EECS2021

LAB B

Lab Objectives

This lab introduces you to the RVS RISC-V simulator and to basic assembly programming. By the end of this lab, you should be able to:

1. Use RVS to compile and run assembly programs.
2. USE RVS to check memory contents and registers contents.
3. Write a small straight line assembly program.

Part 1

Read the Assembler and Manual help files for the simulator. The manual and supporting materials are in http://www.eecs.yorku.ca/course_archive/2017-18/F/2021S/

Under the “Documentation” section please read the “RVS User Manual” and “RVS Input/Output System Calls Manual”. Use version 5 “v005”.

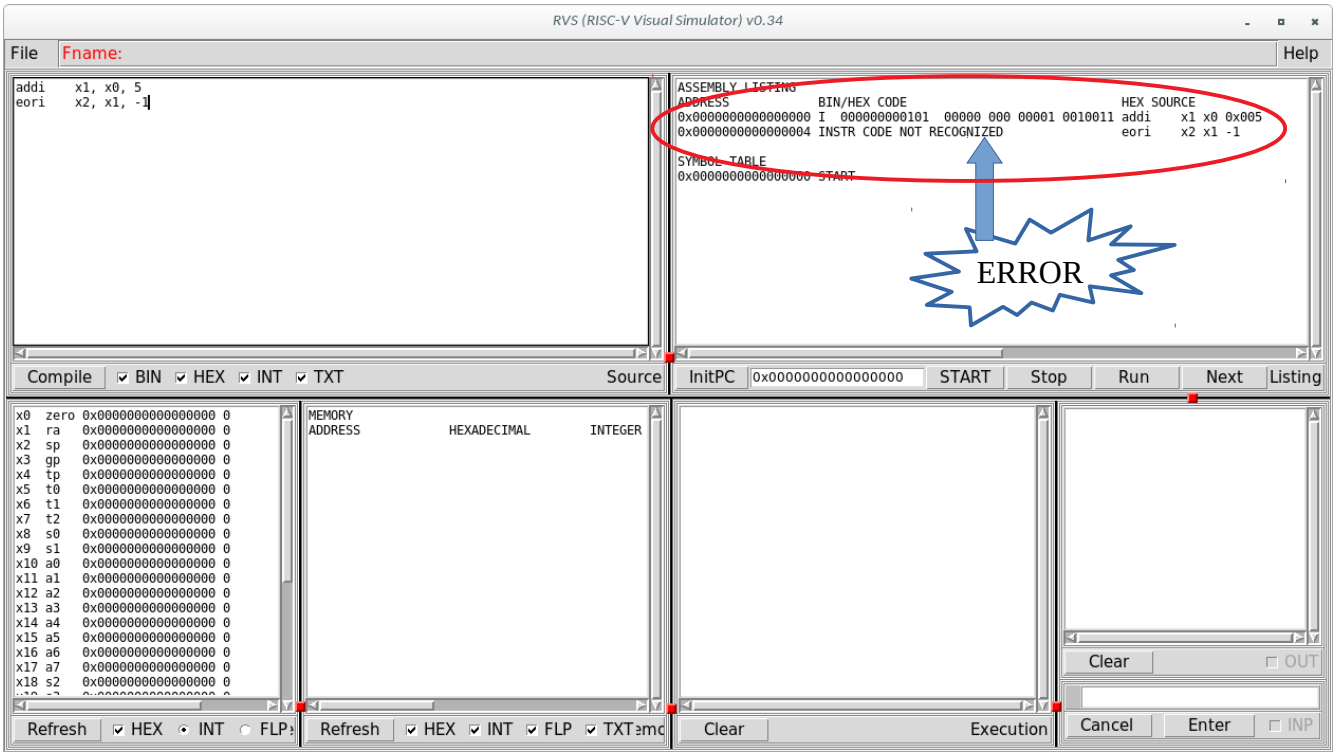
To run the program, type RVS038 at the command prompt. Note that it might take sometime for the simulator to be installed in one of the standard locations. In this case, you can download the RVS038 to your local directory and run it from there.

Syntax Errors

Before we start, let us try to run a program that contains syntax error.

Run the simulator and enter the following code in the source code window or load a text file containing the code shown below File → Load (the file must be terminated by a newline character, which most editors do anyway).

```
addi    x1, x0, 5
eoi     x2, x1, -1
```



The first line is addi x1, x0, 5 is a legitimate instruction, the second line is not (eoi is not an instruction in RISC-V).

Click on “compile” and check the Assembly listing window (shown within red ellipse). The first line in the window (below “ADDRESS”) shows the memory location 0x0000 ..0 (since we did not specify where to put the code, it starts from location zero) and the representation of the first instruction

```
addi x1, x0, 5
```

It shows the instruction type (I for Immediate) followed by the binary, hex, and text representation of the instruction.

The following line, is memory location 0x00..04 followed by “INSTR CODE NOT RECOGNIZED”. That is a syntax error, you have to go back and correct that instruction (eoi is xor in the ARM assembly language, easy mistake to do if you are familiar with ARM).

In the rest of part 1, we will go through simple examples on how to use the simulator. Then in Part 2 that is the part you have to do in the lab.

Register manipulation

Launch the simulator

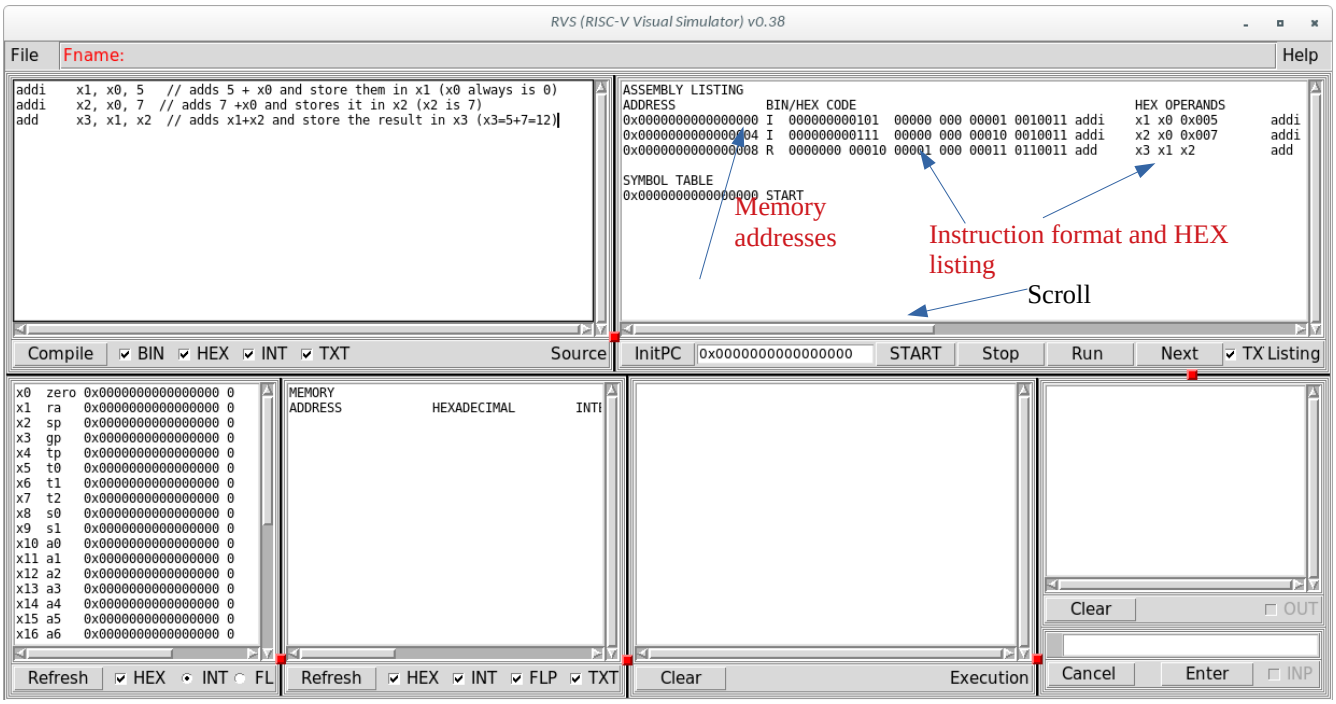
Then follow these steps for your first program/interaction with the simulator.

Write the following into the Assembly Source Window (or load a file contains the text below).

```
addi    x1, x0, 5       // adds 5 + x0 and store them in x1 (x0 always is 0)
addi    x2, x0, 7       // adds 7 +x0 and stores it in x2 (x2 is 7)
add     x3, x1, x2      // adds x1+x2 and store the result in x3 (x3=5+7=12)
```

click on compile.

The simulator window will look like this

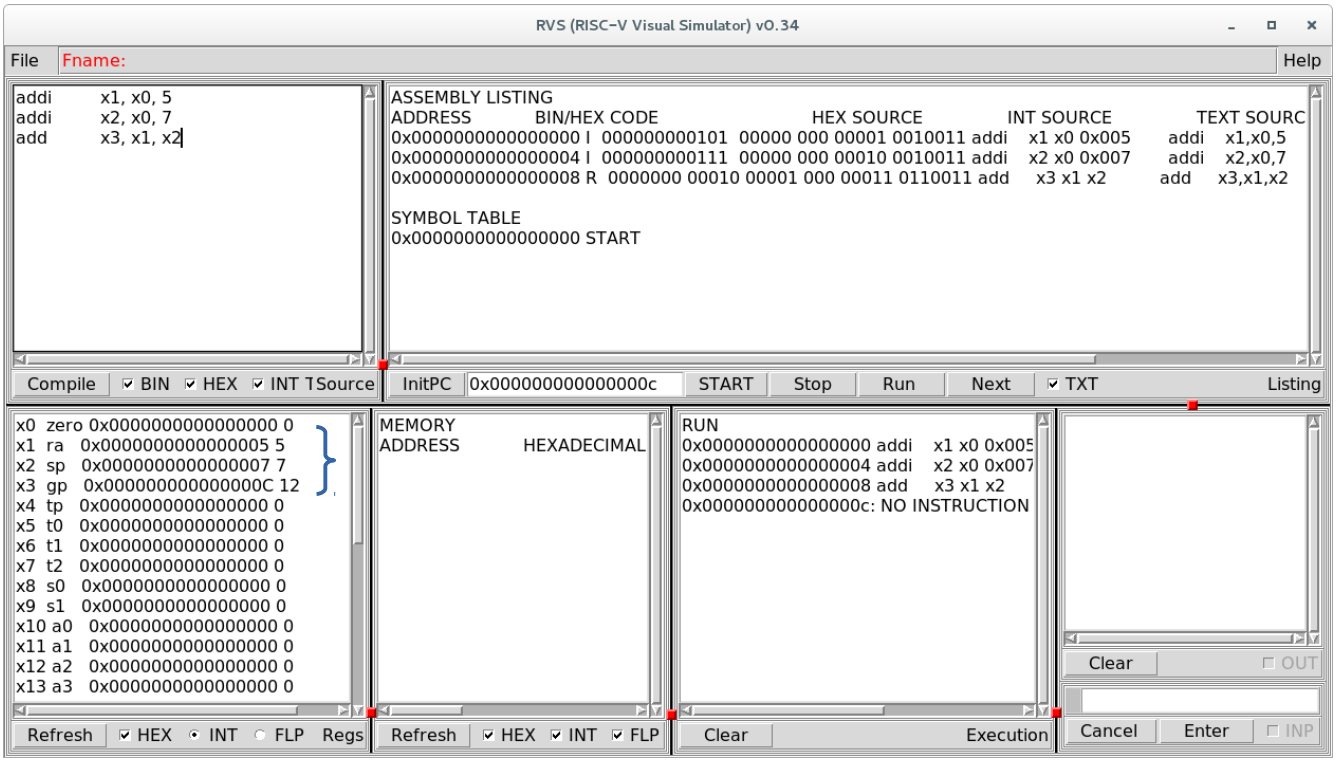


Note that the register window shows the contents of all the 32 registers in the CPU. The registers are initialized to zero.

In the assembly listing window, you can see the memory addresses where the instructions are stored. Since we did not specify the starting memory address, the default is 0 (we can change this using ORG directive as we will see later). Three instructions are stored in locations 0, 4 and 8 respectively. If you remember from the class, RISC-V instructions are 32-bit each (4 bytes).

Under BIN/HEX CODE you can see the binary representation of the instructions showing the different fields as discussed in the class and the textbook. Before the representation, the instruction type is shown (The first two instructions show I, for immediate, and the third one is R for R-type or Register-Register ALU operation). Following that it shows the different operands for the three instructions in HEX. If you scroll to the right, you will see the operands in decimal.

Now, click on START or RUN, the program runs and the register valus changes as shown in the following Figure.



Only the registers that we modified in the program have changed values (x1=5, x2=7, and x3=12).

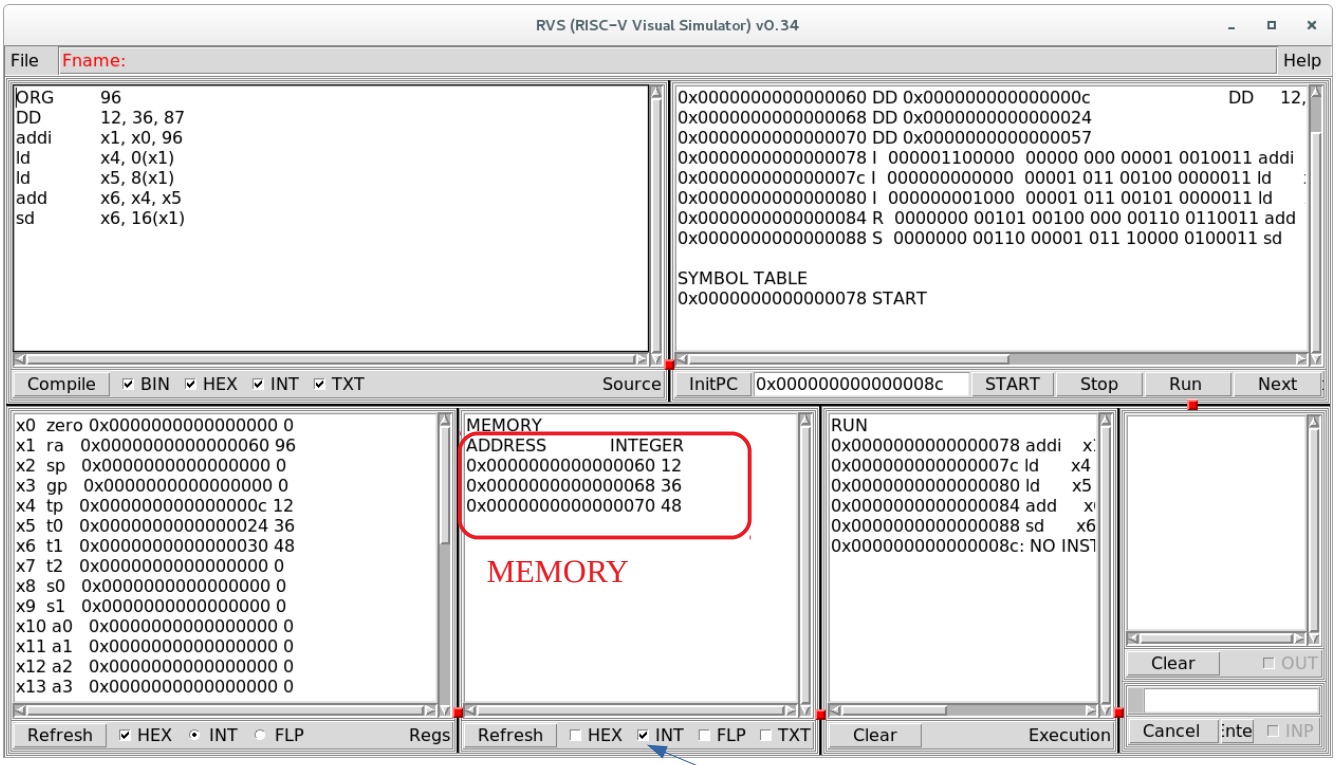
Memory manipulation

Launch the simulator

Enter the following code (or put it in a file and open it in the simulator)

```
ORG    96
DD     12, 36, 87
addi   x1, x0, 96
ld     x4, 0(x1)
ld     x5, 8(x1)
add    x6, x4, x5
sd     x6, 16(x1)
```

Compile and run the program. The window should look like this



Show the value in decimal

Let us us take a close look at the simulator.

1. The first line

   ORG 96

   States that you start at location 96 (hex = 0x60). That basically means the memory assignment will start at location 96. So, the next line data or instruction will be in location 96.

2. The second statement

   DD 12, 36, 87

   Define three double words 12, 36, and 87 and stores them in locations 96, 96+8=104, 104+8=112 (in hex 0x60, 0x68, 0x70).

   Now, look at the memory contents window, it shows the contents of the memory.

   0x0000000000000060 12

   In memory location 0x60 (decimal 96) the contents is the integer value 12. The next two lines shown the two integers 36 and 87 in memory location 0x68 and 0x70 (decimal 104 and 112). The above figure does not show the value 87, since it has changed in the program as we will see soon.

The values is shown in decimal since we clicked the INT box, if you click on HEX, the value will be shown both in decimal and hex

3. The next line is

   ```
   addi    x1, x0, 96
   ```
   It adds the contents of x0 (by definition 0) to the integer 96 and stores the result in x1. The register window shows the content of x1 to be 96. sine the value of x1 is 96, we can say that x1 is pointing to the memory where we started storing data (96 or 0X60).

4. The following line is

   ```
   ld      x4, 0(x1)
   ```
   It loads a doubleword from location 0+contents of X1 REG(x1) that is 96, to register x4. The instruction loads the contents of memory location 96 (0x60) into register x4. From statement 2 above, we stored 12 in location 96. Now x4 contains 12.

5. The following line is

   ```
   ld      x5, 8(x1)
   ```
   it loads from location 8+REG(x1), i.e. 8+96=104 (in hex 0x68) into register x5. If you look at the memory window, in location 0x96 there is 36. now x5 contains 36 (check the register window to be sure it contains 36, if not, you made a mistake. Find out your mistake and correct it).

6. The following line is

   ```
   add     x6, x4, x5
   ```
   Adds REG(x4), that is 12, to REG(x5), that is 36, and the result is stored in x6. If you check the contents of register x6 in the register memory, you find it 48.

7. The following line is

   ```
   sd      x6, 16(x1)
   ```
   Stores the contents of x6 (that is 48) in memory location pointed to by the contents of x1 + 16 (REG(x1)+16 = 96+16 = 112, in hex 0x70). Now check location 0x70 (or 112 decimal) in the memory window it is 48.

Part 2

Consider the following program(s). The two programs do exactly the same thing, one in C and one in Java.

```
int main()
{
   int Y=15;
   int Z=6;
   int C=-5;
   int D=12;
   int L=3;
   int M=11;
   int X;

   X=(Y+M) – (L-D) +(Z+C) – D;
}
```

```
public class HelloWorld {
   public static void main(String[] args) {
      int Y=15;
      int Z=6;
      int C=-5;
      int D=12;
      int L=3;
      int M=11;
      int X;

   X=(Y+M) – (L-D) +(Z+C) – D;
   }
}
```

Some of you may be familiar with C, the majority are familiar with Java. No matter which language you are familiar with, you can do this problem. ***However***, from now on, I will be using C code. If you notice, the syntax is the same. For the programs covered in this course, these two languages are identical.

Write RISC-V assembly code to implement this programs. Consider two cases.

1. All the input variables are to be loaded into a register (you have to store them in the memory using compiler directives). The X should be stored in the memory Also.
2. Y, Z, C are to be loaded immediately into a register. D, L, M, and X are stored in the memory, also X should be stored in the memory (in this case, you don't need to declare A,B, and C).

After completing the lab, you have to show the TA that your code is working.

Lab Report:

Your lab report should be submitted in pdf format.

Your report should include

1. cover page with your name and student ID
2. The code for the 2 programs.
3. For every program, put a screen shot showing the memory locations where the data is stored before and after the run (do that for the register window too).
4. Submit it as **`submit 2021E labb file.pdf`**