Computer Architectures **02LSEOV** **Laboratory 2** **SILVIA BONENTI** **S339232**	Delivery date: October 16th 2025 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:
The first thing I noticed is that, since I forgot to change the data address bus size, WinMIPS64 continuously crashed since ti wasn't able to address 1536 memory locations with only 10 bits ($2^{10} = 1024$).
Another thing I noticed is that with my default execution order it takes 2471 (1926) cycles to complete the execution, but the movement of some instructions before or after increased the performance of my program (the cycles between brackets indicate the amount of them with forwarding enabled).
Moving the load in f3 to after the first multiplication (f4 = f1*f1) it took 2471 (1894).
Moving the subtraction f6 = f4-f1 above, after the f5 = f4/f3 division was also a good choices, taking the cycles to 2279 (1830).
An interesting thing I noticed is that, although I thought that putting the *s.d* instructions in the middle would take less time to execute, I realised that putting them in the end was actually the best choice since in this way the first two *s.d*'s were executed in parallel to the last multiplication (f6*f5). This allowed me to reduce the cycles needed to complete the execution from to 2215 (1830).

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:

$$\text{CPU time} = \left(\sum_{i=1}^{n} CPI_i \times IC_i\right) \times \text{Clock cycle period}$$

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.

CPI (daddui, daddi, l.d, s.d, bne, halt) = 1
CPI (sub.d) = 4
CPI (mul.d) = 6
CPI (div.d) = 30

IC (daddui, first daddi, halt) = 1
IC (other daddi, l.d, s.d, sub.d, mul.d, div.d, bne) = 32
CK period = 1/(15MHz) = 66 ns

So, CPU time =

(1*1+1*1+3*(1*32)+2*(6*32)+3*(4*32)+30*32+3*(1*32)+1*32+1*32+1*1) * 66 * 10^{-9} = 1987 * 66 * 10^{-9} = 131.11μs

- 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	131.11μs	114.25μs	88.9μs

- 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	146.2μs	129.29μs	103.95μs

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

Your answer:
Of course there are differences: times are higher with simulation, since for our calculations we considered only ideal cases without stalls and hazards in the pipeline, thus referring only to the execution stage length.
In fact, while I calculated by hand 1987, 1731, 1347 cycles respectively for standard case, multiplication and division speed ups, the corresponding cycles in WinMIPS64 were 2215, 1959, 1575.

- 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	1830	0.246

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	0.246	1830	0.206	2188	0.224	76	0.250	1803

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	1	1.056	1.05	1.049
By simulation	1	1.030	1.061	1.406

Configuration 1:

WinMIPS64: 2215/2151 = 1.030

By hand:

$fraction_{enhanced} = 3 * 32 / 451 = 0.213$

$speedup_{enhanced} = 4 / 3 = 1.333$

$speedup_{overall} = 1/((1-fraction_{enhanced}) + (fraction_{enhanced}/speedup_{enhanced}))$
$= 1.056$

Configuration 2:

WinMIPS64: 2215/2087 = 1.061

By hand:

$fraction_{enhanced} = 2 * 32 / 451 = 0.142$

$speedup_{enhanced} = 6 / 4 = 1.5$

$speedup_{overall} = 1/((1-fraction_{enhanced}) + (fraction_{enhanced}/speedup_{enhanced}))$
$= 1.05$

Configuration 3:

WinMIPS64: 2215/1575 = 1.406

By hand:

$fraction_{enhanced} = 1 *32 / 451 = 0.071$

$speedup_{enhanced} = 30 / 10 = 3$

$speedup_{overall} = 1/((1-fraction_{enhanced}) + (fraction_{enhanced}/speedup_{enhanced}))$
$= 1.049$

IMPORTANT NOTE: By now, you should have noticed that WinMIPS tends to differ from 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	Comments	Clock cycles
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		
daddui r1,r0,0	r1 ← pointer	5
daddui r2,r0,10	r2 ← 10	1
cyc: l.d f1,v1(r1)	f1 ← v1[i]	1/2
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]	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		1
halt		1
	5+1+(46*9 + 45)+1	486
Total number of CC to run the entire program:		

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	D	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
s.d f7,v4(r1)												F	F	F	F	D	D	D	D	D	D	D	D	D	D	D	D	D
daddui r1,r1,8																F	F	F	F	F	F	F	F	F	F	F	F	F
daddi r2,r2,-1																												
bnez r2, cyc																												
halt																												

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	E	M	W										
daddi r2,r2,-1																F	D	E	M	W									
bnez r2, cyc																	F	D	E	M	W								
halt																		F	X										

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.

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

.data	Comments	Clock cycles
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		
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 number of CC to run the entire program:	5+1+(54*10)+1	**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
s.d f7,v4(r1)															F	F	F	F	F	F	D	D	D	D	D	D	D	D	D
daddui r1,r1,8																					F	F	F	F	F	F	F	F	F
daddi r2,r2,-1																													
bnez r2, cyc																													
halt																													

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	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		