

Lab1: Your First Assembly Program

Learning Outcomes:


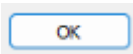
After completing this lab, you will be able to:

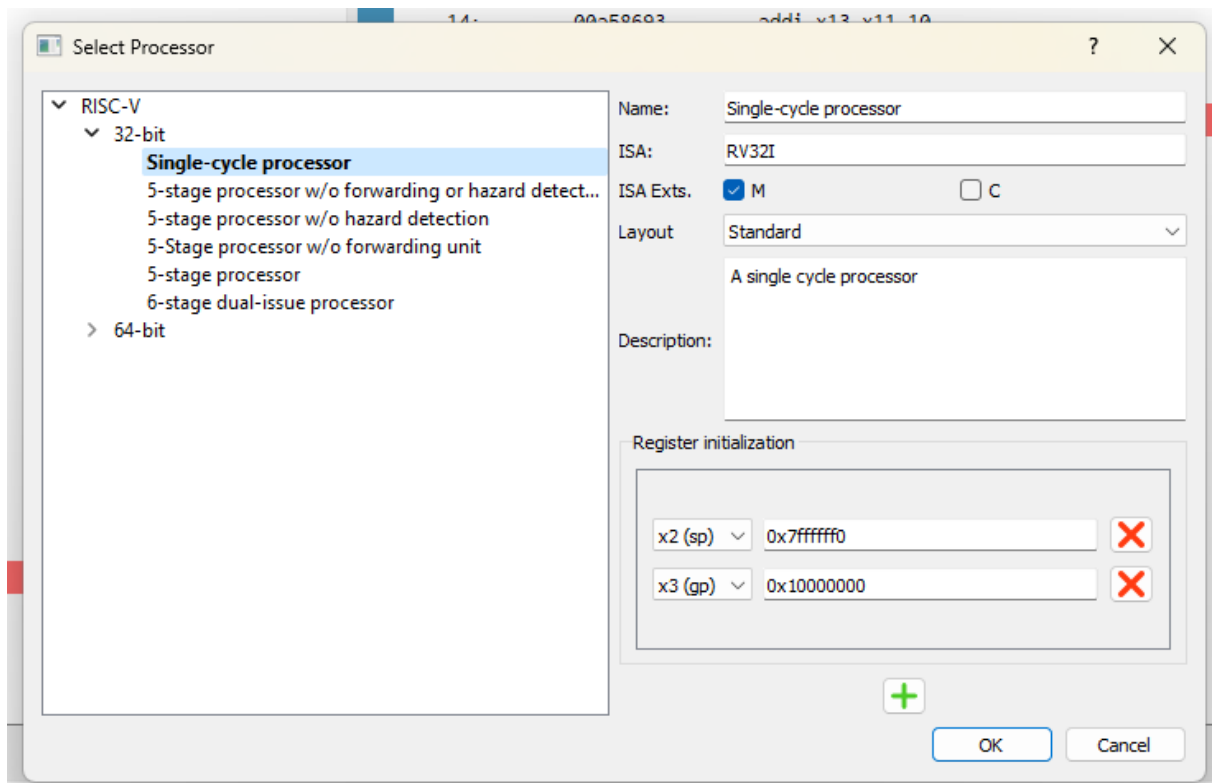
- Write a very simple assembly program using basic RISC-V instructions using the Ripes simulator.
- Observe internal signals/registers within the processor.

Prerequisites:

Follow the Pre-Lab Installation Instructions.

Example: Understanding I-type Instructions

1. Open the Ripes application and click  icon on the top left corner.
2. In the opened *Select Processor* window, select *Single-cycle processor* option under *32-bit* and click  button to save the changes.



3. Paste the following assembly code in the editor tab.

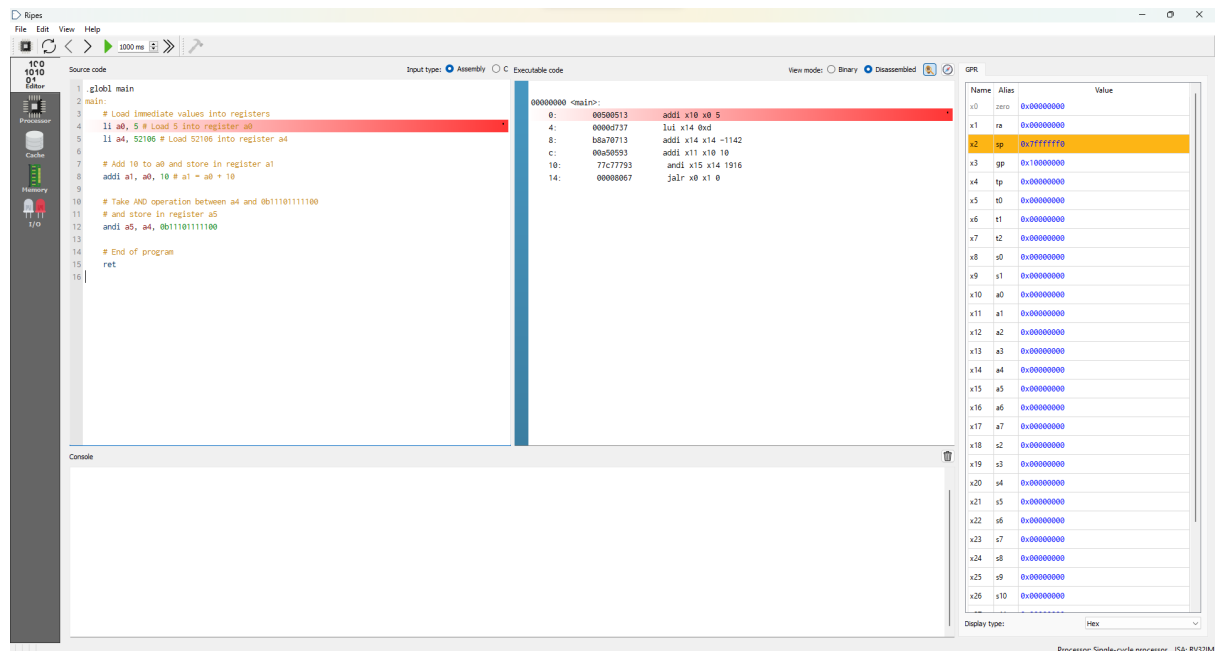
```
.globl main
main:
    # Load immediate values into registers
    li a0, 5 # Load 5 into register a0
    li a4, 52106 # Load 52106 into register a4

    # Add 10 to a0 and store in register a1
    addi a1, a0, 10 # a1 = a0 + 10

    # Take AND operation between a4 and 0b11101111100
    # and store in register a5
    andi a5, a4, 0b11101111100

    # End of program
    ret
```

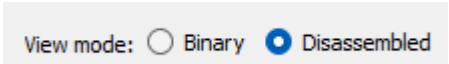
You will be able to see the following screen.



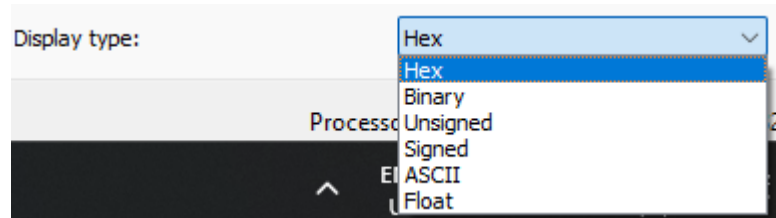
There are 3 main sections in the user interface:

- The *source code* section shows the assembly code you have written.
- The *executable code* section shows the compiled executable code in both the machine code and the disassembled machine code in the Assembly language.


You can switch between these two views by setting the required view mode

using  option.


- The *GPR* section shows the register bank of the processor. You can change the format of the register values using the display type option at the bottom right corner of the screen.



4. Change the display type to *Signed*.

5. Execute the first instruction by clicking the  icon in the top left. Now you can see in the register bank that the value 5 is loaded into the register a0.

x10	a0	5
-----	----	---

6. Go another step further using the  option and observe the value 53248 is loaded into the register a4.

x14	a4	53248
-----	----	-------

Go another step further. Now 1142 will be deducted from register a4 and the final value will be 52106.

x14	a4	52106
-----	----	-------

Notice that the assembler has placed two machine instructions for one assembly instruction.

7. Go one step further. It will execute the 'addi a1, a0, 10' instruction. This instruction is an I-type instruction. It will add 10 (which is the immediate value given with the instruction) to the value in register a0 and store the result in register a1.

Confirm the operation using the value in the register bank.

x10	a0	5
x11	a1	15

8. Change the Display Type of the register bank to Binary.
9. Go one step further. It will execute the 'andi a5, a4, 0b11101111100' instruction. This instruction is also an I-type instruction. It will take the Bitwise AND operation

x14	a4	0b00000000000000001100101110001010
x15	a5	0b0000000000000000000000001100001000

Write a program to check whether a number is an odd number or not. Store the final output in register a0 - it should be set to 1 if the number is odd, otherwise 0.

- AND
- Set Less Than

Write a program to multiply any given number by 7. Store the final result in register a0. You **must use shift instructions** for this exercise.

1. There is a limitation to the number that can be given as an immediate value in an I-type instruction. Observe the largest possible immediate value. What will happen if you specify a value larger than that?
2. Store a negative number in a register. Check the difference between Shift Right Logical (srl) and Shift Right Arithmetic (sra) instructions using the above value.
3. Observe that sometimes instructions in the assembly code may differ from the disassembled machine code. Pseudo-instructions are one of the reasons for this scenario. What are the pseudo-instructions that you have observed?