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`	(41+64)*10+(17+64)*22+5= 2837	(41+64)*10+(17+64)*22+5= 2837	(41+64)*10+(17+64)*22+5= 2837	5+41*10+16*22+64*32= 2815
`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	5072	4987	4602	4960

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

<u>Eventual explanation:</u>

<u>Comparison of Results</u>
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.

<u>The difference between the manually calculated clock cycles and the simulated ones arises due to two main reasons:</u>
-Integer Division Handling by the Simulator: In the manual calculation, we account for integer ALU operations, such as ddiv (integer division), as taking only 1 clock cycle. However, the simulator incorrectly counts integer divisions as floating-point operations, applying a 30-clock cycle penalty for these operations. This leads to a significant inflation in the number of cycles when using the simulator, compared to the more accurate manual approach where only 1 cycle is considered for integer ALU operations.

-Ignoring Stalls in Manual Calculation: When calculating the clock cycles by hand, only the execution phase of each operation is taken into account, without considering potential stalls or pipeline hazards. The simulator, on the other hand, includes stalls and delays caused by instruction dependencies, memory access times, and other factors that can introduce delays in real execution. This results in a higher cycle count in the simulation compared to the theoretical hand-calculated values where these delays are ignored.
Thus, while the manual method focuses purely on the operation execution times, the simulator adds complexity, such as stalls and over-counted division cycles, leading to higher clock cycle estimates.

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.