COMP9032 Lab 0

Getting Started

July, 2017

Before labs start from week 2, it is highly recommended that you install AVR Studio 4 on your computer at home. We will use the software to develop and simulate AVR programs. The software is available on the course website (follow the link References →Tools→ AVR Studio).

1. Objective

- Learn how to use AVR studio to debug 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 AVR Tools → AVR Studio 4.

2.2 Create a New Project

To create a new project, go to the "Project" menu and select "New Project", or on the Welcome screen, click the "New Project" button. The dialog box shown in Figure 1 will appear.

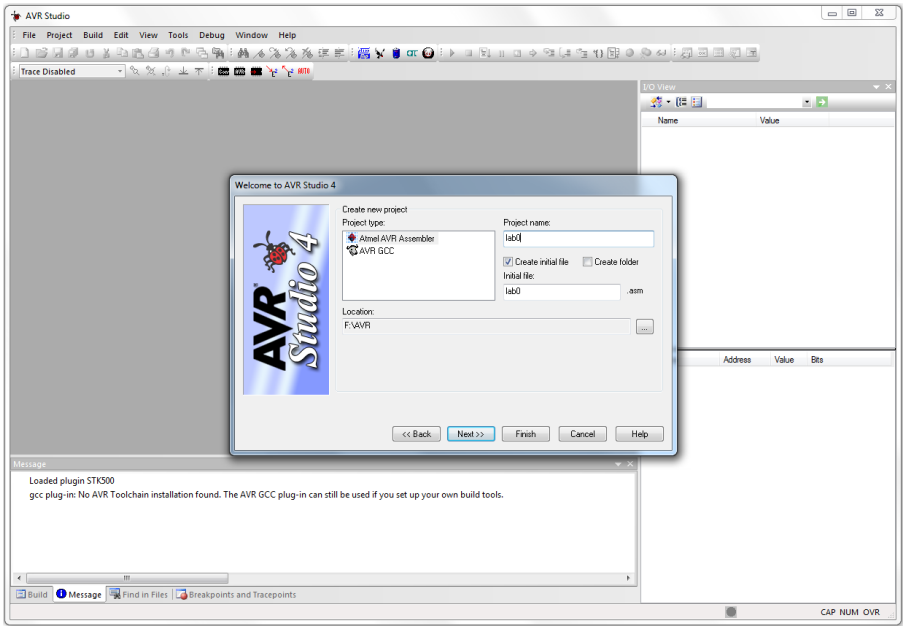

Figure 1: New Project

There are two types of projects: Atmel AVR Assembler and AVR GCC. Choose Atmel AVR Assembler and enter a project name (e.g. "*lab0"*). 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 need to create your working directory on the Desktop as you don't have write permission anywhere else. After choosing the project type, name and location, press the "Next" button to continue. Then you will be asked to

choose the “Debug Platform” and “Device” from the list as shown in Figure 2. Choose “AVR Simulator 2” as the “Debug Platform” and “ATmega2560” from the Device list. Then press “Finish” to continue.

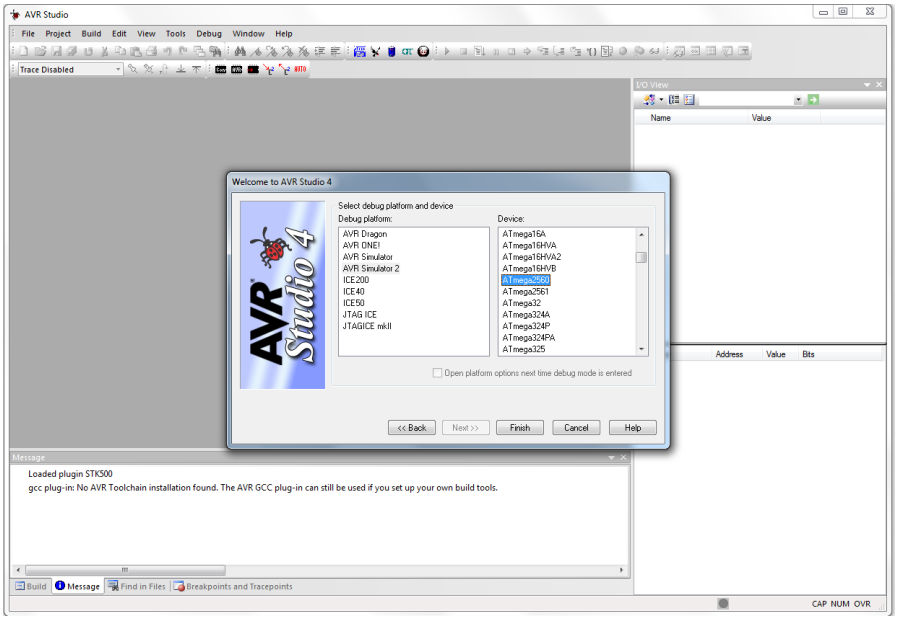

Figure 2: Debug Platform & Device

The Project Manager will now appear and looks like Figure 3. In this view, you will see files associated with your project. At the moment, there is one file in this project called *lab0.asm*. This file is automatically marked as “Assembly Entry File” (with a red arrow on the file icon), which indicates that the assembler will start with this file when assembling the project. Each project can only have one entry file. When you have more than one file associated with the project, you will have to manually mark the file you want to debug and run as the “Assembly Entry File” by right clicking on the file and choosing “Set as Entry File”.

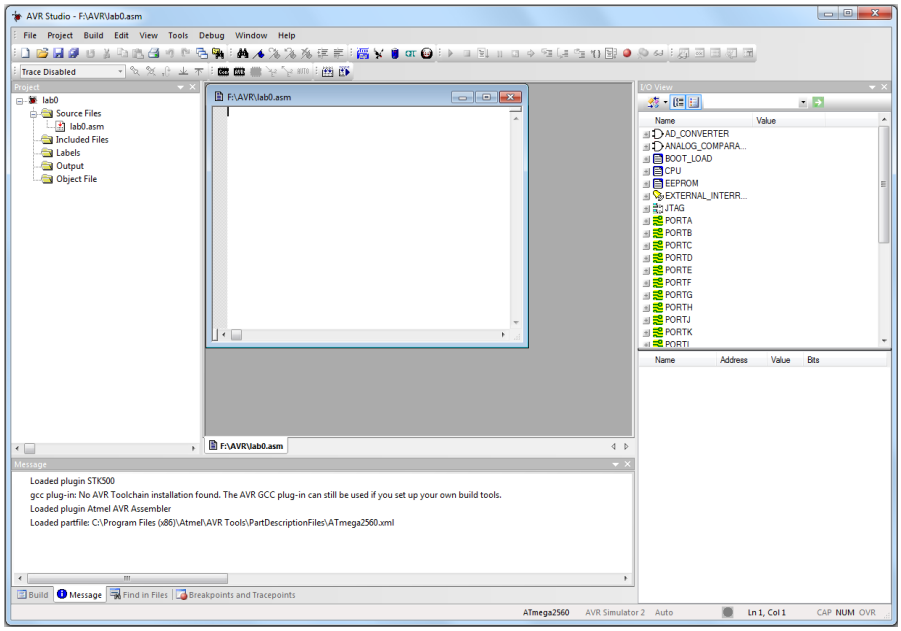

Figure 3: Project Manager

We have now an empty file in our project, called lab0.asm. The next step is to fill this file with our code. Figure 4 and Figure 5 show, respectively, the C program and its hand-compiled AVR assembly code. Here we assume that each integer variable is ONE byte. The assembly program assigns registers r16, r17, r10, r11, r12 to the C variables *a, b, c, d, e*, respectively. Note that the function of pseudo instruction "define" is similar to that in C.

```
# include <stdio.h>

int a = 10,
    b = -20,
    c, d, e;

int main(void)
{
        c = a + b;
        d = a - b;
        e = c*2 + d/2;
        return 0;
}
```

Figure 4: Program lab0.c

```
.include "m2560def.inc"
.def a =r16             ; define a to be register r16
.def b =r17             ; define b to be register r17
.def c =r10             ; define c to be register r10
.def d =r11             ; define d to be register r11
.def e =r12             ; define e to be register r22

main:                   ; main is a label
        ldi a, 10       ; load  value 10 into a
        ldi b, -20
        mov c, a        ; copy the value of a to c
        add c, b        ; add c and b and store the result in c
        mov d, a
        sub d, b        ; subtract b from d and store the result in d
        lsl c           ; refer to AVR Instruction Set for the semantics of
        asr d           ; lsl and asr
        mov e, c
        add e, d
halt:
        rjmp halt       ; jump to halt
```

Figure 5: Program lab0.asm

In lab0.asm, the file m2560def.inc is included. It contains the definitions required by the assembler for the microcontroller. The next step is to build (i.e. assemble the assembly program into machine instructions) and run/debug the program. This is done by selecting "Build and Run" from the "Build" menu (or press Ctrl+F7). The Build window then appears, as illustrated in Figure 6, which shows 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.

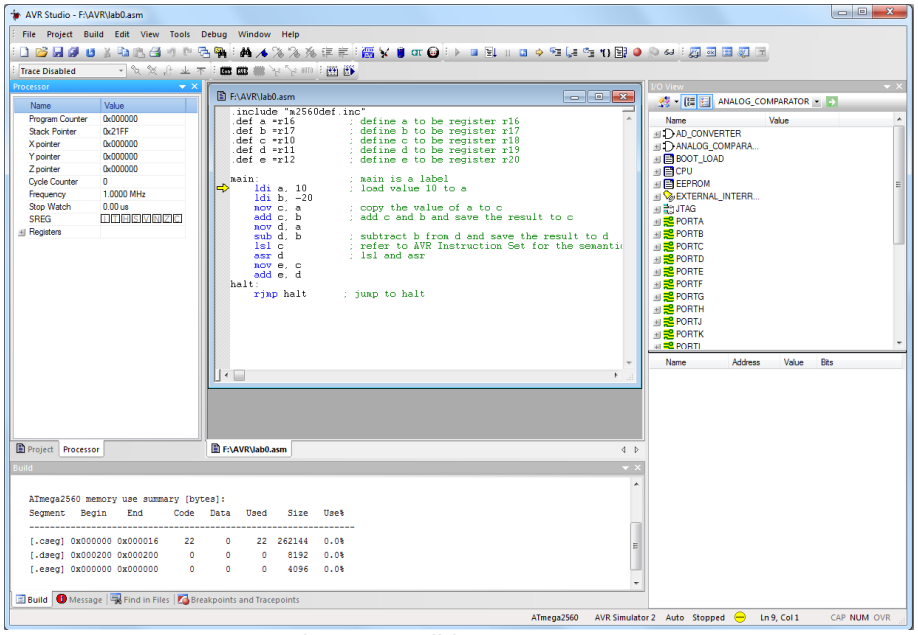

Figure 6: Build Your Program

We are now ready to run the code in the debug mode. Notice that a yellow arrow is pointing to the first instruction “ldi a, 10” in the Editor window. 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. Double click on “Registers” in the Processor window. You will see a map of all registers, as demonstrated in Figure 7.

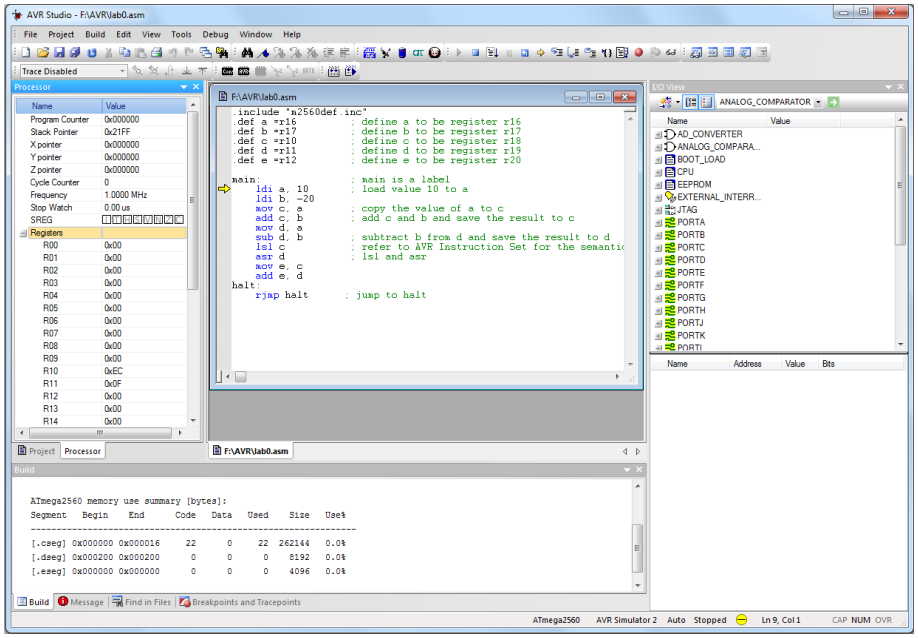

Figure 7: Register View

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 double clicking on the value of a register and entering the appropriate value at any time during the simulation.

2.3 Run Simulation

There are two 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.

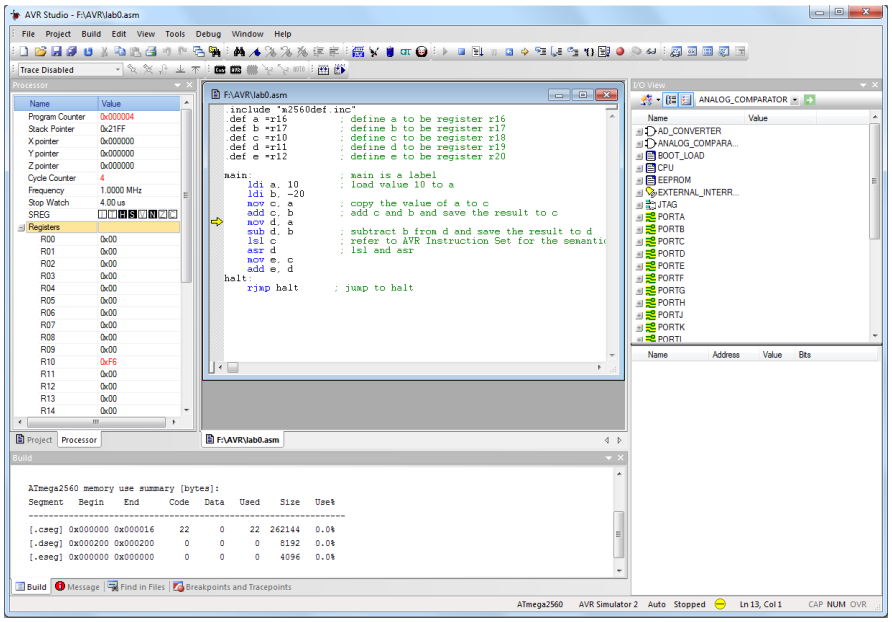

Figure 8: Single-Step Execution

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 Toggle Breakpoint. Figure 9 shows a breakpoint (indicated by the brown dot) is added on the *asr* instruction.

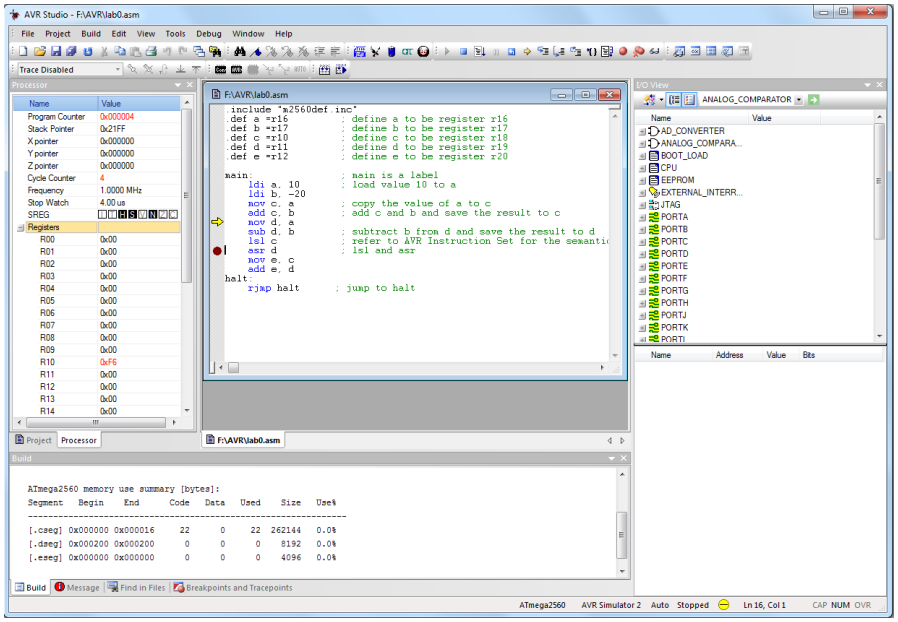

Figure 9: Setting Breakpoint

Now press the run (F15) button. The simulation will continue and then stop at the breakpoint, as shown in Figure 10. Toggle the breakpoint, the breakpoint will be removed.

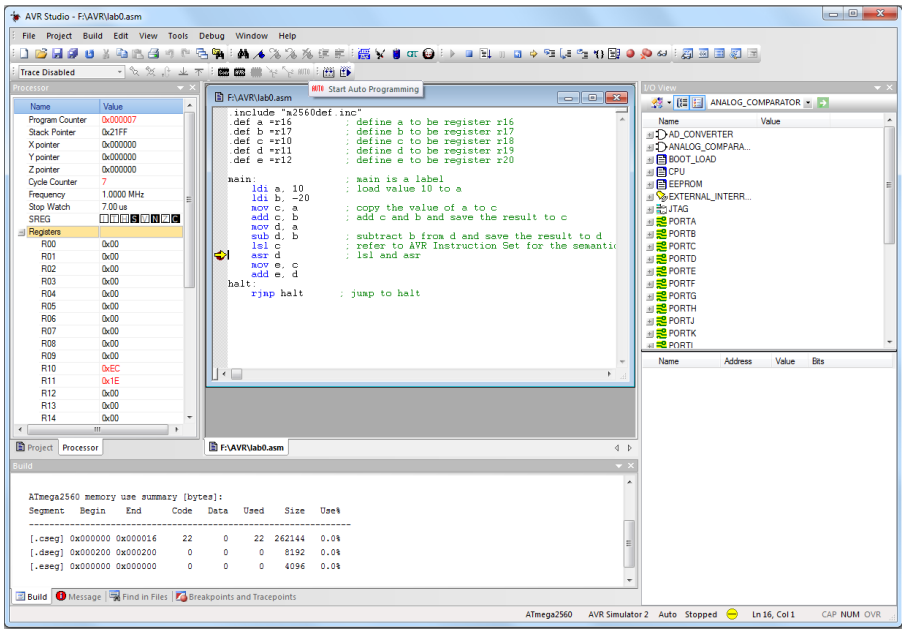

Figure 10 Execution with Breakpoint

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 → Mega256 Data Sheet). These flags are dynamically updated during the program execution, and also can be observed in the Processor window. When a flag bit is set, the corresponding bit block is blacked. 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 Disassembler

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 click View and then Disassembler, as shown in Figure 11. You will see a listing of your disassembled program. 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 name (the last column) are given. Take the instruction “ldi a, 10” for example. Its memory address is 0x00, and the machine code is 0xE00A (one word long). Its assembly code is “LDI R16,0x0A”.

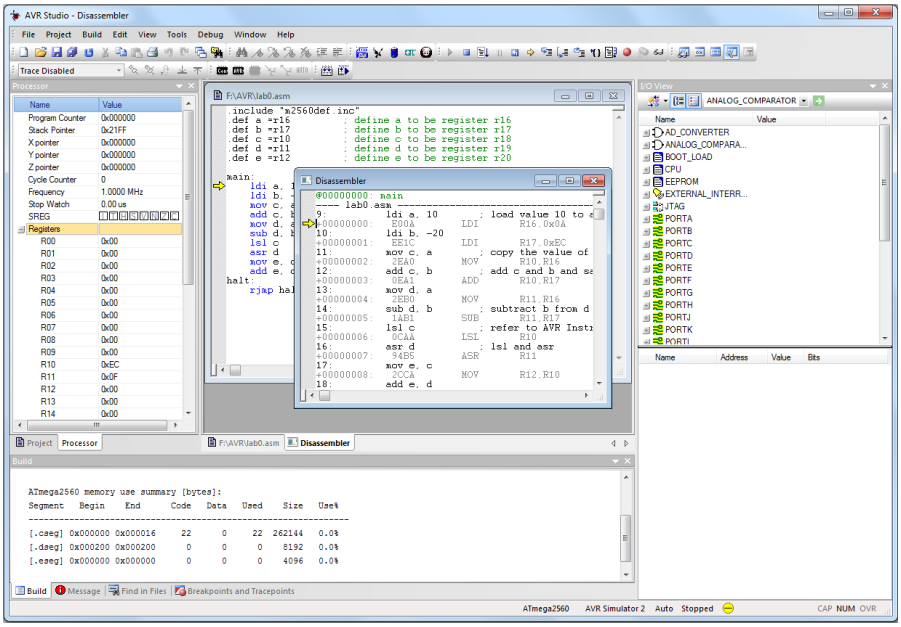

Figure 11: Disassemble Code

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.