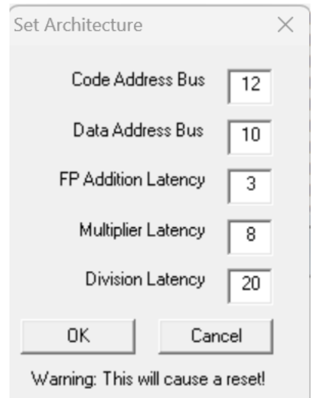
Architetture dei Sistemi di Elaborazione **Laboratory 2**	Delivery date: October 25th 2023 Expected delivery of lab_02.zip must include: - **`program_1.s`** and **`program_2.s`** - This file, filled with information and possibly compiled in a pdf format.

Please, configure the winMIPS64 simulator with the *Base Configuration* provided in the following (*in italics not user controllable configuration*):

- Code address bus: 12
- Data address bus: 10
- Pipelined FP arithmetic unit (latency): 3 stages
- Pipelined multiplier unit (latency): 8 stages
- divider unit (latency): not pipelined unit, 20 clock cycles
- Forwarding is enabled
- Branch prediction is disabled
- Branch delay slot is disabled
- *Integer ALU: 1 clock cycle*
- *Data memory: 1 clock cycle*
- *Branch delay slot: 1 clock cycle*.



1) Write an assembly program (**`program_1.s`**) for the *winMIPS64* architecture described before able to implement the following piece of code described at high-level:

```
for (i = 0; i < 64; i++) {
        v5[i] = ((v1[i]* v2[i]) + v3[i]) + v4[i];
        v6[i] = v5[i]/(v4[i]+v1[i]);
        v7[i] = v6[i]*(v2[i]+v3[i]);
}
```

Assume that the vectors v1[], v2[], v3[], and v4[] are allocated previously in memory and contain 64 double precision **floating point** values; assume also that v1[] and v4[] do not contain 0 values. Additionally, the vectors v5[], v6[], v7[] are empty vectors also allocated in memory.

<u>Calculate</u> the data memory footprint of your program:

Data	Number of Bytes
V1	512
V2	512
V3	512
V4	512
V5	512
V6	512
V7	512
Total	3584

Are there any issues? Yes, where and why? No ? Do you need to change something?

Your answer:
The configuration for the winMIPS64 simulator provided in the previous page suggests to insert “10” as Data Address Bus value. By doing so, the available data memory is equal to 1024 bytes, which clearly are not enough to store all the arrays.

This problem can be solved by increasing the Data Address Bus value to “12”, in this case we have 4096 bytes available to store variables.

ATTENTION: winMIPS64 has a limitation due to the underlying software.
There is a limitation in the string length when declaring a vector. Split the vectors elements in multiple lines (it also increases the readability) .

```
Example: my_fancy_vector:  .byte 4, 5 ,7, 8
                           .byte 5,77, 8
                           .byte ……
```

a. Compute the CPU performance equation (CPU time) of the previous program following the next directions, assume a clock frequency of 1MHz:

$$\textbf{CPU time} = \left(\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{CPI_i} \times \mathbf{IC_i}\right) \times \textbf{Clock cycle period}$$

- Count manually, the number of the different instructions (IC_i) executed in the program
- Assume that the CPI_i for every type of instructions equals the number of clock cycles in the instruction EXE stage, for example:
    - integer instructions CPI = 1
    - LD/SD instructions CPI = 1
    - FP MUL instructions CPI = 8
    - FP DIV instructions CPI = 20
    - …

b. Compute by hand again the CPU performance equation assuming that you can improve the FP Multiplier or the FP Divider by speeding up by 2 only one of the units at a time:

- Pipelined FP multiplier unit (latency): 8 → 4 stages
  Or
- FP Divider unit (latency): not pipelined unit, 20 → 10 clock cycles

Table 1: CPU time by hand

	CPU Time initial (a)	CPU Time (b – MUL speed up)	CPU Time (b – DIV speed up)
`program_1.S`	3.715 ms	3.203 ms	3.075 ms

c. Using the simulator calculate again the CPU time and complete the following table:

Table 2: CPU time using the simulator

	CPU Time initial (a)	CPU Time (b – MUL speed up)	CPU Time (b – DIV speed up)
`program_1.S`	3.206 ms	2.758 ms	2.566 ms

Are there any difference? If yes, where and why? If Not, provide some comments in the following:

Your answer:

Yes, by running the program using the winMIPS64 simulator we can notice a little enhancement in the CPU time values.
This probably happens because the simulator is able to execute in parallel more instructions that don't share registers as operands. In other words, if an instruction doesn't depend on the result of the previous one, it is possible to execute it without stalls.
On the other hand, by manually calculating the CPU time, we are not considering all the possible instructions that can be executed in parallel.

d. Using the simulator and the *Base Configuration*, disable the Forwarding option and compute how many clock cycles the program takes to execute.

Table 3: forwarding disabled

	Number of clock cycles	IPC (Instructions Per Clock)
`program_1.S`	3783	0.288

Enable one at a time the **optimization features** that were initially disabled and collect statistics to fill the following table (fill all required data in the table before exporting this file to pdf format to be delivered).

Table 4: **Program performance for different processor configurations**

Program	Forwarding		Branch Target Buffer		Delay Slot		Forwarding + Branch Target Buffer	
	IPC	CC	IPC	CC	IPC	CC	IPC	CC
Program_1.S	0.340	3206	0.293	3724	0.288	3784	0.347	3147

2) Using the WinMIPS64 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 with multicycle instructions (`Menu→Configure→Architecture`) in the following way:

1) Configuration 1
   - Starting from the *Base Configuration*, change only the FP addition latency to 6
2) Configuration 2
   - Starting from the *Base Configuration*, change only the Multiplier latency to 4
3) Configuration 3
   - Starting from the *Base Configuration*, change only the division latency to 10

Compute by hand (using the Amdahl's Law) and using 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.	Base config. [c.c.]	Config. 1	Config. 2	Config. 3
`By hand`	3715	0.810	1.062	1.030
`By simulation`	3206	0.943	1.162	1.249

3) Write an assembly program (**`program_2.s`**) for the winMIPS64 architecture able to compute the output (y) of a **neural computation** (see the Fig. below):

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

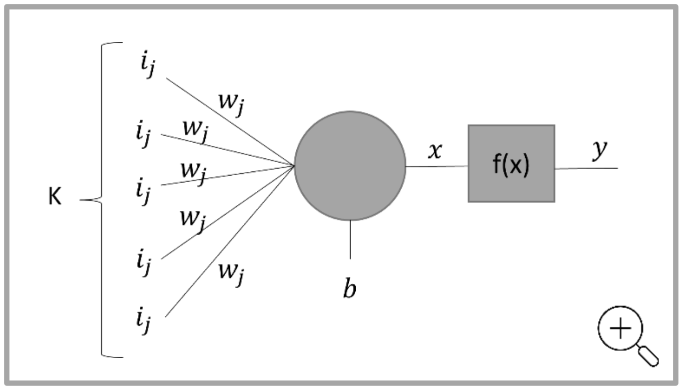
$$y = f(x)$$

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

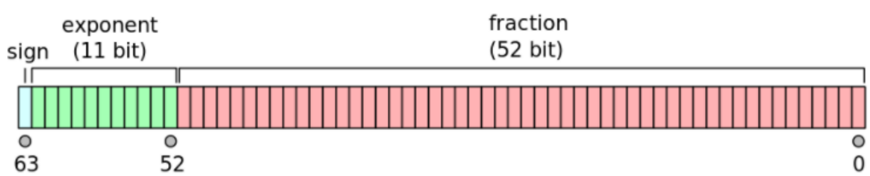
$$f(x) = \begin{cases} 0, & \text{if the exponent part of } x \text{ is equal to } 0x7ff \\ x, & \text{otherwise} \end{cases}$$

Assume the vectors i and w respectively store the inputs entering the neuron and the weights of the connections. They contain $K=30$ double precision **floating point** elements. Assume that b is a double precision **floating point** constant and is equal to $0xab$, and y is a double precision **floating point** value stored in memory.

Compute y.



Below is reported the encoding of IEEE 754 double-precision binary floating-point format:



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

	Number of clock cycles	IPC (Instructions Per Clock)	CPI (Clock per Instructions)
`program_2.S`	355	0.628	1.592