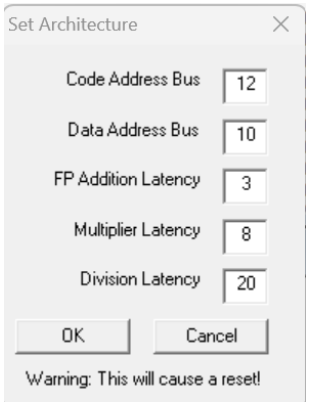
Architetture dei Sistemi di Elaborazione 02GOLOV	Delivery date: **9th November 2023**
Laboratory 3	Expected delivery of lab_03.zip must include: - `program_1_a.s`, `program_1_b.s` and `program_1_c.s` - this file compiled and if possible in pdf format.

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): 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) Enhance the assembly program you created in the previous lab called **`program_1.s`**:

```
int m=1 /* 64  bit */
double k,p
for (i = 0; i < 64; i++){
        if (i is even) {
            p= v1[i] * ((double)( m<< i)) /*logic shift */
            m = (int)p
        } else {
        /*  i is odd */
             p= v1[i] / ((double)m* i))
             k =  ((float)((int)v4[i]/ 2^i)
        }

        v5[i] = ((p * v2[i]) + v3[i])+v4[i];

        v6[i] = v5[i]/(k+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 2 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	5318	4262	4199	3687
By simulation	4966	3910	3847	3335

Collect the IPC (from the simulator) for different programs.

	program_1.s	program_1_a.s	program_1_b.s	program_1_c.s
IPC	0.355	0.451	0.458	0.452

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

Eventual explanation:

The results that I obtained among the different experiments are different. In particular:

- **program 1** is the slowest because there are no optimizations enabled (apart from the forwarding), instructions are not scheduled in the best possible way (so we have a great amount of data and structural hazards) and the branch instructions take 2 cycles each one when the branch is taken.
- **program 1a** is much faster than the previous one due to a rescheduling of the instructions in order to avoid the majority of stalls caused by data and structural dependencies.
- **program 1b** has got the branch delay slot optimization enabled, in this way is possible to save a cycle of clock for each branch taken by putting a useful instruction after the branch one (because it will be executed anyway instead of being flushed from the pipeline).
- **program 1c** implements the loop unrolling feature, which consists in expanding the loop by replicating the body part multiple times. In this way there are more sequential

instructions that are being executed, so it is possible to better exploit the CPU parallelism mechanisms and to reduce the overhead due to the branch instructions.