ELEC3441: Computer Architecture
Second Semester, 2023–24

Lab 4

`Objective: Experiment with Pipeline Operations`

In this lab, you will practice how processor pipeline works using a RISC-V processor simulator called `ripes`. Similar to the assembler `rars` that you used in Lab 2, `ripes` can also execute RISC-V assembly programs in software. However, `ripes` goes into a lot more details of the inner working of a processor, and simulate its pipeline and cache behaviors as well. You will learn to use this software in this lab.

1 Getting the Code

You may find the source files needed for this lab from:

`https://www.eee.hku.hk/~elec3441/sp24/handout/elec3441lab4.zip`

2 Getting the Software

`ripes` is an open source software. You can find its code from:

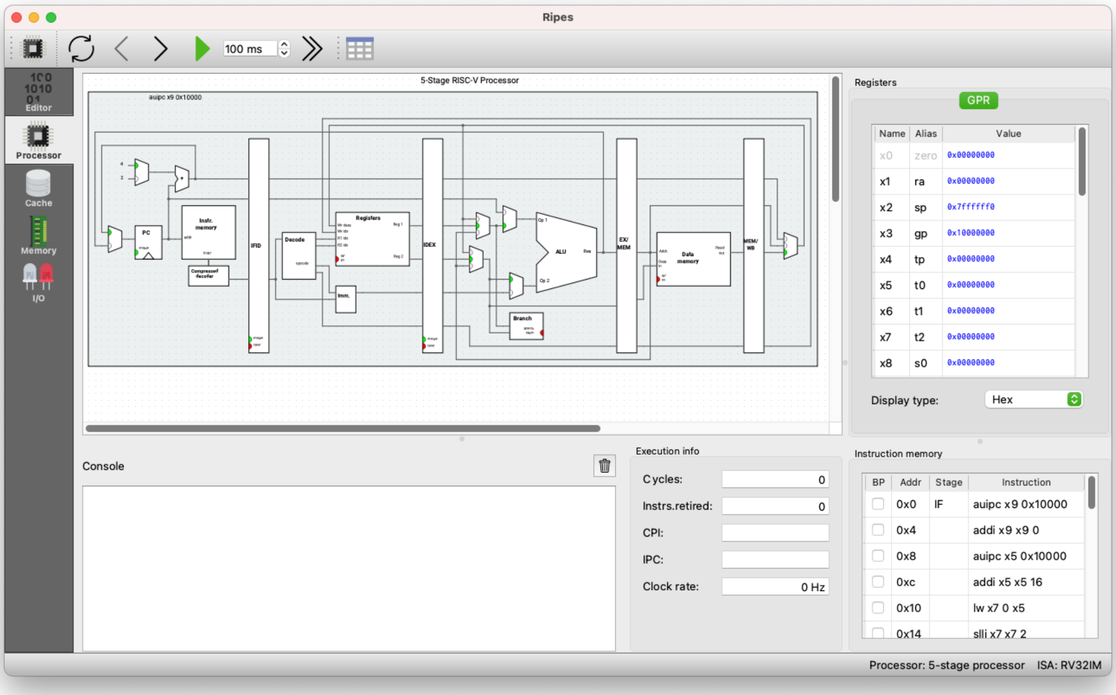
`https://github.com/mortbopet/Ripes`

The easiest way to start using the software is to download the pre-built files from the release page:

`https://github.com/mortbopet/Ripes/releases`

3 Your First Program

3.1 Starting `ripes` Run the `ripes` program. You should see the initial UI similar to the following:



Execution info

Cycles:	0
Instrs.retired:	0
CPI:	
IPC:	
Clock rate:	0 Hz

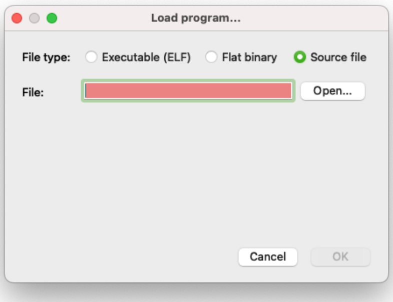
Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000

Display type: Hex

BP	Addr	Stage	Instruction
☐	0x0	IF	auipc x9 0x10000
☐	0x4		addi x9 x9 0
☐	0x8		auipc x5 0x10000
☐	0xc		addi x5 x5 16
☐	0x10		lw x7 0 x5
☐	0x14		slli x7 x7 2

Hint: The UI of `ripes` doesn't look too well if you had dark mode enabled on your computer. Try to use a light mode color scheme.

3.2 Loading Program Load the file `lab4/mystery.s` by using the menu item **File** → **Load Program**. This is the same mystery program from Lab 2. A new window will pop up like below.

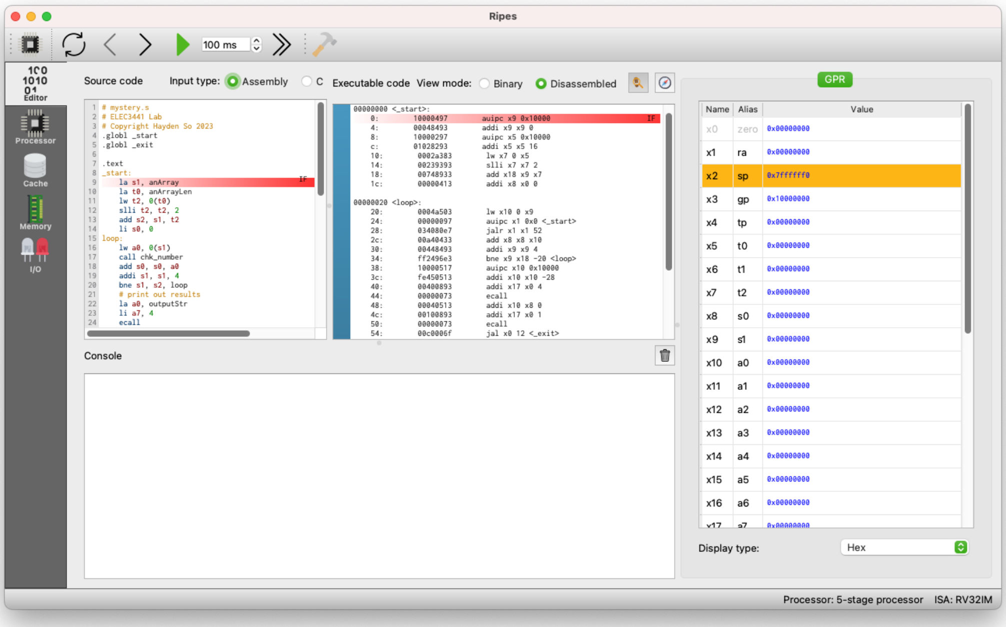


File type: Executable (ELF) Flat binary [✓ Source file]

File:

Make sure you select the **Source File** radio button. Then select **Open. . .** to load the file.

3.3 Running the Code You can see the currently loaded program in the **Editor** pane (Editor). In this pane, you should see the code of `mystery.s` and their machine code loaded with a UI similar to the following:



```
# mystery.s
# ELEC3441 Lab
# Copyright Hayden So 2023
.globl _start
.globl _exit

.text
_start:
    la s1, anArray                    IF
    la t0, anArrayLen
    lw t2, 0(t0)
    slli t2, t2, 2
    add s2, s1, t2
    li s0, 0
loop:
    lw a0, 0(s1)
    call chk_number
    add s0, s0, a0
    addi s1, s1, 4
    bne s1, s2, loop
    # print out results
    la a0, outputStr
    li a7, 4
    ecall
```

```
00000000 <_start>:
    0:      10000497      auipc x9 0x10000          IF
    4:      00048493      addi x9 x9 0
    8:      10000297      auipc x5 0x10000
    c:      01028293      addi x5 x5 16
    10:     0002a383      lw x7 0 x5
    14:     00239393      slli x7 x7 2
    18:     00748933      add x18 x9 x7
    1c:     00000413      addi x8 x0 0

00000020 <loop>:
    20:     0004a503      lw x10 0 x9
    24:     00000097      auipc x1 0x0 <_start>
    28:     034080e7      jalr x1 x1 52
    2c:     00a40433      add x8 x8 x10
    30:     00448493      addi x9 x9 4
    34:     ff2496e3      bne x9 x18 -20 <loop>
    38:     10000517      auipc x10 0x10000
    3c:     fe450513      addi x10 x10 -28
    40:     00400893      addi x17 x0 4
    44:     00000073      ecall
    48:     00040513      addi x10 x8 0
    4c:     00100893      addi x17 x0 1
    50:     00000073      ecall
    54:     00c0006f      jal x0 12 <_exit>
```

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000

Processor: 5-stage processor ISA: RV32IM

While inside the Editor pane, execute the program loaded by pressing the **auto clock** button ( ). If everything goes well, you should see the program running with multiple red stripes moving. At the end, the results of the program should be printed.

You can also check the processor running if you switch to the processor pane ( ).

3.4 Check Yourself

Run the program, check what is being displayed on the Editor and the Processor pane. Then answer these questions:
(i) What is printed on the output console?
(ii) What is the value stored in the register `a3`?
(iii) What is the average CPI of mystery program when run on this processor?

4 Understanding Processor Pipeline

As a processor simulator, `ripes` simulates the inner working of a processor cycle by cycle. It includes simulating the effect of pipeline and cache.

4.1 Basic Pipeline Operations Load the program `lab4/rtype.s` into the processor.

- Go to the Editor pane. You should now see that the first instruction is highlighted in red.
- Go to the Processor pane. You should see that the first instruction is also shown at the top of the IF stage.

4.2 Stepping Execution Now instead of using the auto clock button, step through the execution of the code one cycle at a time by using the **clock** button ( ).

- Step through the program `rtype.s` by pressing the step button **ONE** (1) time.

In the Editor pane, you should notice that both instruction 1 and instruction 2 are now highlighted in red. In addition, the stage that the instruction is currently at (e.g. IF, ID) is shown on the right. Make sure you understand how the first instruction (`addi`) is now in the ID (2nd) stage, while the second instruction (`ori`) is in the first stage (IF).

Go to the Processor pane. You should now also see the instructions have moved by 1 stage to the right in the pipeline.

4.3 Complete the Program Continue stepping through the program with the **clock** button ( ) until the program ends. Observe how the instructions are executed in the pipeline.

4.4 Check Yourself

- After your program completes, what is the value in the register `s0`?

  Value of `s0`: $\boxed{}$
- After your program has completed, what is the CPI shown?

  CPI = $\boxed{}$
- Why is the CPI number not 1?
- What is the maximum number of instructions that would be highlighted in red throughout the execution of the program?

5 Pipeline Stall & Data forwarding

So far, the default processor loaded by `ripes` is the classic 5-stage processor we studied in class with *full data forwarding* and *hazard detection*. However, the program also support other processor models such that you can experiment with hazard detection and forwarding.

5.1 No forwarding Click the **select processor** button ( ) in `ripes`. A new dialog will pop up.

- Select the processor listed as "5-Stage processor w/o forwarding unit"

This is the same 5-stage processor as we studied in class, except that it does not support data forwarding. This processor does support hazard detection and will stall the processor (insert `nop`) if it detects a data hazard.

5.2 Step through the program Now, step through the program by using the **clock** button ( ). Pay particular attention after the `add a1, a0, a0` instruction is loaded. Advance the processor cycle by cycle and observe what happens when the `add` instruction is in the EX stage.

5.3 Checkoff 1

Run the program in `rtype.s` until it ends with the new processor that does not support data forwarding. Answer the following questions:

(i) List all the RAW data dependencies in the program.
(ii) What happens to the processor pipeline when a data hazard occur?
(iii) What is the value of `s0` after your program completes?

Value of `s0`: $\boxed{}$

(iv) What is the CPI value shown after your program completes?

CPI = $\boxed{}$

Is it the same value as the one you obtained above (in Check Yourself) with the default processor?

6 No Data Hazard

6.1 Now change to the processor "5-stage processor w/o forwarding or hazard detection" in the Select Processor dialog.

6.2 Step through the program `rtype.s` in this processor until it completes. Observe the behavior of the pipeline while you execute the program manually. Pay particular attention around the `add` instruction.

6.3 Checkoff 2

Compare the execution of the same program in this processor without hazard detection and data forwarding versus the earlier two processors. Answer the following:

- What happens to the processor pipeline when a data hazard occur?
- What is the value of `s0` after your program completes?

  Value of `s0`: $\boxed{}$
- What is the CPI value shown after your program completes?

  CPI = $\boxed{}$

  Is it the same value as the one you obtained above (in Check Yourself) with the default processor?
- Comparing the 3 processors, which one results in the highest CPI? Which one results in the correct answer?

7 Multiplication Function

In this part of the lab, you will complete a simple program to perform multiplication and test its performance in `ripes`

7.1 By using the select processor dialog, select the default "5-stage processor".

7.2 Load the file `itermult.s` in `ripes`. It contains a skeleton for you to implemeent a simple multiplication routine. In particular, you need to implement the function called `itermult`:

- The 2 values to be multiplied are passed to your function in `a0` and `a1`.
- Your function should return the answer in `a0`.
- Your function *should not* modify the values in `s0` and `s1`.

A simple way to multiply two values, a and b, is to simply add a to itself b number of times. For example $3 \times 4 = 3 + 3 + 3 + 3$.

7.3 Complete the code in `itermult.s`. Run the code to verify that it is performing the correct multiplication function. You can try different values for multiplication by changing the values of `a0` and `a1` in the first 2 lines of the program.

7.4 Checkoff 3

- Verify your code is running correctly with different input.
- Compare the performance of your code when run with (1) default 5-stage processor with forwarding and (2) 5-stage processor *without forward.* Which one has higher CPI and higher performance when running `itermult.s`?

8 Submission

Submit your answers to Checkoff 1, 2, and 3 above. Also, submit your completed `itermult.s` program.