Computer Architectures 02LSEOV **Laboratory 3**	Delivery date: October 22nd 2024 Expected delivery of lab_03.zip must include: - program_1_a.s, program_1_b.s, and program_1_c.s - This file, filled with information and possibly compiled in a pdf format.

This lab will explore some of the concepts seen during the lessons, such as hazards, rescheduling, and loop unrolling. The first thing to do is to configure the WinMIPS64 simulator with the *Initial Configuration* provided below:

- *Integer ALU: 1 clock cycle*
- *Data memory: 1 clock cycle*
- Code address bus: 12
- Data address bus: 12
- FP arithmetic unit: pipelined, 4 clock cycles
- FP multiplier unit: pipelined, 6 clock cycles
- FP divider unit: not pipelined, 30 clock cycles
- Forwarding is enabled
- Branch prediction is disabled
- Branch delay slot is disabled

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

```
int m=1 /* 64  bit */
double a, b
for (i = 31; i >= 0; i--){
        if (i is a multiple of 3) {
                a = v1[i] / ((double) m<< i) /*logic shift */
                m = (int) a
        } else {
                a = v1[i] * ((double) m* i))
                m = (int) a
        }
        v4[i] = a*v1[i] – v2[i];
        v5[i] = v4[i]/v3[i] – b;
        v6[i] = (v4[i]-v1[i])*v5[i];
}
```

   a. Manually detect the different data, structural, and control hazards that cause a pipeline stall.

Alcuni esempi di:

- Data hazards →
  - Tra s.d F6, v6(R10) e mul.d F6,F9,F5
    - Si tratta di un possibile WAW
  - Tra sub.d F4,F7,F2 e mul.d F6,F9,F5 (RAW)

- Control Hazards →
    - Tutte queste istruzioni dipendono dal controllo bnez R11,ELSE

```
dsllv R13,R12,R28 ;shift logico di m di i posizioni
;trasformo m shiftato in floating point
mtc1 R13,F11
cvt.d.l F11,F11
div.d F11,F1,F11
j ENDIF
ELSE:
dmul R12,R12,R28

;trasformo R12 in floating point
mtc1 R12,F11
cvt.d.l F11,F11
mul.d F11,F1,F11
```

b. Optimize the program by re-scheduling the program instructions to eliminate as many hazards as possible. Manually calculate the number of clock cycles for the new program (**`program_1_a.s`**) to execute and compare the results with those obtained by the simulator.

c. Starting from **`program_1_a.s`**, enable the *branch delay slot* and re-schedule some instructions to improve the previous program execution time. Manually calculate the number of clock cycles needed by the new program (**`program_1_b.s`**) to execute and compare the results obtained with those obtained by the simulator.

d. Unroll the program (**`program_1_b.s`**) 3 times; if necessary, re-schedule some instructions and increase the number of registers used. Manually calculate the number of clock cycles to execute the new program (**`program_1_c.s`**) and compare the results obtained with those 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`	2590	2488	2350	3500
`By simulation`	2602	2509	2366	3646

2) Collect the Cycles Per Instruction (CPI) from the simulator for different programs

	`program_1.s`	**`program_1_a.s`**	**`program_1_b.s`**	**`program_1_c.s`**
CPI	3.397	3.275	2.914	3.739

Compare the results obtained in 1) and provide some explanation if the results are different.

Eventual explanation:

Mi aspettavo un CPI e un numero di clock più basso nel programma_1_c, credo di non aver sufficientemente ottimizzato oppure credo di aver sbagliato qualcosa nel rescheduling delle istruzioni (difatti il numero di clock dovrebbe diminuire e di certo non essere più grande rispetto alle altre casistiche).

I cicli di clock differiscono tra by hand e by simulation perchè winMIPS64 gestisce gli stalli in maniera diversa rispetto a quanto detto a lezione → winMIPS64 stalla (per le eventuali dipendenze di dato) nella fase di execute mentre a lezione ci è stato detto di stallare nella fase di decode.