

# Design Assignment 0

CpE 301

Due: Monday, Jan. 30<sup>th</sup>, 2017 by 11:59pm

Spring 2017 (Dr. Harris)

## Introduction

The purpose of this design assignment is to help you become familiar with the programming environment for the Atmega 328p, the microcontroller you will use in this course. This document will guide you in completing the following tasks:

1. Downloading and installing Atmel Studio 7 (also referred to as simply Atmel Studio)
2. Creating an Assembler project in Atmel Studio
3. Writing a sample assembly program
4. Compiling the sample assembly program
5. Simulating and debugging the sample program using Atmel Studio's built-in simulator
6. Creating your own assembler project, compiling, debugging, and simulating it.

Be sure to read the entire document and refer to the **What to Turn in** section at the end of the document.

## Step 1. Download Atmel Studio 7

First, you will download Atmel Studio 7 to your laptop or desktop. If possible, download the program to your laptop so that your system is portable. Go to the following website (see Figure 1) to download Atmel Studio 7, the programming environment (also called integrated development environment: IDE) for Atmega chips:

<http://www.atmel.com/tools/ATMELSTUDIO.aspx>

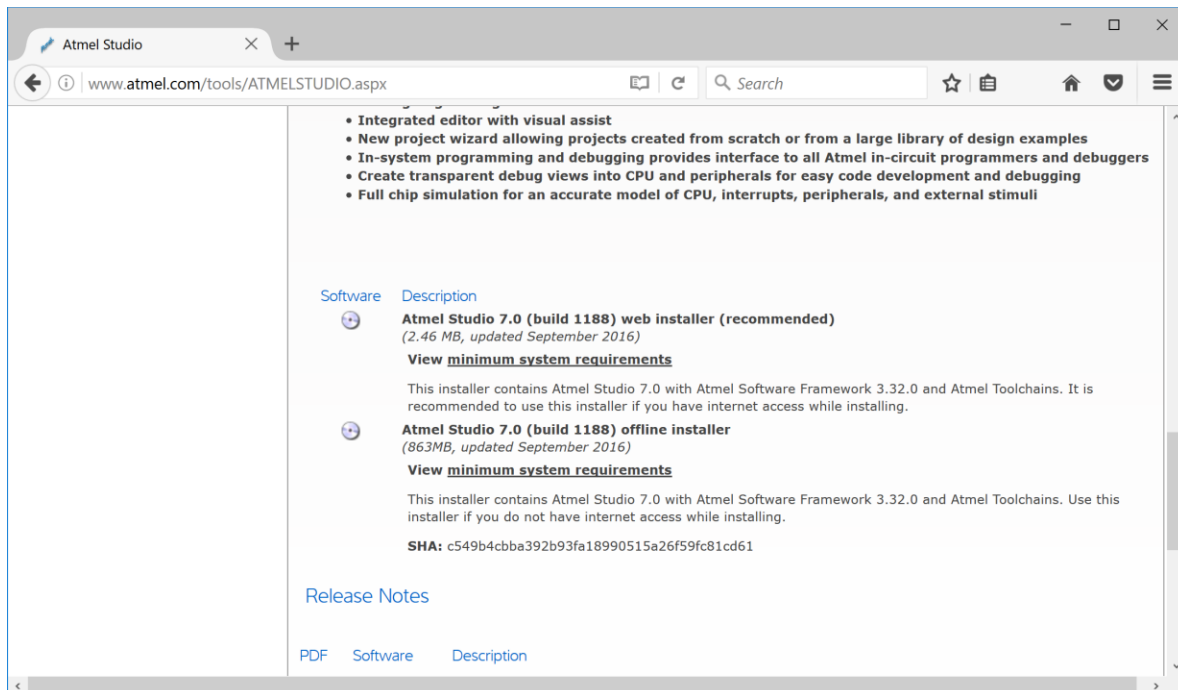
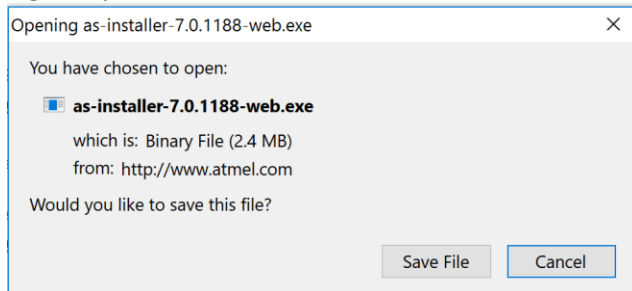


Figure 1. Download Atmel Studio 7.0

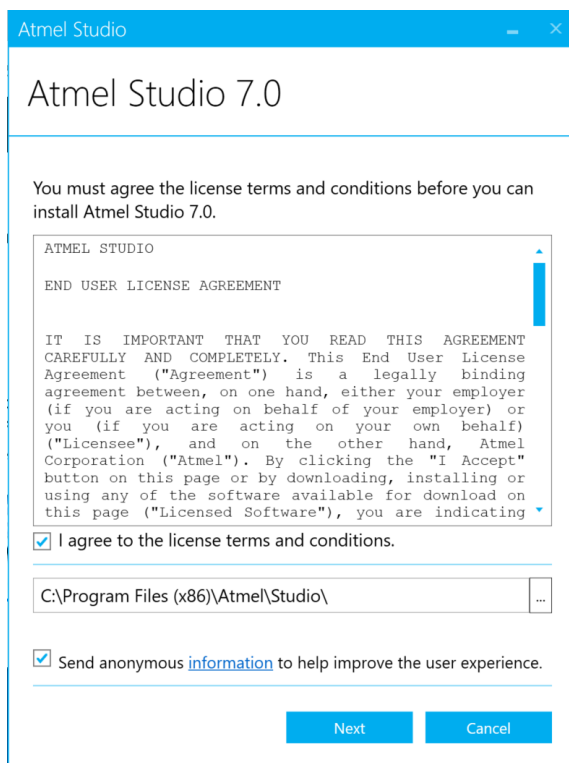
Scroll down to the Software/Description section and click on the disc symbol next to **Atmel Studio 7.0 (build 1188) web installer (recommended)** (see Figure 1). Then click on Save File (see Figure 2).



**Figure 2. Save Atmel Studio 7.0 installer file**

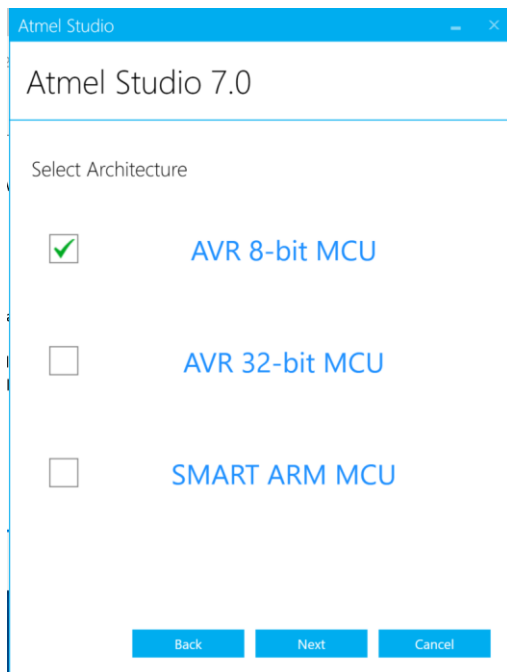
Now run the executable. A window will pop up asking if you're sure, click OK.

A license window will pop up. Click on I agree and note the installation directory (C:\Program Files (x86)\Atmel\Studio\), and click Next (see Figure 3).



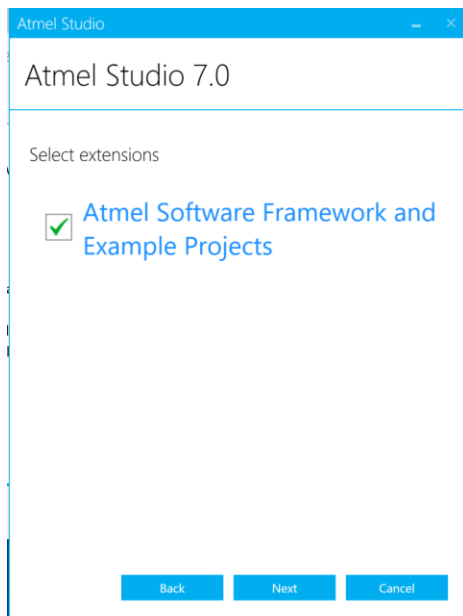
**Figure 3. Atmel Studio 7.0 license agreement**

Now you will be prompted for which architecture you are using (see Figure 4). You only need the 8-bit AVR, but feel free to download the 32-bit and SMART ARM architectures if you'd like. Then click Next.



**Figure 4. Atmel Studio 7.0 supported architectures**

Select the Atmel Software Framework and Example Projects (see Figure 5) and click Next and Next. Then click Install.



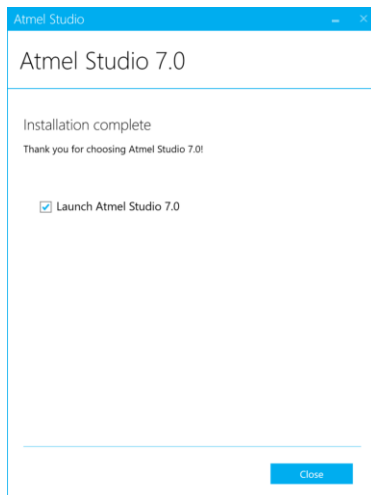
**Figure 5. Select extensions**

When it prompts you whether you want to install Jungo Connectivity Jungo, click Install (see Figure 6).



**Figure 6. Install driver prompt**

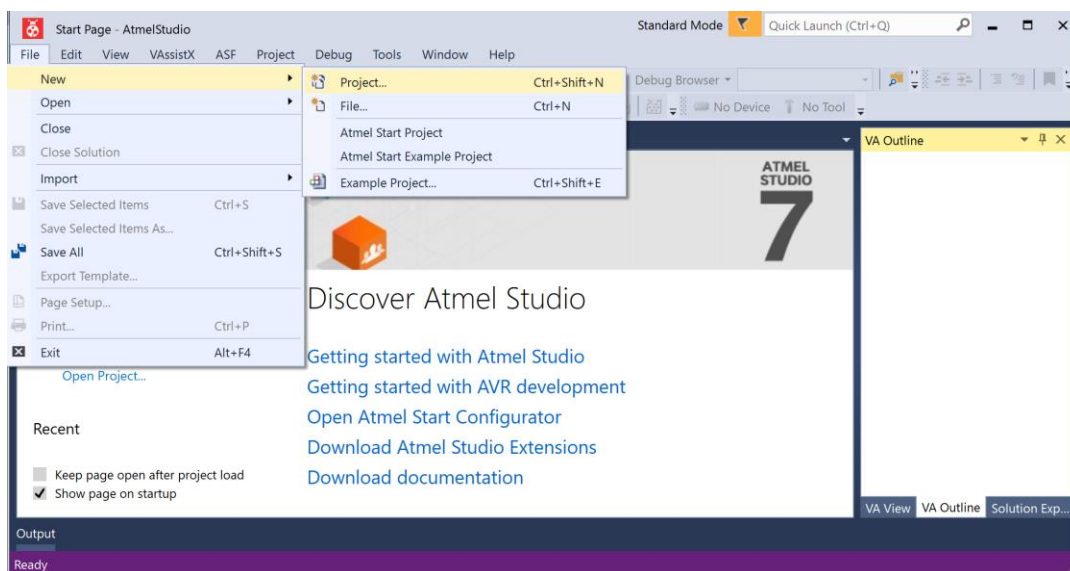
After installation has completed, keep the Launch Atmel Studio 7.0 box selected and click Close (see Figure 7).



**Figure 7. Installation complete and Launch Atmel Studio 7.0**

## Step 2. Create an Assembler project in Atmel Studio 7

After AtmelStudio opens, create a new project by selecting File → New → Project (see Figure 8).



**Figure 8. Create a new project**

Select Assembler project (see Figure 9).

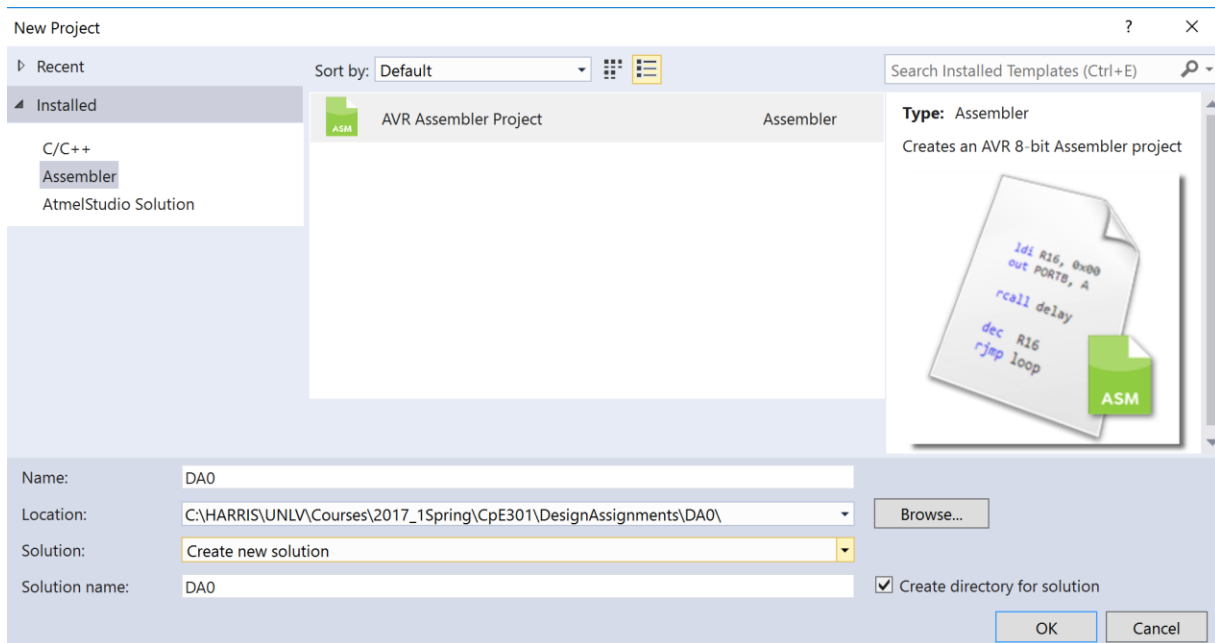


Figure 9. AVR Assembler Project

Select the device to be ATmega328P (see Figure 10). You can type in the name in the search box at the top left, as shown.

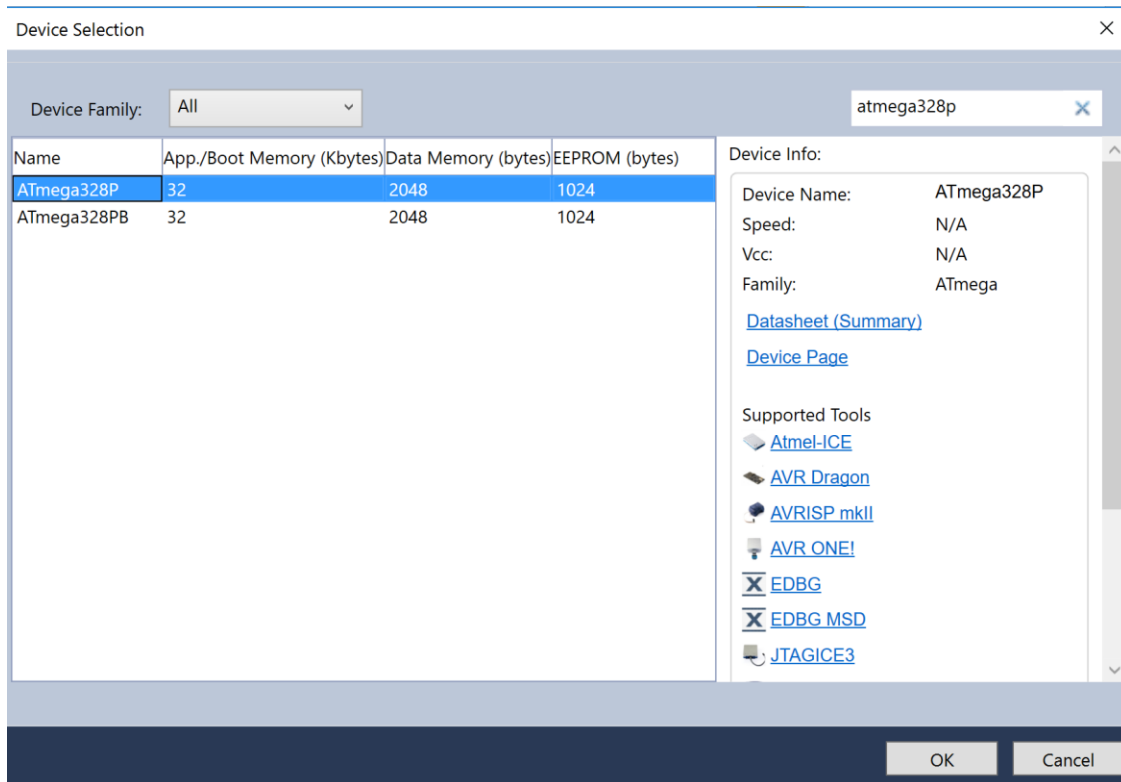
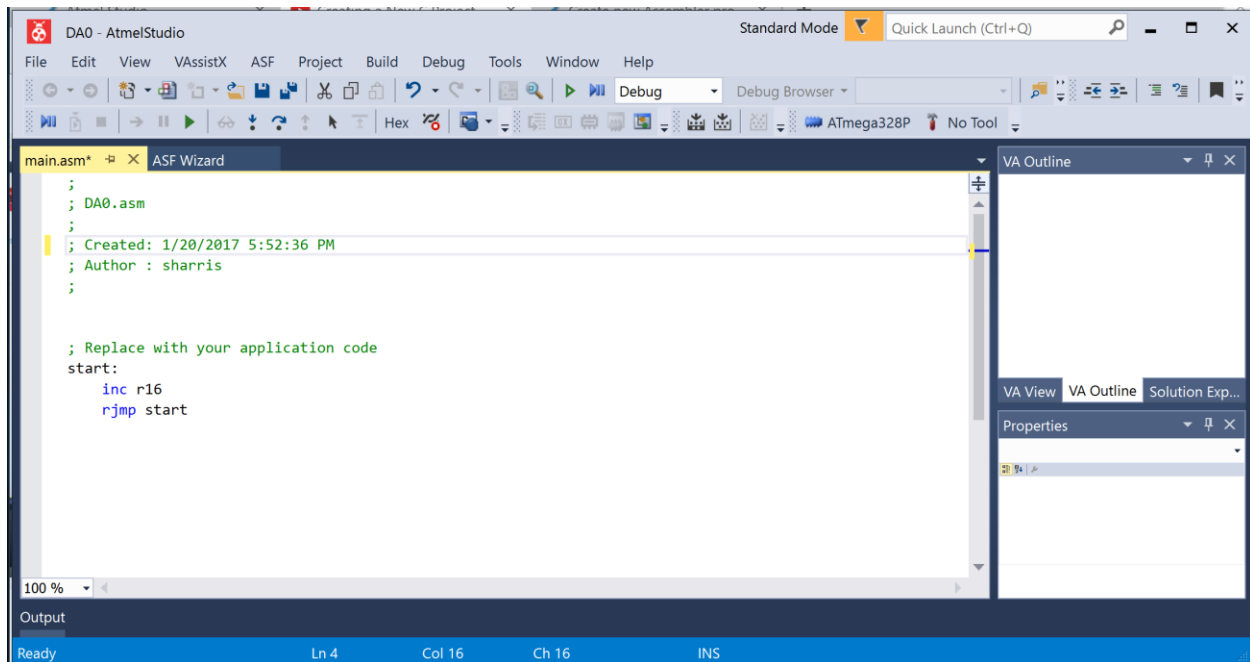


Figure 10. Device Selection: ATmega328p

A default assembly file template (main.asm) will open up (see Figure 11).



**Figure 11. Assembly program template**

### Step 3. Write a simple assembly program

Now you will enter a simple assembly program into your project. Type or copy the assembly program from Figure 12 into your template (i.e., replace all of the code after the comments).

start:

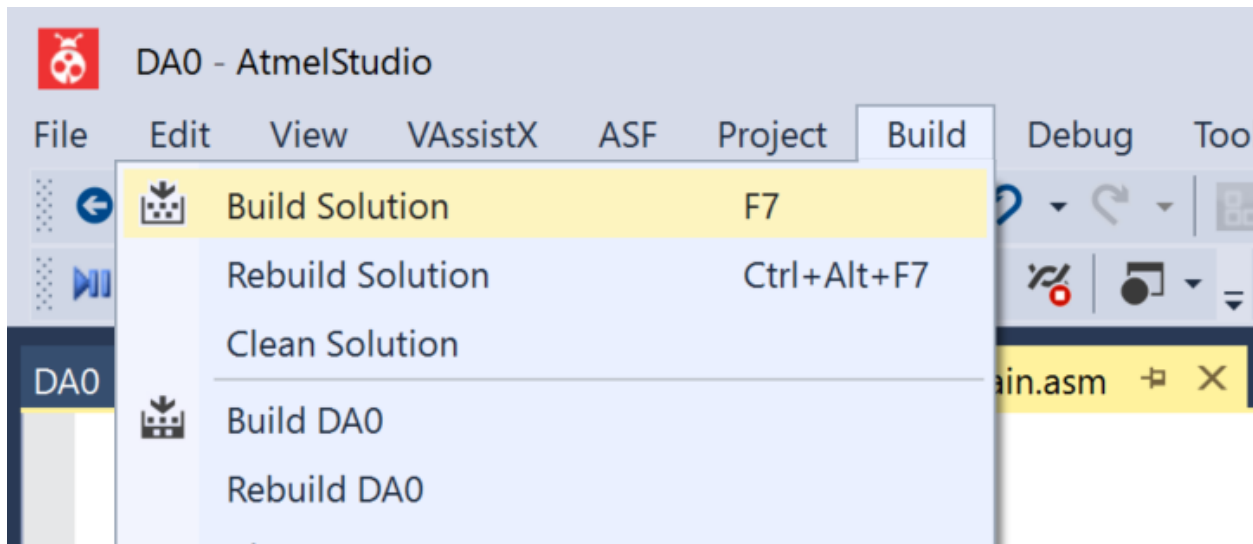
```
LDI r16, 0xF          ; r16 = 0b00001111
OUT DDRB, r16         ; configure PortB bits 3:0 as outputs, bits 7:4 as inputs
LDI r16, 0x4          ; r16 = 0b00000100
OUT PORTB, r16        ; turn PortB2 on
rjmp start            ; repeat
```

**Figure 12. AVR assembly program**

This program makes the upper 4 bits of PORTB inputs and the lower 4 bits outputs and then asserts PortB bit 2 and repeats. Save the file (ctrl-s).

#### Step 4. Compile the assembly program

Now compile the program by selecting Build → Build Solution from the file menu (see Figure 13) or pressing F7. This will compile the program.



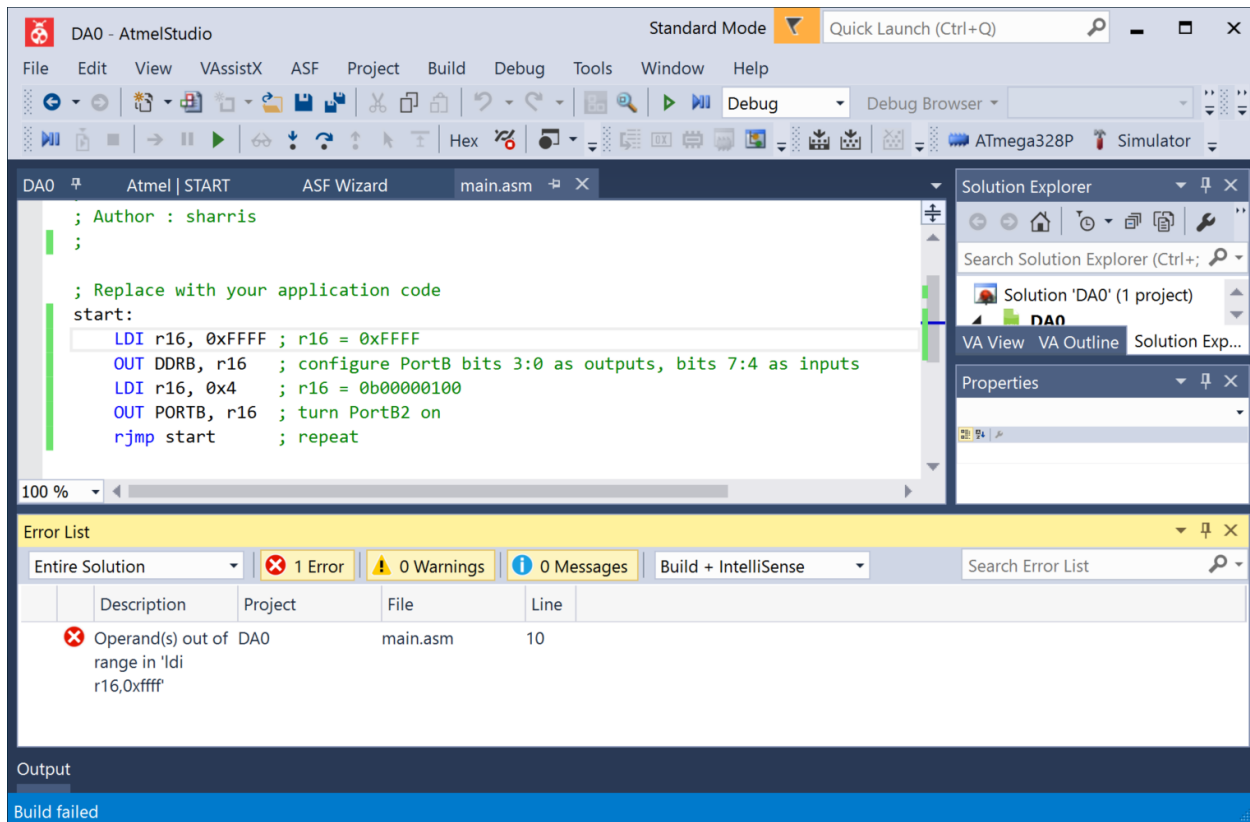
**Figure 13. Build Solution (i.e., compile the program)**

Note the other options: Rebuild Solution and Clean Solution. If it doesn't seem to be compiling/building correctly, you can choose Clean Solution followed by Build Solution to recompile the program.

If there are errors, they will be shown in the window at the bottom of Atmel Studio. For example, replace the first LDI instruction with the following line (see Figure 14):

```
LDI r16, 0xFFFF ; r16 = 0xFFFF - invalid instruction
```

Now build the program again (press F7). As shown in Figure 14, Atmel Studio will report that the Build failed at the bottom of the window and list the errors, in this case that on Line 10 the operand is out of range. Click on the error message in the Error List, and it will bring you to the offending line.

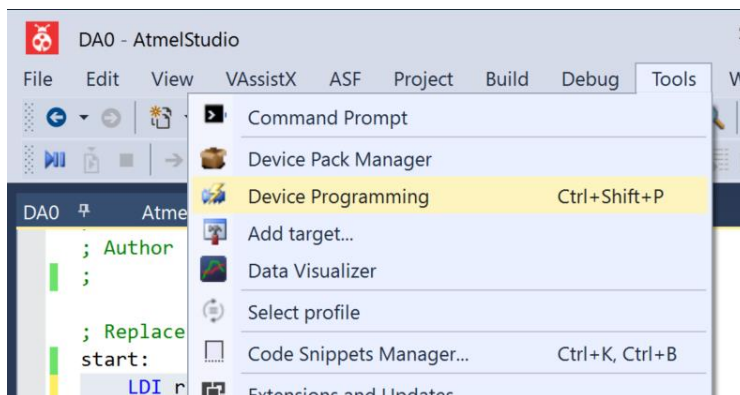


**Figure 14. Building a program with errors**

Replace the line with the original value (`LDI r16, 0xFF ; r16 = 0b00001111`) and rebuild.

### Step 5. Simulate and debug the program

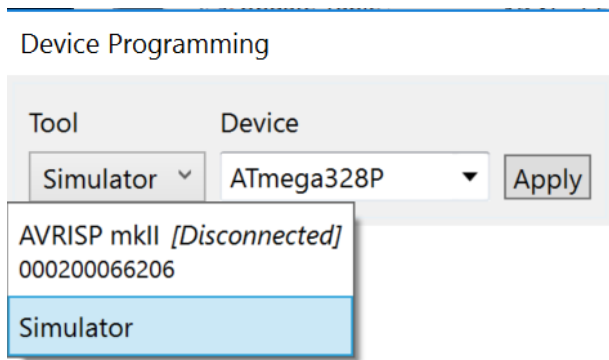
Now you will run the program in the simulator to test and potentially debug it. Click on Tools → Device Programming – or press Ctrl-Shift-P to select the simulator as the device to program (see Figure 15).




**Figure 15. Selecting Device Programmer**

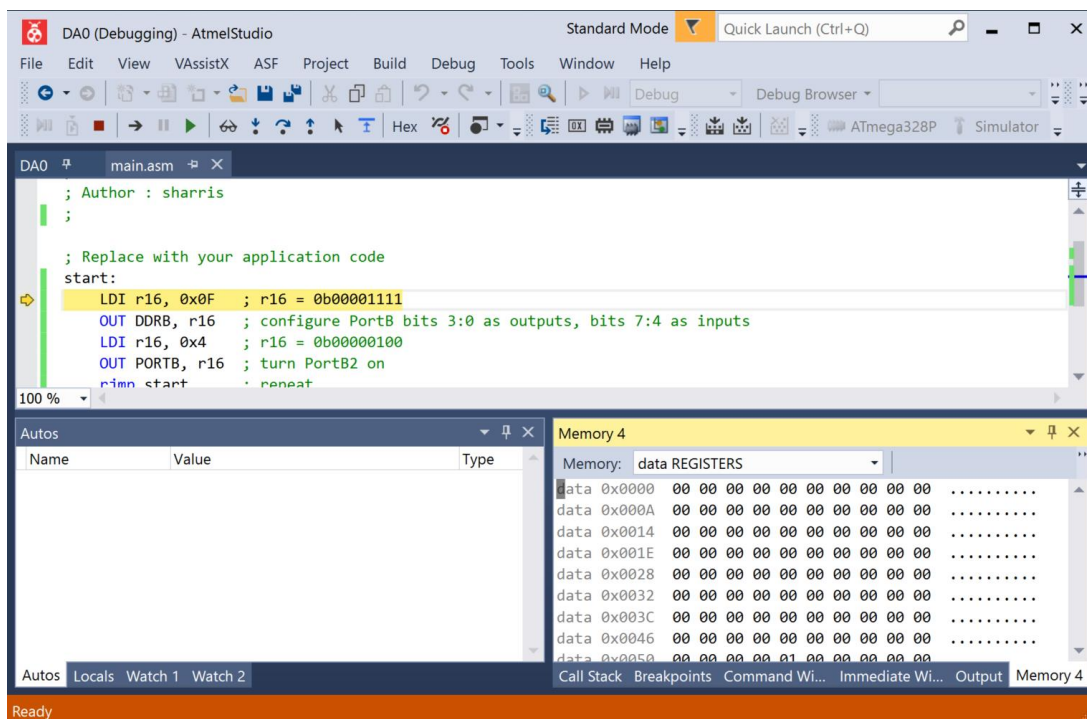
Click on the button under Tool and select **Simulator** (see Figure 16). Then click on Apply and Close.





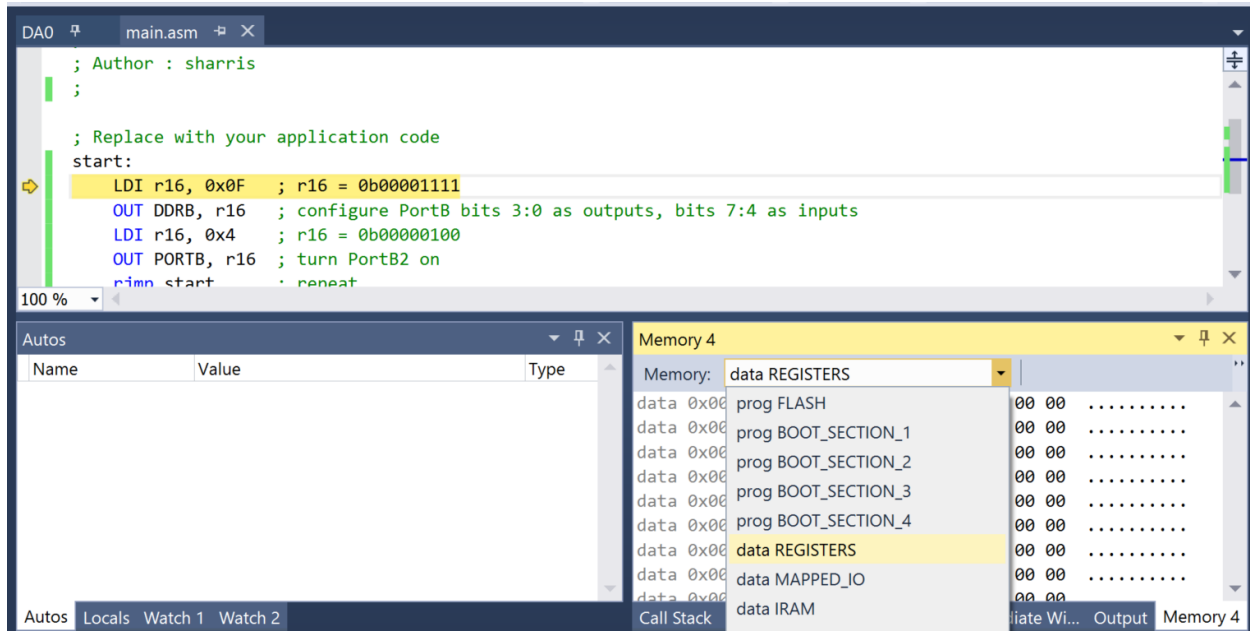
**Figure 16. Select Simulator as the Tool**

Now click on the Start Debugging and Break button:  (or press Alt-F5). Atmel Studio will begin simulating the program and stop at the first instruction, as highlighted in yellow (see Figure 17).



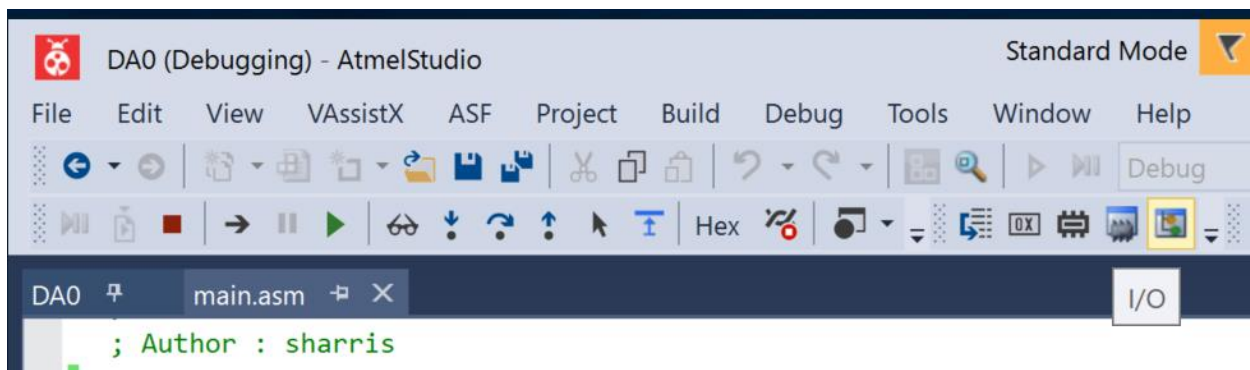
**Figure 17. Begin simulation**

Notice that the lower-right corner shows the contents of various areas of memory. Click on the pull-down menu and select data REGISTERS (see Figure 18). Recall that the 32 data registers are mapped to memory addresses 0-31 (0x00 – 0x1F). Data registers 0-9 are listed from left-to-right in the top row, registers 10-19 (0xA-0x13) are listed from left-to-right in the next row, etc.

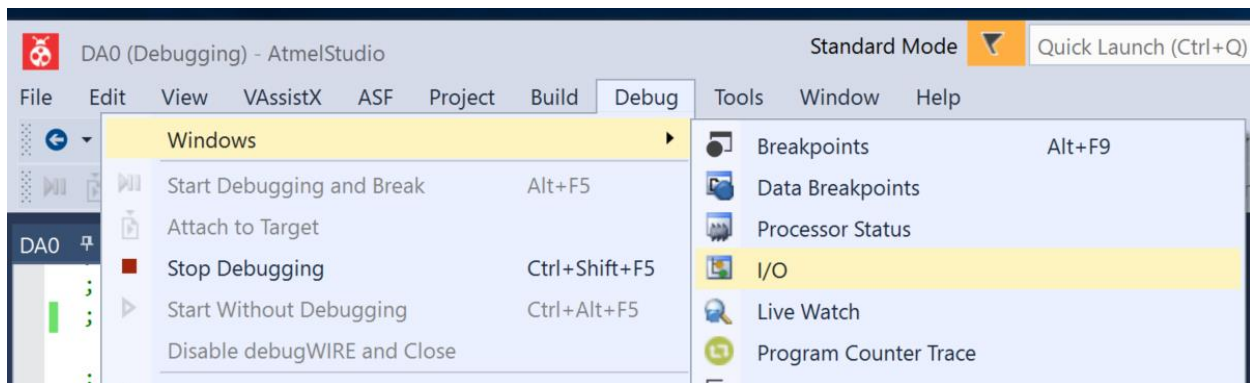


**Figure 18. View data REGISTERS**

Now view some of the I/O registers, particularly the registers associated with PORTB. Do so by either clicking the I/O button from the toolbar (Figure 19) or selecting it from the menu (Figure 20).



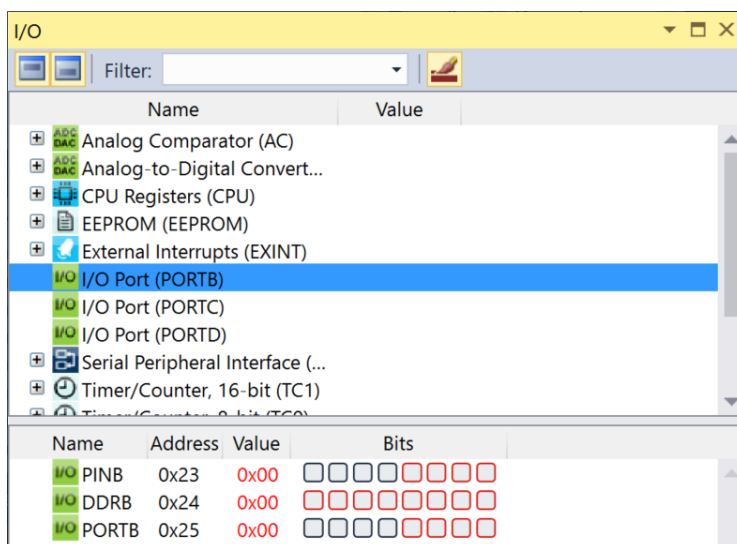
**Figure 19. View I/O registers – I/O button**





**Figure 20. View I/O registers – from menu**

An I/O window will pop up. Highlight I/O Port (PORTB), as shown in Figure 21. The 3 registers associated with that port are listed:

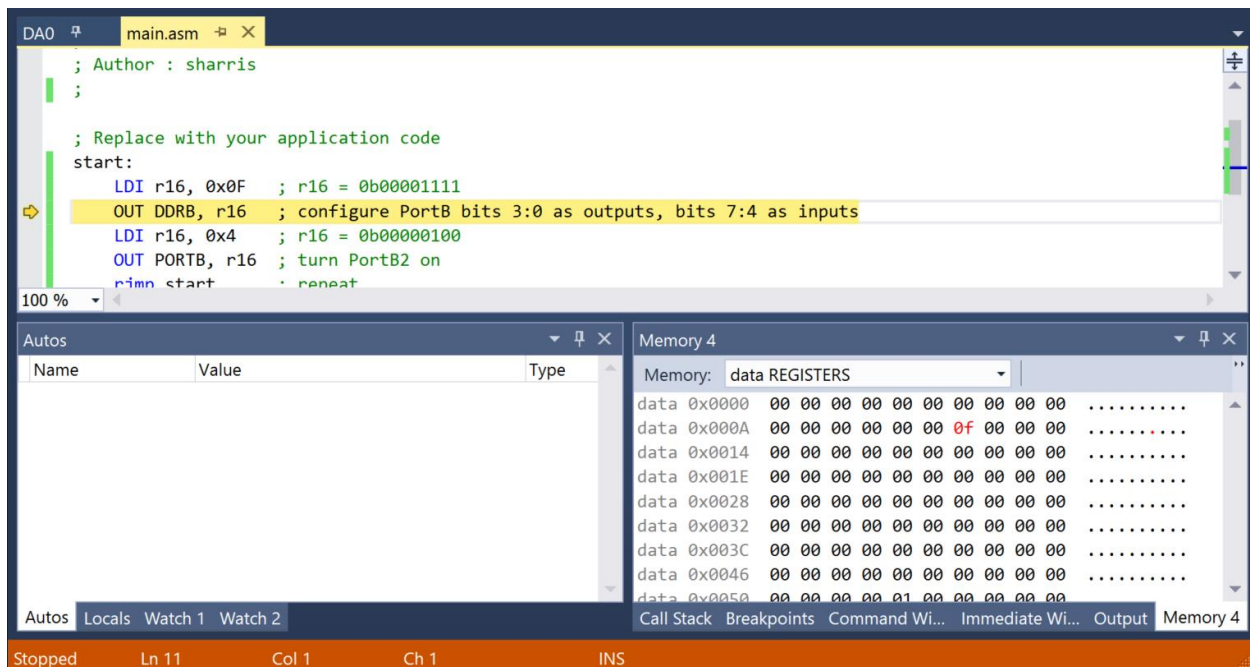
- **PINB:** values read from PORTB
- **DDRB:** direction of PORTB bits (1 = output, 0 = input)
- **PORTB:** value written to PORTB



**Figure 21. View values of PORTB registers**

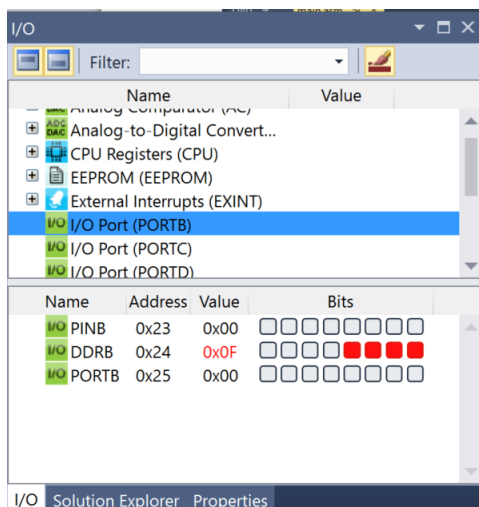
Now, simulate the code by pressing the Step Into (  or press F11) or Step Over (  or press F10) button. Because there are no function calls in this code, both options have the same effect: execute one instruction and stop at the next instruction. (If the simulator were at a line of code that called a function, the Step Over option would run the function and stop at the next instruction after the function return; the Step Into button on the other hand would enter the called function.)

The simulator executes the LDI r16, 0x0F instruction and the cursor moves to the next instruction (see Figure 22). Notice that r16 (data register 16 (0x10) in the Memory pane) has been updated to 0x0F, as expected.



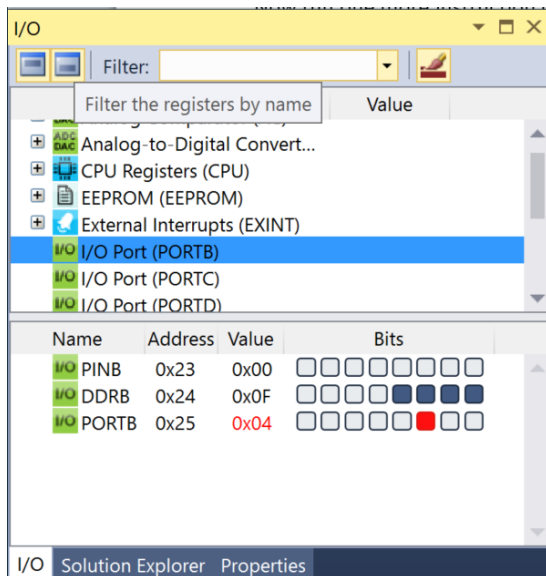
**Figure 22. After Step Into/Step Over command**

Now run one more instruction (F10 or F11). The `OUT DDRB, r16` instruction executes and writes the value in `r16` (`0x0F`) to the memory-mapped register `DDRB` (see Figure 23).



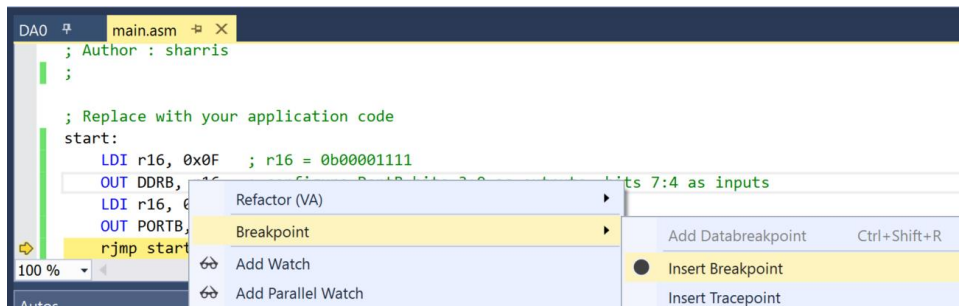
**Figure 23. DDRB register is written with 0x0F**

Press F10 or F11 again to execute the next instruction (`LDI r16, 0x4`). Notice `r16` (i.e., memory address `0x10`) update to `0x04` in the Memory pane. Execute one more instruction (`OUT PORTB, r16`) to see bit 2 of `PORTB` written to a 1 (see Figure 24). An LED connected to `PORTB` bit 2 (`PB2`) would now light up.



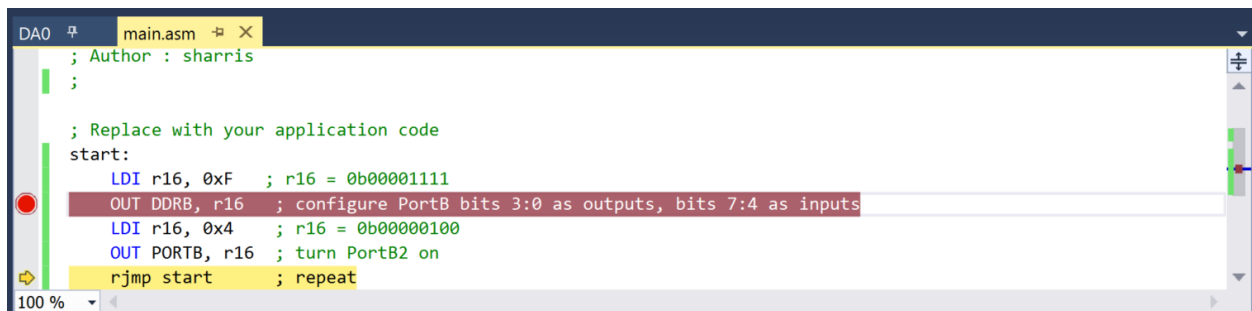
**Figure 24. PortB bit 2 (PB2) asserted**

You can also add breakpoints to the simulation by right-clicking on any instruction and selecting Breakpoint → Insert Breakpoint (see Figure 25). Notice that you can also delete breakpoints using the same process.



**Figure 25. Inserting a breakpoint**

The line with the breakpoint will then be highlighted as shown in Figure 26.



**Figure 26. Breakpoint added**



You can then press the Continue button: and execution will continue until the next breakpoint.

### Step 6. Create your own assembler project and compile, debug, and simulate it

Now you will create your own assembly program and test it in simulation. Do the following:

1. Create a new project called DA0\_1.
2. Write code that adds five numbers of your choosing that are each greater than 30 and less than 60. If the sum produces an overflow set PORTB pin 4 (PB4) HIGH. Otherwise PB4 should be low.
3. Build the program and fix any bugs.
4. Run the program in simulation. View the registers and PORTB registers to determine that it operates as you expect.
5. Run the program in simulation for the cases where (1) overflow occurs and (2) overflow does not occur.
6. Determine the execution time of your program. You can determine the number of clock cycles of your algorithm using the simulation. For your calculation use a clock speed of 8 MHz.

### What to Turn In

The following must be submitted via WebCampus by the due date/time to receive credit for the assignment. Messy, difficult to understand, or disorganized work will receive no credit.

#### Total points available: 100

0. **Time:** Indicate the amount of time this assignment took in hours. This will not affect your grade (unless omitted) but will help gauge the workload for this and future semesters. **[-5 points if omitted]**
1. **A pdf or word document** that contains **(in this order)**:
  - a. A 1-paragraph description of your design. **[10 pts]**
  - b. The assembly code. The AVR assembly code must have been built (i.e., compiled/assembled) and working. The assembly code should be well-documented with a comment for each instruction. **[35 pts]**
  - c. Screenshots of Atmel Studio during debugging at the beginning and end of Step 6.5 for both cases (first overflow, then no overflow). Be sure to also show the register values as well as the PORTB register values. **[30 pts]**
  - d. A link to a 1-2 minute Youtube video showing your program working in simulation in Atmel Studio. **[15 pts]**
  - e. The execution time of your program – show your work and put a box around your final answer. **[10 pts]**