Architetture dei Sistemi di Elaborazione 02GOLOV	Computer Architectures 02LSEYG
Laboratory 0x02	**Expected delivery of lab_02.zip including:** **- program_1.s** **- lab_02.pdf (fill and export this file to pdf)**
Delivery date 23/10/2025	

All the previous steps for compiling the source code, simulating with gem5 and visualizing the pipeline have been collected into a workspace available at:

https://github.com/cad-polito-it/ase_riscv_gem5_sim

To create your simulation workspace, you can run the following git commands:

- For HTTPS clone:

```
~/my_gem5Dir$ git clone https://github.com/cad-polito-it/ase_riscv_gem5_sim.git
```

- For SSH:

```
~/my_gem5Dir$ git clone git@github.com:cad-polito-it/ase_riscv_gem5_sim.git
```

If you wish to install on your machine the tools (without using the VM or LABINF's PCs) the repository contains information (and scripts) on installing the necessary tools out of the box (and generate a *setup_default* file accordingly). See README

IMPORTANT: The repository contains three *setup_default* files:

- *setup_default*: an example of out of box installation and different tool paths.
- *setup_default.vm*: tool paths used in the virtual machines.
- *setup_default.labinf*: tool paths used in the LABINF machines.

Follow the HOWTO instructions available on the GitHub Repository for simulating a program. You simply run the following command:

```
~/ase_riscv_gem5_sim$ ./simulate.sh
```

And select the program you want to simulate and see on the pipeline visualizer.

A useful list of RISC-V instructions can be found at:

https://msyksphinz-self.github.io/riscv-isadoc/#_rv32i_rv64i_instructions

- Exercise 1

The `riscv_in_order_hen_patt.py` (in the gem5 folder inside the workspace) file contains the pipeline configuration. Here you can modify the operation latency and the issue latency of integer and floating-point functional units (ALU, Multiplier, Divider).

You must configure the Gem5 simulator with the ***Initial Configuration*** as described below:

```
INTEGER_ALU_LATENCY = 1
INTEGER_MUL_LATENCY = 1
INTEGER_DIV_LATENCY = 1
FLOAT_ALU_LATENCY = 8
FLOAT_MUL_LATENCY = 24
FLOAT_DIV_LATENCY = 42
```

Write an assembly program (**`program_1.s`**) for the *RISCV* architecture described before being able to implement the following high-level code:

```
for (i = 31; i >= 0; i--) {
        v4[i] = v1[i]*v1[i] – v2[i];
        v5[i] = v4[i]/v3[i] – v2[i];
        v6[i] = (v4[i]-v1[i])*v5[i];
}
```

Assume that the vectors v1[], v2[], and v3[] have been previously allocated in memory and contain 32 single-precision **floating-point values;** also assume that v3[] does not contain any '0'. Additionally, the vectors v4[], v5[], v6[] are empty vectors allocated in memory.

Calculate the data memory footprint of your program:

Data	Number of bytes
V1	128
V2	128
V3	128
V4	128
V5	128
V6	128
Total	768

- Calculate the CPU performance equation (CPU time) of the above program by assuming a clock frequency of 15 MHz:

$$\textbf{CPU time} = (\sum_{i=1}^{n} \textbf{CPI}_i \times \textbf{IC}_i) \times \textbf{Clock cycle period}$$

By definition:

- CPI_i is equal to the number of clock cycles required by the related functional unit to execute the instruction (EX stage);

- IC_i is the number of times an instruction is repeated in the referenced source code.

- Recalculate the CPU performance equation assuming that you can triple the speed of just one unit at a time:
  b) FP multiplier (MUL) unit latency: 24 → 8 clock cycles
  c) FP divider (DIV) unit latency: 42 → 14 clock cycles

Table 1: Calculate the CPU time ***by hand***

	Initial CPU time (a)	CPU time (b – MUL speeded up)	CPU time (c – DIV speeded up)
program_1.s	0.2799 ms	0.2116 ms	0.2201 ms

- Using the simulator, calculate the CPU time again and fill in the following table:

Table 2: Collect the CPU time ***using the simulator***

	Initial CPU time (a)	CPU time (b – MUL speeded up)	CPU time (c – DIV speeded up)
program_1.s	0.2658 ms	0.1975 ms	0.2061 ms

Are there any differences? If so, where and why? If not, please provide some comments in the box below:

Your answer: Le differenze nascono perché nel calcolo a mano vengono sommate latenze in serie, mentre nel simulatore gem5 molte si sovrappongono: durante `fmul` e `fdiv` istruzioni indipendenti avanzano e nascondono cicli.

- Exercise 2

Using the Gem5 simulator, validate experimentally the Amdahl's law, defined as follows:

$$\text{speedup}_{\text{overall}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} = \frac{1}{(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}}$$

a. Using the program developed before **program_1.s**
b. Modify the processor architectural parameters related (in `riscv_in_order_hen_patt.py`) to multicycle instructions latency in the following way:

Configuration 1:

```
INTEGER_ALU_LATENCY = 1
INTEGER_MUL_LATENCY = 1
INTEGER_DIV_LATENCY = 1
FLOAT_ALU_LATENCY = 6
FLOAT_MUL_LATENCY = 20
FLOAT_DIV_LATENCY = 38
```

Configuration 2:

```
INTEGER_ALU_LATENCY = 1
INTEGER_MUL_LATENCY = 1
INTEGER_DIV_LATENCY = 1
FLOAT_ALU_LATENCY = 4
FLOAT_MUL_LATENCY = 16
FLOAT_DIV_LATENCY = 30
```

Configuration 3:

```
INTEGER_ALU_LATENCY = 1
INTEGER_MUL_LATENCY = 1
INTEGER_DIV_LATENCY = 1
FLOAT_ALU_LATENCY = 2
FLOAT_MUL_LATENCY = 4
FLOAT_DIV_LATENCY = 8
```

Compute both manually (using the Amdahl's Law) and with the simulator the speed-up for any one of the previous processor configurations. Compare the obtained results and complete the following table.

Table 5: **program_1.s speed-up computed *by hand and by simulation***

Proc. Config. / **Speed-up comp.**	Initial config. [c.c.]	Config. 1 [c.c.]	Config. 2 [c.c.]	Config. 3 [c.c.]
By hand	1	1.159	1.439	3.348
By simulation	1	1.147	1.406	3.230