Architetture dei Sistemi di Elaborazione 02GOLOV	Computer Architectures 02LSEYG
Laboratory 0x04	Expected delivery of lab_04.zip must include: - `program_1.s`, `program_1_a.s`, and `program_1_b.s` - This file, filled with information and possibly compiled in a **PDF** format. Delivery date: November 7th 2025

This lab will explore some of the concepts seen during the lessons, such as hazards, rescheduling, and loop unrolling. The first thing to do is to configure the GEM5 simulator with the ***Initial Configuration*** provided below:

```
INTEGER_ALU_LATENCY = 1
INTEGER_MUL_LATENCY = 1
INTEGER_DIV_LATENCY = 1
FLOAT_ALU_LATENCY = 4
FLOAT_MUL_LATENCY = 8
FLOAT_DIV_LATENCY = 20
```

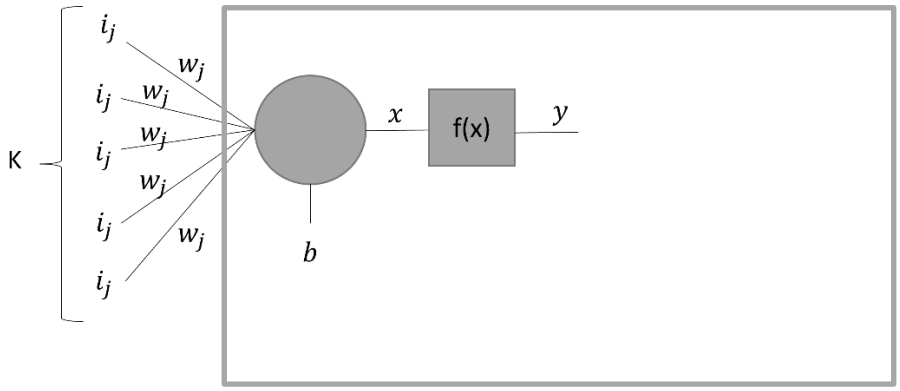
1) Write an assembly program (`program_1.s`) for the RISCV architecture able to compute the output (y) of a **neural computation** (see the Fig. below):

$$x = \sum_{j=0}^{K-1} i_j * w_j + b$$
$$y = f(x)$$

where, to prevent the propagation of NaN (Not a Number), the activation function f is defined as:

$$f(x) = \begin{cases} 0, & if\ the\ exponent\ part\ of\ x\ is\ equal\ to\ 0x7FF \\ x, & otherwise \end{cases}$$

Assume the vectors i and w respectively store the inputs entering the neuron and the weights of the connections. They contain $K=16$ single precision **floating point** elements. Assume that b is a single precision **floating point** constant and is equal to $0xab$, and y is a single precision **floating point** value stored in memory.
Compute y.



Given the *Base Configuration*, run your program and extract the following information.

	Number of clock cycles	Total Instructions	CPI (Clock per Instructions)
`program_1.s`	342	163	2.098

a. Optimize the program by re-scheduling instructions to eliminate as many hazards as possible. Manually calculate the number of clock cycles for the new program (**`program_1_a.s`**) to execute and compare the results with those obtained by the simulator.

b. Unroll the program (**`program_1_a.s`**) two times; If necessary, re-schedule instructions and increase the number of registers used. Manually calculate the number of clock cycles to execute the new program (**`program_1_b.s`**) and compare the results obtained with those obtained by the simulator.

Complete the following table with the obtained results:

Program	**program_1.s**	**program_1_a.s**	**program_1_b.s**
Clock cycles by hand	261	224	166
Clock cycles by simulation	342	237	201

Collect the Cycles Per Instruction (CPI) from the simulator for different programs

	program_1.s	**program_1_a.s**	**program_1_b.s**
CPI	2.098	1.419	1.467

Compare the results obtained in 1) and provide some explanation if the results are different.

Eventual explanation: Le discrepanze tra stima a mano e con l'uso del simulatore derivano da hazard strutturali (contenzione in write-back), dipendenze RAW/WAW amplificate da latenze FP, penalità di branch del datapath base e dall'espansione di pseudoistruzioni; nel codice iniziale questi effetti sono marcati, la rischedulazione li attenua, mentre l'unrolling riduce l'overhead di controllo ma intensifica le collisioni su WB e memoria, che il simulatore modella con stall aggiuntivi.

c. Try different unrolls for the program (**`program_1_a.s`**); If necessary, re-schedule instructions and increase the number of registers used. Manually calculate the number of clock cycles to execute the new program (**`program_1_b.s`**) and compare the results obtained with those obtained by the simulator.

Complete the following table with the obtained results:

Program	Number of Loop Unrolling	Code size [bytes]	Clock Cycles [CC]
`program_1_a.s`	-	188	342
	2	228	201
	4	276	201
	8	372	175
	16	412	149

In order to calculate the code size of your program you can open in *./programs/you_program/your_program.dump* file to retrieve the compiled disassembled code of your executable in which you have the effective addresses from which the code will be executed.

For example (on the right), for *your_program* you have your disassembled file on 32 bit addresses.