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.

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	6 + 10 * 102 + 22 * 77 = 2720	6 + 10 * 106 + 22 * 81 = 2848	6 + 10 * 106 + 22 * 81 = 2848	6 + 73 * 11 + 74 * 11 + 98 * 10 +1 = 2604
By simulation	3,824	3,527	3,475	2,429

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	3844	4029	3757	3636

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

Eventual explanation:

In general, when optimizing the program, the cycle count calculated by hand increases, while in the simulation, it decreases. This is because optimization often increases the number of instructions. However, in program_1_a and program_1_b, the cycles calculated by hand remain constant. This happens because, with the activation of the branch delay slot, the last branch no longer fetches the halt instruction, but instead executes a useful command. Additionally, in the if/else branch, a **nop** instruction is used to prevent the program from crashing, which keeps the hand-calculated instruction count the same, while the simulation shows a reduction in clock cycles.

Typically, the hand calculations result in lower cycle counts than the simulations, as they only account for execution time and do not consider stalls. However, in the last program where loop unrolling is applied, the stalls are significantly reduced. The simulation demonstrates that some instructions are overlapped with others, leading to fewer cycles than calculated by hand.