

Laboratory
3

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:

- Detect manually the different data, structural and control hazards that provoke a pipeline stall
- 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.
- 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	program_2.s	program_2_a.s	program_2_b.s	program_2_c.s
Clock cycle computation				
By hand	<u>1266cc</u>	<u>1236cc</u>	<u>1177cc</u>	<u>1133cc</u>
By simulation	<u>1236cc</u>	<u>1206cc</u>	<u>1147cc</u>	<u>1113cc</u>

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:

Teoria:

sub.d f5,f1,f2 ;f5 = v1[i]-v2[i] FDaaaaMW

...

mul.d f5,f5,f3 ;f5 = (v1[i]-v2[i])*v3[i] FDSmmmmmmmmMW

Winmips:

sub.d f5,f1,f2 ;f5 = v1[i]-v2[i] FDaaaaMW

...

mul.d f5,f5,f3 ;f5 = (v1[i]-v2[i])*v3[i] FDmmmmmmmmMW

Winmips inizia immediatamente l'esecuzione dopo che la sub.d ha finito la fase di execute, mentre sarebbe meglio attendere un colpo di clock, per evitare di perdere il valore di f5 nel caso in cui mul.d lanci un'eccezione appena entra nella fase di execute. (1cc*30 cicli=30cc)