Computer Architectures **02LSEOV 02LSEOQ [MA-ZZ]**	Delivery date: <u>Monday 22 October 2018</u>
Laboratory **1**	Expected delivery of lab_01.zip including: - program_1.s - lab_01.pdf (fill and export this file to pdf)

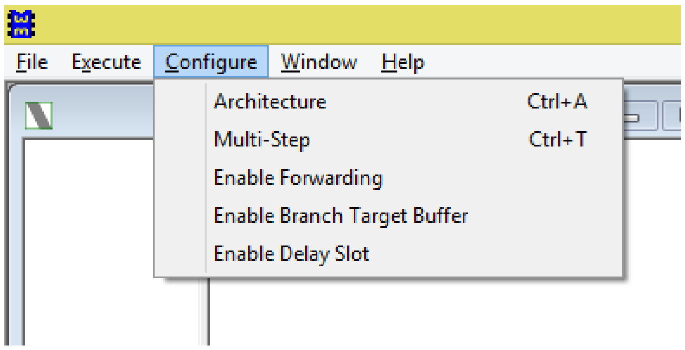
Please, configure the winMIPS64 processor architecture with the configuration provided in the following, it is call *Base Configuration*:

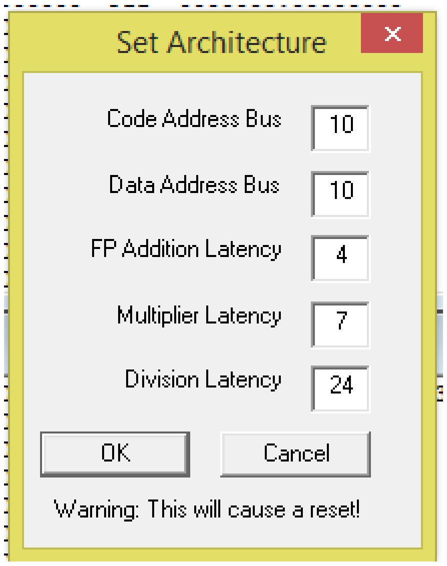
- Code address bus: 12
- Data address bus: 12
- Pipelined FP arithmetic unit (latency): 6 stages
- Pipelined multiplier unit (latency): 8 stages
- divider unit (latency): not pipelined unit, 28 clock cycles
- Forwarding optimization is disabled
- Branch Target Buffer is disabled
- Branch delay slot is disabled
- *Integer ALU: 1 clock cycle*
- *Data memory: 1 clock cycle*
- *Branch delay slot: 1 clock cycle.*

Use the <u>C</u>onfigure menu:

- remove the flags (where activating Enable options)
- Activate the *Architecture* menu (`Ctrl+A`) →

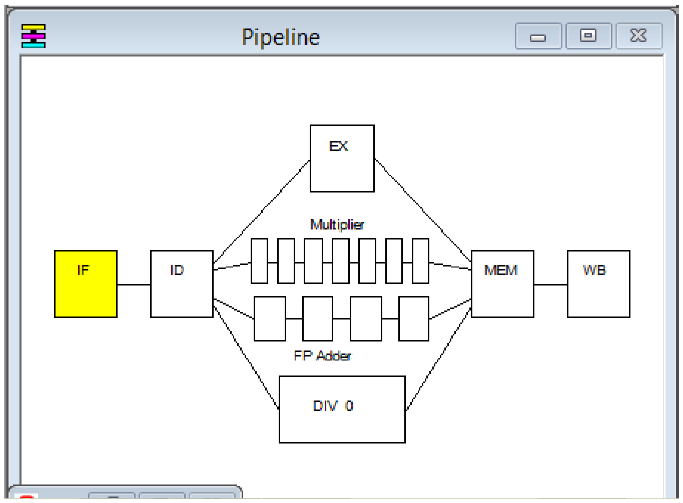
- Modify the defaults Architectural parameters (if needed)





Code Address Bus	10
Data Address Bus	10
FP Addition Latency	4
Multiplier Latency	7
Division Latency	24

Warning: This will cause a reset!



1) Improve your assembly skills and learn by example about pipeline optimizations.

Write an assembly program called **program_1.s (to be delivered as part of the laboratory**) for the win*MIPS64* architecture and execute it. The program has to search for the maximum integer value in a vector composed of 100 integer elements previously stored in the memory; every element of the vector is 64-bit wise and contains signed integer values. The program saves the obtained value in a variable allocated in memory, called *result*.

Identify and use the main components of the simulator:

a. Running the *WinMIPS* simulator
    - Launch the graphic interface
    ...\winMIPS64\winmips64.exe

b. Assembly and check your program:
    - Load the program from the **`File`→`Open`** menu (*CTRL-O*). In the case the of errors, you may use the following command in the command line to compile the program and check the errors:

```
...\winMIPS64\asm program_1.s
```

c. Run your program step by step (*F7*), identifying the whole processor behavior in the six simulator windows:
    Pipeline, **Code**, **Data**, **Register**, **Cycles** and **Statistics**

d. 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 1: **Program performance for different processor configurations**

	Number of clock cycles			
Program	No optimization	Forwarding	Branch Target Buffer	Delay Slot
program_1	1512	1012	1430	1639

2) <u>Perform execution time measurements.</u>
   Search in the *winMIPS64* folder the following benchmark programs:
   a. `isort.s`
   b. `mult.s`
   c. `series.s`
   d. `program_1.s` (your program)

Starting from the basic configuration with no optimizations, compute by simulation the number of cycles required to execute these programs; in this initial scenario, it is assumed that the programs weight is the same (25%) for everyone. Assume a processor frequency of 1MHz.

Then, change processor configuration and vary the programs weights as following. Compute again the performance for every case and fill the table below **(fill all required data in the table before exporting this file to pdf format to be delivered)**.

1) Configuration 1
   a. Enable Forwarding
   b. Disable branch target buffer
   c. Disable Delay Slot
   Assume that the weight of all programs is the same (25%).
2) Configuration 2
   a. Enable Forwarding
   b. Enable branch target buffer
   c. Disable Delay Slot
   Assume that the weight of all programs is the same (25%).
3) Configuration 3
   Configuration 1, but assume that the weight of the program *your_program* is 50%.
4) Configuration 4
   Configuration 1, but assume that the weight of the program `series.s` is 50%.

Table 2: **Processor performance for different weighted programs**

Program	No optimization	Conf. 1	Conf. 2	Conf. 3	Conf. 4
isort.s	0.25*2591	0.25*1894	0.25*1701	0.167*1894	0.167*1894
mult.s	0.25*1880	0.25*980	0.25*922	0.167*980	0.167*980
series.s	0.25*550	0.25*233	0.25*234	0.167*233	0.5*233
program_1.s	0.25*1512	0.25*1012	0.25*930	0.5*1012	0.167*1012
TOTAL TIME	1633.25	1029.75	946.25	1024.87	765.462

For time computations, use a clock frequency of 1MHz.

Appendix: *winMIPS64 Instruction Set*

WinMIPS64

The following assembler directives are supported

.data - start of data segment
.text - start of code segment
.code - start of code segment (same as .text)
.org <n> - start address
.space <n> - leave n empty bytes
.asciiz <s> - enters zero terminated ascii string
.ascii <s> - enter ascii string
.align <n> - align to n-byte boundary
.word <n1>,<n2>.. - enters word(s) of data (64-bits)
.byte <n1>,<n2>.. - enter bytes
.word32 <n1>,<n2>.. - enters 32 bit number(s)
.word16 <n1>,<n2>.. - enters 16 bit number(s)
.double <n1>,<n2>.. - enters floating-point number(s)

where <n> denotes a number like 24, <s> denotes a string like "fred", and
<n1>,<n2>.. denotes numbers seperated by commas.

The following instructions are supported

lb - load byte
lbu - load byte unsigned
sb - store byte
lh - load 16-bit half-word
lhu - load 16-bit half word unsigned
sh - store 16-bit half-word
lw - load 32-bit word
lwu - load 32-bit word unsigned
sw - store 32-bit word
ld - load 64-bit double-word
sd - store 64-bit double-word
l.d - load 64-bit floating-point
s.d - store 64-bit floating-point
halt - stops the program

daddi - add immediate
daddui - add immediate unsigned
andi - logical and immediate
ori - logical or immediate
xori - exclusive or immediate
lui - load upper half of register immediate
slti - set if less than or equal immediate
sltiu - set if less than or equal immediate unsigned

beq - branch if pair of registers are equal
bne - branch if pair of registers are not equal
beqz - branch if register is equal to zero
bnez - branch if register is not equal to zero

j - jump to address
jr - jump to address in register
jal - jump and link to address (call subroutine)
jalr - jump and link to address in register (call subroutine)

dsll - shift left logical
dsrl - shift right logical
dsra - shift right arithmetic
dsllv - shift left logical by variable amount
dsrlv - shift right logical by variable amount
dsrav - shift right arithmetic by variable amount
movz - move if register equals zero
movn - move if register not equal to zero
nop - no operation
and - logical and
or - logical or
xor - logical xor
slt - set if less than
sltu - set if less than unsigned
dadd - add integers
daddu - add integers unsigned
dsub - subtract integers
dsubu - subtract integers unsigned

add.d - add floating-point
sub.d - subtract floating-point
mul.d - multiply floating-point
div.d - divide floating-point
mov.d - move floating-point
cvt.d.l - convert 64-bit integer to a double FP format
cvt.l.d - convert double FP to a 64-bit integer format
c.lt.d - set FP flag if less than
c.le.d - set FP flag if less than or equal to
c.eq.d - set FP flag if equal to
bc1f - branch to address if FP flag is FALSE
bc1t - branch to address if FP flag is TRUE
mtc1 - move data from integer register to FP register
mfc1 - move data from FP register to integer register