Architetture dei Sistemi di Elaborazione	Delivery date: October 29th 2021
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 *Base Configuration* provided in the following:

- 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) Starting from the assembly program you created in the previous lab called **`program_2.s`**,:

```
for (i = 0; i < 40; i++){
        v5[i] = v1[i]+(v2[i] * v3[i]);
        v6[i] = v5[i]*v4[i];
        v7[i] = v6[i]/v2[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 much hazards as possible. Compute manually the number of clock cycles 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 some instructions in order to improve the previous program execution time. Compute manually the number of clock cycles the new program (**`program_2_b.s`**) requires to execute, and compare the obtained results with the ones obtained by the simulator.

   d. Unroll 4 times the program (**`program_2_b.s`**), if necessary re-schedule some instructions and renaming the used registers. Compute manually the number of clock cycles the new program (**`program_2_c.s`**) requires to

execute, and compare the obtained results with the ones obtained by the simulator.

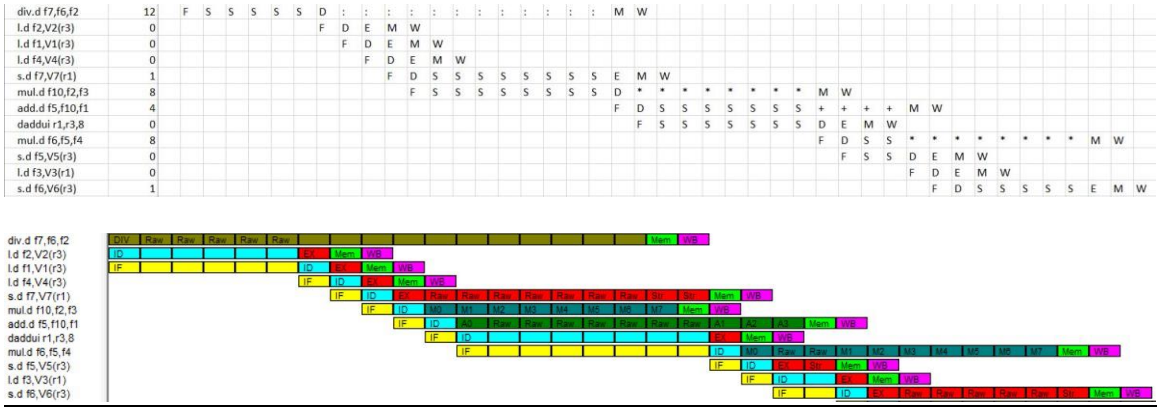
Complete the following table with the obtained results:

Program / Clock cycle computation	`program_2.s`	`program_2_a.s`	`program_2_b.s`	`program_2_c.s`
`By hand`	1656	1616	1616	1416
`By simulation`	1658	1618	1618	1158

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

Eventual explanation:

I risultati ottenuti calcolando a mano i cicli di clock impiegati per I programmi 2 , 2_a, 2_b, risultano coerenti con i risultati ottenuti effettuando una simulazione con winmips; per il programma 2_c invece la discrepanza tra I risultati è notevole ma questo è dovuto al fatto che winMips gestisce gli stalli della pipeline in modo diverso da quanto visto a lezione. Nel caso ci sia una dipendenza di dato per un operazione aritmetica calcolando a mano in classe abbiamo deciso di stallare la pipeline sulla fase di decode, questo influisce sulle istruzioni successive in quanto esse stalleranno già nella fase di fetch. WinMips invece gestisce questo tipo di hazard entrando nello stadio di esecuzione e stallando questo stadio invece di quello di decode. Implementando questa soluzione lasciamo libero lo stadio di decode dal quale potrà partire l'esecuzione di una istruzione successiva nel caso in cui questa debba uasare una diversa unità della ALU di fatto andando a completare le istruzioni in maniera out of order.
Più sotto si può vedere la differenza nelle due implementazioni:



Lo stesso metodo è applicato anche gli altri problemi però l'effetto è meno evidente rispetto ai cicli di clock impiegati per l'esecuzione