PRINCIPLES AND APPLICATION OF MICROCONTROLLERS
AVR Assembly Lab10: Assembly Program Simulation

Introduction

In this lab, you will gain familiarity with assembly program tracking and debugging using Atmel Studio. After completing this lab you should be able to:

- Simulate an assembly program
- Disassemble a program from machine codes

Procedure

Creating a New Project:

Create a new assembly project. Type the following program in the text editor and build it.

```
        LDI R20, 0
        LDI R21, 55
        LDI R22, 10
        ADD R20, R21
        ADD R20, R22
HERE:   RJMP HERE
```

Figure 1: Example program code

Program Tracing and Debugging:

Select **Start Debugging and Break** from the **Debug** menu. A window will pop up and ask for the debugging tool. Select **AVR Simulator** and press the **OK** button.

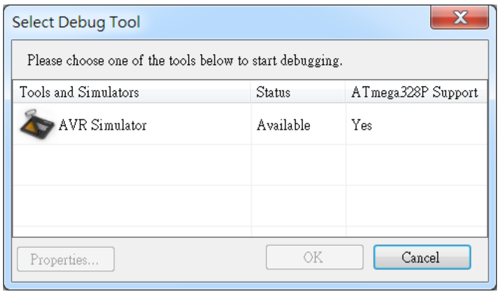

Figure 2: Selecting simulator

A yellow arrow appears next to the first instruction of the program which shows that the next instruction which will be executed. To execute the next instruction press F11 or click the **Step Into** button in the **Debug** toolbar (blue rectangle in Fig. 3). There are also other tracing tools in the toolbar. The **Step Over** executes the next instruction like **Step Into**. The only difference between them is that, if the next instruction is a function call, the **Step Into** goes to the function. In contrast, **Step Over** executes the function completely and goes to the next instruction. For more information about the Tracing tools you can see the AVR Studio's help.

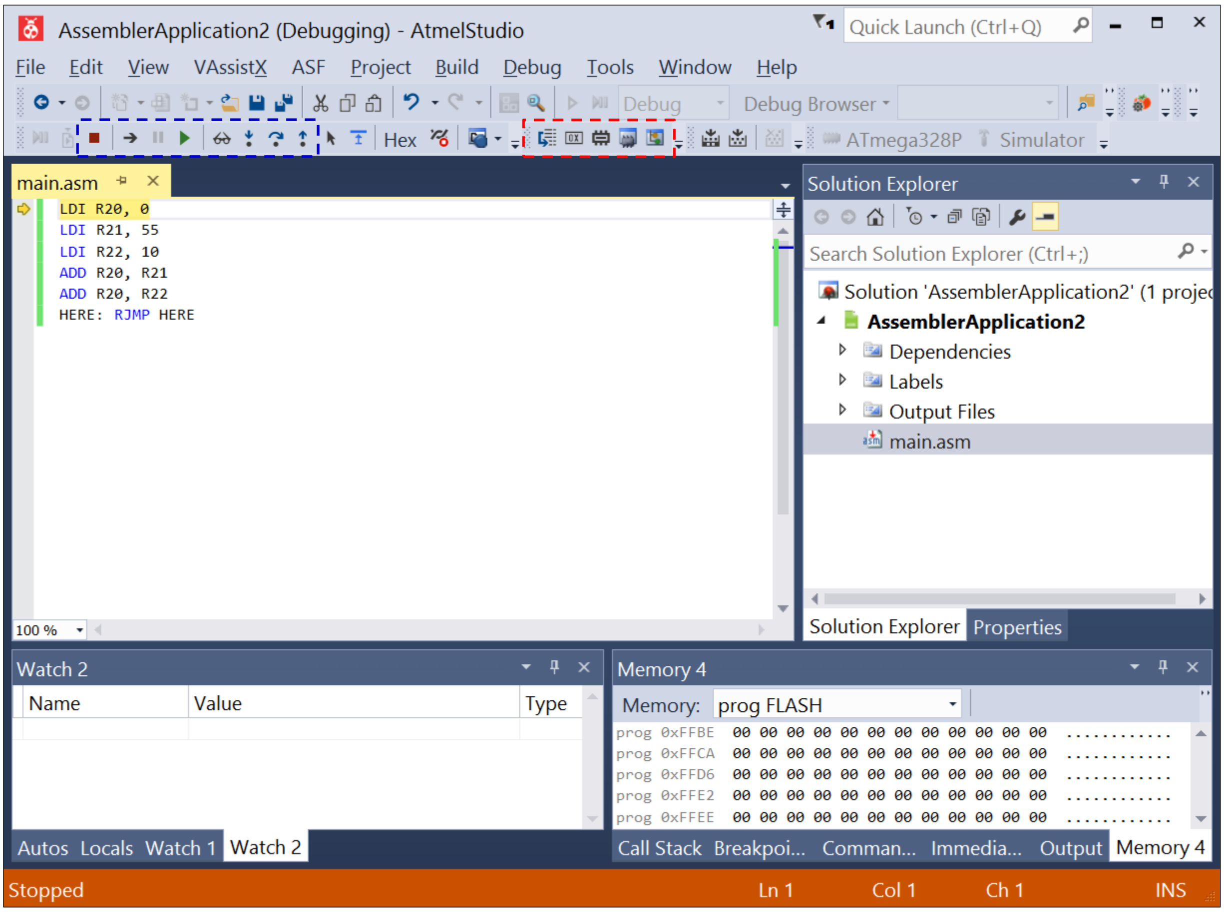

Figure 3: Debug and Atmel Debugger toolbars

Useful Tools:

In this part you learn to use the different tools for checking the program. Click the buttons on **Atmel Debugger** toolbar (red rectangle in Fig. 3) to use these tools. The buttons are: a) **Disassembly**, b) **Registers**, c) **Memory1**, d) **Processor View**, and e) **I/O View** (from left to right).

a) **Disassembly**: This window shows the contents of the program memory (flash ROM). In the window:

1) The black texts display your program (Fig. 4).
2) Below each instruction of your program, a gray number mentions at which address in the flash ROM an instruction is located. For example, in Fig. 4, "LDI R20, 0x00" is located at address 0000. Check the "Viewing Options" if you don't see the address of the instruction.
3) The gray numbers after an instruction location are the machine code of each instruction. For example, in Fig. 4, the machine code of "LDI R20, 0x00" is "40 e0".
4) The last column describes what the assembly instruction does. For example, as you see in Fig. 4, LDI is Load Immediate, and RJMP is Relative Jump.
5) The yellow arrow, points to the next instruction which will be executed.

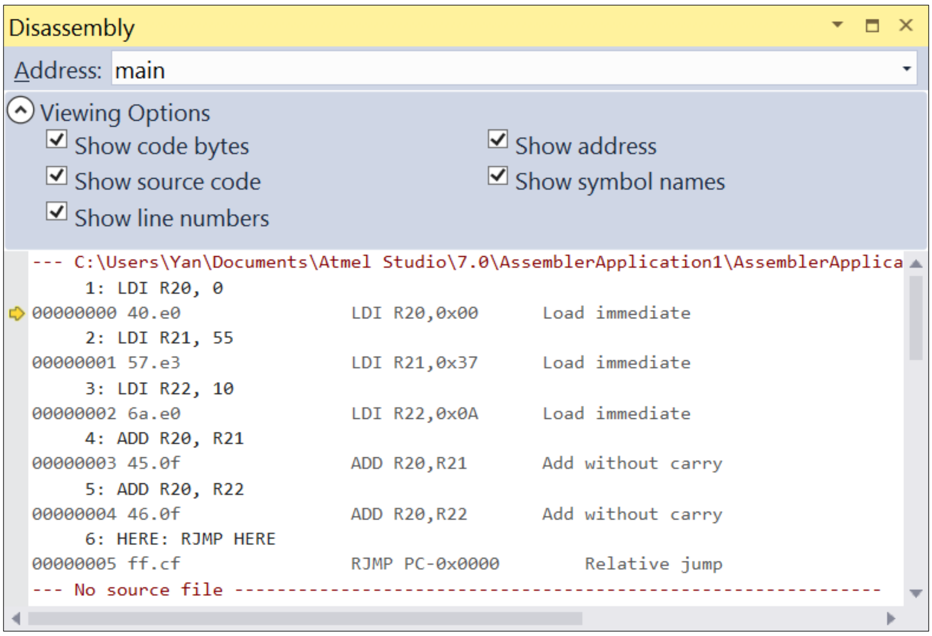

Figure 4: Disassembly window

b) **Register:** The Register window shows the contents of all of the general purpose registers, at the current time (Fig. 5).

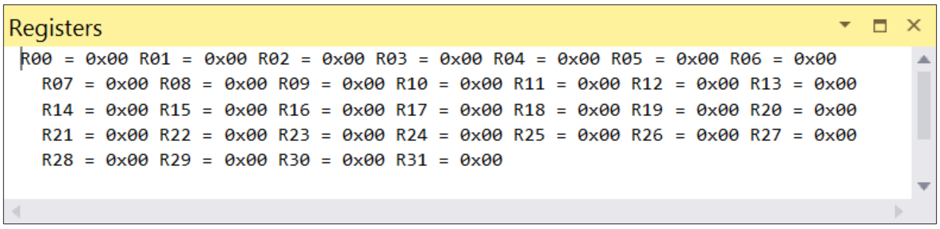

Figure 5: Register window

c) **Memory:** In the Memory window you can see the contents of different locations of memory, at the correct time. In the window:

1) The gray column shows the address of the first location in each row.
2) You can choose which of the memories (e.g., prog FLASH, data REGISTERS, data MAPPED_IO, etc) to be displayed on the top left corner of the window (Fig. 6).

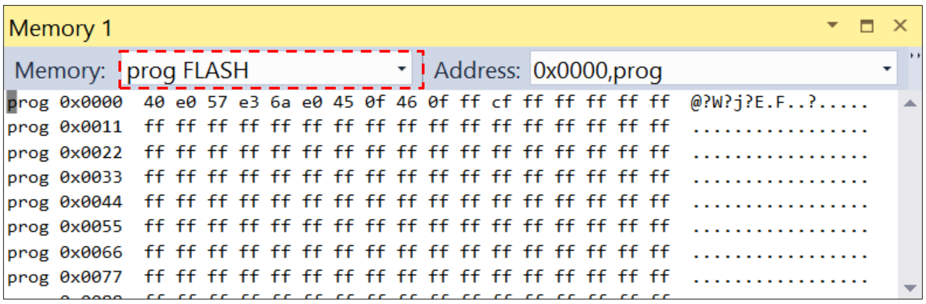

Figure 6: Memory window

d) **Processor:** The Processor window shows the contents of the registers which are related to the CPU: general purpose registers (R0 to R31), Program Counter, Stack Pointer, Status Register, X, Y, and Z registers (Fig. 7). "Cycle Counter" counts the number of machine cycles which have been passed and the "Stop Watch" represents how much time has elapsed. You can use the parameters to measure the execution time of your program. You can reset them as well, by right clicking on them and choosing reset.

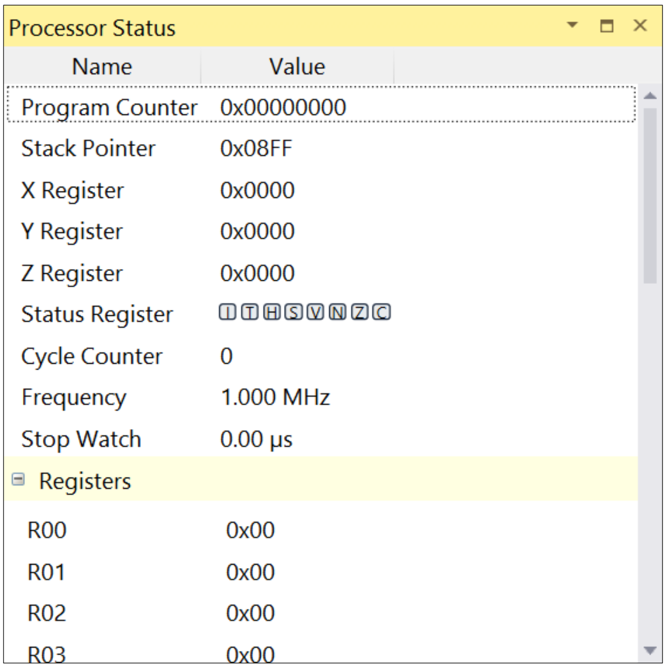

Figure 7: Processor window

e) **I/O View:** In this window, you can see the value of the different I/O registers (Fig. 8). In the upper box, the related I/O registers are grouped. Click on PORTB and see the values of PINB, DDRB, and PORTB.

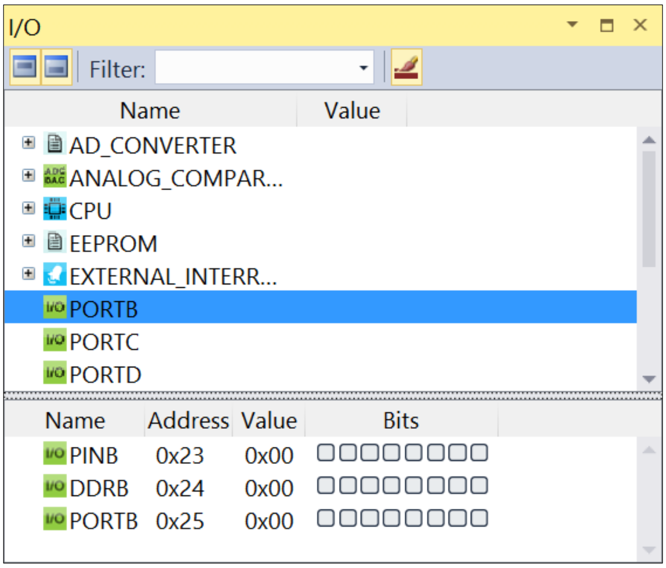

Figure 8: I/O view window

Deliverables

Create an assembly project, type the program in Fig. 9, build it, and answer the questions.

```
    .EQU SUM = 0x300
    LDI  R16, 0x25
    LDI  R17, 0x34
    LDI  R18, 0x31
    ADD  R16, R17
    ADD  R16, R18
    LDI  R17, 0x74
    ADD  R16, R17
    LDI  R20, 0
    ADD  R16, R20
    LDI  R21, 0xFF
    OUT  DDRD, R21
L2: INC  R20
    OUT  PORTD, R20
    STS  SUM, R20
    RJMP L2
```

Figure 9: Deliverable program code

1. Use the Atmel Studio debugging functions to find the machine codes for the program in Fig. 9. Make a copy of the machine codes to your report.

2. Identify the opcodes for the instructions in the following table. Report them in binary forms. Leave the digit cross (X) if it is a part of an operand.

Instruction	Opcode
LDI	1110 XXXX XXXX XXXX
ADD	
OUT	
INC	
STS	
IN	
SUBI	
SBI	

3. Observe the value changes in general purpose registers (GPR), status register (SREG), and program counter (PC) "after" the execution of every instruction. Report your observations in a table.

	Instruction	PC	SREG	R16	R17	…
1	LDI R16, 0x25					
2	LDI R17, 0x34					
…						

4. Find out how the value of memory address at 0x300 varies? What do you do if you want to store the value of R20 to the addresses of 0x150 instead of 0x300?

Appendix

Lock Bits and Fuse:

AVR microcontrollers have 3 memory areas: FLASH dedicated to program code, SRAM for run-time variables, and EEPROM used by users to store data that have to be preserved when the microcontrollers are turned off. Lock Bits and Fuses form a fourth memory area available for programming. The Lock Bits and Fuses are used for configuring peripheral and determining some very general system settings, such as memory access, clock source, and source divider, start-up options, and etc. ATmega328P has one byte for Lock Bits and 3 bytes for Fuses (i.e., low, high and extended). Setting some wrong combination of the Lock Bits and Fuses effectively renders microcontrollers unworkable. Datasheets provide extensive information on the Lock Bits and Fuses. Read docs carefully before applying any changes to this area.

Lock	-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1
Fuse Low	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0
Fuse High	RSTDISBL	DWEN	SPIEN	WDTON	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST
Fuse Ext.	-	-	-	-	-	BODLEVEL2	BODLEVEL1	BODLEVEL0

Figure 10: Lock bits and fuses