COMP9032 Lab 0

Getting Started

July, 2018

Before labs start fr[illegible] recommended that you install AVR Studio 7 on your computer at home. [illegible]re to develop and simulate AVR programs. The software is availabl[illegible]e (follow the link References → AVR Studio).

1. Objective

- Learn how [illegible]ebug and run an AVR assembly program.

2. Introduction to AVR Studio

2.1 Start AVR Studio

To start AVR Studio, double click the AVR Studio icon, or click on Start → Programs → Atmel Studio 7.0 → Atmel Studio 7.0.

2.2 Create a New Project

To create a new project, on the Start Page screen, click the “New Project …” link. The dialog box will appear.

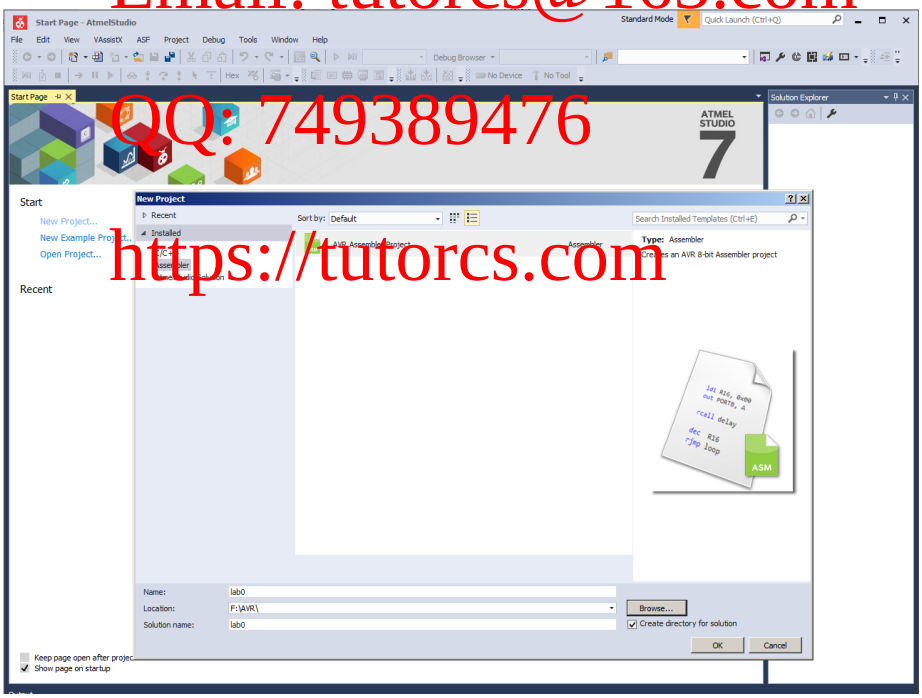

Figure 1: New Project

Among the installed programs, select Assembler and enter a project name (e.g. “*lab0*”), as shown in Figure 1. Next, select the project location. This is the location where AVR Studio will store all files associated with the project. It is a good practice to create a separate directory for each lab. In the laboratory, you may need to create your working directory on the Desktop as you don't

have write permission anywhere else. After choosing the project software type, name and location, press the "OK" button to continue. Then you will be asked to choose the device for your project. Choose "ATmega2560" from the device list, as shown in Figure 2. Then press "OK" to continue.

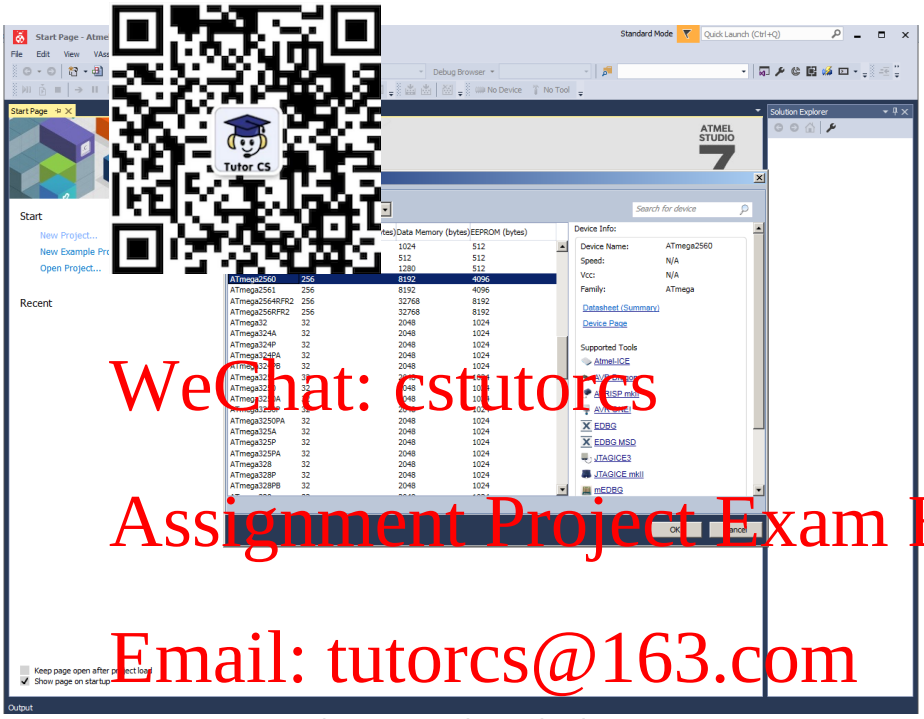

Figure 2: Device Selection

Now the project window with a file called main.asm appears, as shown in Figure 3.

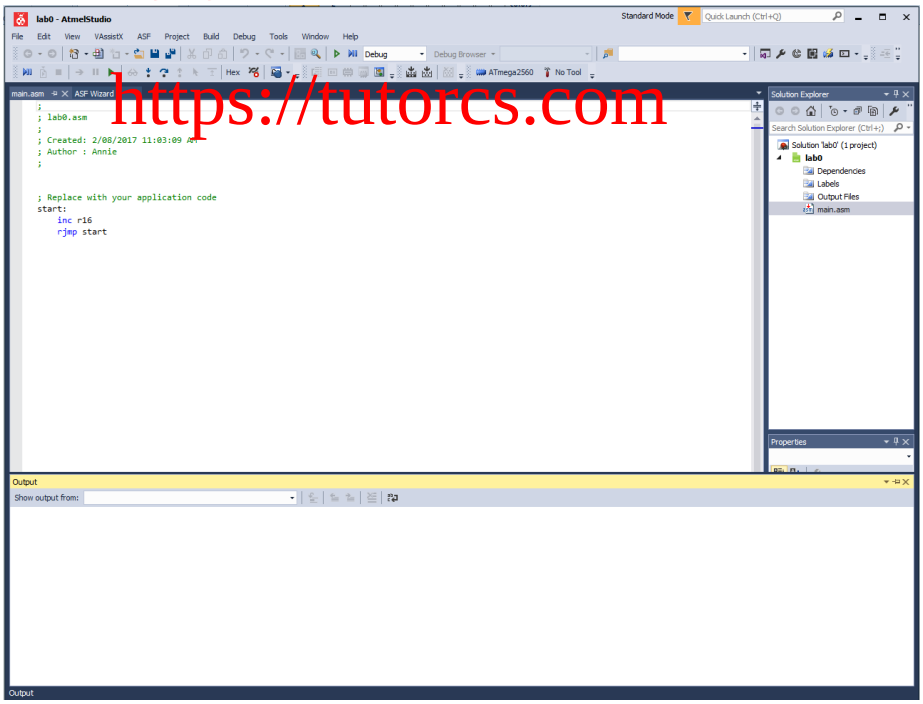

Figure 3: Project Window

Replace the sample code (that was automatically generated by the studio) with your code given in Figure 4.


```
.include "m2560def.inc"
.def a =r16                ; define a to be register r16
                           fine b to be register r17
                           fine c to be register r10
                           fine d to be register r11
                           fine e to be register r12

                           ain is a label
                           ad  value 10 into a

                           py the value of a to c
                           d c and b and store the result in c

sub d, b                   ; subtract b from d and store the result in d
lsl c
asr d
mov e, c
add e, d
halt
rjmp halt                  ; halt the processor execution
```


Figure 4: Your assembly code

In the assembly code, the file m2560def.inc is included. It contains the definitions required by the assembler for the microcontroller. The next step is to build the code (i.e. assemble the assembly program into machine instructions). This is done by selecting “Build Solution” from the “Build” menu (or press F7). The Output window then displays the information from the assembler. From this window we can see that the assembly process was completed with no errors and the executable file is 22 bytes long.

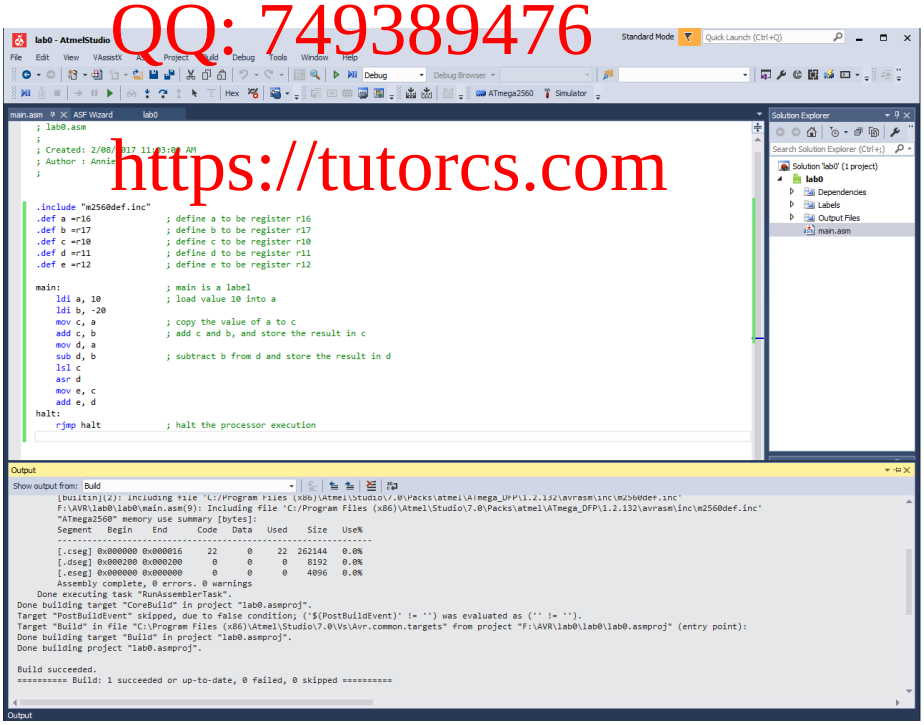

Figure 5: Build Your Program

Now we can investigate the execution of the code in the debug mode by selecting “Start Debugging and Break” under the Debug menu. Notice that a yellow arrow is pointing to the first instruction “ldi a, 10” in the assembly code window, as shown in Figure 6. This arrow indicates the position of the next instruction to be executed. Before simulation, we may want to set up the Register View so that we can see the value of each register during the program execution. Select Debug→Windows→ and drag the created window to the side (see Figure 6).

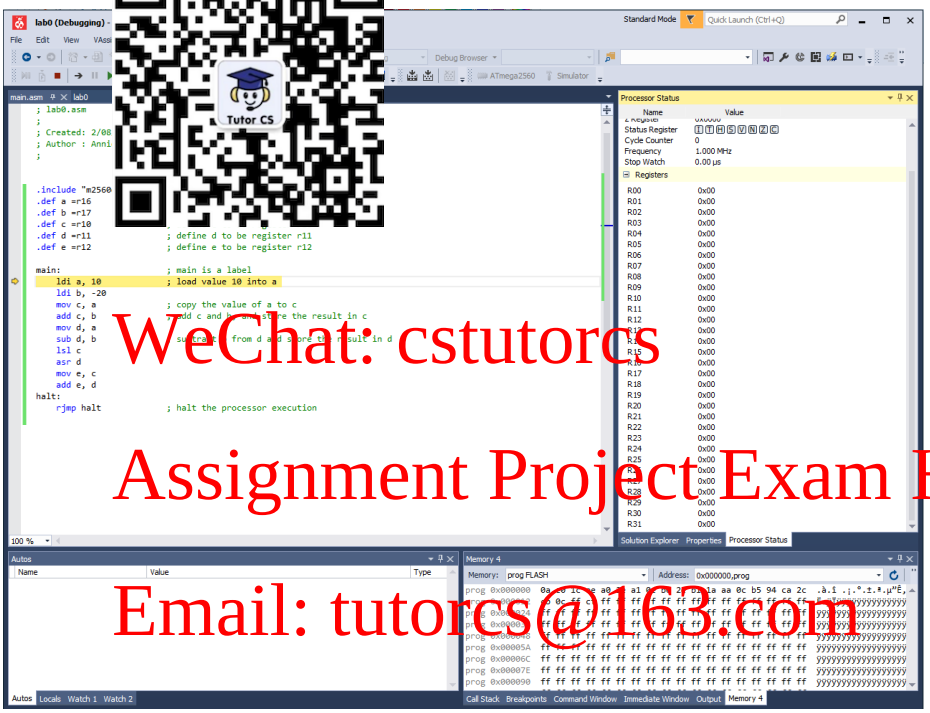

Figure 6: Start Debugging

The value of each register is initially 0. These registers will be dynamically updated during the program execution. You can also assign and change values of these registers by clicking on the value of a register and entering the appropriate value at any time during the simulation. You can also change the value display format by right clicking on the value and selecting the format you want to use.

2.3 Run Simulation

There are commands to single step through the code: “Step Over” F10 and “Step Into” F11. The difference between these commands is that F10 does not trace into subroutines. Since our example does not contain any subroutines, there is no difference between the two here. Now, single-step down to the fifth line of the code by repeatedly pressing the F11 key. Notice how the color changes from black to red on the registers when their values are updated. This makes it easy to identify which register changes its value when an instruction is executed.

Figure 7: Single-Step Execution

You can also run the code with breakpoints. To add a breakpoint, move the cursor to the instruction you want to stop on, and right click mouse and select Breakpoint→Insert Breakpoint. Figure 8 shows a breakpoint (indicated by the red dot) is added on the *asr* instruction.

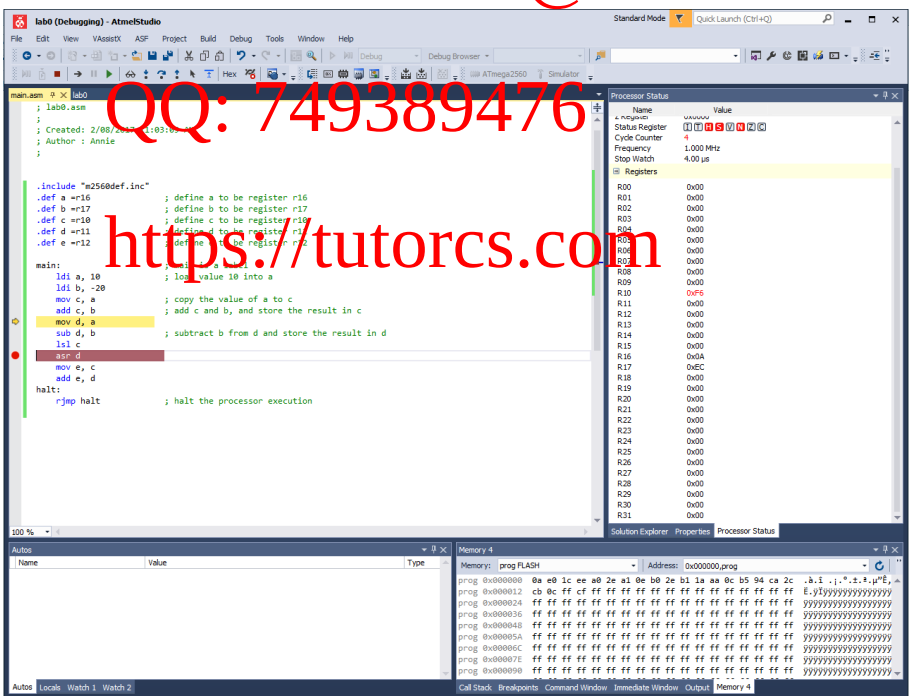

Figure 8: Setting Breakpoint

Now press the Continue (F5) button. The simulation will continue and then stop at the breakpoint, as shown in Figure 9. The breakpoint can be removed/disabled by selecting Breakpoint → Deleting Breakpoint/Disable Breakpoint.

Figure 9: Execution with Breakpoint

To end the debugging session, press “Ctrl+Shift+F5” or selecting Debug → Stop Debugging from the menu.

2.4 Status Register

Each AVR microcontroller has a Status REGister (named as SREG) that keeps 8 flags such as C, S and V. The definitions of all 8 flags in SREG can be found in Mega2560 Data Sheet (page 14), which is available on the course website (follow the link References → Documents → Mega2560 Data Sheet). These flags are dynamically updated during the program execution, and can also be observed in the Processor Status window. When a flag bit is set, the corresponding bit block is highlighted in red. For example, after the instruction “add c, b” is finished, bit 2 (N), bit 4 (S), and bit 5 (H) are set, as shown in Figure 8.

2.5 Disassembly

AVR Studio provides a disassembler which lists the assembly details. Using the disassembler, you can see the corresponding machine code and the memory address of each instruction. To run the disassembler, start debugging, and select Debug→Windows→Disassembly. You will see a listing of your disassembled program, as shown in Figure 10. For each instruction its memory address (in the first column), machine code (the second column, in the hex format), assembly code (the third column), and its operation (the last column) are given.

Figure 10: Disassemble Code

Note that you don't need to attend laboratory for this lab. However, you are strongly recommended to complete it at home. If you have any questions, feel free to ask the lab tutor in week 2.