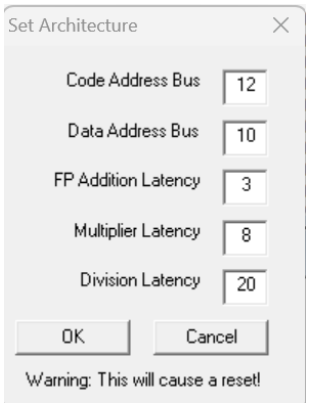
Architetture dei Sistemi di Elaborazione 02GOLOV	Delivery date: **9th November 2023**
Laboratory 3	Expected delivery of lab_03.zip must include: - `program_1_a.s`, `program_1_b.s` and `program_1_c.s` - this file compiled and if possible in pdf format.

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	6187	6187	6123	5899
By simulation	5578	5162	5099	4842

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	0,322	0,348	0,365	0,332

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

Eventual explanation:
I risultati sono, ovviamente, diversi in quanto nel primo programma sono presenti diversi hazard, i quali causano una molteplicità di stalli rallentando l'esecuzione complessiva del codice. Tali stalli vengono, per quanto possibile, correnti nel program_1_a usando principalmente il rescheduling sfruttando la pipeline di certe operazioni fp e il parallelismo tra operazioni intere e fp trasformando certe operazioni in "costo 0", successivamente si evita di perdere un colpo di clock ad ogni ciclo del programma sfruttando il "branch delay slot" per decrementare l'iteratore (il branch causato dalla parità dell'iteratore causa un'operazione di shift che verrà sempre eseguita ma non comporta "danni collaterali"). Per ridurre maggiormente il numero di clock "persi" per controllo si può srotolare il loop per 2 (come nel program_1_c) e così effettuare il controllo del loop la metà delle volte e in aggiunta considerare che la parità di un iteratore con variazione "1" ha periodo 2 si può eliminare il branch causato dal controllo eliminando così il controllo del salto per ogni ciclo e creando un algoritmo con prevedibilità del flusso di codice quasi completa (rimane solo l'imprevedibilità della fine

del ciclo complessivo causato dall'iteratore). Per quanto riguarda la differenza sostanziale di valori ottenuti tra il calcolo di clock cycle a mano e tramite simulatore, ciò viene ottenuto poiché il calcolo a mano è stato effettuato in modo naif considerando ogni istruzione semi-pipelined (solo la fase di execute non vengono considerate tali) né parallelized, considerando quindi solo la durata dei corrispettivi cicli exe per istruzione (per la prima si consideri 5 clk come tempo iniziale per portare a regime il processore con un'istruzione che usi le unità intere). Si noti quindi che il calcolo a mano non riporta nessuna modifica causata da rescheduling (nonostante il simulatore evidenzi un'efficienza maggiore del codice sia in termini di istruzioni totali che di ipc) al contrario del "branch delay slot" che elimina le istruzioni interrotte a metà (come per esempio l'halt finale) sostituendole con istruzioni utili per il ciclo. Anche lo srotolamento risulta più efficiente sia a mano che da simulatore.