Computer Architectures **02LSEOV** **Laboratory 2** **REVALOR RICCARDO** **s339423**	Delivery date: October16th2025 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 simulatorwith the *Initial Configuration* provided belowc):

- *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 32double-precision **floating-point values;** also assume that v3[] doesnot contain 0 values. Additionally, the vectors v4[], v5[], v6[]are empty vectors also allocated in memory.

<u>**Calculate**</u> 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:
I noticed that the mathematical results of the operations is right, but there are performance problems affecting the pipeline.For example with the instructions `mul.d f4, f1, f1` and `sub.d f4, f4, f2`: since both write on f4 the compiler stalls one of them to finish the execution of the other.
I executed the program with **forwarding disabled**.
To improve the performance, I tried to move some **independent instructions** between dependent ones to reduce pipeline stalls. For example, I reordered the `mul.d f4, f1, f1` and `sub.d f4, f4, f2` instructions by placing `l.d f3, v3(r1)` in between. At first, the required clock cycles were 2599, after the enhancements they became 2407.

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:.byte8, 12 ,2, 9
                        .byte49,77, 28
                        .byte ……
```

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

CPU time =

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.

DISCLAIMER: I am considering an ideal case, with no stall or hazards
Clock cycle period = T = 1 / 15 MHz = 66.67 * 10^$^{-9}$ s

The first two instructions (two daddui …….) are not in the cycle so for them:
$CPI*IC_{\text{first 2}} = (1*1 + 1*1) = 2$
The last instruction (halt) is also not part of the cycle so:
$CPI*IC_{\text{halt}} = (1*1) = 1$

Regarding the instructions in the cycle (we have 32 iterations):

Instruction	CPI	IC
l.d …….	1	32 * 3 = 96
s.d …….	1	32 * 3 = 96
mul.d …….	6	32 * 2 = 64
div.d …….	30	32
sub.d …….	4	32 * 3 = 96
slt …….	1	32
daddi …….	1	32
daddui …….	1	32
beq …….	1	32

So, for the cycle we have:

$CPI*IC_{cycle}$ = (1*96 + 1*96 +6*64 + 30*32 + 4*96 + 1*32 + 1*32 + 1*32 + 1*32) = (6*96 + 6*64 + 34*32) = 2048

The total CPU time is therefore calculated as:
CPU time = ($CPI*IC_{first\ 2}$ + $CPI*IC_{halt}$ + $CPI*IC_{cycle}$) * T = (2+1+2048) * 66.67 * 10^$^{-9}$ s = 0.00013674 s = **136.74 µs**

- 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→2clock cycles
    or
  - FP divider unit: 30→10 clock cycles

CPU time calculation with DIV speeded up:
$CPI*IC_{cycle}$ = (1*96 + 1*96 +6*64 + 10*32 + 4*96 + 1*32 + 1*32 + 1*32 + 1*32) = (6*96 + 6*64 + 24*32) = 1728

CPU time = ($CPI*IC_{first\ 2}$ + $CPI*IC_{halt}$ + $CPI*IC_{cycle}$) * T = (2+1+1728) * 66.67 * 10^$^{-9}$ s = 0.00011541 s = **115.41 µs**

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	136.74 µs	/	115.41 µs

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

Table 2: CPU time using the simulator

Initial config:
2407 total clock cycles
Cpu time = 2407 * T = 2407 * 66.67 * 10^-9 s = 0.00016474 s = 164.74 µs

With DIV speeded up:
1767 total clock cycles
Cpu time = 1767 * T = 1767 * 66.67 * 10^-9 s = 0.000117805 s = 117.81 µs

	Initial CPU time (a)	CPU time (b – MUL speeded up)	CPU time (b – DIV speeded up)
program_1.s	164.74 µs	/	117.81 µs

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

Your answer:
The simulator results are slightly **higher**, as expected, because the provided formula for CPU time, used in calculations by hand, does not take into account potential stalls or hazards in the pipeline, and thus only refers to the ideal length of the EX stage, as specified by the units latencies.
In fact, by hand I estimate a number of clock cycles equal to 2051 (Initial config.) and 1728 (DIV enhanced), when in reality they are 2407 (Initial config.) and 1767 (DIV enhanced).

- 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	1862	0.276

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	0.276	1862	0.214	2410	0.231	82	0.281	1835

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.

All the **optimization features** (including forwarding) have been **DISABLED** during this test.

1) By hand:
Speedup $_{\text{enhanced}}$ = 4 / 3 = **1.33** for all FP sums and subs instructions
Instructions $_{\text{enhanced}}$ = 3 instructions * 32 cycles = **96 instructions**
Fraction $_{\text{enhanced}}$ = 96 / 515 = 0.186 = **18.6%**
Speedup $_{\text{overall}}$ = 1 / (1-0.186+ 0.186/ 1.33) = 1.048
Using WinMIPS64:
Speedup overall = 2407 cycles / 2311 cycles = 1.042

2) By hand: Speedup $_{\text{enhanced}}$ = 6 / 4 = **1.5** for all FP mul instructions
Instructions $_{\text{enhanced}}$ = 2 instructions * 32 cycles = **64 instructions**
Fraction $_{\text{enhanced}}$ = 64 / 515 = 0.124 = **12.4%**
Speedup $_{\text{overall}}$ = 1 / (1-0.124 + 0.124 / 1.5) = 1.043
Using WinMIPS64:
Speedup overall = 2407 cycles / 2279 cycles = 1.056

3) By hand:
Speedup $_{\text{enhanced}}$ = 30 / 10 = **3** for all FP div instructions
Instructions $_{\text{enhanced}}$ = 1 instruction * 32 cycles = **32 instructions**
Fraction $_{\text{enhanced}}$ = 32 / 515 = 0.062 = **6.2%**

Speedup $_{\text{overall}}$ = 1 / (1-0.062 + 0.062 / 3) = 1.043
Using WinMIPS64:
Speedup overall = 2407 cycles / 1767 cycles = 1.362

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	1	1.048	1.043	1.043
By simulation	1	1.042	1.056	1.362

IMPORTANT NOTE: By now, you should have noticed that WinMIPS tends to differfrom the notation provided during the lessons. One of these is that it tends to enter the execution state even if the operands are not yet available. This can affect the number of clock cycles required to execute the program since the instruction, entering the execution phase, frees up the decoding phase, which another instruction can then use.

3) Consider the *Initial Configuration*.
   Then, assume that:
   - branch delay slot is not enabled
   - data forwarding is enabled
   - the EX stage could also be completed in an out-of-order fashion

   a) Given the following source code, esteem the number of clock cycles (CC) needed for completion using notation given during the lessons (so, not using WinMIPS). In this regard, fill in the table on the next page with the pipeline stages for a single iteration of the loop using the same notation as the first two lines. If necessary, duplicate it.Then, use the information obtained to fill in the table on this page.

```
; ********************* C ***********************
;  for (i = 10; i > 0; i--){
;    v4[i] = (v1[i]+v1[i]+v2[i])/(v1[i]*v2[i])
; }
; ****************** MIPS64 ********************
```

```
.data

v1: .double  43, 23, 57, 34, 79, 6, 30, 44, 82, 18
v2: .double  37, 12, 94, 59, 31, 77, 58, 21, 62, 1
v3: .double  44, 94, 56, 67, 23, 78, 37, 14, 9, 52
v4: .space 80

.text
```

	Comments	Clock cycles
daddui r1,r0,0	r1 ← pointer	5
daddui r2,r0,10	r2 ← 10	1
cyc: l.d f1,v1(r1)	f1← v1[i]	1
l.d f2,v2(r1)	f2← v2[i]	1
l.d f3,v3(r1)	f3← v3[i]	1
add.d f5,f1,f1	f5 ←v1[i]+v1[i]	4
mul.d f6,f1,f2	f6 ← v1[i]*v2[i]	3
add.d f5,f5,f2	f5←f5+ v2[i]	1
div.d f7,f5,f6	f7 ← f5/f6	30
s.d f7,v4(r1)	v4[i] ← f7	1
daddui r1,r1,8	r1 ← r1+8	1
daddi r2,r2,-1	r2 ← r2-1	1
bnez r2, cyc		2
halt		1
Total numberof CC to run the **entire** program:		**467**

daddui r1,r0,0	F	D	E	M	W																									
daddui r2,r0,10		F	D	E	M	W																								
l.d f1,v1(r1)			F	D	E	M	W																							
l.d f2,v2(r1)				F	D	E	M	W																						
l.d f3,v3(r1)					F	D	E	M	W																					
add.d f5,f1,f1						F	D	E_A	E_A	E_A	E_A	M	W																	
mul.d f6,f1,f2							F	D	E_M	E_M	E_M	E_M	E_M	E_M	M	W														
add.d f5,f5,f2								F	D	D	D	E_A	E_A	E_A	E_A	M	W													
div.d f7,f5,f6									F	F	F	F	D	D	D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D
s.d f7,v4(r1)													F	F	F	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
daddui r1,r1,8																	F	D	D	D	D	D	D	D	D	D	D	D	D	D
daddi r2,r2,-1																		F	F	F	F	F	F	F	F	F	F	F	F	F

div.d f7,f5,f6	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	M	W													
s.d f7,v4(r1)	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	E	M	W												
daddui r1,r1,8	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	D	D	E	M	W										
daddi r2,r2,-1																F	F	D	E	M	W									
bnez r2, cyc																		F	D	D	E	M	W							
halt																			F	F	X									

Total CC = 5+1+(46*10)+1 = 467

HINT: Use letters other than E for FP functional units or enter a subscript. For example, you can use E_A for the FP adder, E_M for the FP multiplier, and E_D for the FP divider.
X = because PC was updated and halt removed

b) Repeat the same operations performed above assumingthat data forwarding is disabled. If necessary, duplicate the table in the next pageto have the necessary space to complete the required work.

```
.data

v1: .double  43, 23, 57, 34, 79, 6, 30, 44, 82, 18
v2: .double  37, 12, 94, 59, 31, 77, 58, 21, 62, 1
v3: .double  44, 94, 56, 67, 23, 78, 37, 14, 9, 52
v4: .space 80

.text
```

	Comments	Clock cycles
daddui r1,r0,0	r1 ← pointer	5
daddui r2,r0,10	r2 ← 10	1
cyc: l.d f1,v1(r1)	f1← v1[i]	2
l.d f2,v2(r1)	f2← v2[i]	1
l.d f3,v3(r1)	f3← v3[i]	1
add.d f5,f1,f1	f5 ←v1[i]+v1[i]	4
mul.d f6,f1,f2	f6 ← v1[i]*v2[i]	3
add.d f5,f5,f2	f5← f5 + v2[i]	3
div.d f7,f5,f6	f7 ← f5/f6	32
s.d f7,v4(r1)	v4[i] ← f7	3
daddui r1,r1,8	r1 ← r1+8	1
daddi r2,r2,-1	r2 ← r2-1	1
bnez r2, cyc		3
halt		1
Total numberof CC to run the **entire** program:		**547**

daddui r1,r0,0	F	D	E	M	W																									
daddui r2,r0,10		F	D	E	M	W																								
l.d f1,v1(r1)			F	D	D	E	M	W																						
l.d f2,v2(r1)				F	F	D	E	M	W																					
l.d f3,v3(r1)						F	D	E	M	W																				
add.d f5,f1,f1							F	D	E_A	E_A	E_A	E_A	M	W																
mul.d f6,f1,f2								F	D	E_M	E_M	E_M	E_M	E_M	E_M	M	W													
add.d f5,f5,f2									F	D	D	D	D	D	E_A	E_A	E_A	E_A	M	W										
div.d f7,f5,f6										F	F	F	F	F	D	D	D	D	D	D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D
s.d f7,v4(r1)															F	F	F	F	F	F	D	D	D	D	D	D	D	D	D	D
daddui r1,r1,8																					F	F	F	F	F	F	F	F	F	F

div.d f7,f5,f6	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	E_D	M	W								
s.d f7,v4(r1)	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	E	M	W					
daddui r1,r1,8	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	D	E	M	W				
daddi r2,r2,-1																							F	D	E	M	W			
bnez r2, cyc																								F	D	D	D	E	M	W
halt																									F	F	F	X		

Total CC = 5+1+(54*10)+1 = 547