|  |  |
| --- | --- |
| **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*):

* Immagine che contiene testo, schermata, Carattere, numero

  Description automatically generatedCode 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: Non c’è abbastanza memoria; si aumenta il data bus a 12 bit, per ottenere uno spazio di indirizzamento di 4K bytes, adatti a contenere i 3584 bytes necessari. |

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 ……

* + 1. Compute the CPU performance equation (CPU time) of the previous program following the next directions, assume a clock frequency of 1MHz:
       - Count manually, the number of the different instructions (*ICi*) executed in the program
       - Assume that the *CPIi* 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
         * …
    2. 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 | 3.782 ms | 3.270 ms | 3.142 ms |

* + 1. 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 | 3.468 ms | 2.956 ms | 2.828 ms |

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

|  |
| --- |
| Your answer: Sì, sono presenti differenze.  Il simulatore opera con pipelining e forwarding: nonostante la presenza di stalli (dovuti ad esempio alle dipendenze di dato o agli accessi concorrenti allo stesso modulo), la possibilità di parallelizzare le operazioni consente di terminare l’esecuzione del programma rispetto ai calcoli effettuati manualmente. In questo caso, non si considerava la possibilità di stalli, ma nemmeno la parallelizzazione delle operazioni. |

* + 1. 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 | 4749 | 0.244 |

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 | 0.334 | 3468 | 0.247 | 4688 | 0.244 | 4749 | 0.340 | 3407 |

1. Using the WinMIPS64 simulator, validate experimentally the Amdahl’s law, defined as follows:



1. Using the program developed before: **program\_1.s**
2. 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

* + - * 1. Configuration 2

Starting from the *Base Configuration*, change only the Multiplier latency to 4

* + - * 1. 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** | 3782 | 0.80 | 1.37 | 1.20 |
| **By simulation** | 3468 | 0.90 | 1.17 | 1.23 |

1. 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):

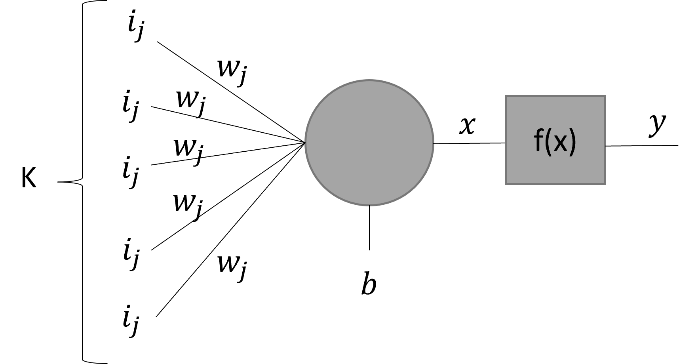
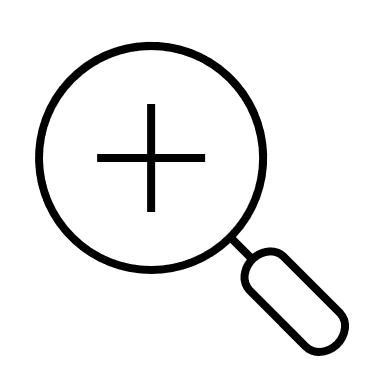
*x =*

*y = f(x)*

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

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:

Immagine che contiene testo, musica, pianoforte

Descrizione generata automaticamente

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | Number of clock cycles | Total Instructions | CPI (Clock per Instructions) |
| program\_2.S | 358 | 257 | 1.393 |