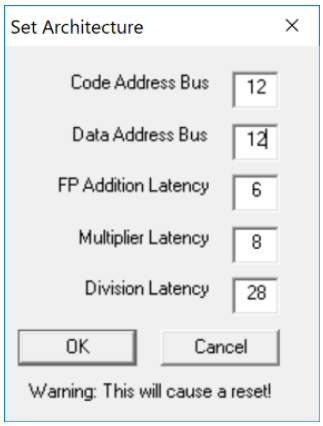
Architetture dei Sistemi di Elaborazione 02GOLOV [A-L]	Delivery date: October 29, 2020
Laboratory 2	Expected delivery of lab_02.zip must include: - `program_2.s` and `program_3.s` - This file, filled with information and possibly compiled in a pdf format.

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 < 5; i++){
      for (j = 0; j < 5; j++) {
            v3[i][j] = v1[i][j] * v2[i][j];
            v5[i][j] = v3[i][j] * v4[i][j];
      }
}
```

Assume that `v1`, `v2` and `v4` are two 5x5 matrixes allocated previously in memory and containing double precision floating-point values. Additionally, the matrixes `v3`,`v5` are initially empty and allocated in memory.

a. Using the simulator and the *Base Configuration*, compute how many clock cycles take the program to execute.

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_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`	*791*	*1*	*1.40 **	*1*
`By simulation`	*791*	*791 / 791 = 1*	*791 / 571 = 1.39*	*791 / 791 = 1*

*

$$fraction_{enhanced} = \frac{(8 \times 5) + (9 \times 25) + (9 \times 25)}{791} = 0.619$$

$$speedup_{enhanced} = \left(\frac{8}{4} + \frac{9}{5} + \frac{9}{5}\right)/3 = 1.867$$

$$speedup_{overall} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}}} = \frac{1}{(1 - 0.619) + \frac{0.619}{1.867}} = 1.403$$

program_2.s - Base Configuration

	.text		
main:	daddui R5, R0, 5	F D E M W	5
	daddui R1, R0, 0	F D E M W	1
for_i:	daddui R2, R0, 0	F D E M W	1
	dmulu R3, R1, R5	F D * * * * * * * * M W	8
for_j:	daddu R4, R3, R2	F D S S S S S S S S E M W	1
	dsll R4, R4, 3	F S S S S S S S S D E M W	1
	l.d F1, v1(R4)	F D E M W	1
	l.d F2, v2(R4)	F D E M W	1
	mul.d F3, F1, F2	F D S * * * * * * * * M W	9
	s.d F3, v3(R4)	F S D E S S S S S S S M W	1
	l.d F4, v4(R4)	F D S S S S S S S S E M W	1
	mul.d F5, F3, F4	F S S S S S S S S D S * * * * * * * * M W	9
	s.d F5, v5(R4)	F S D E S S S S S S S M W	1
	daddui R2, R2, 1	F D S S S S S S S S E M W	1
	bne R2, R5, for_j	F S S S S S S S S D E M W	2
	daddui R1, R1, 1	F D E M W	1
	bne R1, R5, for_i	F S D E M W	2
	halt	F - - - -	1
	Total	**6 + (9 + (29 * 5) + 3) * 5**	**791**

program_2.s - Configuration 2

	.text		
main:	daddui R5, R0, 5	F D E M W	5
	daddui R1, R0, 0	F D E M W	1
for_i:	daddui R2, R0, 0	F D E M W	1
	dmulu R3, R1, R5	F D * * * * M W	4
for_j:	daddu R4, R3, R2	F D S S S S E M W	1
	dsll R4, R4, 3	F S S S S D E M W	1
	l.d F1, v1(R4)	F D E M W	1
	l.d F2, v2(R4)	F D E M W	1
	mul.d F3, F1, F2	F D S * * * * M W	5
	s.d F3, v3(R4)	F S D E S S S S M W	1
	l.d F4, v4(R4)	F D S S S S E M W	1
	mul.d F5, F3, F4	F S S S S D S * * * * M W	5
	s.d F5, v5(R4)	F S D E S S S S M W	1
	daddui R2, R2, 1	F D S S S S E M W	1
	bne R2, R5, for_j	F S S S S S D E M W	2
	daddui R1, R1, 1	F D E M W	1
	bne R1, R5, for_i	F S D E M W	2
	halt	F - - - -	1
	Total	**6 + (5 + (21 * 5) + 3) * 5**	**571**

3) Write an assembly program (**program_3.s**) for the winMIPS64 architecture that implements the following piece of code:

```
unsigned char a[30];
unsigned char b[30];
unsigned char res[30];

for (i = 0; i < 30; i++){
    while (b[i] > 0)
    {

        if (isOdd (b[i])) {
            res[i] = res[i] + a[i];
          }

        a[i] = a[i] * 2;
        b[i] = b[i] / 2;
     }
}
```

Assume vectors a and b are previously allocated in memory. Populate the vectors with values chosen by you. Assume also that res is and empty vector in memory. The function `isOdd` returns 0 when the number is even, 1 when odd. **Please note that the function should be replaced with the proper piece of code (function call not required)**.

Which is the operation implemented by the above code?

Your Answer:
The program multiplies each element of array *a* with the corresponding element of array *b*, and the results are stored in array *c*:
c[i] = a[i] * b[i]

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*.

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

Number of clock cycles:	*1245*

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

Number of clock cycles:	*1245*

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:

The number of clock cycles depends on the elements of array *b*. The results in 4.a and 4.b are obtained with the following array *b*:

[2, 11, 8, 9, 7, 0, 5, 5, 13, 12, 1, 13, 14, 15, 14, 7, 15, 9, 4, 9, 13, 4, 11, 7, 8, 13, 3, 8, 5, 11]

It is possible to calculate by hand the total number of clock cycles with the following formula, obtained from the graph below:

$$c.c. = 6 + (6 \times 30) + (9 \times n\_while) + (5 \times 30)$$

program_3.s

	.text		
main:	daddui R5, R0, 30	F D E M W	5
	daddui R1, R0, 0	F D E M W	1
for:	lb R11, a(R1)	F D E M W	1
	lb R12, b(R1)	F D E M W	1
	daddui R13, R0, 0	F D E M W	1
while:	beqz R12, next	F S D E M W	2
	andi R2, R12, 1	F D E M W	1
	beqz R2, even	F S D E M W	2
	daddu R13, R13, R11	F D E M W	1
even:	dsll R11, R11, 1	F D E M W	1
	dsrl R12, R12, 1	F D E M W	1
	j while	F D E M W	1
next:	sb R13, res(R1)	F D E M W	1
	daddui R1, R1, 1	F D E M W	1
	bne R1, R5, for	F S D E M W	2
	halt	F - - - -	1
	Total	**6 + (6 * 30) + (9 * {n_while}) + (5 * 30)**	