CEG3156-Computer System Design

12/2/2024

Lab#2:

Floating-Single-Cycle RISC 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 single cycle RISC-Type using VHDL coding implementation. After completion of this laboratory, we will be able We will also be able to demonstrate and have a clear understanding of RISC processors and instructions set architecture.

Theoretical Part

1 Introduction of the laboratory

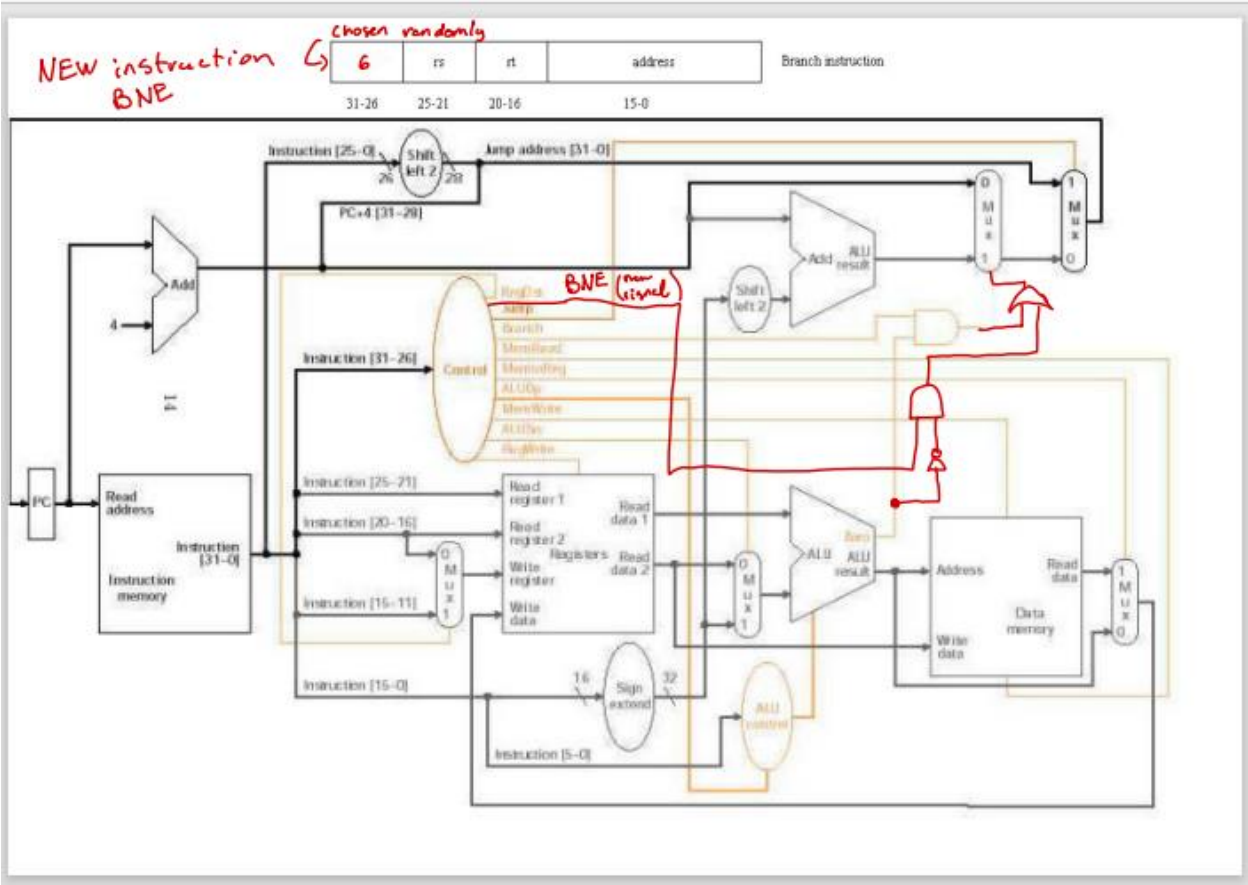
The goal of this laboratory is to design a RISC type processor simple cycle in VHDL and more precisely in this laboratory we will implement a MIPS processor with its main functionality including memory reference instructions, arithmetic instructions and control instructions, this laboratory is important 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 Modify the Datapath shown for the single-cycle processor in the textbook and included in this lab (figure 9), to support a new instruction such as the branch if not equal (bne) instruction. And show the modifications of the change datapath, as well as any control signal changes that need to be done to support the new instruction.

First off, a random number is chosen from the Branch Instruction register, secondly, a new Branch Not Equal(bne) signal is added in the control signal to perform the logical operation. Thirdly, an And logic is performed between the branch not equal and the inverted zero value coming from the ALU unit. And finally an Or logic operation is then done between the branch and the result from the first operation to send a 1 to the multiplexer. See the modified Datapath below.



3 Algorithmic procedures and discusion

This laboratory experiment consists of designing a processor with simple MIPS cycle while using 8-bit data paths and 32-bit instructions and to do this, we had to use a register file of eight eight-bit registers, memory instruction (32 bits) that contains 256 instructions while the data memory will also be eight bits and has 256 words. The main feature of this processor is that all Instructions must be executed in one clock cycle that means we had to use a clock cycle allowing to accommodate the longest instruction that was supported. In our case, it was the load word instruction, as we will be implementing later. The schematic of this processor is described in the following figure and tables below.

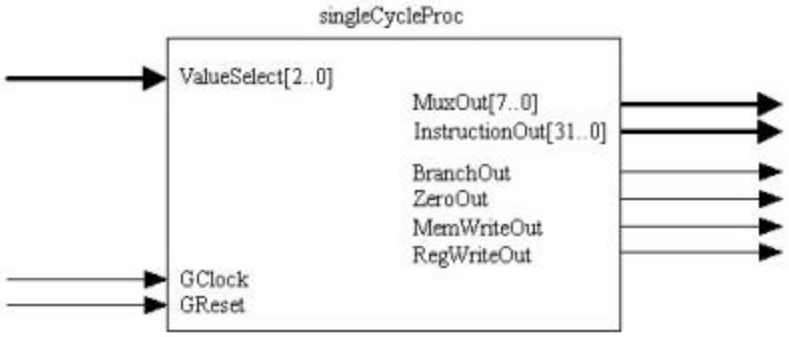


Figure 8: Single-cycle processor entity

The MuxOut[7..0] and ValueSelect[2..0] output and input are intertwined in the following manner:

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

- **Fetch and Increment**
  The development strategy for this processor consisted of developing each component separately, then group the components together to memory and to

have a response to a clock signal that controlled the flow of instructions, such as components were all developed using the structural VHDL language, it was easy to group them together once each component was simulated independently. To run the CPU we first needed to do the fetching and incrementation for which it is necessary to decode the instructions in the ALU and the fetch is done relatively to the fetch calculates in the ALU. Regarding the fetch instruction, the address of the instruction is the contents of the program counter (PC) and the contents of the last component is used as address to access memory instruction for the purpose of obtaining the instruction code, concerning incrementation during sequential execution, we incremented the value of pc by 4 and during a connection instruction the value of pc will be the target address of the connection. The data path of fetch and increment is given in the figure below.

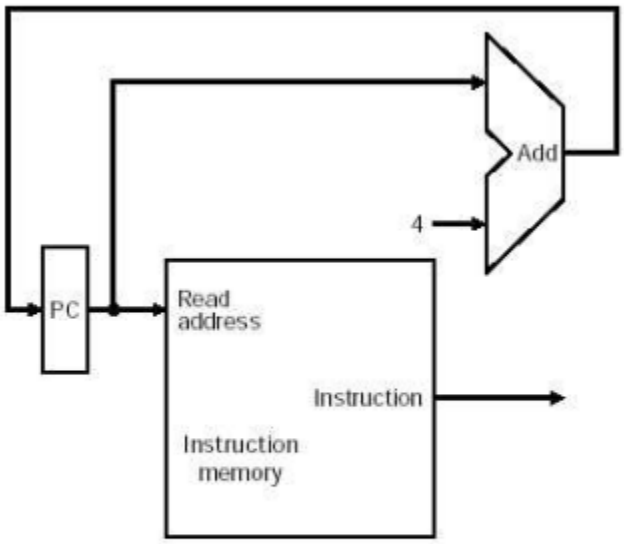


Figure 4: Fetch and increment datapath

- **R-type Instruction Datapath**

The R-type instruction Datapath is shown in the figure below, and utilizes two basic building blocks: a register file and an ALU. All data signals are 32-bits wide, and we have thirty-two 32-bit registers, hence a 5-bit register addressing is required for the register file. The latter contains two read ports and one write port, allowing three simultaneous accesses (two reads and one write) to the register file. The ALU, on the other hand, is very similar to the one designed in class, and consists of a 32-bit MIPS ALU, capable of addition,

subtraction, AND/OR logical operations and a set less than (comparison) operation. It provides a Zero status signal indicating whether the ALU result is equal to zero (Zero = 1) or not (Zero = 0). As a forementioned, the ALU result must be written back into the register file for later use. Finally, note that a control signal (RegWrite) is needed to control the writing mechanism of the register file. (Instruction taken from the class lab2 note).

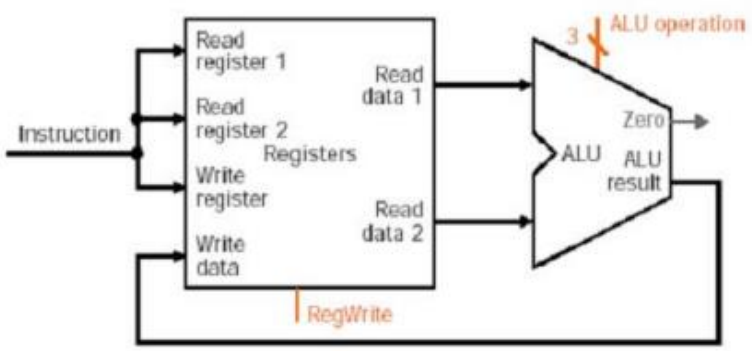


Figure 5: R-type instruction datapath

- **I-type Instruction Datapath**

Regarding the data path of type-l instructions, we started by retrieving the instruction from the register instruction (instr MEM), and the PC will then be incremented the contents of register 1 are read from the register file then The ALU adds the data read in register 1 to the value 16-bit lower part of the extended sign instruction when result of the ALU operation the latter is written in the registry file, and the address of the registry file is determined by rt. The following figure below illustrates the Datapath of the I-Type instruction path.

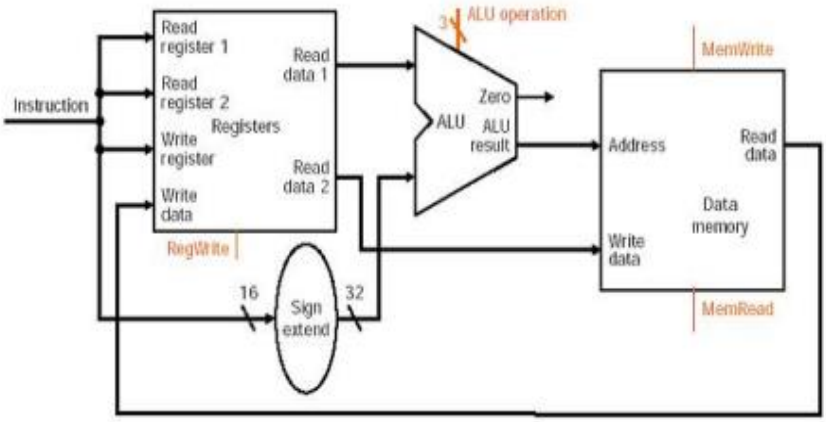


Figure 6: I-type instruction datapath

- **J-type Instruction Datapath**

  The J-type instruction Datapath is shown in the figure below and utilizes three basic building blocks: a register file, an ALU and a sign-extension unit. All data signals are 32-bits wide, except for the 16-bit address offset value. The register file outputs the contents of the two operands, in the case of a branch instruction, whilst the ALU proceeds to compare the two numbers. If the latter are found to be equal, then the Zero status flag is asserted. The main control unit then decides whether to take the branch or not. If the branch is to be taken, the branch target is written to the PC. The branch target consists a base address (PC + 4) and the sign-extended offset, shifted by 2 to the left, in order to make the jump a word offset. In the case of a jump instruction, shown later on, we calculate the new address simply through a shift left by 2 of the address field in the instruction. (This instruction refers to the class lab2 note).

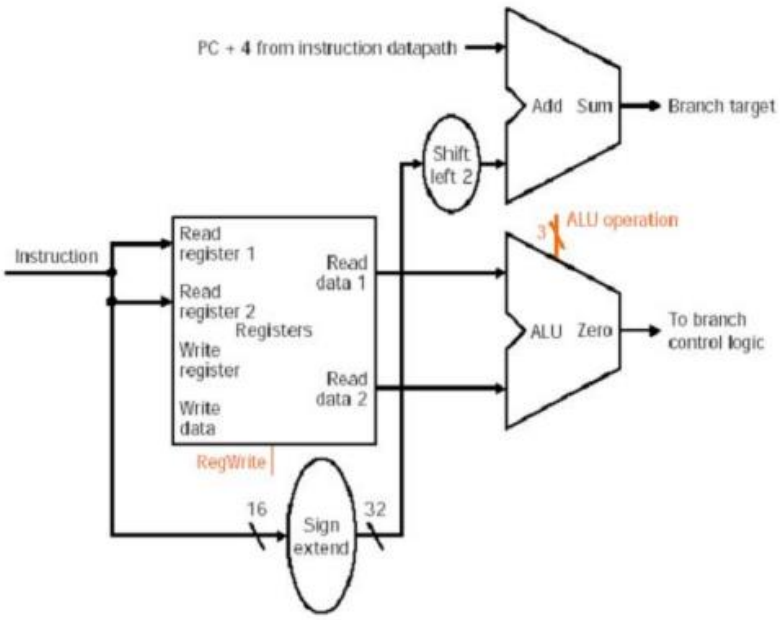


Figure 7: J-type instruction datapath

5 Complete Datapath

The completed Datapath consists of grouping all components built in the previous sections and adding all associated multiplexers and control signals, as well as the ALU control and seven (7) units and the main control unit, to construct our single-cycle RISC processor seen in class.

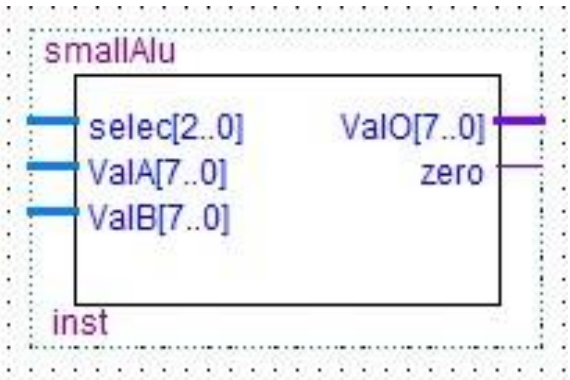
II Design Part

1 Discussion of Used Components

1.1 The ALU component

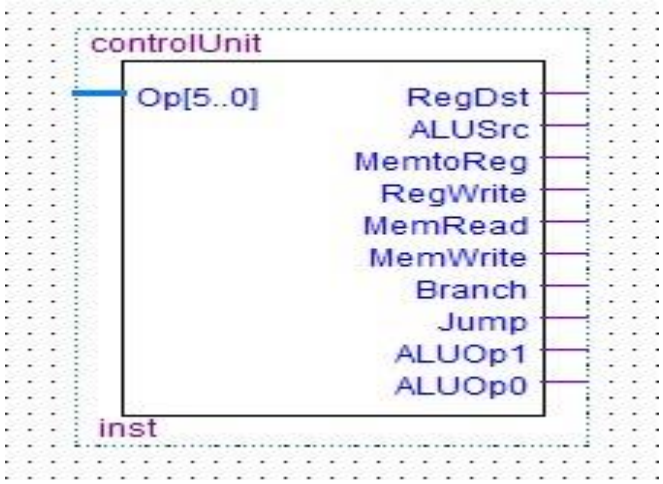
The ALU performs several operations depending on the value of the ALU controller. Logical operations AND, OR and SET ON LESS THAN were supported, as well as arithmetic operations of addition and subtraction. A CLA unit was incorporated into the ALU to improve efficiency and reduce delays, to do this we will use a multiplexer five(5) to 1 with the ALU entities as input

value notably AND,OR,ADD,SUB and as output we find the value chosen by the multiplexer and an output for a value null (Zero)



1.2 Control Unit component

The control unit is the main component of the processor which sends signals to activate certain components. The nine(9) different control signals that are sent are determined by decoding the opcode of each instruction. The opcode is defined as the first 6 bits of each instruction. An opcode of 0 identifies the instruction as R-Type, 35 for the loading word, 43 for the storage, 4 for connection and 2 for instructions jump, to do this we had to use the logic gate OR, AND and NOT to combine between the opcodes and obtain the value obtained.

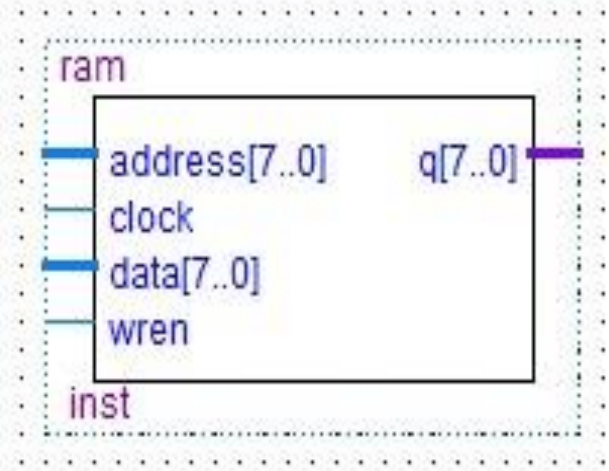


1.3 The Registrer File

The register file was built as a register file of 8 registers. To write to a register file, it was necessary to update the value of the WriteReg signal. Each of these registers had an activation signal which became active depending on the value of WriteReg, so that only the specified register was written. Two registers can be read from the registry file. The date of any register was read by specifying the correct register using the ReadData signal, then setting the value of the register output to a signal, to do this we used two 9 to 1 multiplexers each of them has an input to read the data and these two multiplexers aim to choose between the 8 registers of 8 bits which are themselves connected to AND gates between the WriteReg and Write register without forgetting the write data which is connected to all registers.

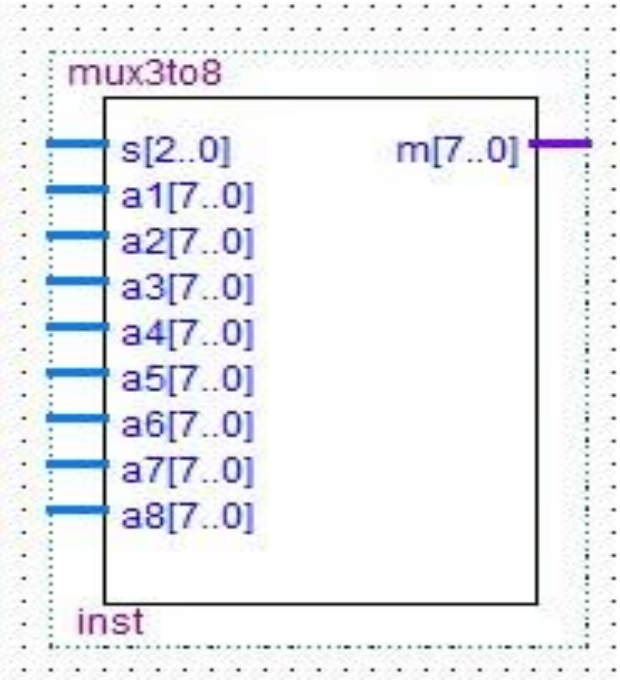
1.4 Data memory

The Data memory contains stored values that can be read and written to the processor. She uses the Altera's "altsyncram" function. It has an address length of 8 bits and 256 addresses, each memory location has a 8-bit width. The 8-bit address is provided by the result of the ALU, it uses the same address port for reading and writing. Write data comes from the ReadData port of the registry file. 8-bit read data is outputs into a mux to choose the data that goes to writing the register. Sound control signals MemRead and MemWrite, indicating whether the data memory is reading or writing to memory.



1.5 Multiplexer 3 to 8

This unit represents a multiplexer representing the final values which will be displayed. These values will be determined according to the selection value that we will choose as input there also has other inputs including the different value of the control unit, the counter program register, the value of the ALU, the two outputs of the ReadData which are the outputs of the output register.

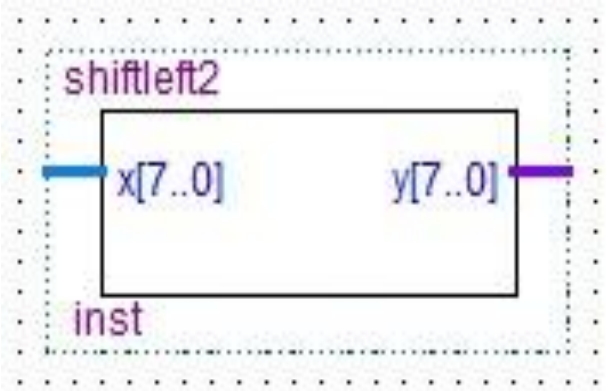


1.6 ROM Instructions

This unit simply contains the instructions which must be executed.The unit choses between two to one to synchronize with the clock and the other concerns the output of the counter program.

1.7 Shift-Left2

This register allows us to shift the value from 2 to the left and it will be used when an instruction is in the J-Type it will be made in the ALU with an incrementation of PC+4.



1.8 Multiplexer 2 to 1 with eight bits

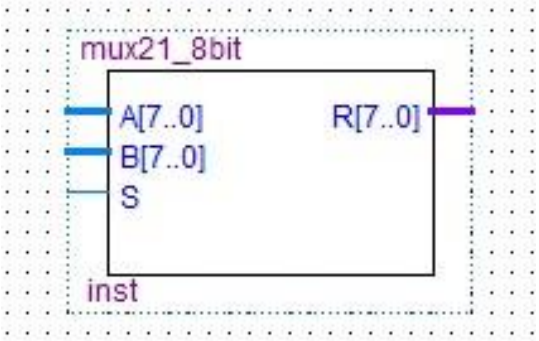
These are multiplexers with 3 inputs 2 of them are the values to choose and one is the selectordepending on each case:

The SelectWrite which takes as input the value of aluminum and the value of the data memory and the selector is the signal memoReg coming out of the control unit.

The SelectTread takes as input the value of the instructions and the value of read data 2 of the Register File and the selector. This is the AluSRC signal coming out of the control unit.

The BranchSelect takes as input the value of the register of PC+4 and the value of the next PC value for the selector we make an AND gate with the Branch signal coming out of the control unit and the zero-signal coming out of the ALU.

The JumpSelect takes a value as an input and and the selector is the signal of the Control unit.

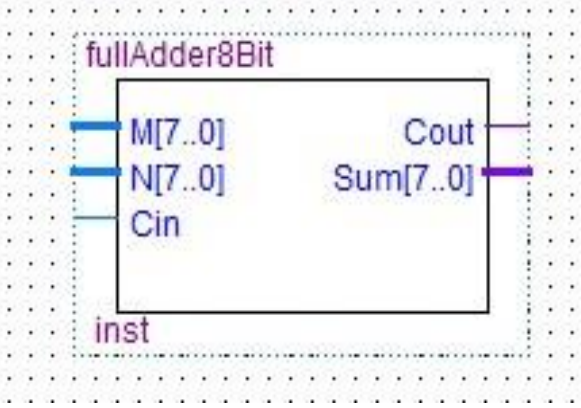


1.9 Eight bits FullAdder

This register which is an 8-bit adder representing the addition of the next address of PC and PC+4 mainly has three inputs one of which is the carry-in and the two more are the entries we needed to add which are.

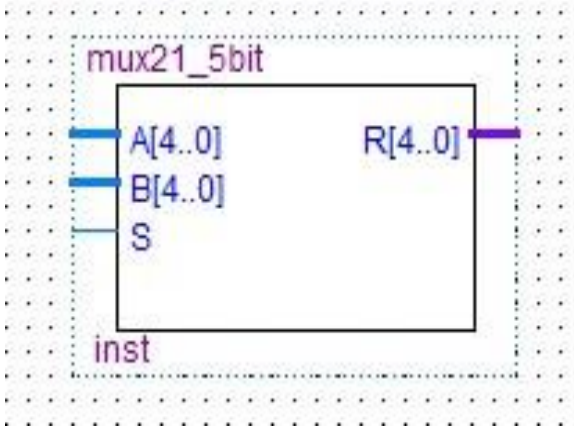
For PC+4, the Program counter value, and the value of 4(to increment by 4)

For the next PC value, the value of the memory instruction and the output of the PC+4.



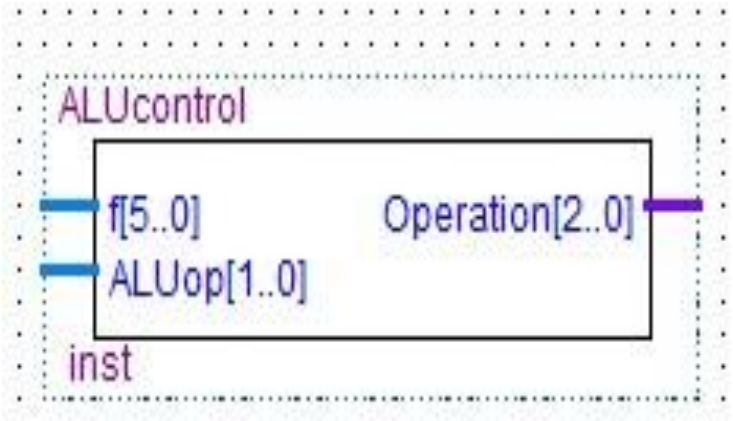
1.10 Multiplexer 2 to 1 with five bits

This multiplexer represents the WriteReg which is activated when we want to write to the register and as input it takes the instructions that we want to execute, and the selector this is the MemoReg signal coming out of the control unit.



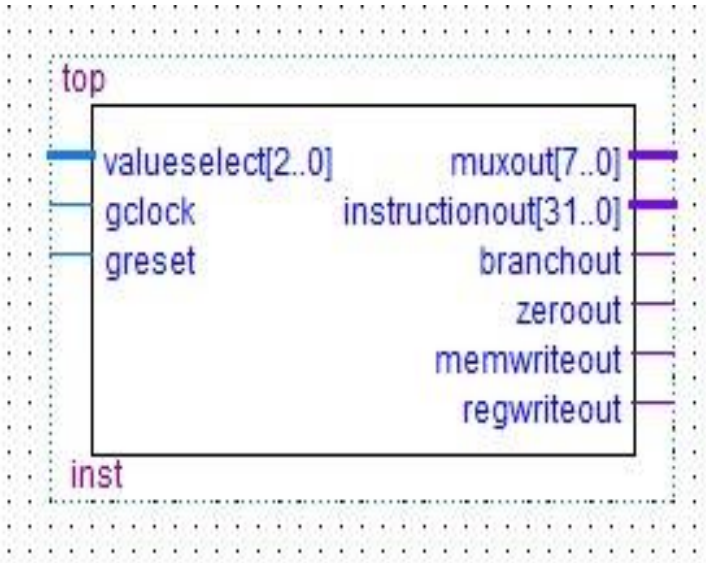
1.11 ALU control signal

This unit consists of choosing a value to select in the ALU it already has as input ALUOP of the control unit and the instruction is done by logic gates AND, OR and NOT.



1.12 Top level.

It represents the upper level of the processor and consists of grouping all components, thus creating the single-cycle processor.



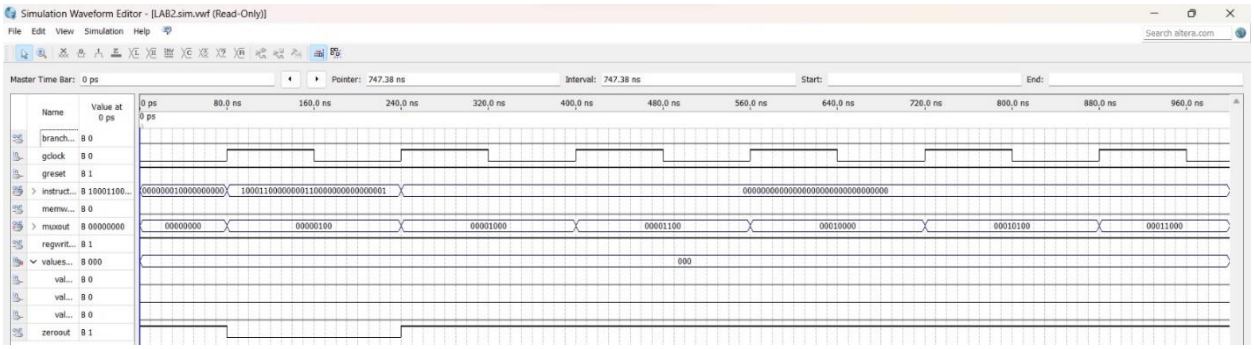
2 Discussion of encountered problems

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

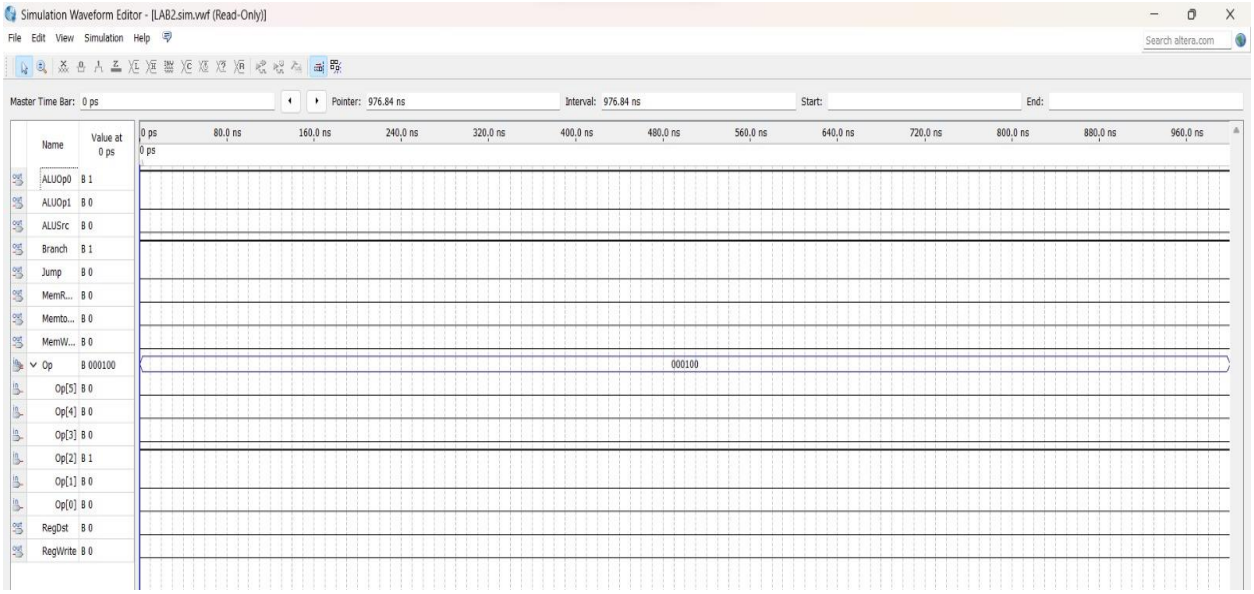
1.1 Top level unit simulations



As we can notice in the final top level figure below where the valu value select a 00 we notice that the Mux Out incremented by 4 which is the result expected but we can notice that we have a problem concerning the instructions because they do not change over time we notice that there are only 2 values and this can be explained by a poor implementation of the memory instruction register.

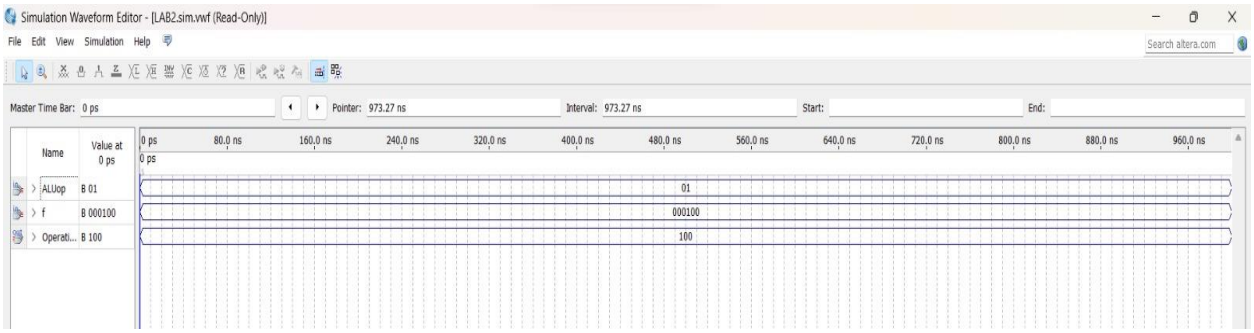
1.2 Control unit simulation

Here we can notice that the simulation of the control unit represents the different bits of the CPU control. The ALUop is 01 and Branch is 1. This represents a branching action with the ALU to perform an addition to represent a connection to the offset.



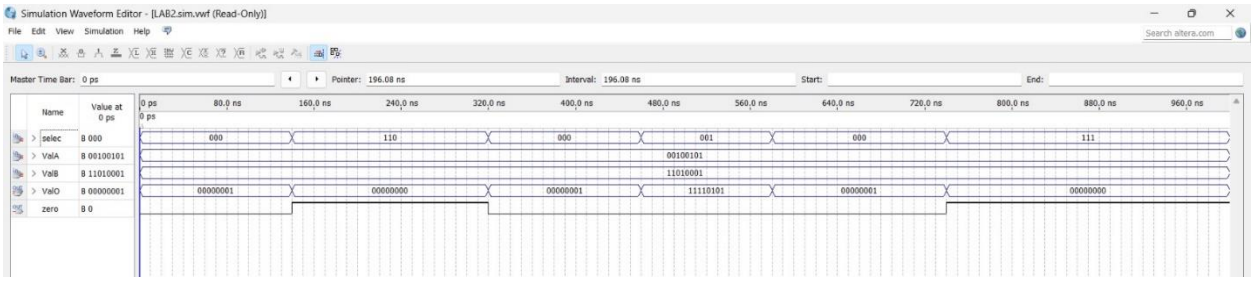
1.3 The ALU simulation

As we can notice the operation number presented with the values of AluOP in the figure below which is to connect to the ALU select. Comparing them, we can see that the resulting operation is indeed the same.



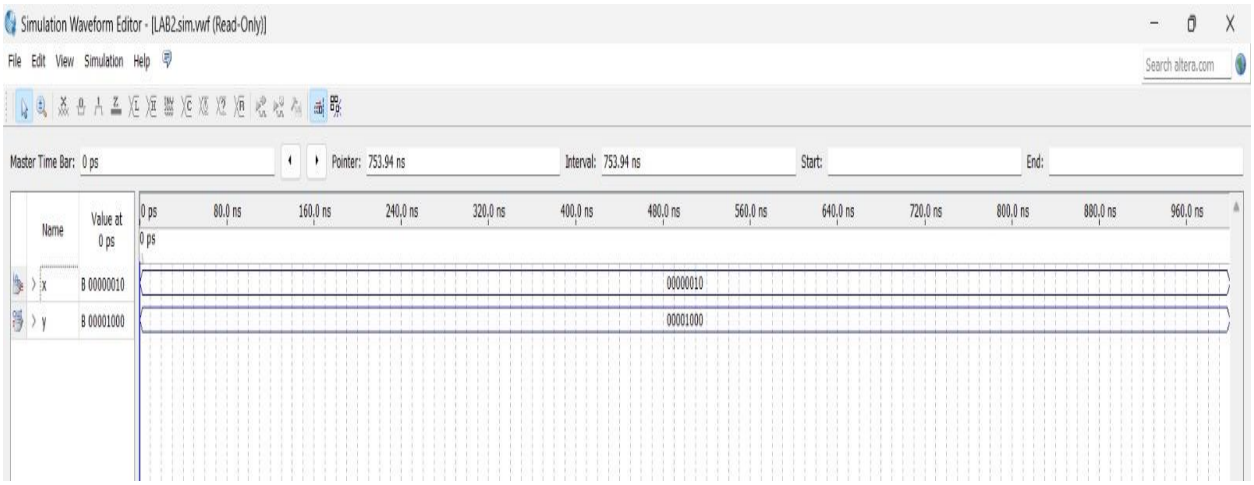
1.4 Small ALU simulation

Here we notice the valA and valB which are in the rt and rs registers. We put a selec value of 000 which gives us an OR operation here, which gives us the result on valO of 00000001, value different from 0 where the zero flag is 0.



1.5 Shift-Left2 simulation

We notice that the value of the output Y is the value of output X in shift which is therefore the expected result.

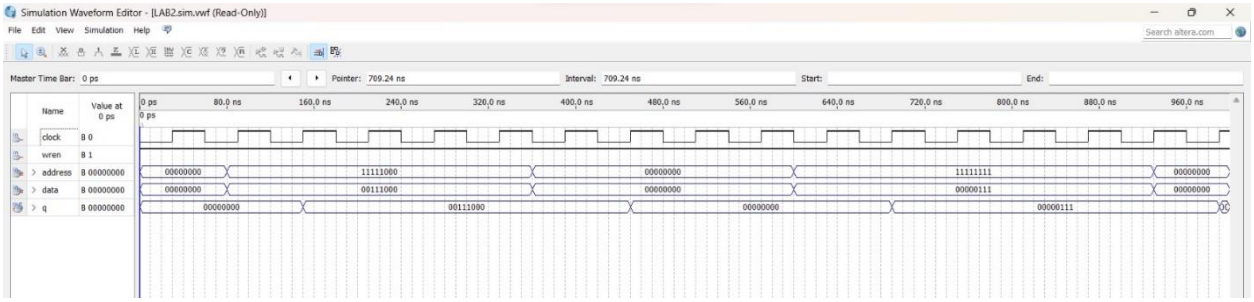


1.6 ROM instruction simulation

We notice that the memory instructions are not as expected because from address 04 to address 05 we notice that there is not one incrementing by 4 as it is supposed to appear.

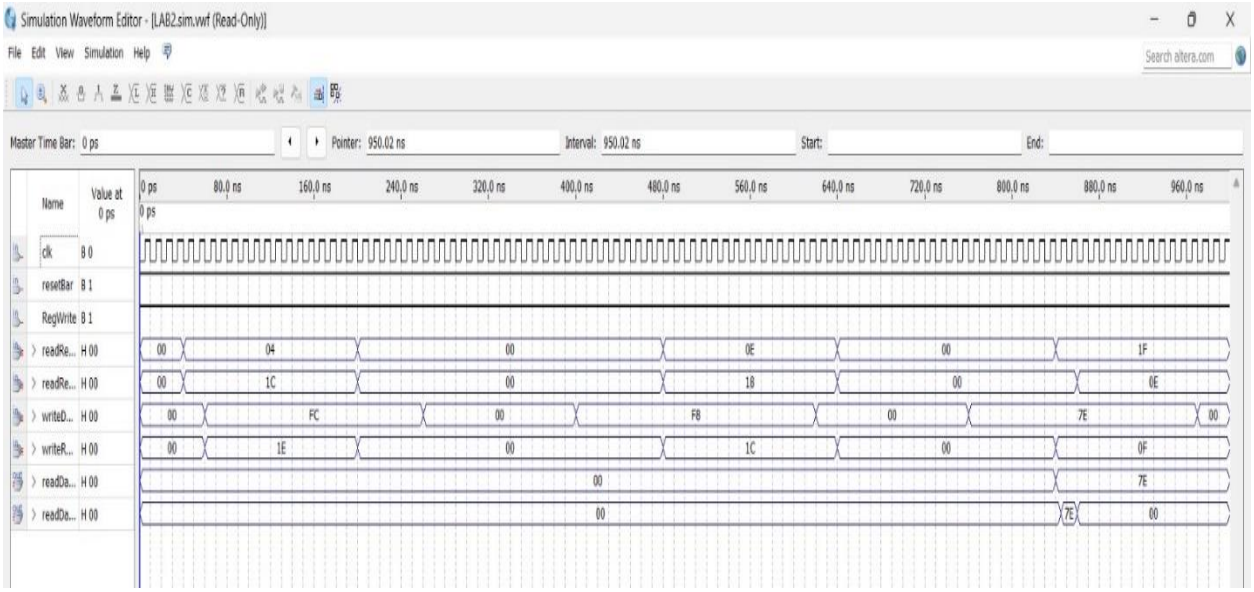
1.7 Data memory simulation(RAM)

We notice that our simulation is correct because if the entry written which plays the role of an enable in our case is then activate our output is none other than the data that was entered as input.



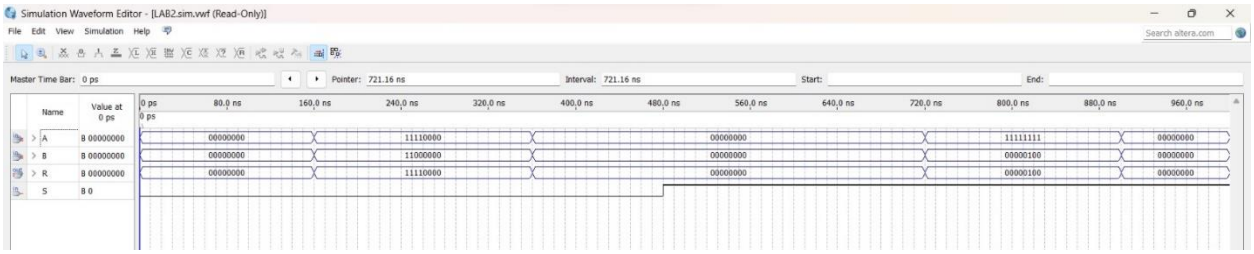
1.8 Memory register simulation

We can see that our simulation of the register entity is correct because what we wrote in the register with the entry (WRITE DATA) was read in the two-read data with the value of reg write and reset bar activate.



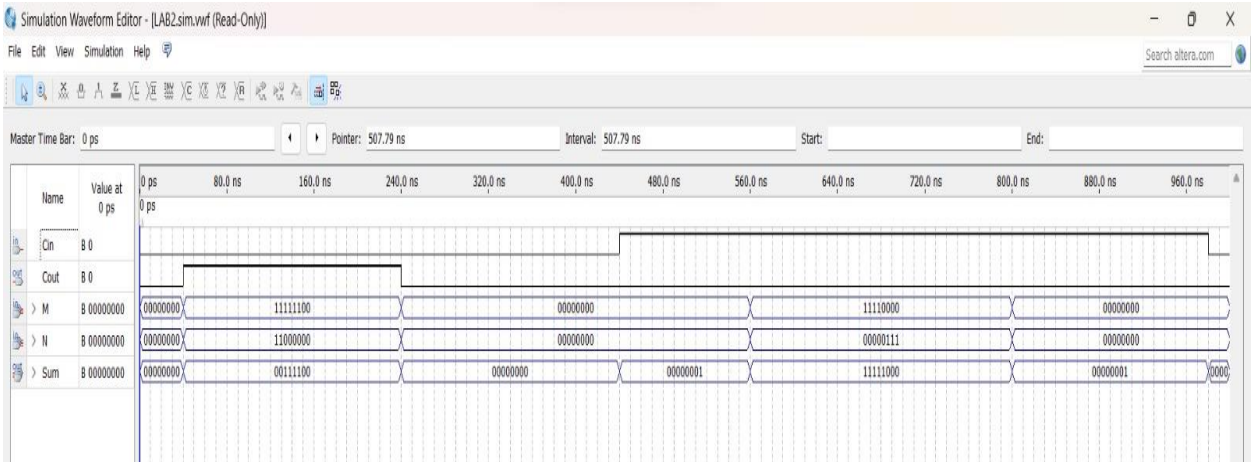
1.9 Multiplexer 2 to 1 with eight bits simulation

We notice that our results are correct because according to what we chose with our selector s in inputs A and B, we notice that the value exact was chosen and written to the R output.



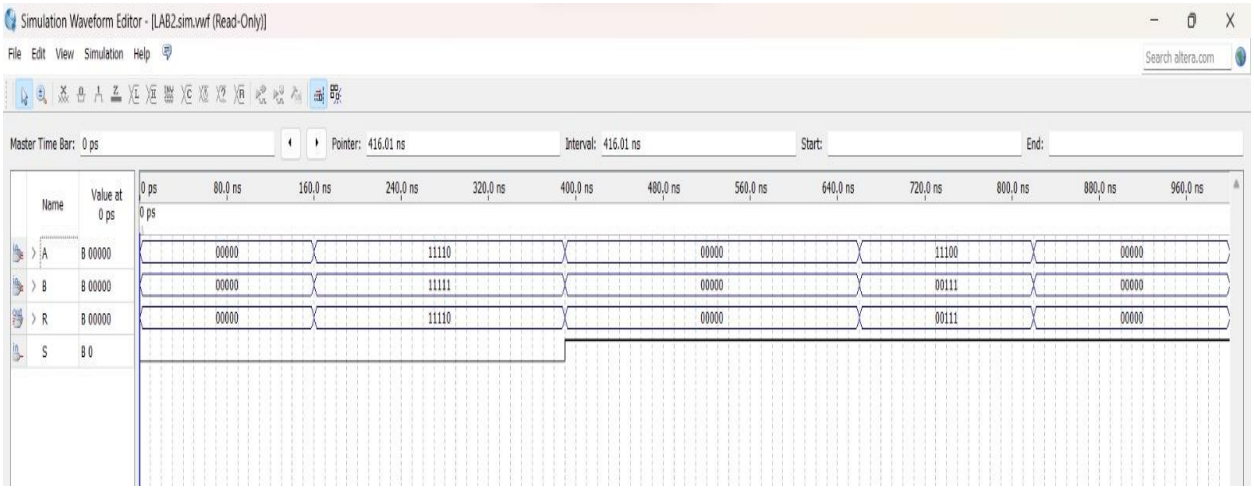
1.10 FullAdder-8bits

our results are correct because according to what we chose with our selector s in the M and N inputs, we notice that the valueexact was chosen and written to the sum output



1.11 Multiplexer 2 to 1 with 5 bits simulation

In the multiplexer 2x1 with 5 bits, we notice that our results are correct because according to what we chose with our selector s in inputs A and B we notice that the value exact was chosen and written to the R output.

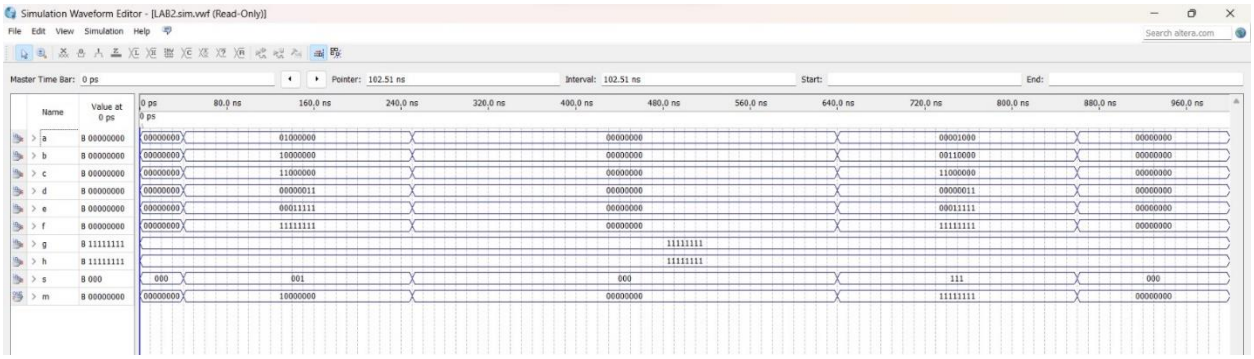


1.12 Eight-bit-Register

We notice that our simulation is correct because when enable is not activated, no value is chosen in our output but when it last is activated the value that we put in input is displayed in output. Enable must therefore be activated for the value of the input to pass to output.

1.13 Multiplexer 3 to 8 simulation

We notice that our results are correct because according to what we chose with our selector(s) in the entries (a,b,c,d,e,f) we notice that the exact value was chosen and written to the output (out).



2 Design verification.

Finding the Maximum clock frequency:

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

$+ ALU$ + 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

CPU Execution Time Calculation:

$$\text{CPU} = \frac{\text{CPI} * \text{Instruction time}}{\text{Clock rate}}$$

$$\text{Cpu} = \frac{12}{12.35 * 10\ 3} = 9.73 * 109\text{p}(-4)\ s$$

Finding worst case delay scenario that the ALU can have.

Worst delay = 31*2+3=65

Knowing that :Carry spread:=2, Sum =3

And so the Worst delay = 65*0.01=0.65ns

IVDiscussion

Each component of our single cycle processor RISC-Type 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 single cycle RISC-Type using structural VHDL coding implementation, and we got familiarized of designing and testing a single cycle RISC-Type processor on altera board. We were finally able to demonstrate and have a clear understanding of RISC processors and instructions set architecture which justifies the main objective of the lab.

VIReference

The Lectures notes seen in class about MIPS single and multiples 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.