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):

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

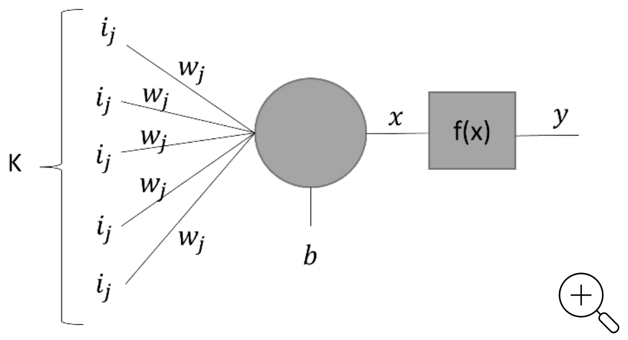
$$y = f(x)$$

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

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

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`	354	188	1.88298

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	352	333	277
Clock cycles by simulation	354	335	279

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

	program_1.s	**program_1_a.s**	**program_1_b.s**
CPI	1.88298	1.78191	1.62209

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

Eventual explanation: I risultati variano per il riscaldamento delle cache che introduce 5 stalli e I 3 colpi di clock finali che invece non vengono eseguiti, portando la differenza a 2.

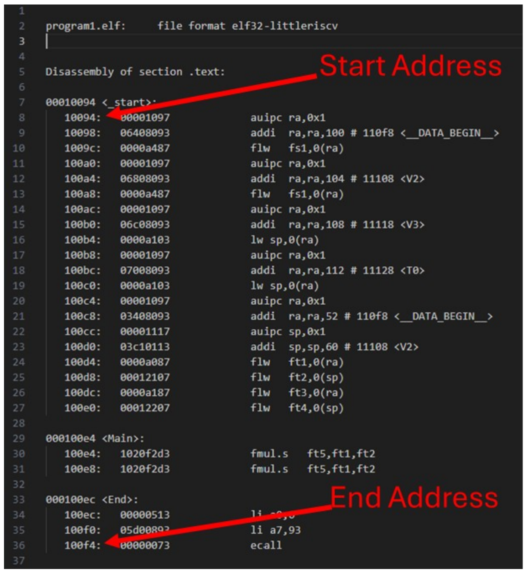
c. Try different unrolls for the program (`program_1_a.s`); If necessary, re-schedule instructions and increase the number of registers used. Manually calculate the number of clock cycles to execute the new program (`program_1_b.s`) and compare the results obtained with those obtained by the simulator.

Complete the following table with the obtained results:

Program	**Number of Loop Unrolling**	**Code size [bytes]**	**Clock Cycles [CC]**
`program_1_a.s`	-	169	335
	2	201	279
	4	265	245
	8	393	233
	16	649	229

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.



```

program1.elf:     file format elf32-littleriscv


Disassembly of section .text:

00010094 <_start>:
   10094:   00001097            auipc ra,0x1
   10098:   06408093            addi  ra,ra,100 # 110f8 <__DATA_BEGIN__>
   1009c:   0000a487            flw   fs1,0(ra)
   100a0:   00001097            auipc ra,0x1
   100a4:   06808093            addi  ra,ra,104 # 11108 <V2>
   100a8:   0000a487            flw   fs1,0(ra)
   100ac:   00001097            auipc ra,0x1
   100b0:   06c08093            addi  ra,ra,108 # 11118 <V3>
   100b4:   0000a103            lw sp,0(ra)
   100b8:   00001097            auipc ra,0x1
   100bc:   07008093            addi  ra,ra,112 # 11128 <T0>
   100c0:   0000a103            lw sp,0(ra)
   100c4:   00001097            auipc ra,0x1
   100c8:   03408093            addi  ra,ra,52 # 110f8 <__DATA_BEGIN__>
   100cc:   00001117            auipc sp,0x1
   100d0:   03c10113            addi  sp,sp,60 # 11108 <V2>
   100d4:   0000a087            flw   ft1,0(ra)
   100d8:   00012107            flw   ft2,0(sp)
   100dc:   0000a187            flw   ft3,0(ra)
   100e0:   00012207            flw   ft4,0(sp)

000100e4 <Main>:
   100e4:   1020f2d3            fmul.s   ft5,ft1,ft2
   100e8:   1020f2d3            fmul.s   ft5,ft1,ft2

000100ec <End>:
   100ec:   00000513            li a0,0
   100f0:   05d00893            li a7,93
   100f4:   00000073            ecall

```