

Laboratory 1

Expected delivery of lab_01.zip including:

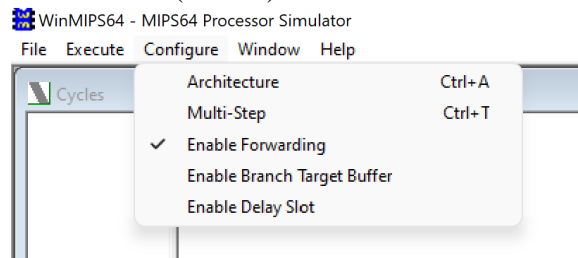
- program_0.s
- program_1.s
- lab_01.pdf (fill and export this file to pdf)

Please, configure the winMIPS64 processor architecture with the *Base Configuration* provided in the following:

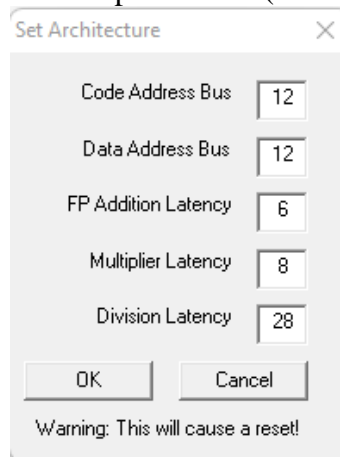
- *Integer ALU: 1 clock cycle*
- *Data memory: 1 clock cycle*
- *Branch delay slot: 1 clock cycle*
- Code address bus: 12
- Data address bus: 12
- Pipelined FP arithmetic unit (latency): 6 stages
- Pipelined FP multiplier unit (latency): 8 stages
- FP divider unit (latency): not pipelined unit, 28 clock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

Use the Configure menu:

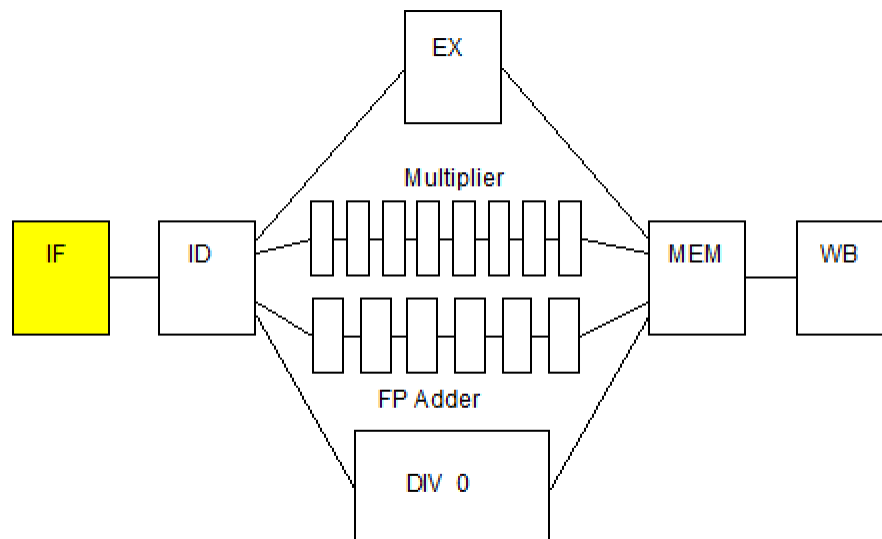
- Running the *WinMIPS* simulator, launching the graphical user interface
(*folder_to_simulator*)...\winMIPS64\winmips64.exe
- Disable ALL the optimization (a mark appears when they are enabled)
- Browse the Architecture menu (Ctrl-A)



- Modify the defaults Architectural parameters (where needed)



- Verify in the Pipeline window that the configuration is effective (usually in the left bottom window)



1) Exercise your assembly skills.

To write an assembly program called **program_0.s (to be delivered)** for the *MIPS64* architecture and to execute it.

The program must:

1. Given 2 statically initialized arrays (a and b), compute their sum and store each result in a third array (i.e., $c[i] = a[i] + b[i]$). Each array contains 50 8-bit integer numbers.
2. Search for **both** the maximum and minimum in the array c. The program saves the obtained value in two variables allocated in memory, called `max` and `min` respectively.

2) Exercise your assembly skills.

To write an assembly program called **program_1.s (to be delivered)** for the *MIPS64* architecture and to execute it.

The program must:

1. Given one array of 15 8-bit integer numbers (`v1`), check if the content corresponds to a **palindrome** sequence of numbers. If yes, use a 8-bit unsigned variable (`flag`) to store the result. The variable will be equal to 1 if the sequence is palindrome, 0 otherwise.

Example of a palindrome sequence:

`v1:` `.byte` 2, 6, -3, 11, 9, 11, -3, 6, 2

- 3) Once that **program_0.s** and **program_1.s** are written, use the *WinMIPS* simulator to check their correctness.

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 programs:
 - 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_0.s
- c. Run your programs step by step (*F7*), identifying the whole processor behavior in the six simulator windows:
Pipeline, Code, Data, Register, Cycles and Statistics
- d. Collect the clock cycles 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 the specific processor configurations**

Program	Clock cycles	Number of Instructions
program_0	1372	712
Program_1	95	56

4) Perform execution time measurements.

Search in the winMIPS64 folder the following benchmark programs:

- a. testio.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 weight of the programs is the same (25%) for everyone. Assume a processor frequency of 1.75 MHz.

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 70%.

4) Configuration 4

Configuration 1, but assume that the weight of the program `series.s` is 70%.

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

Program	No opt	Conf. 1	Conf. 2	Conf. 3	Conf. 4
<code>testio.s</code>	4,20E-04	2,70E-04	2,45E-04	2,70E-04	2,70E-04
<code>mult.s</code>	1,07E-03	5,57E-04	5,24E-04	5,57E-04	5,57E-04
<code>series.s</code>	3,13E-04	1,32E-04	1,33E-04	1,32E-04	1,32E-04
<code>program_1.s</code>	5,40E-05	5,00E-05	4,94E-05	5,00E-05	5,00E-05
TOTAL TIME	4,64E-04	2,52E-04	2,38E-04	1,31E-04	9,70E-04

For time computations, use a clock frequency of 1.75 MHz.

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
.ascii <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 separated 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
ld - load 64-bit floating-point
sd - 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