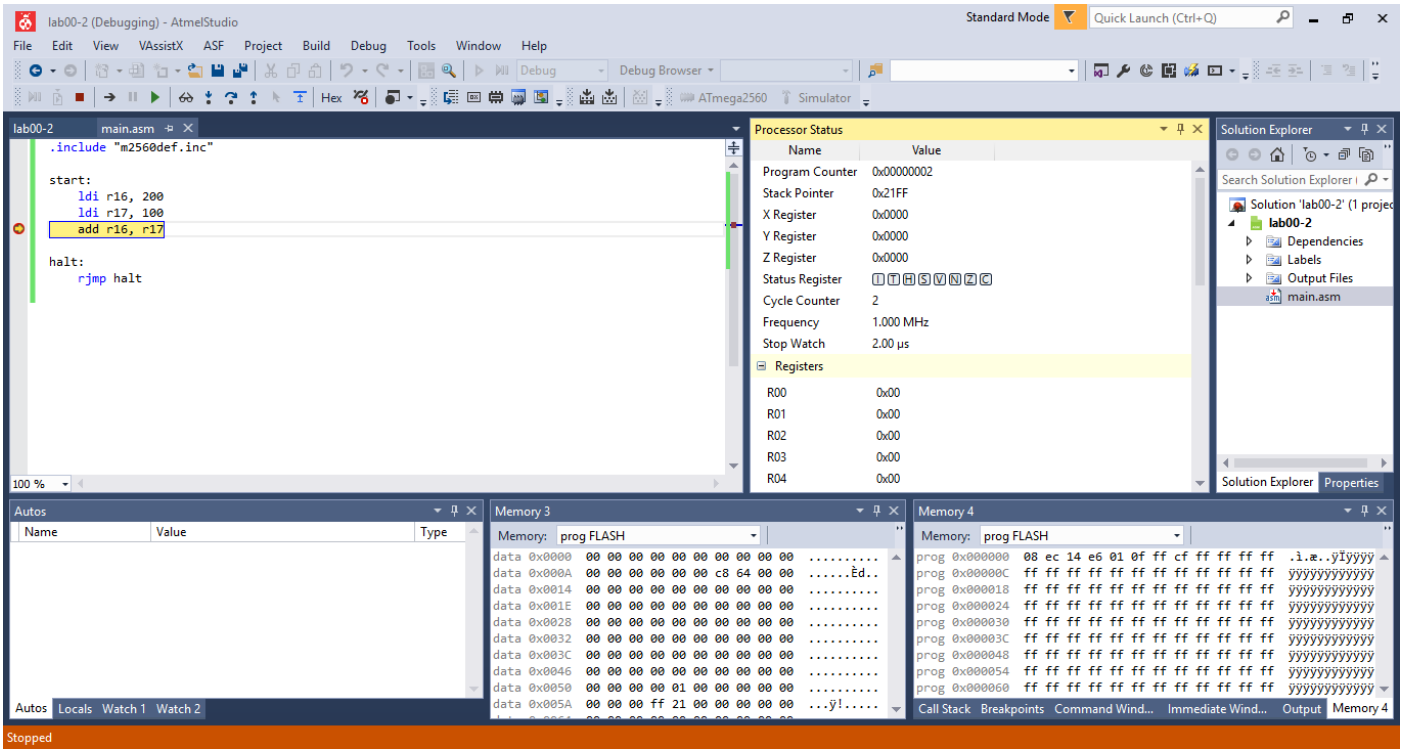
Now clear the code again (or create a new project) and enter the following.

```
.include “m2560def.inc”

start:
        ldi r16, 200
        ldi r17, 100
        add r16, r17

halt:
        rjmp halt
```

Breakpoints allow us to specify instructions that we want the simulator to halt on so we can examine the system before the instruction gets called. We are going to put a breakpoint on the add instruction. Clicking on the grey bar on the left side of the code window will place a red dot denoting a breakpoint.

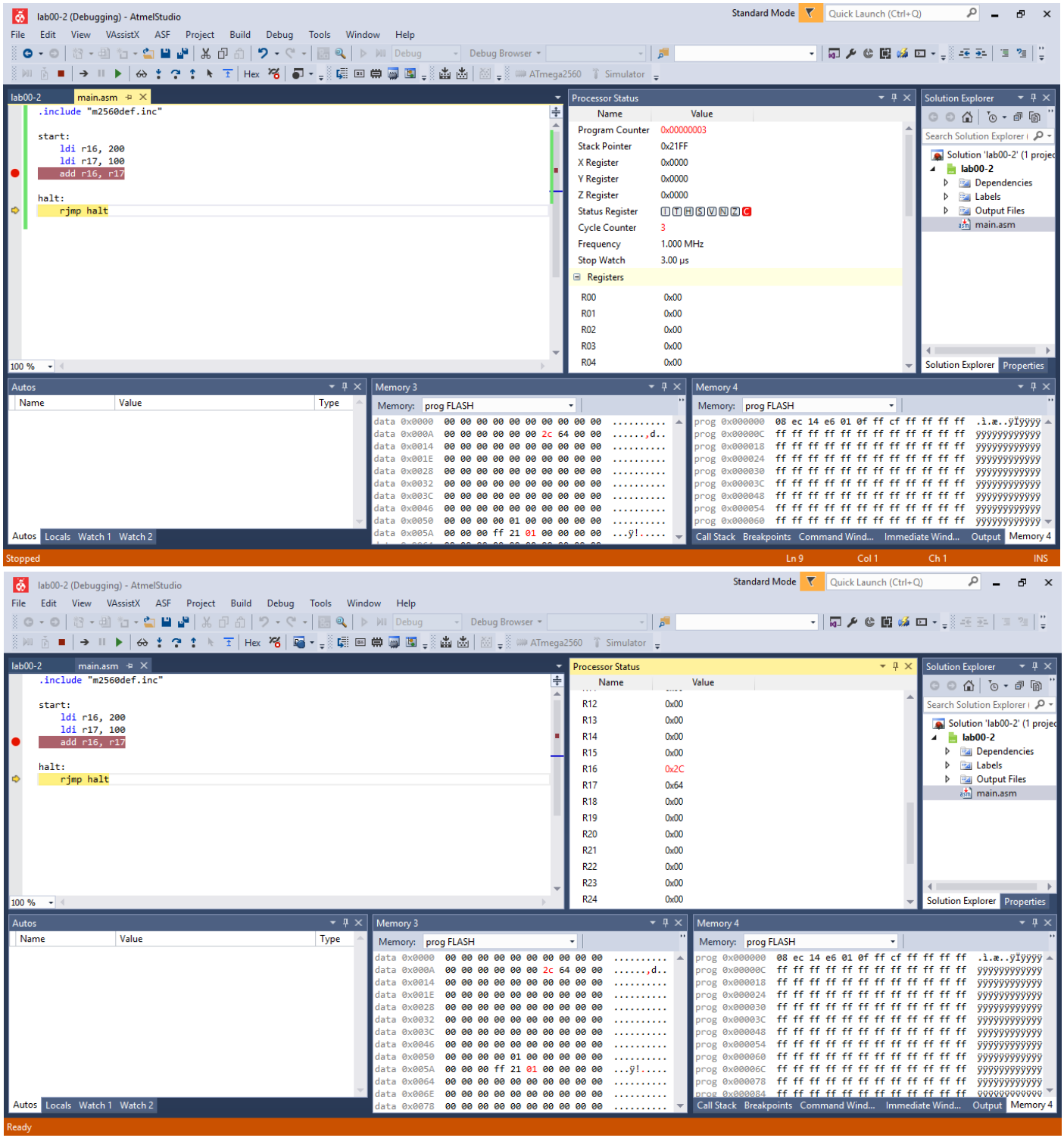


Now press F5 alone (remember that pressing Alt+F5 will cause the simulator to break on the first instruction). The simulator will progress until it hits the breakpoint on the add instruction.



Press F10 once again to execute the add instruction.

This simple program has finished executing and we can see that the previous add instruction has triggered one of the status flags of the register. Scrolling down the processor status window will allow you to see the final values in the register.



As indicated by the raised carry flag, the stored value in R16 has become too large to fit in an 8-bit integer so rather than containing 300 as expected it contains 0x2C (44).