

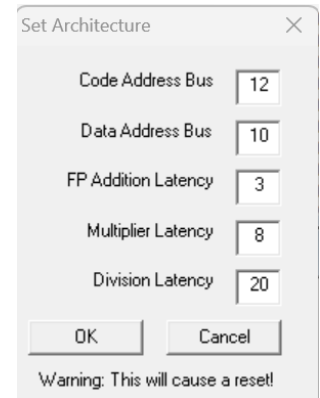
Laboratory
2

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.

Calculate 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: con data address bus a 10 non mi riesce ad allocare v3, invece se viene messo a 12 non sorge più il problema di allocazione.
Questo perché con data address bus a 10 si possono allocare $2^{10}=1024$ bytes, considerando che ho 4 vettori da 64 elementi, 8bit ciascuno, quindi 1 vettore occupa 512 byte, mi serviranno almeno 2048byte riservati. Aumentando il data address bus a 12 si avranno $2^{12}=4096$ e saranno sufficienti per allocare i 4 vettori

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:

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

- 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 $\text{CPI} = 1$
 - LD/SD instructions $\text{CPI} = 1$
 - FP MUL instructions $\text{CPI} = 8$
 - FP DIV instructions $\text{CPI} = 20$
 - ...

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): $8 \rightarrow 4$ stages
Or
- FP Divider unit (latency): not pipelined unit, $20 \rightarrow 10$ 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	3.78ms	3.27ms	3.14ms

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	3.47ms	2.96ms	2.82ms

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

Your answer: ci sono differenze in tutte e 3 le configurazioni. Per quanto riguarda quelle a mano ho calcolato sempre più cicli di clock rispetto a quelli calcolati dal simulatore. Credo che le differenze siano date grazie al simulatore che risolve i pipeline hazards grazie al forwarding, risparmiando cicli di clock

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	4748	0.24

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	0.33	3467	0.24	4750	0.26	83	0.34	3406

- 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.	Base config. [c.c.]	Config. 1	Config. 2	Config. 3
Speed-up comp.				
By hand	3780	0.91	1.04	1.09
By simulation	3470	0.9	1.17	1.22

- 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):

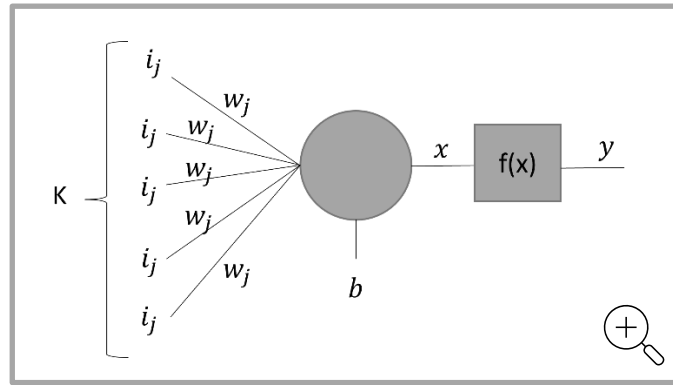
$$x = \sum_{j=0}^{K-1} i_j * w_j + b$$

$$y = f(x)$$

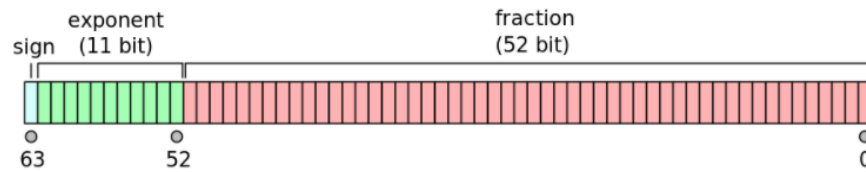
where, to prevent the propagation of NaN (Not a Number), the activation function f is defined as:

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

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	356	0.63	1.574