INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

COMPUTER ORGANIZATION

LEIC-A, LEIC-T

Third Lab Assignment: Instruction Level Parallelism

Version 2.0

2025/2026

1 Introduction

The main purpose of this assignment is to provide the students with a direct and close contact to the operation of a pipelined computer architecture, as well as to the techniques that are commonly applied in order to maximize the efficiency of the developed computer programs. For that purpose, a dedicated simulation environment will be adopted: Ripes, the RISC-V simulator used for Introduction to Computer Architecture.

1.1 Environment

Ripes is a visual computer architecture simulator and assembly code editor built for the RISC-V instruction set architecture. Ripes may be used to explore concepts such as:

- How machine code is executed on a variety of microarchitectures (RV32IMC/RV64IMC based)
- How different cache designs influence performance
- How C and assembly code is compiled and assembled to executable machine code
- How a processor interacts with memory-mapped I/O

And more importantly for our use case, how a pipeline affects the performance and execution of programs.

1.2 RIPES Setup

To install Ripes on Windows:

- Download the Windows zip from the official Github releases: `https://github.com/mortbopet/Ripes/releases`
- Extract to a location of your choosing.
- Try to open Ripes.exe. It might fail to execute if you don't have the C++ runtime library installed. If that's the case, you can get it from it's download page `https://www.microsoft.com/en-us/download/details.aspx?id=48145`.

On Ubuntu (and most Linux distros):

- Download the linux 'AppImage' from the official Github releases: `https://github.com/mortbopet/Ripes/releases`
- On a terminal, add execution permissions to the downloaded file with `chmod a+x ~\/Downloads\/Ripes-v2.2.6-linux-x86_64.AppImage` (You may need to change the version number to match the one you downloaded).
- You should be able to run ripes by double clicking it in a file browser or by launching it from the terminal with `~/Downloads/Ripes-v2.2.6-linux-x86_64.AppImage`.

To install Ripes on other systems, checkout the official install instructions at `https://github.com/mortbopet/Ripes?tab=readme-ov-file#downloading--installation`

There is an online version of Ripes, but it appears to have the Pipeline Diagram feature broken (which is relevant for this lab). Please use Ripes locally.

1.3 Application program

A given mathematics library makes use of the following algorithm written in pseudo-code. The outcome value is stored in the variable *C*.

```
#define N 16

double A[N] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
double B[N] = {11,22,33,44,55,66,77,88,99,100,112,122,133,144,155,166};
double C[N] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int64 i = 0;

while(i < N) {
    C[i] = A[i] + B[i] * A[i];
    i++;
}
```

Figure 1 lists the RISC-V source code that implements this mathematical function (see file prog.s). To provide some variety in the results, the program initializes the data vectors with a few example values. Each value is represented with a 32-bit integer.

```
    .data
A:  .word 1,2,3,4,5,6,7,8,9
    .word 10,11,12,13,14,15,16
B:  .word 11,22,33,44,55,66,77
    .word 88,99,111,122,133
    .word 144,155,166
C:  .word 0,0,0,0,0,0,0,0,0
    .word 0,0,0,0,0,0,0

    .text
main:
    addi s1, zero, 0    # i = 0
    addi s2, zero, 16   # value of N
    la s3, A            # base of A
    la s4, B            # base of B
    la s5, C            # base of C

loop:
    lw t1, 0(s3)
    lw t2, 0(s4)
    mul t3, t2, t1
    add t3, t3, t1
    sw t3, 0(s5)
    addi s1, s1, 1
    addi s3, s3, 4
    addi s4, s4, 4
    addi s5, s5, 4
    bne s1, s2, loop

end:
    addi a7, zero, 10
    ecall               # Exit (syscall)
```

Figure 1: Program source code.

2 Procedure

For the following exercises, download 'prog.s' and open Ripes. The main window in Ripes has five different tabs, of which two are relevant for us: the Editor and the Processor tabs. Open the Editor tab, delete any existing code and paste in the code from 'prog.s'. This is the code you will be using for the rest of the project.

2.1 Simple execution, without data forwarding techniques

a) Switch to the Processor tab. You should see a diagram of the currently selected processor. For this first part you will need to select a processor with no data forwarding.

b) Click the `Select processor` button (chip icon on the top left) and select `RISC-V` > `32-bit` > `5-Stage processor w/o forwarding unit` and click OK. The processor diagram should be updated.

c) Return to the Editor tab. To execute the program you can:

```
Manually step the code (with F5)
Automatically step the code at a preset interval (with F6)
Run to end at maximum speed (with F8)
```

d) Back in the Processor tab, click the table button to show the pipeline diagram. Select an arbitrary loop iteration (avoid the first and the last ones) of the executed program. For each instruction of such iteration represent in Table 1 the several executed stages of the pipeline: F, D, X, M, W. Make sure to include in the diagram the first fetch of the next loop iteration, this is what defines the total clock cycles that a single loop iteration takes. Do not forget to represent every *Stall* that may occur.

e) Compute the CPI of the entire program execution. You can use Ripes' Execution Info panel to help you.

f) By analyzing the program execution, characterize the branch prediction policy that is adopted by this simulator. Justify.

2.2 Application of data forwarding techniques

a) Click the `Select processor` button (chip icon on the top left) and select `RISC-V` > `32-bit` > `5-Stage processor` and click OK. The processor diagram should be updated.

b) Represent in Table 2 the execution of the selected iteration of the program loop, by representing, for each instruction, the several executed stages of the pipeline: F, D, X, M, W. Do not forget to represent every *Stall* that may occur.

c) Summarize the program execution profile, by filling the table in the answer sheet with the execution of the entire program.

d) Evaluate the obtained speedup, when compared to the base setup, considered in Section 2.1.

2.3 Source code optimization: minimization of data and structural hazards

a) One common approach to reduce the still existing data and structural hazards is to apply re-order techniques [1] to the instruction sequence of the program. Keeping the same processor selected, analyze the time diagram of the previous section and apply the necessary re-ordering optimization techniques in order to minimize the Structural and Data stalls. Make sure that the resulting output is kept unchanged.

b) Represent in Table 3 the execution of the selected iteration of the program loop, by representing, for each instruction, the several executed stages of the pipeline: F, D, X, M, W. Do not forget to represent every *Stall* that may occur.

c) Summarize the program execution profile, by filling the table in the answer sheet with the execution of the entire program.

d) Compute the obtained speedup, when compared to the base setup, considered in Section 2.1.

2.4 Source code optimization: loop unrolling

a) One approach that is usually adopted to reduce the control hazards is to apply loop unrolling techniques [1] to the program instruction sequence. It consists in merging several loop iterations into a single one. This results in less branch instructions being executed. Higher execution speeds can be achieved at the expense of code size. By analyzing the time diagram of the previous section, apply the loop unrolling technique in order to reduce by a factor of 4 the amount of resulting control hazards. Try to optimize as much as possible the body of the loop.

b) Represent in Table 4 the execution of the selected iteration of the program loop, by representing, for each instruction, the several executed stages of the pipeline: F, D, X, M, W. Do not forget to represent every *Stall* that may occur.

c) Summarize the program execution profile, by filling the table in the answer sheet with the execution of the entire program.

d) Compute the obtained *speedup*, when compared with the base setup, considered in Section 2.1.

References

[1] David Patterson and John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 5th edition, 2014.

STUDENTS IDENTIFICATION:

Number:	**Name:**

2.1 Simple execution, without data forwarding techniques

e)

Clock cycles	
Instructions	
Average CPI	

Stalls:	- Data	
	- Structural	
Flushes:	- Branch Taken	

f)

2.2 Application of data forwarding techniques

c)

Clock cycles	
Instructions	
Average CPI	

Stalls:	- Data	
	- Structural	
Flushes:	- Branch Taken	

d)

2.3 Source code optimization: minimization of data and structural hazards

a) Attach a copy of the new assembly program.

c)

Clock cycles	
Instructions	
Average CPI	

Stalls:	- Data	
	- Structural	
Flushes:	- Branch Taken	

d)

2.4 Source code optimization: loop unrolling

a) Attach a copy of the new assembly program.

c)

Clock cycles	
Instructions	
Average CPI	

Stalls:	- Data	
	- Structural	
Flushes:	- Branch Taken	

d)

Table 1: Pipeline time diagram, without data forwarding techniques.

	INSTRUCTIONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1																																									
2																																									
3																																									
4																																									
5																																									
6																																									
7																																									
8																																									
9																																									
10																																									
11																																									
12																																									
13																																									
14																																									
15																																									
16																																									
17																																									
18																																									
19																																									
20																																									
21																																									
22																																									
23																																									
24																																									
25																																									
26																																									
27																																									
28																																									
29																																									
30																																									

Table 2: Pipeline time diagram, with data forwarding techniques.

	INSTRUCTIONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1																																									
2																																									
3																																									
4																																									
5																																									
6																																									
7																																									
8																																									
9																																									
10																																									
11																																									
12																																									
13																																									
14																																									
15																																									
16																																									
17																																									
18																																									
19																																									
20																																									
21																																									
22																																									
23																																									
24																																									
25																																									
26																																									
27																																									
28																																									
29																																									
30																																									

Table 3: Pipeline time diagram, with minimization techniques to reduce the data and structural hazards.

	INSTRUCTIONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1																																									
2																																									
3																																									
4																																									
5																																									
6																																									
7																																									
8																																									
9																																									
10																																									
11																																									
12																																									
13																																									
14																																									
15																																									
16																																									
17																																									
18																																									
19																																									
20																																									
21																																									
22																																									
23																																									
24																																									
25																																									
26																																									
27																																									
28																																									
29																																									
30																																									

Table 4: Pipeline time diagram: usage of loop unrolling minimization techniques to reduce the control hazards.

	INSTRUCTIONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1																																									
2																																									
3																																									
4																																									
5																																									
6																																									
7																																									
8																																									
9																																									
10																																									
11																																									
12																																									
13																																									
14																																									
15																																									
16																																									
17																																									
18																																									
19																																									
20																																									
21																																									
22																																									
23																																									
24																																									
25																																									
26																																									
27																																									
28																																									
29																																									
30																																									