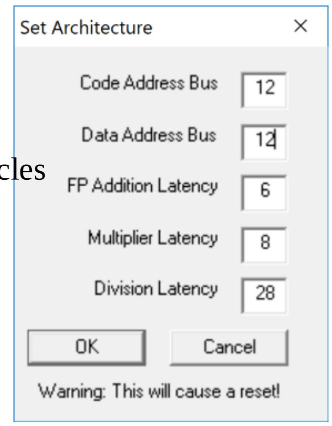
Computer Architectures **02LSEOV 02LSEOQ [AA-LZ]**	Delivery date: <u>Thursday 25/10</u>
Laboratory **2**	Expected delivery of lab_02.zip must include: - `program_2.s` and `program_3.s` - This file, filled with information and possibly compiled in a pdf format.

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): 6 stages
- Pipelined multiplier unit (latency): 8 stages
- divider unit (latency): not pipelined unit, 24 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) Write an assembly program (**`program_2.s`**) for the *winMIPS64* architecture described before able to implement the following piece of code described at high-level:

```
for (i = 0; i < 30; i++){      /* it was 100, corrected */
        v5[i] = v1[i]*v2[i];
        v6[i] = v2[i]/v3[i];
        v7[i] = v1[i]+v4[i];
}
```

Assume that the vectors v1[], v2[], v3[], and v4[] are allocated previously in memory and contains 100 double precision floating point values; assume also that v3[] does not contain 0 values. Additionally, the vectors v5[], v6[], and v7[] are empty vectors also allocated in memory.

a. Using the simulator and the *Base Configuration*, compute how many clock cycles take the program to execute. No-opt: 1056, opt: 966

2) Using the WinMIPS64 simulator, validate experimentally the Amdahl's law, defined as follows:

$$\text{speedup}_{\text{overall}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} = \frac{1}{(1-\text{fraction}_{\text{enhanced}}) + \dfrac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}}$$

a. Using the program developed before: **`program_2.s`**
b. Modify the processor architectural parameters related with multicycle instructions (`Menu→Configure→Architecture`) in the following way:

1) Configuration 1
   - Starting from the *Base Configuration*, change only the FP addition latency to 3
2) Configuration 2
   - Starting from the *Base Configuration*, change only the Multiplier latency to 4
3) Configuration 1
   - Starting from the *Base Configuration*, change only the division latency to 12

Compute by hand (using the Amdahl's Law) and using the simulator the speed-up for any one of the previous processor configurations. Compare the obtained results and complete the following table.

Table 1: **`program_2.s` speed-up computed by hand and by simulation**

Proc. Config. / Speed-up comp.	Base config. [c.c.]	Config. 1	Config. 2	Config. 3
`By hand`	1056 cycles	1056 cycles – speedup: 1	1056 cycles – speedup: 1	696 cycles – speedup: 1.517
`By simulation`	1056 cycles	1056 cycles – speedup: 1	1056 cycles – speedup: 1	696 cycles – speedup: 1.517

The performance of the program depends only on the division latency: the other two operations, in fact, always terminate before the division and therefore do not account for any extra clock cycle (the enhanced fraction is 0 * 30 / 1056 = 0). For the division, instead, we have:

$$fraction_{enhanced} = \frac{24\frac{cycles}{loop} * 30\ loops}{1056\ cycles} = 0.682$$

$$speedup_{enhanced} = \frac{24\ cycles}{12\ cycles} = 2$$

$$speedup_{overall} = \frac{1}{(1 - 0.682) + \frac{0.682}{2}} = \frac{696\ cycles}{1056\ cycles} = 1.517$$

3) Write an assembly program (**`program_3.s`**) for the winMIPS64 architecture able to set the parity bit of a data array X[] allocated in memory.
   a. the data array **X[]** is composed of 100 elements
   b. every element **X[i]** is one byte long divided as follows:
      - $X[i]_{0\text{-}6}$ → data bits
      - $X[i]_7$ → parity bit
   c. consider *even parity*: the parity bit is set to 1 if the number of ones in a given set of bits (not including the parity bit) is odd
   d. the assembly program must be able to elaborate every data as presented by the following high level piece of code:

   for (i = 0; i < 100; i++){

```
        if (parity_in [x[i]_{0..6}])
            x[i]_7 = 1;
        else
            x[i]_7 = 0;
    }
```

For example, if x[i] = X001101, then it becomes x[i] = **<u>1</u>**001101; on the other side, if x[i] = X011101, then it becomes x[i] = **<u>0</u>**011101.

4) Considering the following *winMIPS64* architecture:
    - Code address bus: 12
    - Data address bus: 12
    - Pipelined FP arithmetic unit (latency): 4 stages
    - Pipelined multiplier unit (latency): 8 stages
    - divider unit (latency): not pipelined unit, 12 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*.

    and supposing that 50% of the data elements contain an odd number of ones in the data bits:

    a. calculate by hand how many clock cycles take the program to execute.

Number of clock cycles:	5255 = 5 + (5 + 6 * 6 + 5 + 3,5 + 2) * 100 + 99 + 1 • The first instruction is 5 clock cycles long • The first five instructions of the loop take another 5 • The internal loop takes 6 clock cycles for 6 times (considering the fetched but not executed BEQZ) and 5 clock cycles for once • The odd case accounts for 4 clock cycles (BEQZ, ORI, J, and fetched ANDI) • The even case accounts for 3 clock cycles (BEQZ, fetched ORI, and ANDI) • The last two instructions take another 2 clock cycles, plus 1 in the first 99 repetitions (HALT is fetched but not executed) • HALT takes another clock cycle

b. compute the same calculation using the *winMIPS64* simulator.

Number of clock cycles:	5255

Compare the results obtained in the points 4.a and 4.b, and provide some explanation in the case the results are different.

Eventual explanation: