Computer Architectures 02LSEOV **Laboratory 3**	Delivery date: **By 2:00 AM on October, 23 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	(10.67*143 + 21.33*141)+5= 4537	(10.67*143 + 21.33*141)+5= 4537	(10.67*143 + 21.33*141)+5= 4537	11×(75 cicli nel 'if' +60 cicli comuni)= 11×135=1485 cicli + 21×(51 cicli nel 'else'+60 cicli comuni) =21×111=2331 cicli + 5 =3821
By simulation	3824	3760	3760	3740

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	5.071	4.987	4.602	4.960

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

Eventual explanation:

Comparison of Results

The comparison between program_1_a and program_1_b shows no difference in total clock cycles. This is because the instructions have already been rescheduled to reduce stalls and improve performance, making the branch delay slot unable to insert any independent instruction after a branch. As a result, the branch delay optimization in program_1_b does not further reduce the clock cycles compared to program_1_a. Both programs achieved 3760 clock cycles in the simulation.

Differences between Manual and Simulated Results

In the manual calculation, the number of clock cycles was consistently higher compared to the simulator's results across all programs. For instance, program_1.s had a manually calculated total of 4537 cycles, while the simulator yielded 3824 cycles. These discrepancies can be explained by several factors:

Pipeline Efficiency: The simulator can manage pipeline hazards, stalls, and forwarding more efficiently than what is captured by manual estimation. The manual calculation assumes a more conservative scenario, where hazards like data dependencies or control hazards are more impactful.
Branch Handling: The simulator likely implements branch prediction techniques, and with the use of forwarding and other mechanisms, it reduces the cost of branch delays more effectively than manual estimates.
Automatic Optimization: The simulator might implement optimizations that aren't considered manually, such as reordering instructions to reduce stalls further, beyond what was already done in the rescheduled versions of the programs.

CPI Analysis

In terms of Cycles Per Instruction (CPI), the values show a clear improvement from program_1.s to program_1_b.s due to the rescheduling. For example, program_1.s has a CPI of 5.071, while program_1_b.s reduces it to 4.602, reflecting the enhanced instruction scheduling and fewer stalls. However, in program_1_c.s, where the loop is unrolled, the CPI slightly increases to 4.960. This increase is due to the higher number of operations in each unrolled block, even though the number of branches is reduced.