Computer Architectures 02LSEOV **Laboratory 2**	Delivery date: **By 2:00 AM on October, 16 2024** The expected delivery of lab_02.zip must include: - **`program_1.s`** - This file, filled with information and possibly compiled in a pdf format.

Please configure the WinMIPS64 simulator with the *Initial Configuration* provided below c):

- *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

1) Write an assembly program (**`program_1.s`**) for the *WinMIPS64* architecture described before being able to implement the following high-level code:

```
for (i = 31; i >= 0; i--){
        v4[i] = v1[i]*v1[i] – v2[i];
        v5[i] = v4[i]/v3[i] – v2[i];
        v6[i] = (v4[i]-v1[i])*v5[i];
}
```

Assume that the vectors v1[], v2[], and v3[] have been previously allocated in memory and contain 32 double-precision **floating-point values;** also assume that v3[] does not contain 0 values. Additionally, the vectors v4[], v5[], v6[] are empty vectors also allocated in memory.

Calculate the data memory footprint of your program:

Data	Number of bytes
V1	256
V2	256
V3	256
V4	256
V5	256
V6	256
Total	1536

Are there any issues? Yes, where and why? No? Do you need to change something?

Your answer:
Considering that each array occupies 256 bytes of memory and each double-precision value requires 8 bytes, I believe that, based on the current memory footprint of the program, there are no immediate issues that would require changes. At this stage, I don't see a need for any modifications.
However, potential issues could arise in the future, particularly if the program expands or needs to handle larger arrays or operate in a more memory-constrained environment, in that case, there will be the necessity to reduce the memory footprint of the program.

ATTENTION: WinMIPS64 has a limitation regarding the maximum length of the string when declaring a vector. It is therefore recommended to split the elements of the vectors into multiple lines: this also increases readability.

Example: my_fancy_vector: .byte 8, 12 ,2, 9
.byte 49,77, 28
.byte ……

- Calculate the CPU performance equation (CPU time) of the above program by assuming a clock frequency of 15 MHz:

$$\textbf{CPU time} = (\sum_{i=1}^{n} \textbf{CPI}_i \times \textbf{IC}_i) \times \textbf{Clock cycle period}$$

By definition:

- CPI is equal to the number of clock cycles required by the related functional unit to execute the instruction (EX stage).
- IC_i is the number of times an instruction is repeated in the referenced source code.

- Recalculate the CPU performance equation assuming that you can triple the speed by just one unit of your choice between the FP multiplier or the FP divider:
  - FP multiplier unit: 6 → 2 clock cycles
    or
  - FP divider unit: 30 → 10 clock cycles

Table 1: CPU time by hand

	Initial CPU time (a)	CPU time (b – MUL speeded up)	CPU time (b – DIV speeded up)
program_1.s	0.0835 ms	0.0749 ms	0.0408 ms

- Using the simulator, calculate the CPU time again and fill in the following table:

Table 2: CPU time using the simulator

	Initial CPU time (a)	CPU time (b – MUL speeded up)	CPU time (b – DIV speeded up)
program_1.s	0.1284 ms	0.1114 ms	0.0859 ms

Are there any differences? If so, where and why? If not, please provide some comments in the box below:

Your answer:
There are differences between the by hand calculation and the simulation results because the by hand method assumes ideal execution without accounting for stalls or hazards, while the simulation models real-world processor behavior, including pipeline stalls and branch mispredictions

- Using the simulator and the *Initial Configuration*, enable the Forwarding option and compute how many clock cycles the program takes to execute.

Table 3: forwarding enabled

	Number of clock cycles	IPC (Instructions Per Clock)
program_1.s	1926	0.251

Enable one at a time the *optimization features* that were initially disabled and collect statistics to fill the following table (fill all required data in the table before exporting this file to pdf format to be delivered).

Table 4: **Program performance for different processor configurations**

Program	Forwarding		Branch Target Buffer		Delay Slot		Forwarding + Branch Target Buffer	
	IPC	CC	IPC	CC	IPC	CC	IPC	CC
program_1.s	0.241	1926	0.200	2413	0.214	86	0.254	1899

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

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

a. Using the program developed before: **program_1.s**

b. Modify the processor architectural parameters related to multicycle instructions (Menu→Configure→Architecture) in the following way:

1) Configuration 1
   - Starting from the *Initial Configuration*, change the FP addition latency to 3
2) Configuration 2
   - Starting from the *Initial Configuration*, change the FP multiplier latency to 4
3) Configuration 3
   - Starting from the *Initial Configuration*, change the FP division latency to 10

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

Table 5: **program_1.s speed-up computed by hand and by simulation**

Proc. Config. / **Speed-up comp.**	Initial config. [c.c.]	Config. 1	Config. 2	Config. 3
By hand	2440	1.05	**1.**0536	1.364
By simulation	2440	1.041	1.0554	1.3556