Architetture dei Sistemi di Elaborazione

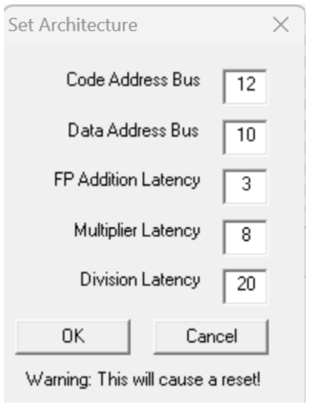
Delivery date:
October 25th 2023

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.*


Set Architecture

Code Address Bus: 12
Data Address Bus: 10
FP Addition Latency: 3
Multiplier Latency: 8
Division Latency: 20

Warning: This will cause a reset!

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: È presente un problema per il caricamento di valori floating point in memoria, il quale si è potuto risolvere modificando il valore (e quindi il numero di linee simulate) del “Data Adress Bus” dal valore predisposto di 10 a 12.

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:

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

- 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 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`	3,840ms	3,328ms	3,200ms

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`	3,256ms	2,801ms	2,606ms

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

Your answer: Si, ci sono differenze e sono presenti in tutti i CPU time calcolati. La motivazione ricade principalmente sul fatto che con il calcolo a mano non si tiene in considerazione il forwarding del dato e quindi delle ottimizzazioni in ambito di parallellismo di determinate sequenze di operazioni.
Esempio:
due addizioni fp consecutive indipendenti non si dovrebbero calcolare come 6 Clock cycles ma come 4: le due operazioni verranno eseguito in parallelo, una in ritardo di un clock cycle rispetto all'altra, per cui alla fine dell'esecuzione della prima istruzione, la seconda impiegherà un singolo CLK per terminare la fase di EXE.
In generale sia nel calcolo a mano che dal simulatore si vede un trend decrescente, come era intuitivo aspettarsi, sia nel CPU Time con il MUL velocizzato e sia con il DIV velocizzato. In entrambi i casi il simulatore calcola un numero di clock minore di quello calcolato da me a mano a causa della motivazione spiegata sopra.
P.S.
nel calcolo effettuato a mano non conteggio il delay iniziale che rappresenta 5 clock cycle per la prima istruzione invece del metodo più semplicistico di considerare da subito il processore a regime in pipeline considerando solo la durata della fase EXE.

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`	4427	0,25

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`	`0,34`	`3256`	`0,25`	`4429`	`0,26`	`4427`	`0,35`	`3195`

2) Using the WinMIPS64 simulator, validate experimentally the Amdahl's law, defined as follows:

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

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`	3840	0,833	1,154	1,20
`By simulation`	3256	0,943	1,162	1,25

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):

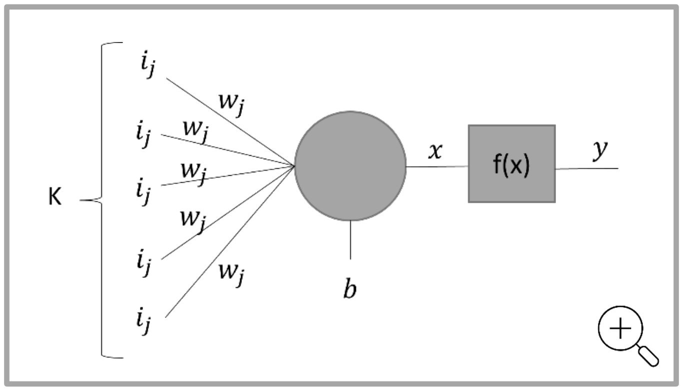
$$x = \sum_{j=0}^{K-1} i_j * w_j + b$$

$$y = f(x)$$

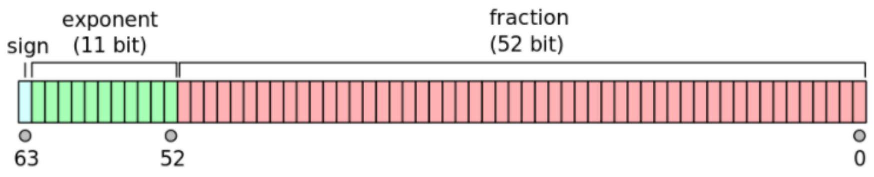
where, to prevent the propagation of NaN (Not a Number), the activation function *f* is defined as:

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

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`	336	0,682	1,467