Architetture dei Sistemi di Elaborazione **02GOLOV [A-L]** **Laboratory** **3**	Delivery date: November 5, 2020 Expected delivery of lab_03.zip must include: - `program_2_a.s`, `program_2_b.s` and `program_2_c.s` - this file compiled and if possible in pdf format.

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

- Code address bus: 12
- Data address bus: 12
- Pipelined FP arithmetic unit (latency): 4 stages
- Pipelined multiplier unit (latency): 8 stages
- Divider unit (latency): not pipelined unit, 12 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_2.s`** that implements the following behavior:

```
for (i = 0; i < 30; i++){
        v5[i] = (v1[i] - v2[i]) * v3[i];
        v6[i] = v5[i]*v3[i]
        v7[i] =(v6[i] + v4[i]) * v2[i]:
}
```

Assume that v1, v2, v3 and v4 are vectors allocated in memory populated with values chosen by you. V5, v6, v7 are initially empty and allocated in memory. Each array holds double-precision floating –point values (64-bit values). You are requested to:

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

b. Optimize the program **`program_2.s`** by re-scheduling the instructions in order to eliminate as much hazards as possible. Compute manually the number of clock cycles that the new program (**`program_2_a.s`**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

c. Starting from **`program_2_a.s`**, enable the *branch delay slot* and re-schedule again the code in order to positively exploit the branch delay slot, or add NOP operations that avoid the code to lost its functionalities. Compute manually the number of clock cycles that the new program (**`program_2_b.s`**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

d. Unroll 3 times the body of the for loop in **`program_2_b.s`** so that there are 3 copies of the body. After the unrolling, reschedule again the instructions for further improving the program performance. Compute manually the number of clock cycles that the new program (**`program_2_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_2.s`	`program_2_a.s`	`program_2_b.s`	`program_2_c.s`
`By hand`	1207	1207	1179	540
`By simulation`	1207	1207	1179	540

Compare the results obtained by hand and by simulation. Use the empty box below to provide some explanation in the case the results are different.

Eventual explanation about results: There is too much data dependence between a computation and the next one that the code can't be optimized by rescheduling of instructions.

Regarding the code optimization with "branch delay slot" enabled, the only way to optimize it is to duplicate the first load instruction and put one before the loop and the other one after the branch instruction.

For the loop unrolling, I used a bit of register renaming in order to exploit better the optimization together with rescheduling of instructions.