PHASE 1&2 REPORT

24/4/2024

PREPARED BY :

Omar Mohamed 22P0197
Ahmed Wael 22P0221
Ezzeldin Ismail 22P0141
Mohamed Haitham 22P0156
Ahmed Mohamed 22P0167

PRESENTED TO :

Dr. Tamer Mostafa Abdelkader
Eng. Yasmine shaban

Table Of Contents

List of figures:

Abstract

This report discusses A design of MIPS processor using VHDL that illustrates a basic computer system by simulating the data and control paths.

After modifying the 32 bit full ALU, and a Register File the designed CPU for phase 1 should be able to perform certain instructions: R-type (AND, OR, ADD, SUB, SLT and NOR).

A control module, MIPS module, memory module, and datapath are all integrated into the MIPS CPU design to efficiently execute R, I-type (lw, sw, beq), and J instructions. The control module produces the RegWrite, MemRead, MemWrite, ALUSrc, Branch, and Jump control signals that are necessary for the datapath. The main datapath for executing instructions is formed by the registers, ALU, sign extension unit, branch logic, and multiplexers that make up the MIPS module. Both read and write operations for data and instructions are supported by the memory module. Under the direction of control signals from the data control unit, the datapath coordinates the phases of instruction fetch, decode, execute, and write-back. This abstract captures the interdependent parts and functions of the MIPS CPU, which are necessary to smoothly carry out various kinds of instructions.

Introduction

This project uses the hardware description language VHDL to design and develop a Simple MIPS CPU. The CPU consists of an 32-bit ALU that includes AND, OR, ADD, SUB, SLT and NOR functions and a Register File that consists of a 32 registers and so the register file selects which registers will be used to read data from and which will be used to write data into then these 2 components are connected together to develop a CPU that can perform R-type instructions.

One popular RISC (Reduced Instruction Set Computing) architecture that is well-known for its ease of use and effectiveness in processing a variety of instruction types is the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture. We describe in this study the comprehensive design and implementation of a MIPS CPU that can carry out J, R, and I-type (lw, sw, beq) instructions. The design includes vital parts such the memory module, control module, MIPS module, and datapath, all of which are critical to the CPU's functioning. This design's main goal is to produce a flexible and effective CPU that can process a wide variety of instructions that are frequently seen in MIPS assembler programs. We want to show how a MIPS CPU takes instructions, executes computations, and efficiently handles data transfers by comprehending the relationships among the control unit, datapath elements, and memory units.

Design Overview:

1) Register File: A module that simultaneously reads from two registers and writes to a third register is called a MIPS RegisterFile.

2) ALU: An ALU with capability for AND, OR, ADD, SUB, and NOR operations is called a modified 32-bit full ALU.

3) Datapath: Instruction execution is enabled by connecting the register file and ALU through the Simple MIPS CPU Datapath.

4) Modify the MIPS CPU as to be able to perform not only R instructions, but also I-type (lw, sw, beq) and J instruction by:

- Implementing the control module.
- Implementing the Mips module.
- Connecting the Mips module with instruction and data memory module together.

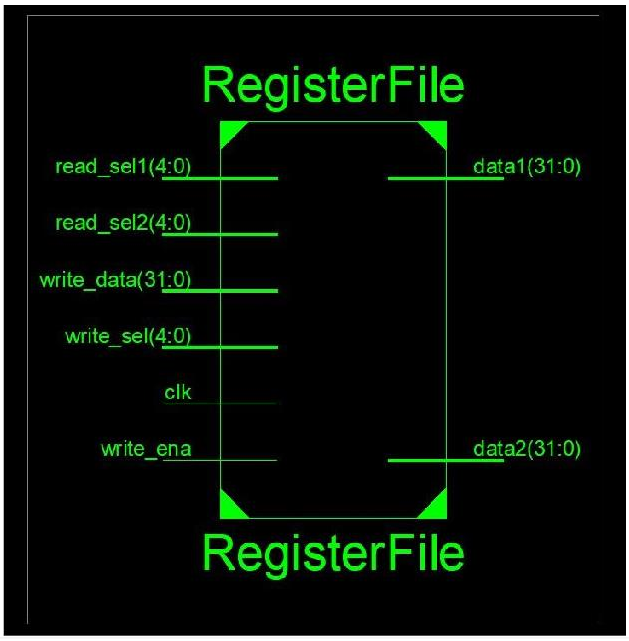
Methods:

MIPS Register File :

- **Inputs:**
  - **read_sel1, read_sel2: Selects the registers to read from.**
  - **write_sel: Selects the register to write into.**
  - **write_ena: Write enable signal.**
  - **clk: Clock signal.**
  - **write_data: Data to be written into the selected register.**
- **Outputs:**
  - **data1, data2: Data read from the selected registers.**

This circuit simply have 2 selectors that selects the registers that data will be taken from and then the data are taken from the registers to be the output and also it has a selector for the register that will be written in and the data that will be written in it and finally a write enable signal that enable writing when equal '1'.

Register File implementation:



[Figure 1]

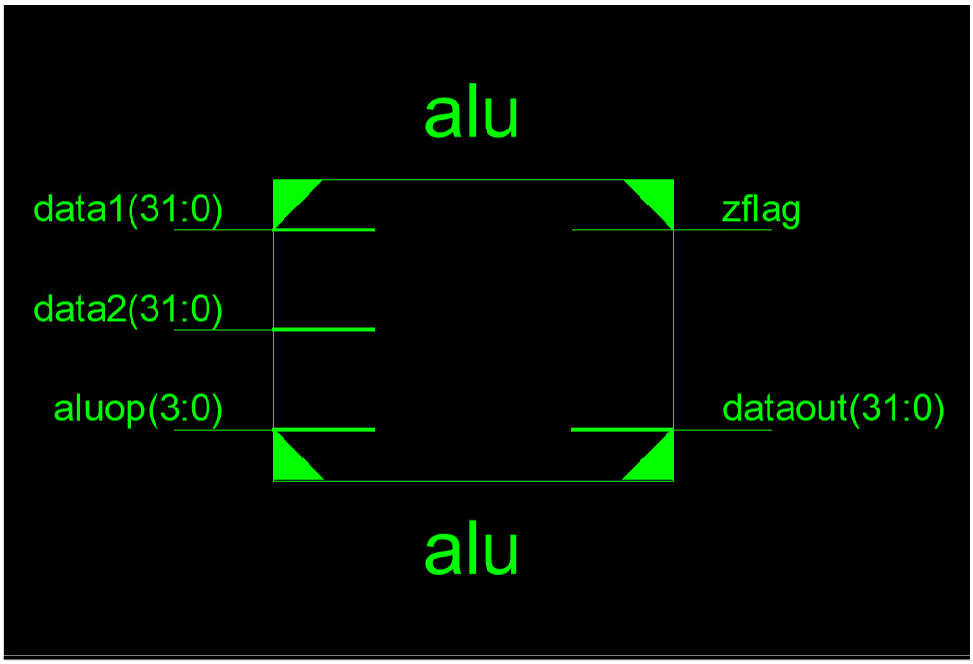
Modified 32-bit Full ALU :

- **Inputs:**
- **data1, data2: Data inputs for ALU operations.**
- **aluop: Operation code for ALU functionality.**
- **Outputs:**
- **dataout: Result of ALU operation.**
- **zflag: Zero flag indicating the result is zero.**

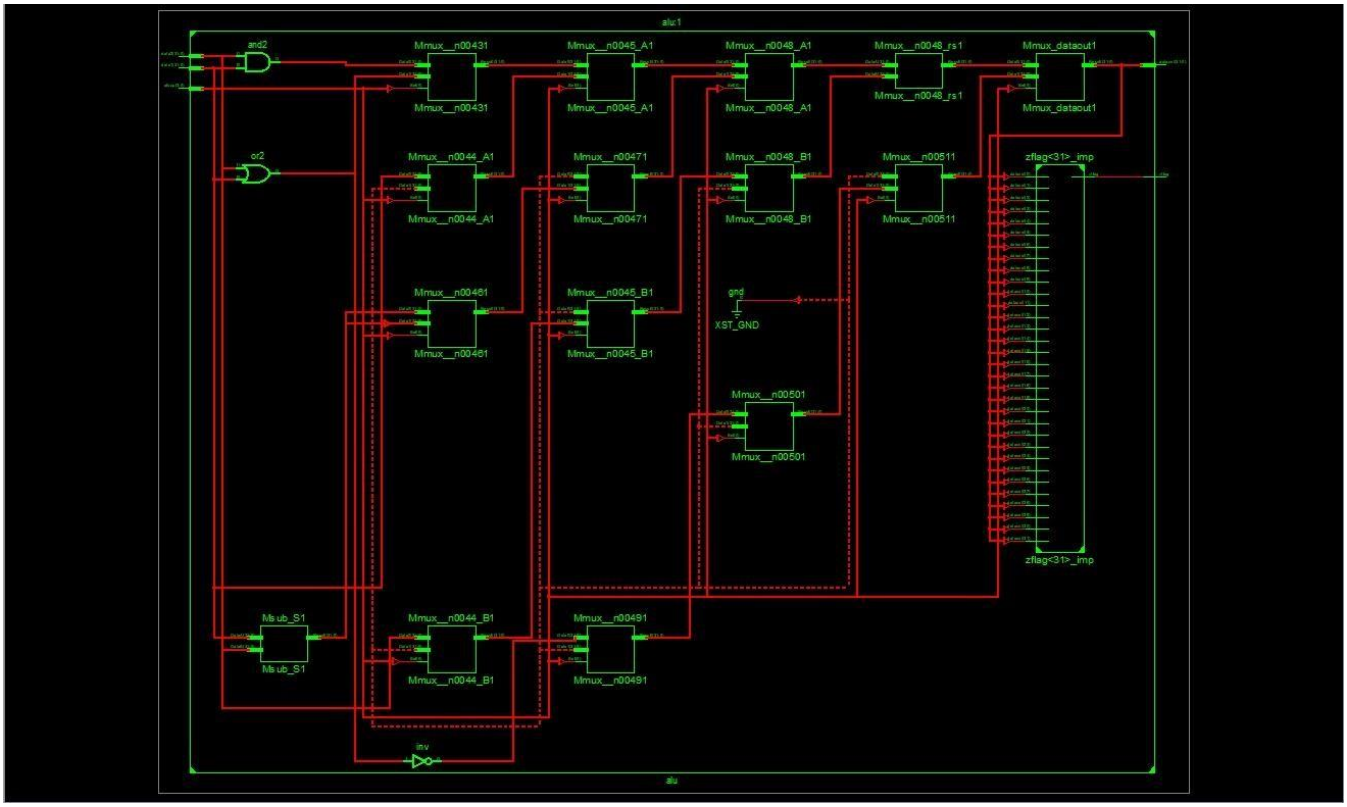
ALU functional specifications:

ALUOp	Function
0000	AND
0001	OR
0010	ADD
0110	SUB
1100	NOR
0111	SLT

Alu implementation:



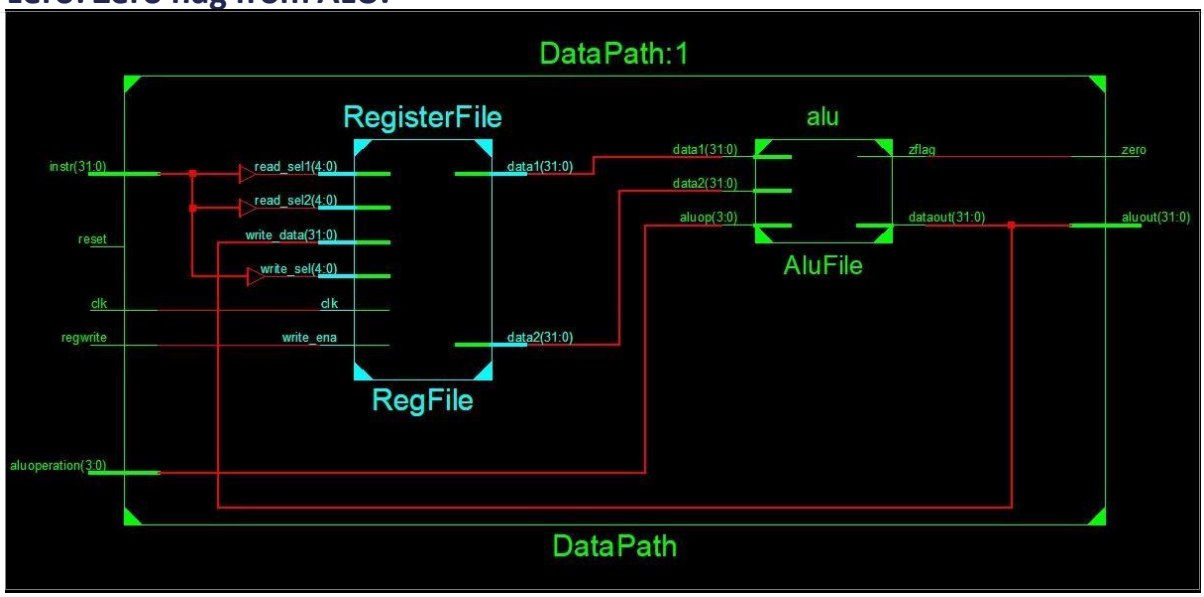
[Figure 2]


alu:1
and2
or2
Mmux__n00431
Mmux__n0045_A1
Mmux__n0048_A1
Mmux__n0048_rs1
Mmux_dataout1
Mmux__n0044_A1
Mmux__n00471
Mmux__n0048_B1
Mmux__n00511
zflag<31>_imp
Mmux__n00461
Mmux__n0045_B1
gnd
XST_GND
Mmux__n00501
Ms ub_S1
Mmux__n0044_B1
Mmux__n00491
inv
alu

[Figure 3]

MIPS CPU Datapath :

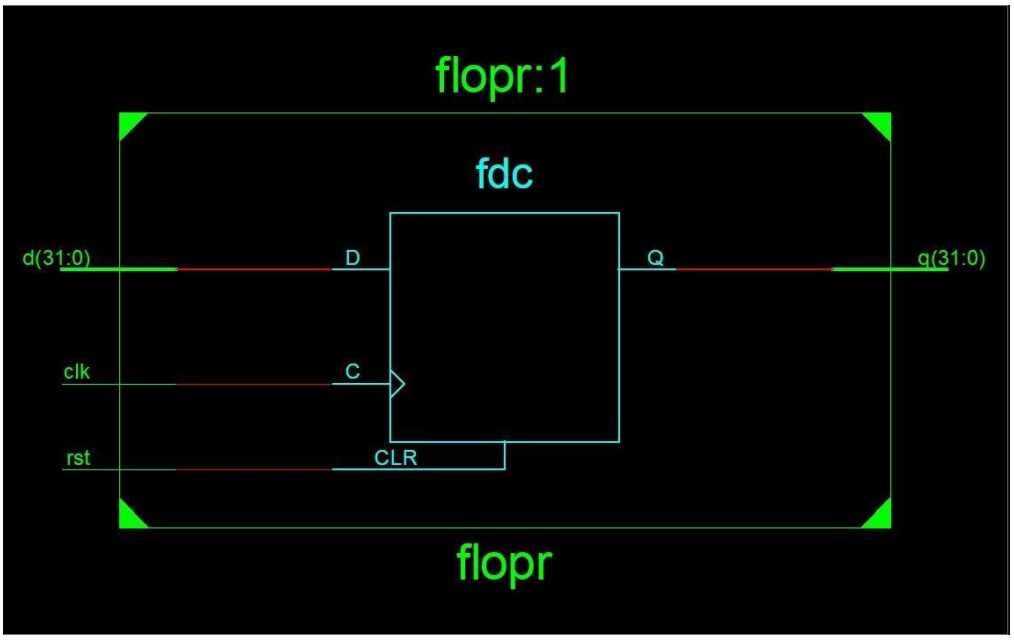
- **Inputs:**
- **clk, reset: Clock and reset signals.**
- **instr: Instruction input for register file and ALU.**
- **aluoperation: ALU operation code.**
- **regwrite: Register write control signal.**
- **Outputs:**
- **aluout: Result of ALU operation.**
- **zero: Zero flag from ALU.**



[Figure 4]

Flop Register :

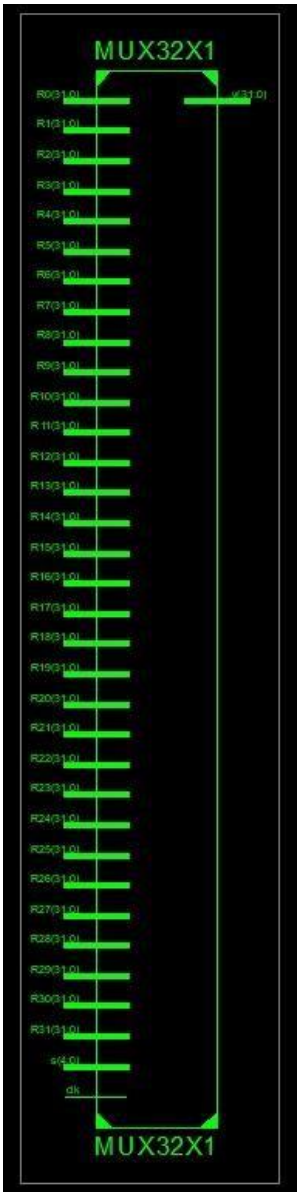
In this code we start by introducing our inputs and outputs. We define n as generic natural number of 32 bits, then add our inputs ‘clk’ , ‘rst’, ‘d’ , and output ‘q’. Then define ‘clk’ and ‘rst’ inside process to be asynchronous inputs. Next, we check for the value of ‘rst’, if equal ‘1’ output is ‘0’ regardless of value of other inputs, otherwise, value of output equals input at every new clock cycle.



[Figure 5]

32-1 MUXs:

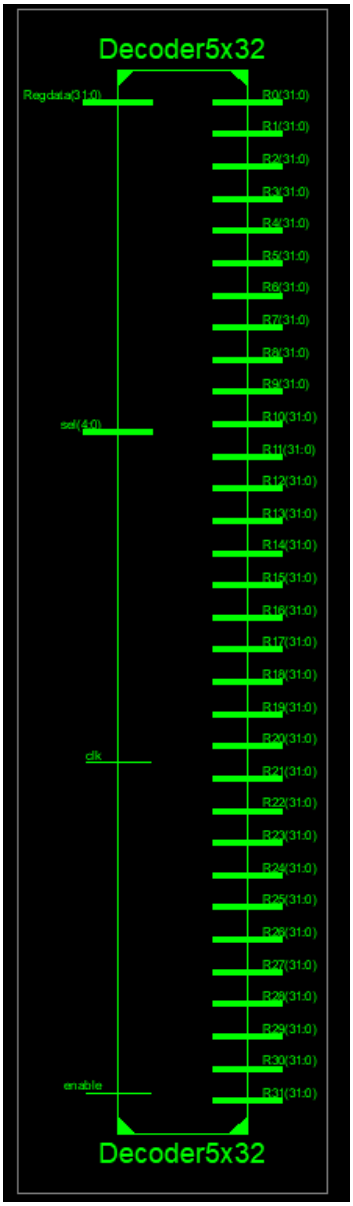
A multiplexer is a combinational circuit that has many data inputs and a single output, depending on control or select inputs. Multiplexers are also known as universal logic circuits. They are mainly used to increase the amount of data that can be sent over a network within a certain amount of time and bandwidth.



[Figure 6]

5X32 Decoder:

The decoder circuit is a circuit that have 3 inputs without the clock which are a selector which selects the register in which data will be written into and also the data that will be written and the data is written in the register that is selected by the selector and for the 3rd input its and enable signal that allows writing in the register if its ‘1’ else if its ‘0’no data will be written.



[Figure 7]

Control module :

There are three primary parts to the design:

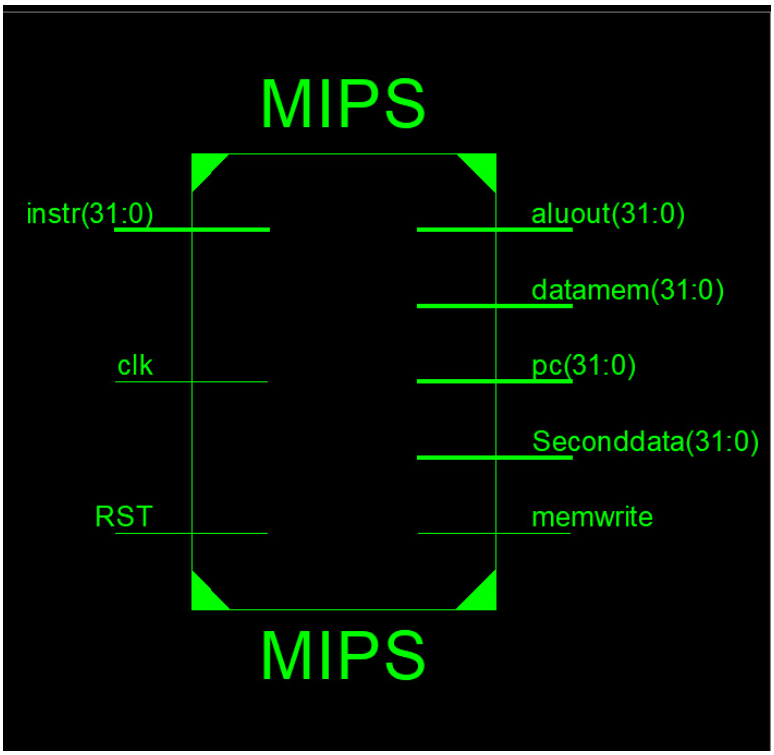
The opcode and funct fields of the instruction serve as the basis for the control signals that are generated by the control module. It chooses the action that the CPU should do in response to each kind of instruction. Among the control signals are Branch, Jump, ALUSrc, RegWrite, MemRead, and MemWrite.

RegWrite: Allows writing to R-type and lw instruction registers.
MemRead: Allows lw instructions to read data from memory.
MemWrite.: Enables writing data to memory for sw instructions.

ALUSrc: Determines the instruction type-dependent second ALU operand.
Branch: Manages instructions for branches such as beq.
Jump: Manages jump commands, such as j.

MIPS Module:

The datapath elements needed for instruction execution are included in the MIPS module. It has branch logic to handle branching, registers, an ALU (Arithmetic Logic Unit), a sign extension unit for I-type instructions, and multiplexers to choose data sources based on control signals.



Datapath components:

Registers: During the execution of an instruction, they store data and intermediate outcomes.

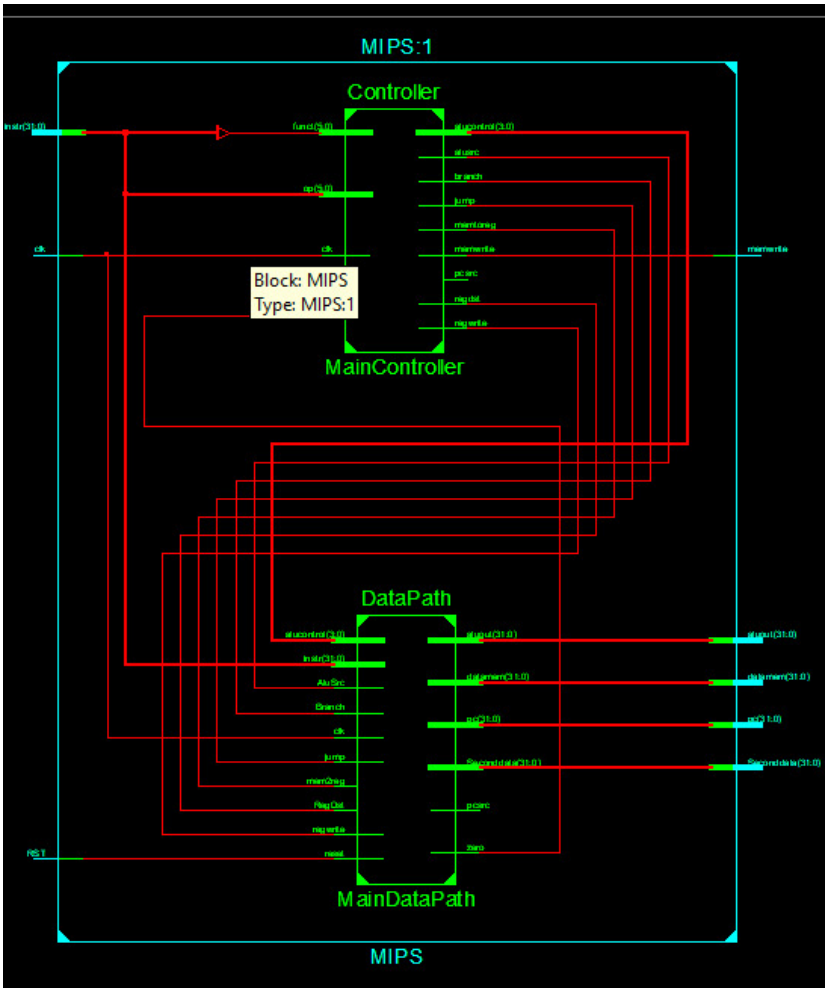
ALU: Works with data operands to execute arithmetic and logic operations.
Sign Extension Unit: Increases I-type instruction instantaneous values to 32 bits.
Branch Logic: Based on control signals and ALU results, branches are determined.

Multiplexers (Mux): Selects inputs to ALU or registers based on control signals.

Datapath Operation:

Instruction Fetch: Using the program counter (PC), retrieves the instruction from memory.
Instruction Decode: Produces control signals by decoding the instruction.
Execute: Depending on the kind of instruction, executes memory accesses or ALU operations.

Write Back: In response to control signals, writes the results back to memory or registers.
Branch and Jump: Utilizing jump or branch commands, this feature updates the computer.



Memory Module:

There are distinct instruction and data memories in the memory module. It is capable of reading and writing data and instructions, respectively. In order to retrieve instructions and access data while they are being executed, the memory module is connected to the MIPS module.

Program instructions loaded from the program code are stored in the instruction memory.
Data Memory: Holds information that the software accesses or modifies while it is running.

Datapath overview :

PC advances in order to get the subsequent instruction.
Based on the PC value, the instruction is read by the instruction memory.

Guidelines Decode: The opcode and funct fields are used to generate control signals.
Register numbers from the instruction are used to read registers.
Execute: An ALU can compute memory addresses or carry out arithmetic and logic operations.
MemRead/MemWrite signals control what data memory can read and write.
Write Back: In response to control signals, the outcomes are written back to registers or data memory.

Parts of the Datapath Registrations:

Goal: Temporarily store data while an instruction is being executed.

Functionality: Based on control signals, read data from registers and write data to registers.
The goal of an arithmetic and logic unit, or ALU, is to process data operands.
Functionality: Takes inputs from immediate values or registers, executes operations (such add, subtract, and logic operations), and outputs the results.
Unit of Sign Extension:
Goal: Increase I-type instruction immediate values to 32 bits.
Functionality: Sign extension (copying the most significant bit to fill the remaining bits) is used to expand the 16-bit immediate values from instructions to 32 bits.

Branch Logic: Goal: Ascertain branch outcomes by using control signals and ALU outputs.
Functionality: Produces signals to regulate PC (Program Counter) updates by comparing ALU results for branch instructions (such as beq).

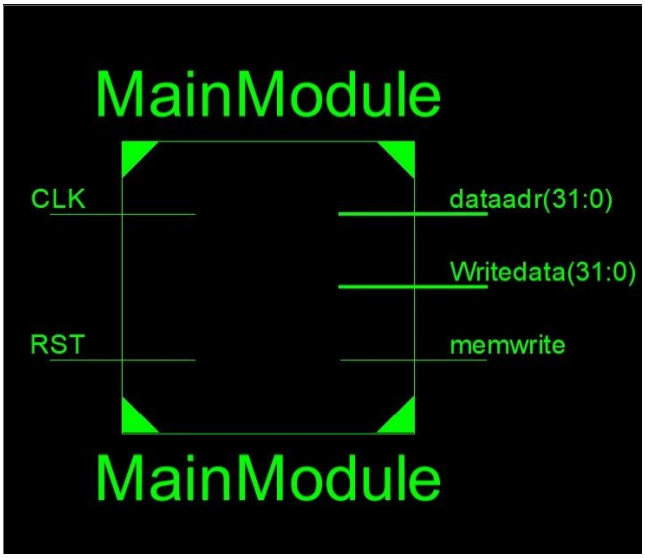
The function of multiplexers, or muxes, is to choose inputs for ALUs or registers according to control signals.
Functionality: Based on control signals from the control unit, selects various data sources (such as ALU results, register values, and immediate values).

Unit of Data Control (Control Module)

The control signals that direct the datapath components' actions during instruction execution are produced by the data control unit, sometimes referred to as the control module.

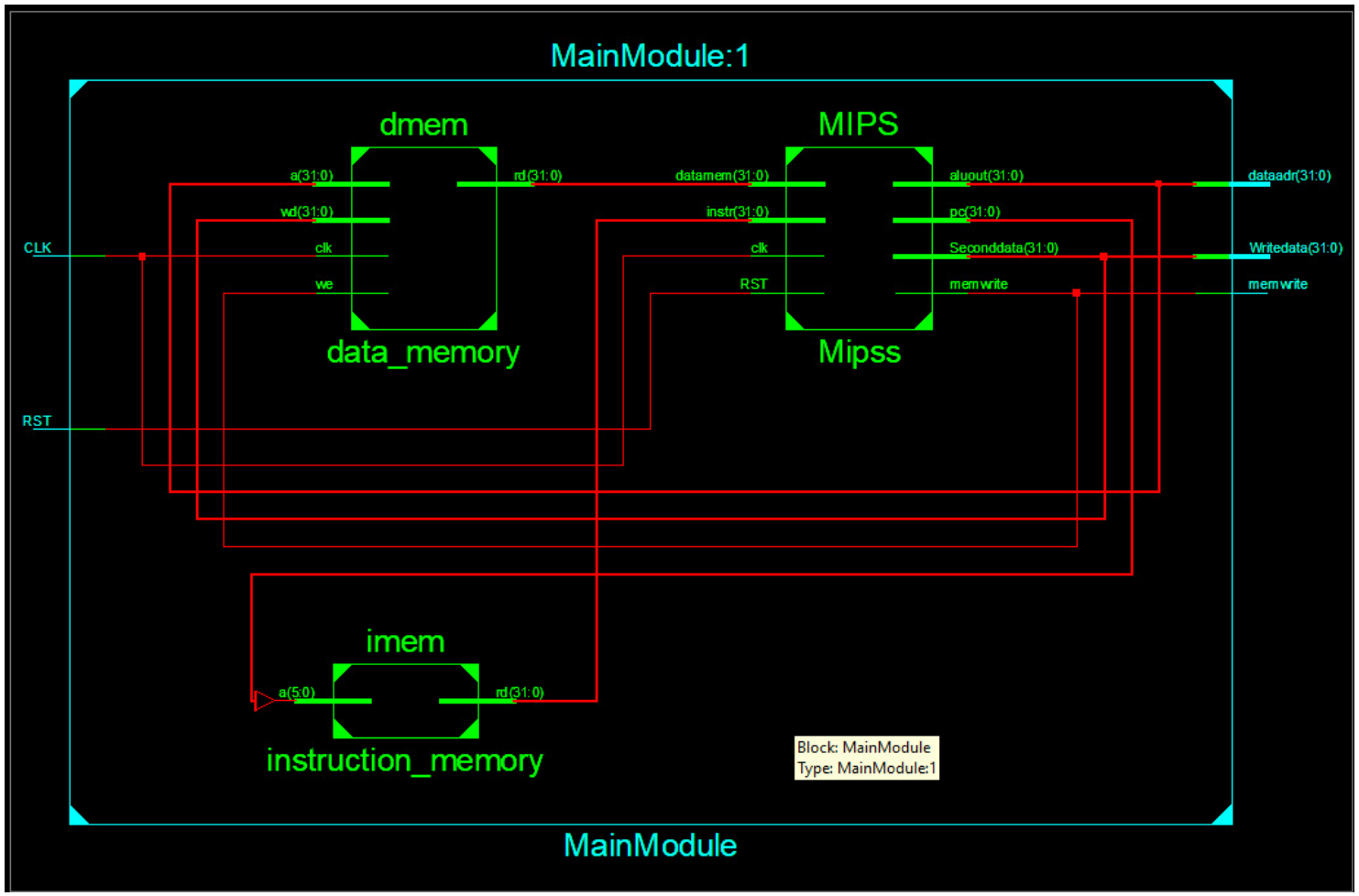
Control Signals: An Overview of Their Uses
Write-to-register functionality is provided by RegWrite.
Use: Enabled to write results to registers using lw (load word) instructions and R-type instructions.
MemRead: Function: Makes it possible to read data out of memory.
Use: Enabled to retrieve data from RAM using lw (load word) commands.
MemWrite: Function: Permits the writing of data to RAM.
Use: Enabled to write data to memory using sw (store word) instructions.
The function of ALUSrc is to select the second ALU operand.
Use: Depending on the kind of instruction, selects either immediate values or register values for ALU operations.

branch:
Function: Manages instructions for branches.
Use: Produces signals (e.g., equality check for beq instructions) to identify branch outcomes.
Jump: Function: Manages the instructions for jumping.
Use: Establishes the jump instruction's target address (e.g., j).



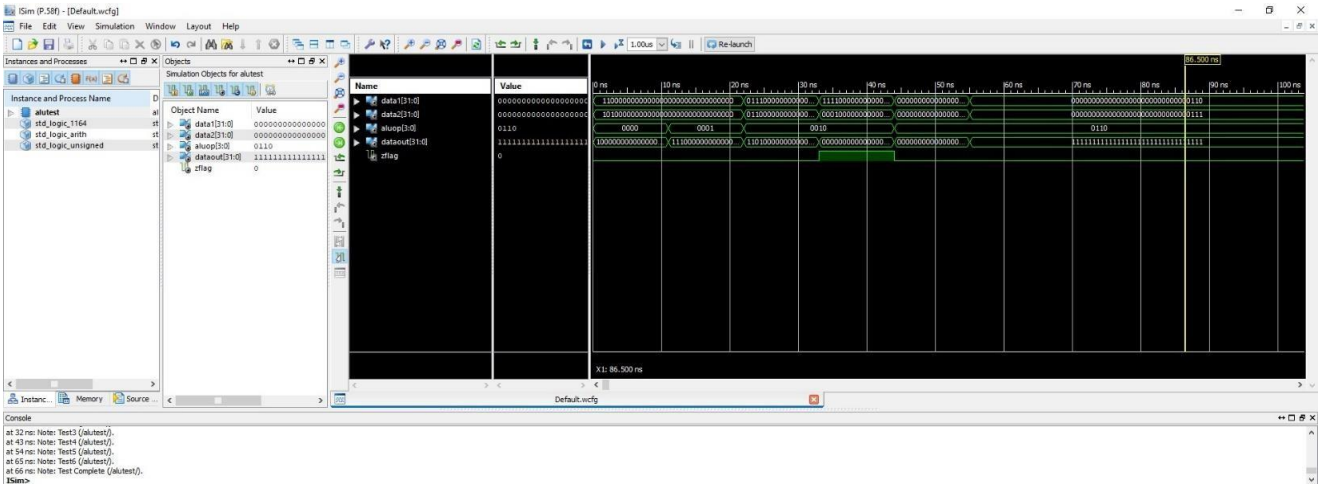
Interaction between Datapath and Control Unit:

By producing control signals in response to an instruction being executed, the control unit coordinates the datapath's activities and data flow. It guarantees that the proper parts are turned on or off at every point throughout the execution of an instruction, resulting in accurate and productive CPU operation.

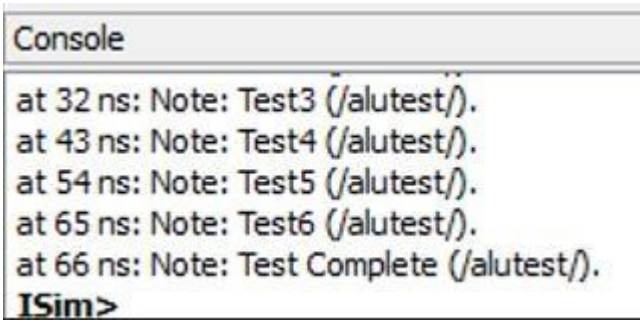


Results:

Alu test:

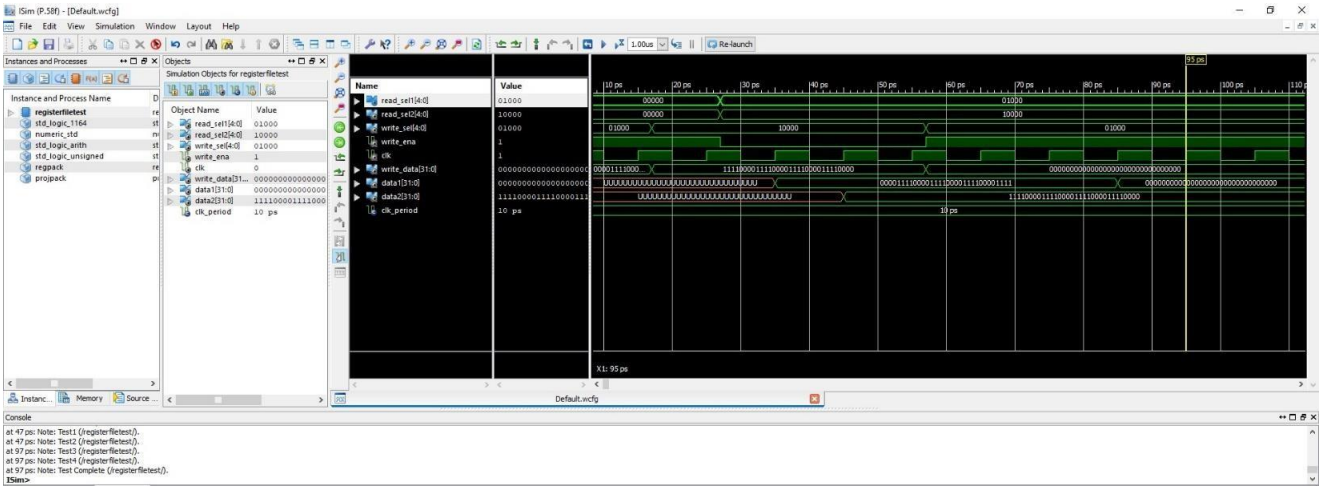


[Figure 8]

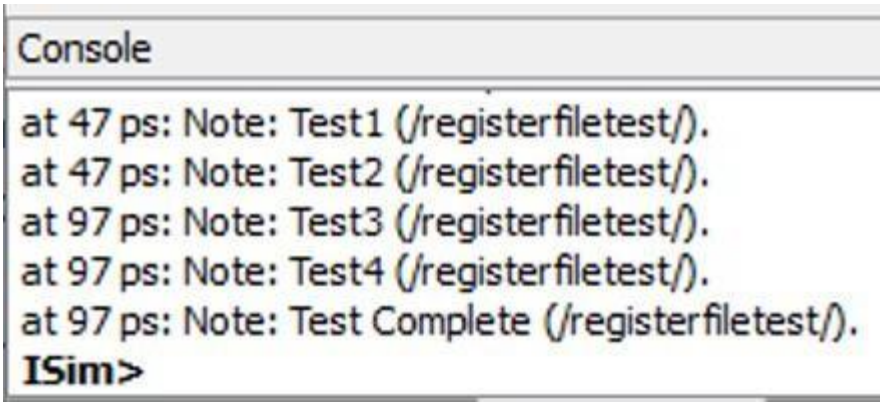


[Figure 9]

RegisterFile test:



[Figure 10]



[Figure 11]

Conclusion :

The R-type instructions indicated in the project requirements are successfully executed by the installed MIPS CPU. It illustrates register operations and ALU calculations, two fundamental MIPS architecture functions.

In summary, the project succeeded in creating and implementing a basic MIPS CPU in VHDL. Potential future work could include performance-enhancing design changes or an expansion of the instruction set.

The control module, MIPS module, memory module, and datapath of the planned MIPS CPU offer a complete framework for effectively executing R, I-type, and J instructions. Future iterations may support more instruction types or optimizations with ease thanks to the modular design's easy extension and customization capabilities.

References:

[1] Patterson and Hennessy's Computer Organization and Design, 5th Ed.

[2] FCIS Ainshams University Spring2022 labs

Team Contribution:

Datapath	Ahmed Wael & Omar Mohamed Mostafa
Register file	Ahmed Wael & Omar Mohamed Mostafa
Alu	Ahmed Wael & Omar Mohamed Mostafa
Mux 32×1	Mohamed Haitham
Decoder 5×32	Ahmed Mohamed Talaat
Flop register	Ezzeldin Ismail
Project package	Ezzeldin Ismail
Register file test	Mohamed Haitham
Alu test	Ahmed Mohamed Talaat
Report	Ahmed Mohamed Talaat & Ezzeldin Ismail & Mohamed Haitham
Error debugging	Ahmed Mohamed Talaat & Mohamed Haitham
Research	Ezzeldin Ismail