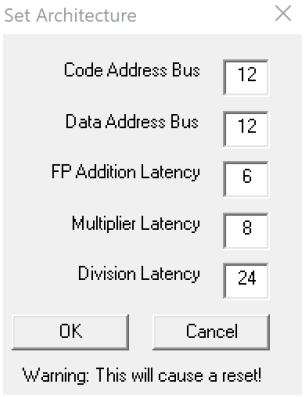
| Architetture dei Sistemi di Elaborazione<br>02GOLOV [GRB-ZZZ] | Delivery date:<br>**1st November 2022** |
|---|---|
| **Laboratory 3** | Expected delivery of lab_03.zip must include:<br>- `program_1_a.s`, `program_1_b.s` and `program_1_c.s`<br>- this file compiled and if possible in pdf format. |

Please, configure the winMIPS64 simulator with the *Base Configuration* provided in the following:

- Code address bus: 12
- Data address bus: 12
- Pipelined FP arithmetic unit (latency): 6 stages
- Pipelined multiplier unit (latency): 8 stages
- Divider unit (latency): not pipelined unit, 24 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) Starting from the assembly program you created in the previous lab called **`program_1.s`**:

```
for (i = 0; i < 60; 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]);
}
```

   a. Detect manually the different data, structural and control hazards that provoke a pipeline stall

   b. Optimize the program by re-scheduling the program instructions in order to eliminate as many hazards as possible. Compute manually the number of clock cycles the new program (**`program_1_a.s`**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

   c. Starting from **`program_1_a.s`**, enable the *branch delay slot* and re-schedule some instructions in order to improve the previous program execution time. Compute manually the number of clock cycles the new program (**`program_1_b.s`**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

   d. Unroll 3 times the program (**`program_1_b.s`**), if necessary re-schedule some instructions and increase the number of used registers. Compute

manually the number of clock cycles the new program (**program_1_c.s**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

Complete the following table with the obtained results:

| Program / Clock cycle computation | program_1.s | program_1_a.s | program_1_b.s | program_1_c.s |
|---|---|---|---|---|
| By hand | 2832 | 2681 | 2652 | 1150 |
| By simulation | 2832 | 2681 | 2652 | 1150 |

Compare the results obtained in point 1, and provide some explanation in the case the results are different.

Eventual explanation:

| | Data Hazard | Structural Stalls | Branch Taken Stalls |
|---|---|---|---|
| program 1.s | 2341 | 240 | 31 |
| program 1 a.s | 1951 | 210 | 31 |
| program 1 b.s | 1951 | 210 | 0 |
| program 1 c.s | 449 | 104 | 0 |

Initially the program was presented with the highest number of RAW criticality stalls, mainly due to two factors: the data dependency between the division instruction and the subsequent multiplication instruction on registers f6 and f7 and more seriously between the division and multiplication instruction on registers f13 and f14 as in addition to the stalls due to them are also added stalls by the two successive store instructions on the same registers. This was resolved by trying to cushion the stalls due to the division and multiplication instruction by inserting store operations made in other parts of the program but independent of these instructions and eliminating some minor stall at the beginning of the program between the instruction of addition (f5) and multiplication (f5).

Instead in program b you are aimed at improving the number of clock cycle by making a re-schedule only of the rising index counter addition instruction having had the opportunity to activate the deloy slot branch and thus to take advantage of this possibility to execute that independent statement anyway, while on the other instructions nothing else could be done.

Finally, in the program c there was a further unroll. In program 1, and consequently also in the others, we had already performed a unroll of the body of the loop with the aim of having a greater performance based more on the different indices (one increasing and one decreasing) and on the decrease above all the number of stalls due to the dependencies of data, this thanks also to the register renaming that has allowed to be able to insert the code without dependencies on the name of the registers. Instead, in the program c the same technique was applied but increasing the number of indices, that is two indexes increasing and two indices decreasing but with different starting points, thus bringing the number of loop execution from a number of 30 times to a number of only 15 times.