CEG3156-Computer System Design

12/2/2024

Lab#3:

Pipelined Processor

Group membres :

Gbegbe Decaho - Jacques	300094197
Ismaël Pedro -	300242291
Abdoulaye Diallo -	7935327

Assistant TA – Pulkit Arora

Prof. Rami Abielmona

DEPARTEMENT COMP. ENG UNIVERSITY OF OTTAWA

Table of Contents

I Objectif

The objective of this laboratory is to design and build a pipelined RISC processor using VHDL implementation. Upon completion of this lab, we must not only be able to design, realize and test a pipelined processor, but also demonstrate an understanding for pipelining concepts, including hazards, in the context of RISC processors and instruction set architectures.

Theoretical Part

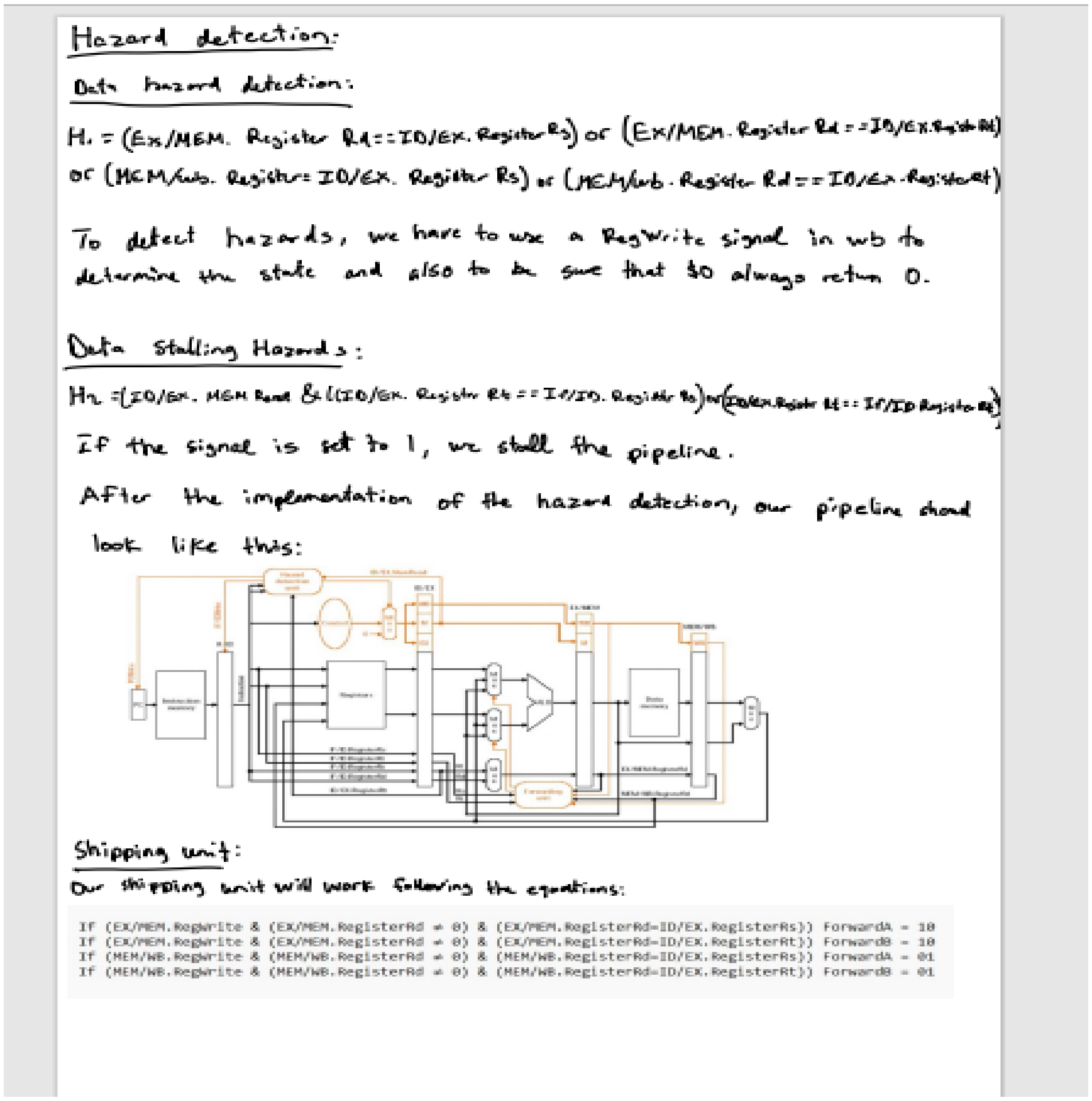
1 Introduction of the laboratory

The goal of this laboratory is to design a RISC type processor simple cycle in VHDL and more precisely in this laboratory we will implement a MIPS processor with its main functionality including memory reference instructions, arithmetic instructions and control instructions, this laboratory is important because it allowed us to build a processor and put into practice what we had studied in class. Furthermore, we will be looking at an implementation of the MIPS processor, which includes the following functionalities like the Memory-reference instructions (lw and sw), the Arithmetic logic instructions (add, sub, and, or and slt) and the Control flow instructions (beq and j).

Our implementation will not include all integer operations supported by the MIPS ISA (e.g. multiply and divide), nor will it support floating-pointoperations, also supported by the MIPS processor. However, the design will be modular and expandable in order to support additional operations if the need arises.

2 Lab Preparation(Pre-lab)

In this task of the lab, we were asked to Design the hazard detection and forwarding units discussed in this laboratory to show all the work, including the internals of your design, such as the truth table, optimization and Boolean realization of the circuits. Demonstrate the correct operation of the circuit by analyzing a sample scenario.



Hazard detection:

Data hazard detection:

H1 = (Ex/MEM. Register Rd==ID/EX. Register Rs) or (EX/MEM. Register Rd==ID/EX.Register Rt) or (MEM/wb. Register= ID/EX. Register Rs) or (MEM/wb. Register Rd==ID/Ex. Register Rt)

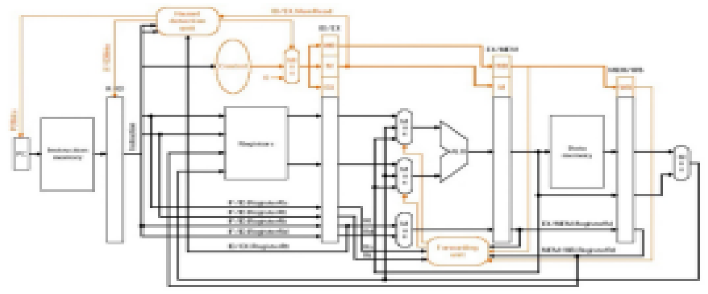
To detect hazards, we have to use a RegWrite signal in wb to determine the state and also to be sure that $0 always return 0.

Data Stalling Hazards:

H2 =(ID/EX. MEM Read &&((ID/EX. Register Rt == IF/ID. Register Rs)or(ID/EX.Register Rt == IF/ID Register Rt))

If the signal is set to 1, we stall the pipeline.

After the implementation of the hazard detection, our pipeline should look like this:



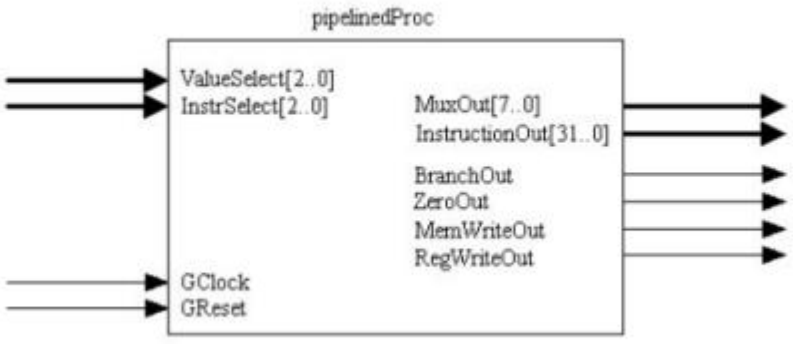
Shipping unit:

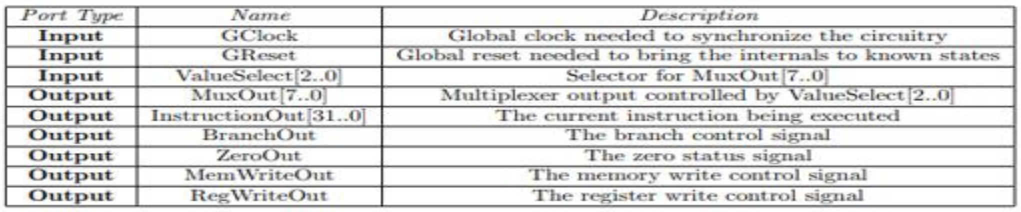
Our shipping unit will work following the equations:

```
If (EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0) & (EX/MEM.RegisterRd=ID/EX.RegisterRs)) ForwardA = 10
If (EX/MEM.RegWrite & (EX/MEM.RegisterRd ≠ 0) & (EX/MEM.RegisterRd=ID/EX.RegisterRt)) ForwardB = 10
If (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0) & (MEM/WB.RegisterRd=ID/EX.RegisterRs)) ForwardA = 01
If (MEM/WB.RegWrite & (MEM/WB.RegisterRd ≠ 0) & (MEM/WB.RegisterRd=ID/EX.RegisterRt)) ForwardB = 01
```

3 Algorithmic procedures and discusion

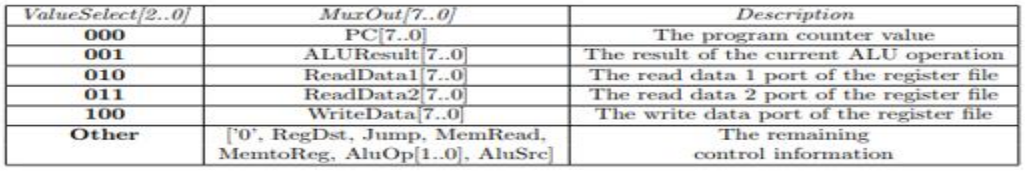
We were asked to build a process RISC pipeline which is the continuity of the previous laboratory where we had implemented a simple cycle processor, for the processor of this laboratory we will also use an 8-bit data paths the latter will allow us to represent operations that include 8 bits concerning the instruction widths they will be 32 bits, the lines controls remain constant, the memory does not change and will always remain 32 bits with a maximum of 256 instructions, the data memory it will be 8 bits and will contain a maximum of 256 words. This implementation will be correct if we assume that our pipeline is an ideal pipeline but unfortunately it is not the case hence the need to use the chances which make it possible to treat the pipeline problems, the diagram of this processor is described in the following figure:





Port Type	*Name*	*Description*
Input	GClock	Global clock needed to synchronize the circuitry
Input	GReset	Global reset needed to bring the internals to known states
Input	ValueSelect[2..0]	Selector for MuxOut[7..0]
Output	MuxOut[7..0]	Multiplexer output controlled by ValueSelect[2..0]
Output	InstructionOut[31..0]	The current instruction being executed
Output	BranchOut	The branch control signal
Output	ZeroOut	The zero status signal
Output	MemWriteOut	The memory write control signal
Output	RegWriteOut	The register write control signal

Table 1: Input/Output Specification



ValueSelect[2..0]	*MuxOut[7..0]*	*Description*
000	PC[7..0]	The program counter value
001	ALUResult[7..0]	The result of the current ALU operation
010	ReadData1[7..0]	The read data 1 port of the register file
011	ReadData2[7..0]	The read data 2 port of the register file
100	WriteData[7..0]	The write data port of the register file
Other	['0', RegDst, Jump, MemRead, MemtoReg, AluOp[1..0], AluSrc]	The remaining control information

Table 2: Output Multiplexer Selection

4 Algorihtmic solution discussion

First, the pipeline consists of breaking down a repetitive process into several subprocesses, each of them running in parallel with other sub-processes and therefore we conclude that several tasks are executed at the same time this process is distinguished by having several characteristics for instructions including:

- Keeping the length as consistent as possible, which allows you to simply retrieve and decode the instruction. Instruction, which is convenient for address calculation: PC+4, Regarding 32-bit MIPS Instructions.
- The format is few, and the location of the register source is the same, which is useful for retrieving the operand when the instruction is unknown and therefore the positions Rs and Rt of the MIPS instruction are fixed, and the values of Rs and Rt can be read when the instruction is decoded.
- Only loading and storage instructions can access memory, which is beneficial in reducing the the number of operation steps, the execution steps of the calculation of the lw/sw address and operation instruction are arranged in the same cycle.
- Storage is aligned in memory, which allows you to reduce the number of extractions. What you need to know is that the pipeline consists of five(5) states as shown in the following table and figure:

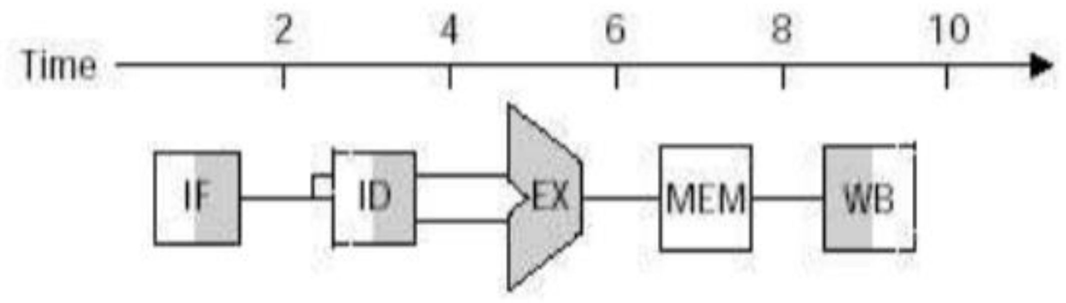


Figure 1: Processeur MIPS à 5 étapes

Instruction Type	Pipeline Stage				
Load	IF	ID	EX	MEM	WB
Store	IF	ID	EX	MEM	
R-Type	IF	ID	EX		WB

- **IF Instruction Fetch**: Allows depending on the memory address stored in the PC, find the instruction at this address in the memory, then the instruction is stored in a register, at the same time time the PC points to the next instruction (PC+4).
- **ID (Instruction Decode)**: uses the instruction extracted from the IF stage, Decodes the instruction, and finally finds the data stored in the register required by the instruction. If the latter only includes one only jump instruction, at this point the acquired values must be compared based on the jump algorithm. If the result of the comparison is true, a jump is made. If the result of the comparison is false, no jump is performed, and the instruction following is executed; if the instruction must fill certain bits of the instruction, it is also executed.bits of the instruction, it is also completed in the ID step, such as filling in four upper bits so that the instruction result is 32 bits and calculate the address of the instruction which can which can jump.
- **EX (Execution):** the ALU calculates the result of step ID. In step ID, the register values necessary for calculating the instruction have been extracted. Then, in the EX step, the values of these registers must be calculated based on the meaning of the instructions. The calculation varies depending on the instruction There are mainly three types of ALU calculations:

1) The ALU calculates the effective address unit based onthe address completed in the ID, and finally fetches the address memory required;

2) Depending on the direction of the instruction, the value extracted from the register, perform operations, such as adding the values oftwo registers;

3) Calculates the immediate result based on the register value and additional value.

- MEM (Memory Access): if the current instruction is an instruction LOAD, then the corresponding value is retrieved from memory by function of the address calculated by EX; if the current instruction is a STORE instruction, then the register value is stored.If the current instruction is a STORE instruction, the register value is stored according to the memory address and the value of the register calculated by EX in the memory address. Others instructions are generally not designed for access to the memory.

- WB (Write Back): writes the final calculated register value to the registry file. This operation includes the value extracted from the memory and the result obtained by an arithmetic operation.The following figures represent a simple cycle processor in 5 steps and instructions that we have just described previously those are provided as support for what was described previously:

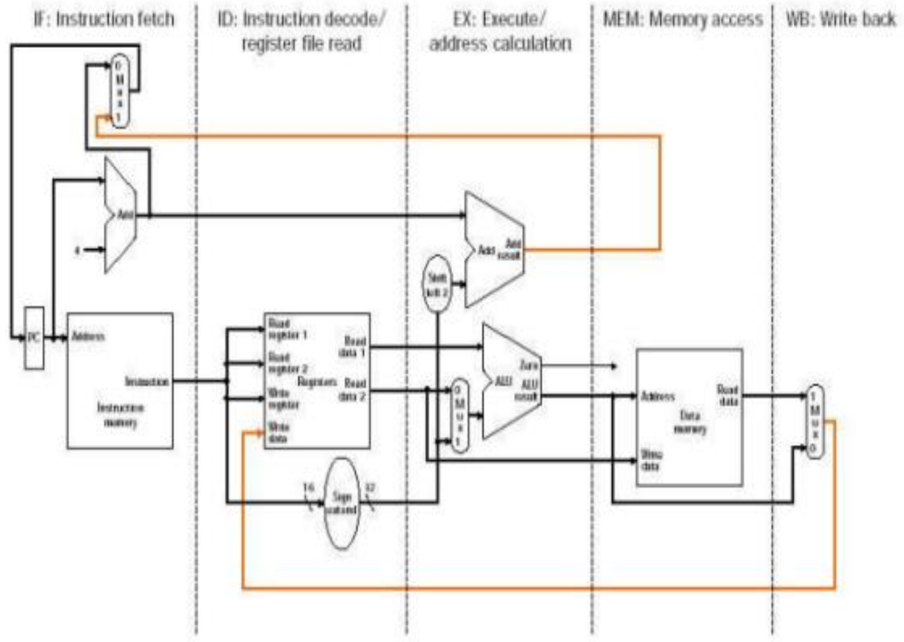


Figure 2: Processeur à cycle simple divisé en 5 étapes

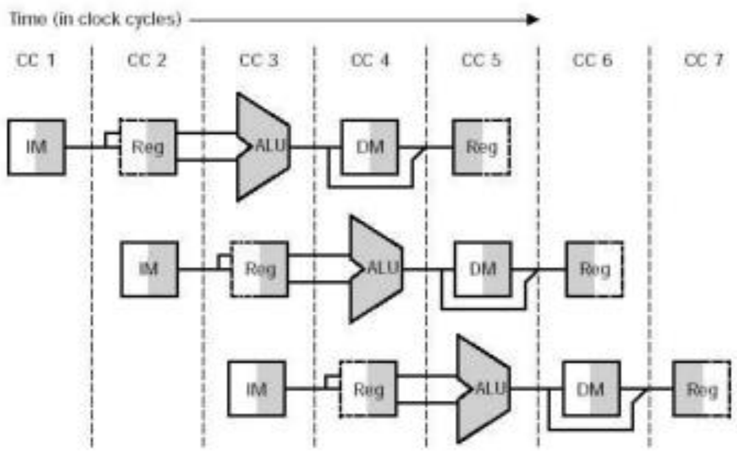


Figure 3: Multiple instructions running on the same datapath

5 Pipeline Hazards Implementation

Pipeline hazards refer to the fact that for a specific pipeline, the next instruction in the instruction flow cannot be executed at a specific clock cycle due to the existence of the correlation. There are three types of pipeline risks: structural risks, control risks and risks linked to data.

- Structural risks: it is a combination of instructions that is not supported by data path.
- Control risk: The latter occurs when a decision is made on the basis results of one instruction, while others execute like the connection instructions for example.
- Data risk: It occurs when an instruction depends on the results of an instruction on the pipeline.

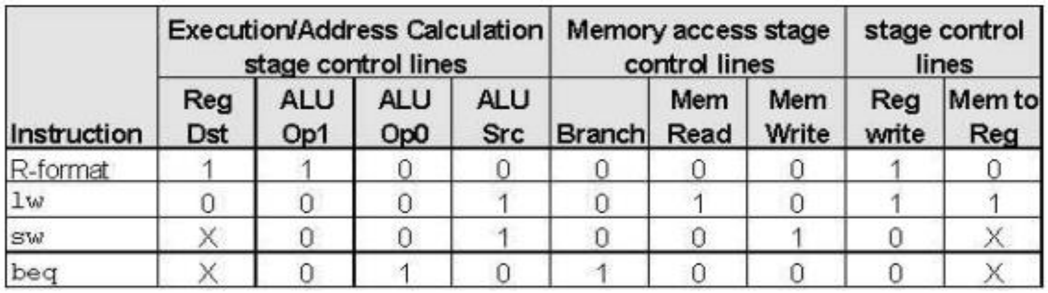
Hazard detection:

Between instruction i+1 and instruction i: ID/EX.WriteReg == IF/ID read-register 1 or 2

Between the instruction i+2 and i (2 bubbles): EX/MEM.WriteReg == IF/ID read-register 1 or2

Between instruction i+3 and i (1 bubble):MEM/WB.WriteReg == IF/ID read-register 1 or 2
Stalls stop instructions in the ID stage. Therefore,have to stop looking for new instructions, otherwise we

Let us interleave the PC and the IF/ID register.



Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Figure 5: Control signal values

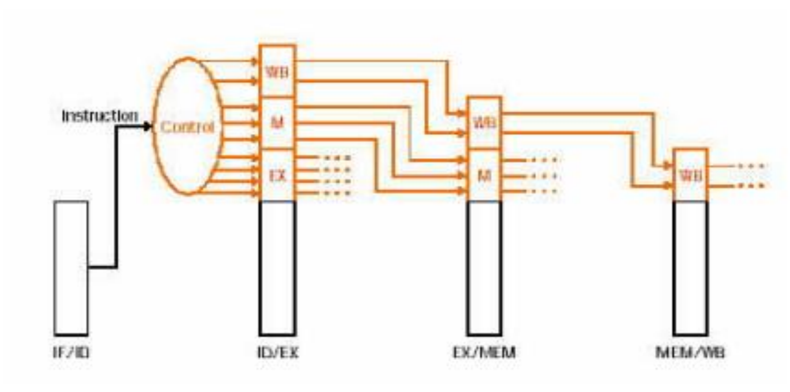


Figure 6: Control signal additions to pipeline registers

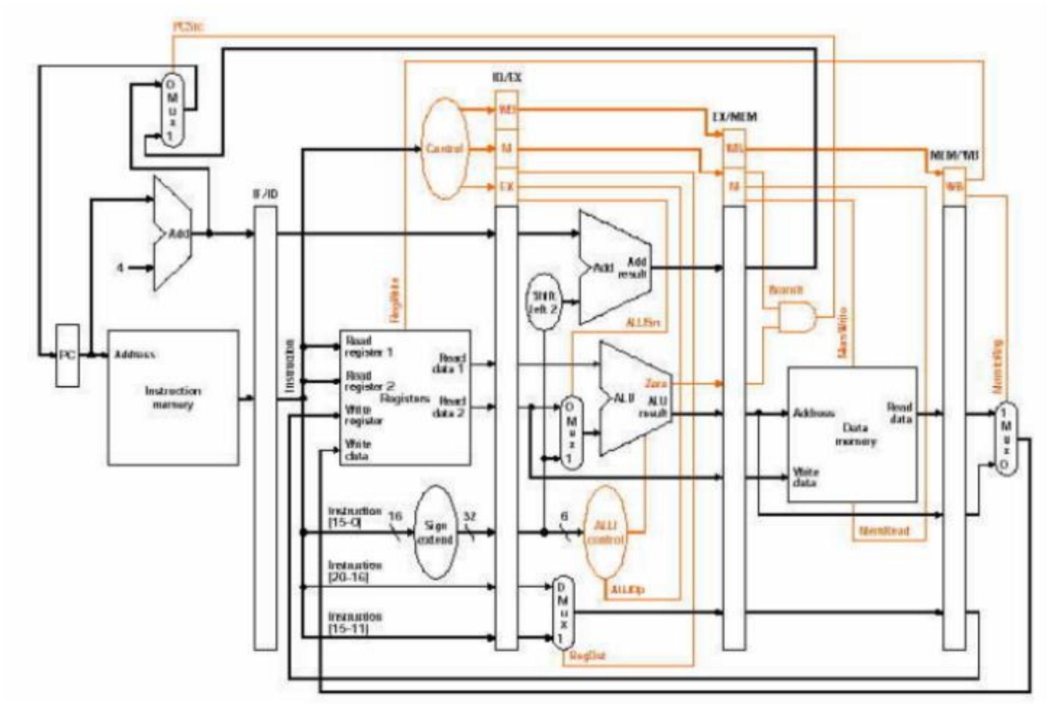


Figure 7: Processeur MIPS en pipeline avec un chemin de contrôle

II Design Part

1 Discussion of Used Components

1.1 The ALU component

This component is explained and detailed in the previous laboratory.

1.2 The ALU Controller

This component is explained and detailed in the previous laboratory.

1.3 The Hazard Unit

This component is mainly made up of logic gate AND, OR, NOT and XOR it takes as input different states of chance in particular:

- the execution value of the memread (EX) in the input “IDEXreadMEM”.

- the execution value of the rt register in the “IDExrt” entry

- The value of the IF ID of the rom which is essentially a 32-bit register which is connected to the “IFIDrt” and “IFIDrs”

The output will be connected to the selector of each of regwrite and memwrite and in each of the enables of pc+4, program counter, and the IF_ID” of the rom

1.4 Full Adder 8 Bits

This component is explained and detailed in the previous laboratory.

1.5 8 Bits Register

This component is explained and detailed in the previous laboratory.

1.6 8 Bits Substractor

This component is explained and detailed in the previous laboratory.

1.7 5 Bits Register

This component is mainly made up of 5 registers to 1 bits and it is used for:

- the MEM/WB registers for the rd register and the write register .

- ID/EX registers for each of rt, rd and rs

- the EX/MEM registers for the rd register and the write register

1.8 Full Adder 1 Bits

This component is explained and detailed in the previous laboratory.

1.9 Forwarding Unit

This component mainly uses logic gates between its different inputs: AND,OR,XOR and NOT, the inputs of this last are connected to: Regwrite (state: EX/MEM and MEM/WB) also for the rd register (the SAME/WB state and EX/MEm), the rs register (the state: ID/EX) and the rt register for the state: ID/EX.

Concerning the output values one of them is connected to the two multiplexers which allows us to make the choice after having read the input data,

1.10 1 Bits Substractor

This component is explained and detailed in the previous laboratory.

1.11 Multiplexer

This component is explained and detailed in the previous laboratory.

1.12 RAM

This component is explained and detailed in the previous laboratory.

1.13 ROM

This component is explained and detailed in the previous laboratory.

1.14 Register File

This component is explained and detailed in the previous laboratory.

1.15 32 Bits Register

This component brings together 32 D rockers (which are supplied) and it is used to model the instruction steps in particular EX/MEM,ID/EX,IF/ID,MEM/WB.

1.16 Top

This is the final entity where we have grouped all the components mentioned previously so that the latter looks like Figure 7 represents in the part of the discussion of algorithmic solution.

2 Discussion of the current solution (VHDL code):

Our solution to the problem has been sent with this file in format VHDL where you will find all the corresponding code, and particularly all components used in this laboratory.

3 Discussion of the tools (Optional):

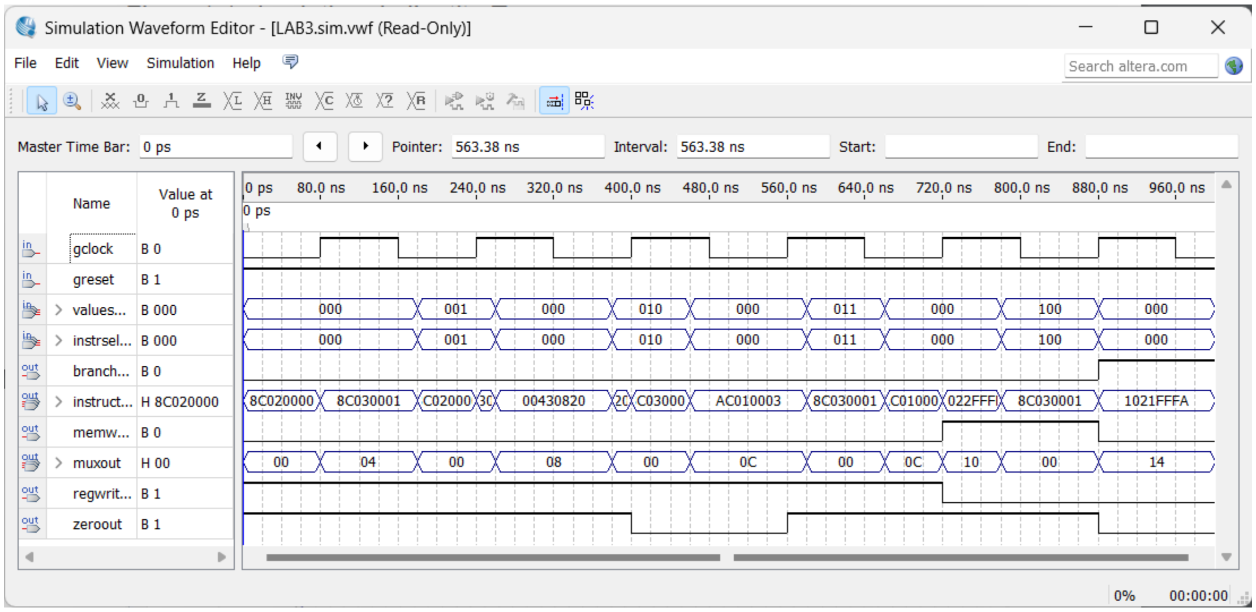
For this laboratory it was a question of implementing our entities with VHDL code on quartus 2 to then demonstrate on the altera card we had a lot of problems with the altera card because we work online and we don't have access to university tools and therefore we couldn't know if our design would work or not we had to wait until the day of the presentation to test our designs and we had only our simulation as a benchmark.

4 Discussion of encountered problems

We encountered several problems in this laboratory for instance we found it difficult to test our designs especially when Quartus gave us errors and we arrived at advanced stages in our code we were obliged to return to view entity by entity and make additional outputs to try to debug our code. For instance, we had difficulties in testing some designs especially for the type-J instruction path because we never obtained the expected result and after a lot of debugging we reached realized that we had forgotten to add the shift before entering thevalue in the ALU with PC+4.

III Real Implementation and Simulation

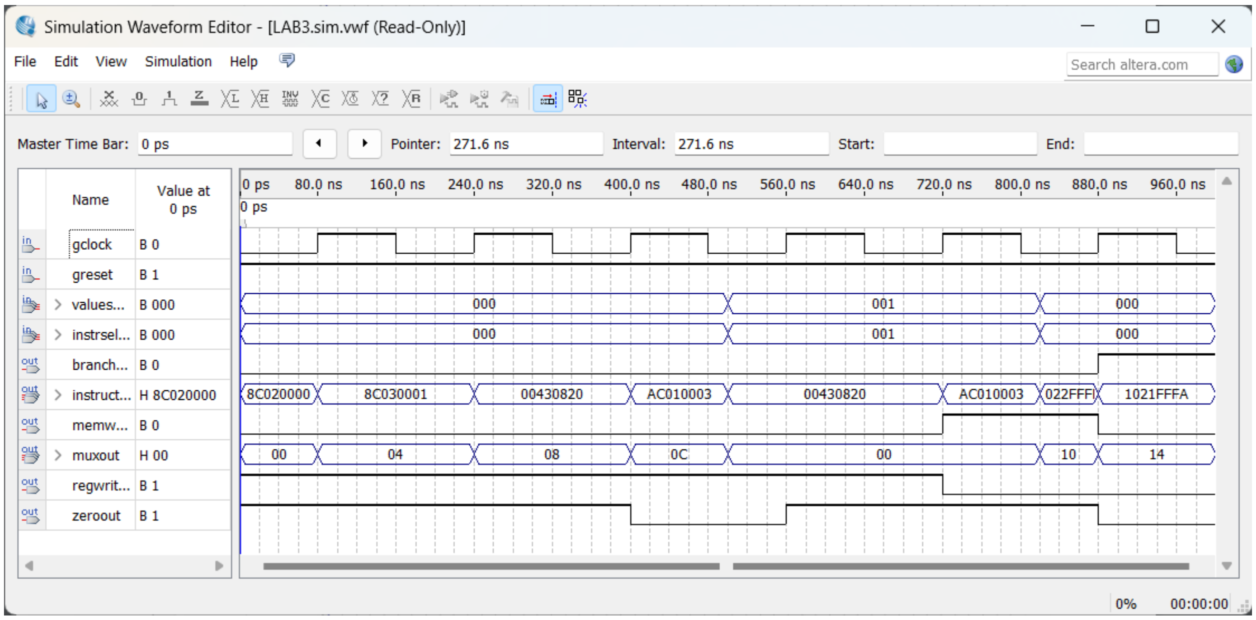
1 Simulations of all components



Master Time Bar: 0 ps Pointer: 563.38 ns Interval: 563.38 ns Start: End:

Name	Value at 0 ps	Waveform
gclock	B 0	
greset	B 1	
values...	B 000	000, 001, 000, 010, 000, 011, 000, 100, 000
instrsel...	B 000	000, 001, 000, 010, 000, 011, 000, 100, 000
branch...	B 0	
instruct...	H 8C020000	8C020000, 8C030001, C02000, 3C, 00430820, 2C, C03000, AC010003, 8C030001, C01000, 022FFF, 8C030001, 1021FFFA
memw...	B 0	
muxout	H 00	00, 04, 00, 08, 00, 0C, 00, 0C, 10, 00, 14
regwrit...	B 1	
zeroout	B 1	

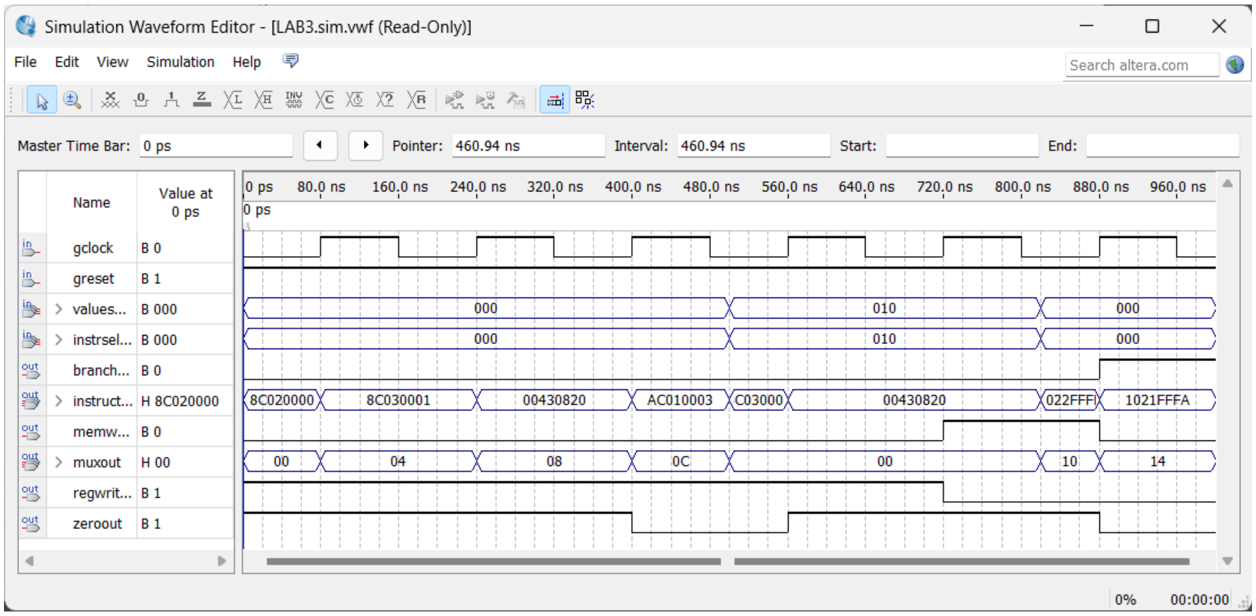
1.1 Top level Entity simulations



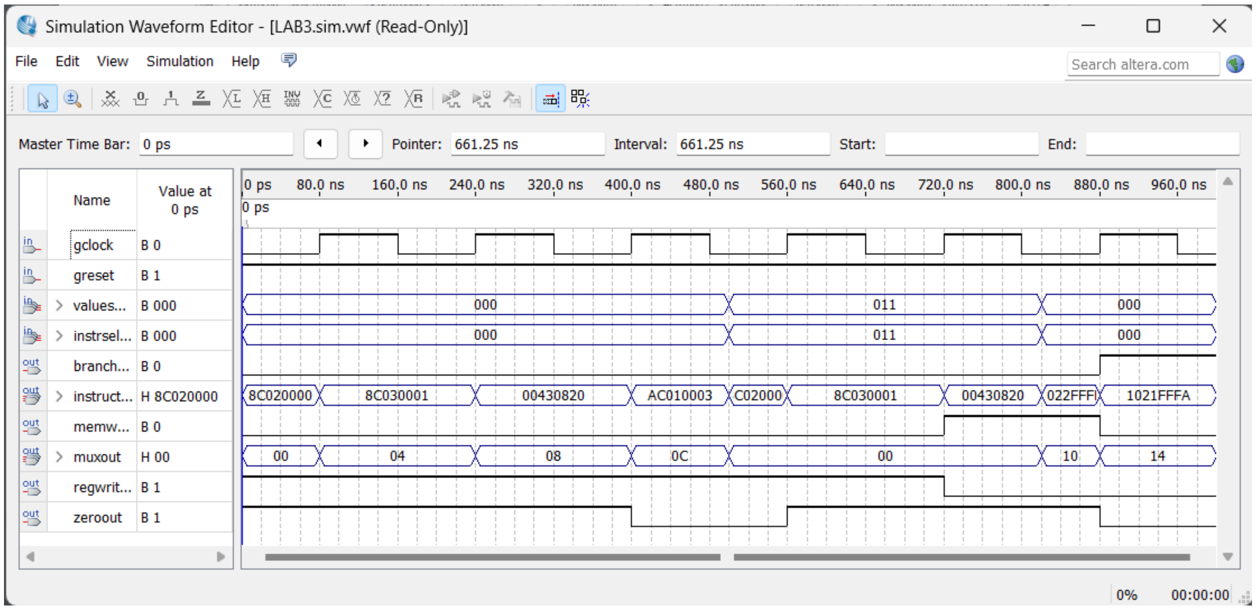
Master Time Bar: 0 ps Pointer: 271.6 ns Interval: 271.6 ns Start: End:

Name	Value at 0 ps	Waveform
gclock	B 0	
greset	B 1	
values...	B 000	000, 001, 000
instrsel...	B 000	000, 001, 000
branch...	B 0	
instruct...	H 8C020000	8C020000, 8C030001, 00430820, AC010003, 00430820, AC010003, 022FFF, 1021FFFA
memw...	B 0	
muxout	H 00	00, 04, 08, 0C, 00, 10, 14
regwrit...	B 1	
zeroout	B 1	

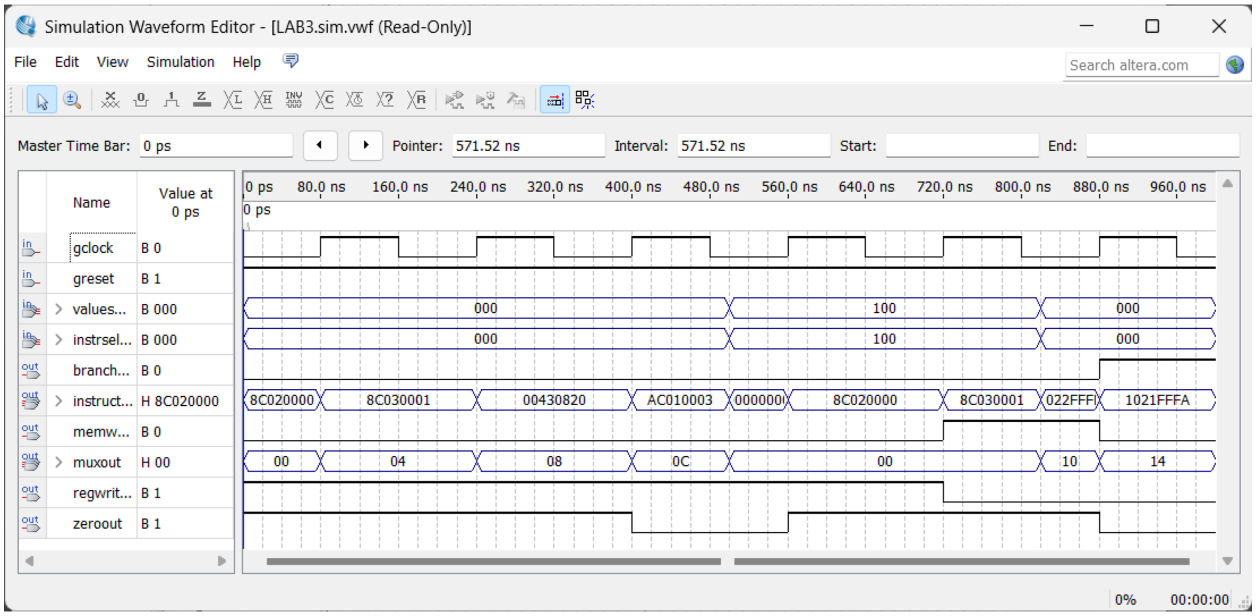
1.1 Top level Entity with instrselect = 001 simulation



1.1 Top level Entity with instrselect = 010 simulation

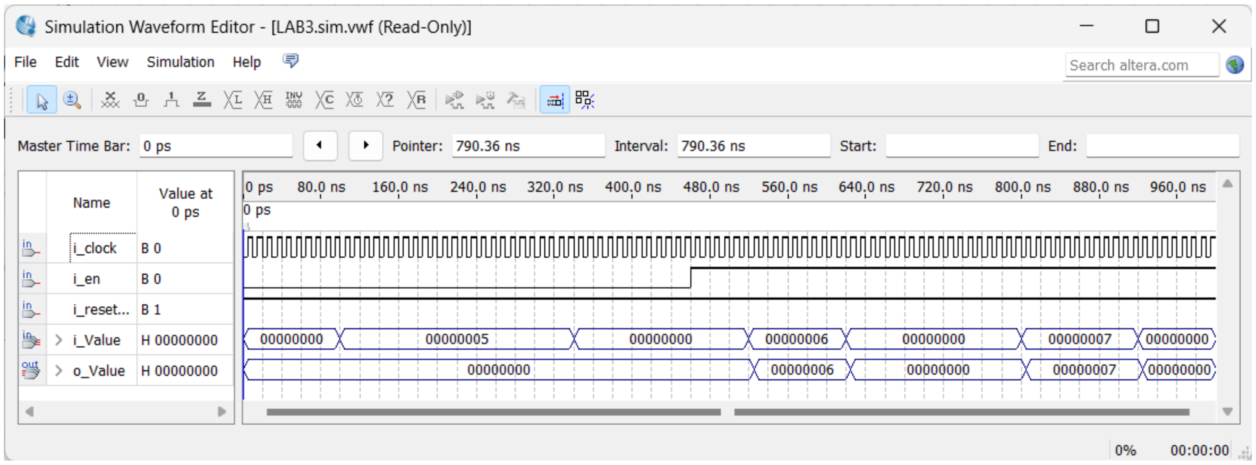

Simulation Waveform Editor - [LAB3.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 661.25 ns
Interval: 661.25 ns
Start:
End:
Name
Value at 0 ps
gclock B 0
greset B 1
values... B 000
instrsel... B 000
branch... B 0
instruct... H 8C020000
memw... B 0
muxout H 00
regwrit... B 1
zeroout B 1
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
000 011 000
000 011 000
8C020000 8C030001 00430820 AC010003 C02000 8C030001 00430820 022FFF 1021FFFA
00 04 08 0C 00 10 14
0% 00:00:00

1.1 Top level Entity with instrselect = 011 simulation



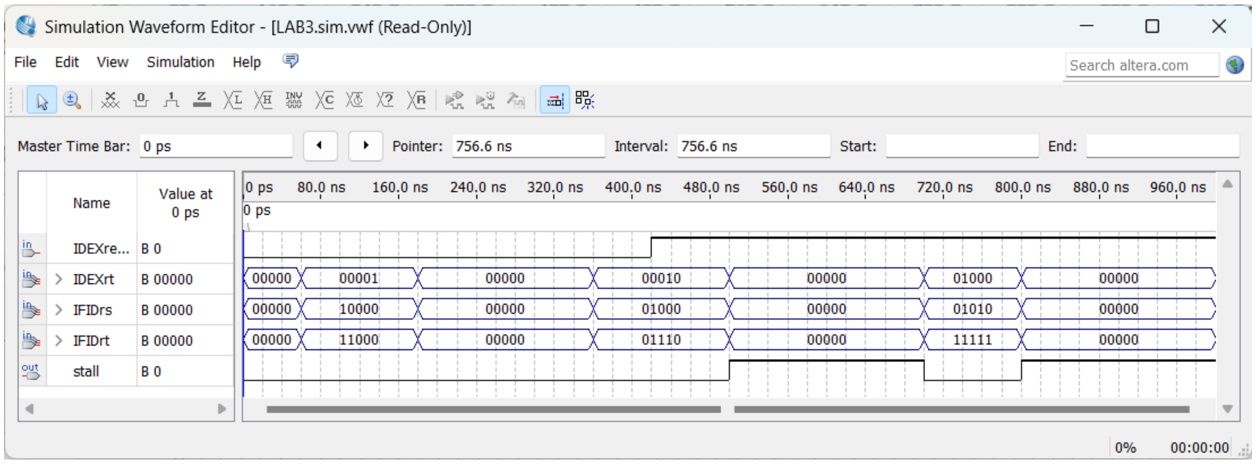
1.1 Top level Entity with instrselect = 100 simulation

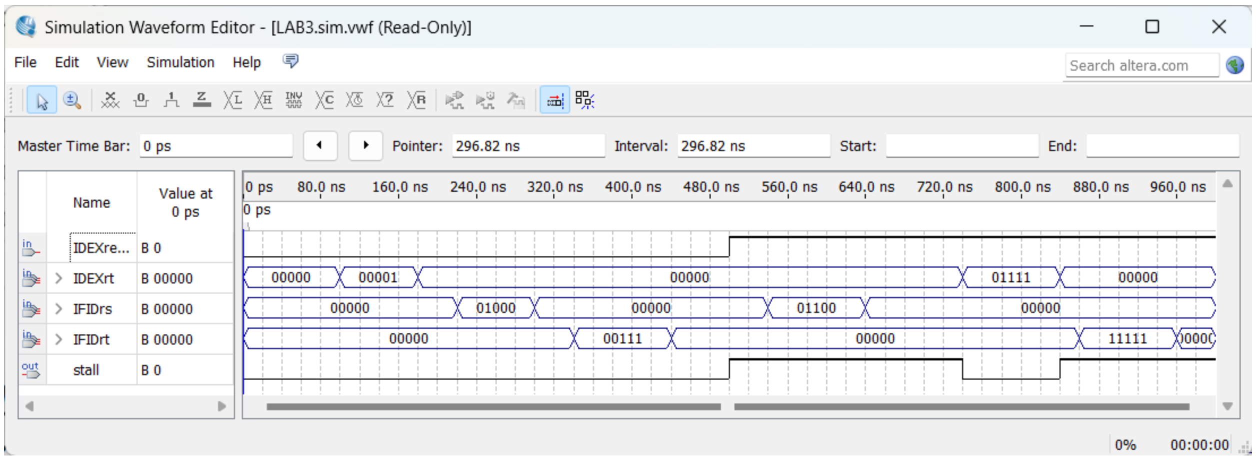
We can notice that in the following figures that our 5 steps of pipeline work perfectly, so as we can see in our instrselect input which are responsible for selecting the instruction multiplexer and therefore pass from instruction to instruction this which we do by changing the values of the latter depending on the case that we want (IF,ID,EX,MEM,WB) and we find the expected result for each of them in our output instruction which corresponds perfectly meets our expectations for example taking the example of WB in 011 when we do a fetch we notice that there are two values in the instructions and that the memwrite is activated during the second value until the end of the statement put in the input which is the expected result since we want to write in memory.



1.2 32 Bits Register simulation

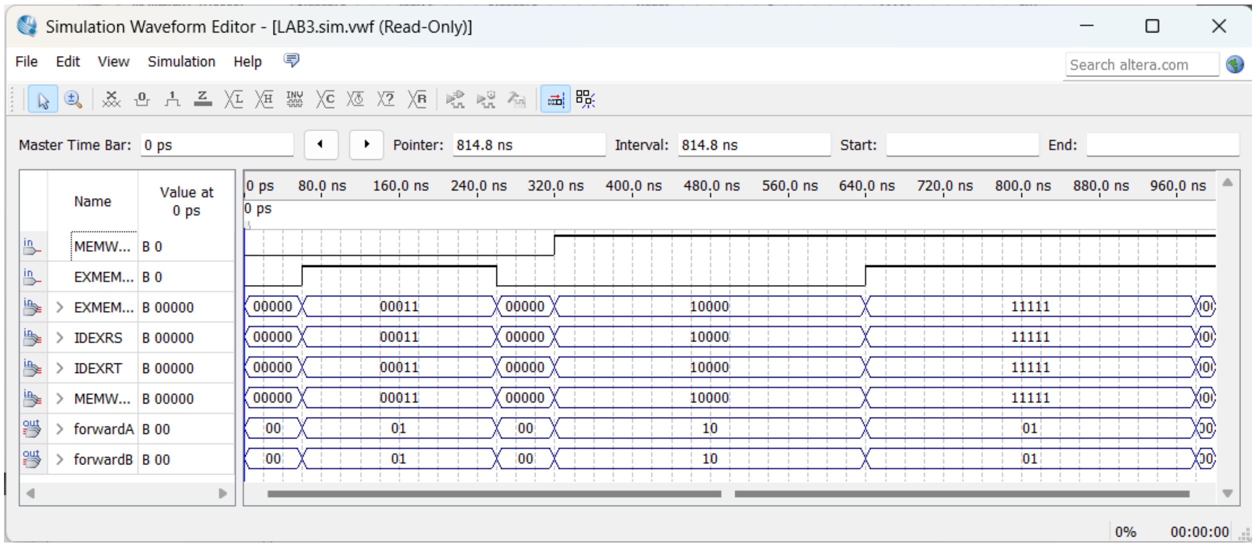
We notice that our simulation is correct because as we can see notice so that the output value gives us the input values wanted the enable (i_en in our case) to be activated in addition from the resetbar.





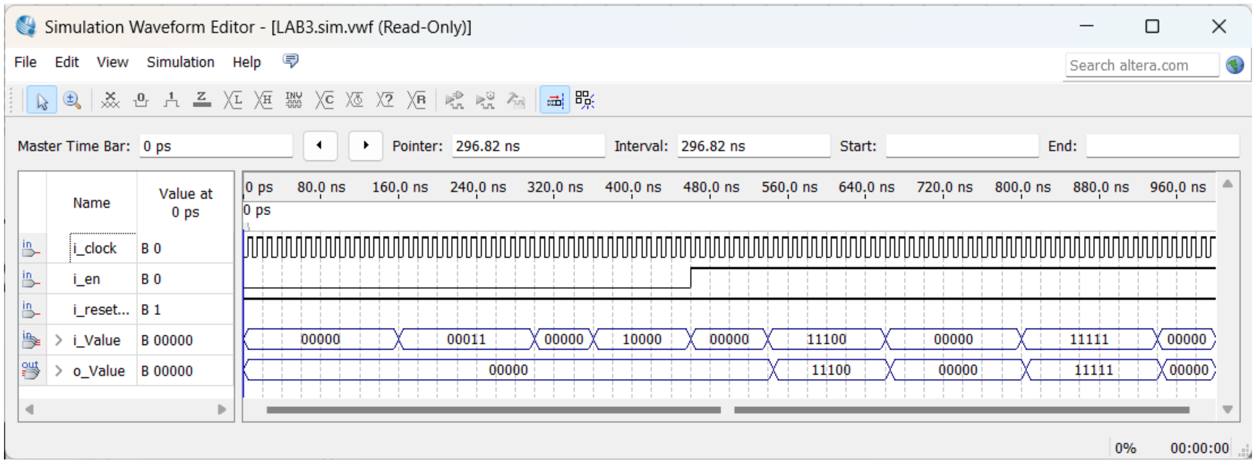
1.3 Hazard Units simulations

Here we notice that the stall begins directly with the passage of IDEX to 1, and the other values IDEXrt, IFIDrs and IFIDrt follow the order as shown during the course. The transition between the stages of pipeline is running correctly.



1.4 Forwarding Unit Entity simulation

The desired results are well presented in the simulation. Forward A and B switch between 01 and 10 and are proportional to the EXMEM, IDEXRT, IDEXRS and MEMWBRD inputs. They also go to 0 as soon as the inputs are 0. The registers entering at each step is like what is found in the rest of the lab.


Simulation Waveform Editor - [LAB3.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 296.82 ns
Interval: 296.82 ns
Start:
End:
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns

Name	Value at 0 ps
i_clock	B 0
i_en	B 0
i_reset...	B 1
i_Value	B 00000
o_Value	B 00000

```
i_Value: 00000 | 00011 | 00000 | 10000 | 00000 | 11100 | 00000 | 11111 | 00000
o_Value: 00000 | 11100 | 00000 | 11111 | 00000
```

1.5 5 Bits Register simulation

We notice that our simulation is correct because as we can see notice so that the output value gives us the input values wanted the enable (i_en in our case) to be activated in addition from the resetbar.

2 Design verification.

- Calculation of the Maximum clock frequency:

The processor at the beginning contains: $= lw\ delay$

$=PC + instruction\ mem + register\ file$

$+ ALU +$ data memory $+ mux + regwrite$

$=1 + 10 + 10 + 2 + 15 + 10 + 2 + 10$

$=60\ ns$

We therefore obtain: 2 mux + 1 ALU+ 1 mux= 51+ 4+ 15+2 = 81 ns

And so for the frequency we obtain: 1/81ns = 12.35 MHz

- Calculate CPU execution time:

$Cpu = CPI * Instruction\ time / Clock\ rate$

Now we have 4 load (5 cycles) + 1 sub (4 cycles)+1 or (4cycles) +3 branch (3cycles) + 2 store (4cycles) +1 add (4cycles) +1 jump (3cycles) = 13

And so

Cpi = 4*5+1*4+1*4+3*3+2*4+1*4+1*3 = 4

Cpu = 13*4/12.35 *10^6 = 9.73 * 10^−6s

In comparison with the simple cycle processor we obtained:

Cpu = 12/12.35 *10^3 = 9.73 * 10^−4s

Therefore, we notice an increase of approximately 198% in the speed of the processor compared to the single cycle processor. This increase in CPU time is due to the increase in CPI resulting from the implementation of the 5 pipeline stages.

-Calculate the worst delay that the ALU can have:

Worst delay = 31*2*3=65

Knowing that :

Carry propagation:=2

Sum =3

And so :

Worst delay = 65*0.01=0.65ns

IVDiscussion

Each component of our pipelined processor has been simulated and tested. We also managed to build all blocks units for each component. Although we had not fully built the top-level entity which was caused to a little of uncertainty. On the other hand, there was no time left for us to continue with the complete demonstration.

V Conclusion

Finally, we can conclude that this laboratory experiment was relevant because not only we were able to design and build a pipelined processor using structural VHDL coding implementation, and we got familiarized of designing and testing a pipelined processor on altera board. We were finally able to demonstrate and have a clear understanding of pipelined processors and instructions set architecture which justifies the main objective of the lab.

VIReference

The Lectures notes seen in class about Pipelined processors.

The instruction given in the lab2 manual.

The required textbook for this course:

Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface , 6th edition, Morgan Kaufmann, 2021.

VII Annex

All captures of the designs , simulations and diagrams used during the lab experiment session are included in the lab report.