GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

COEN 6741: PROJECT REPORT

On

MINI MIPS - A SIMPLE PIPELINED PROCESSOR

Submitted By:

Group 12

Mazdad Anklesaria [40102785]

Anar Pandya [40085636]

Rokibul Hasan Bhuiyan [40091513]

Mohammad Shafayet Islam [40038419]

Date of Submission: April 15th, 2019

TABLE OF CONTENTS

INTRODUCTION 5
Mini-MIPS Processor 5
Pipelined Mini-MIPS Processor 6
Pipelined Stages 6
Hazards in Pipeline 7
Types of Hazards 7
Structural Hazard 7
Data Hazard 7
Control Hazard 8
Exception 8
INSTRUCTION SET ARCHITECTURE 8
ISA Design Format 9
ISA Encoding Format 9
DATAPATH DESIGN 11
Datapath in Five Stages 11
Instruction Fetch Stage (IF) 11
Instruction Decode Stage (ID) 11
Execute stage (EX) 12
Memory Access Stage (MEM) 12
Write Back Stage (WB) 12
Datapath for Instruction Types 12
Datapath for R-type instructions 13
Datapath For Immediate Arithmetic and Logical Instructions 13
Datapath for the Load and Store Instruction 13
Datapath For Branch Instruction 14
Datapath For Jump Instruction 14
Datapath as a Whole 14
Control Unit 15
HAZARD HANDLING 16
Structural Hazards 16
Data Hazard 16
Control Hazard 16
SIMULATION TEST ANALYSIS 17
Instruction Memory 17
Register Memory 17
CONCLUSION 20
WORKS CITED 21

LIST OF TABLES

Table 1: Pipelined MIPS stages 6
Table 2: ISA Design 9
Table 3: R-Type Instruction Encoding Format 10
Table 4: I-Type Instruction Encoding Format 10
Table 5: Legends of Encoding Format 10
Table 6: Datapath for R-Type Instructions 13
Table 7: Control Unit for each instructions 15

LIST OF FIGURES

Figure 1: MIPS Architecture in brief 5
Figure 2: MIPS Architecture w/ Pipelined 6
Figure 3: Datapath of Mini-Mips 14
Figure 4:After 17
Figure 5: before 17
Figure 6: Simulation Output 18
Figure 7: IF stage 18
Figure 8: ID Stage 19
Figure 9: EX Stage 19
Figure 10: WB Stage 19

ABSTRACT

The principal objective of this report is to create a simple pipelined Mini MIPS processor to implement several instructions. Following the fact that the latest MIPS- processor present today is high performance, low power consumption and compact size, the MIPS processors are highly employed in today's mercantile products like game consoles, network devices like routers and in many embedded devices.

In this report the focus has been emphasized towards implementing few provided instructions using the pipelined version of MIPS architecture and eliminating the hazards, using VHDL language in Modelsim software. Few test case scenarios along with simulation results and analysis has also been provided in the report.

INTRODUCTION

MINI-MIPS PROCESSOR

MIPS is an abbreviation to Multiple Instructions Per Second. Mini – MIPS, a subset of full-fledged version of MIPS, uses the (RISC) Reduced Instruction Set Architecture and pipeline was implemented to increase the throughput performance. Pipelining is a powerful way to improve the throughput of a digital system. It also introduces some hazards therefore throughput will not be as high as expected but due to its most important advantage of high speed with little cost, all the modern microprocessors are pipelined. The buffer registers are placed at every stage. In a pipelined processor a single cycle processor is divided into five stages and multiple instructions can be executed simultaneously in each stage. In these processors clock frequency is almost five times faster. Therefore, latency is almost five times better.

The key concepts of the original MIPS architecture are:

- ⇒ five-stage execution pipeline: fetch, decode, execute, memory-access, write-result
- ⇒ regular instruction set, all instructions are 32-bit
- ⇒ three-operand arithmetical and logical instructions
- ⇒ 32 general-purpose registers of 32-bits each
- ⇒ no status register or instruction side-effects
- ⇒ no complex instructions (like stack management, string operations, etc.
- ⇒ optional coprocessors for system management and floating-point
- ⇒ only the load and store instruction access memory

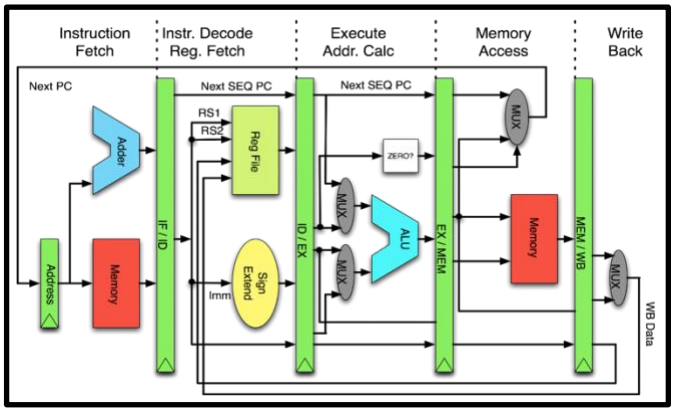

Figure 1: MIPS Architecture in brief

As per the project requirements, this report consists of all the mechanisms of Mini–Mips Architecture taking into account the fact of extending the non-pipelined MIPS processor to a pipelined processor which is hazard free and be capable of executing some instructions. (MIPS Architecture Overview)

PIPELINED MINI-MIPS PROCESSOR

Pipelining enables the MIPS to perform multiple functions and thus making the overall output faster compared to unpipelined processors. Using the idea of 5 stages of MIPS, instructions can be issue in every clock cycle. In this way various instructions can exist in the pipeline at different individual stages. This technique greatly reduces the output time of the processor and boosts the performance.

PIPELINED STAGES

Pipelined MIPS processor goes through five stages as mentioned below.

Table 1: Pipelined MIPS stages

Qty.	Stages	Context
1	IF	Instruction fetch from memory
2	ID	Instruction decode and register read
3	EX	Execute operation
4	MEM	Access Memory Operands
5	WB	Write Back the result to the register

Referred below is the architectural diagram of a pipelined MIPS. (Strukov, Pipelined MIPS Processor)

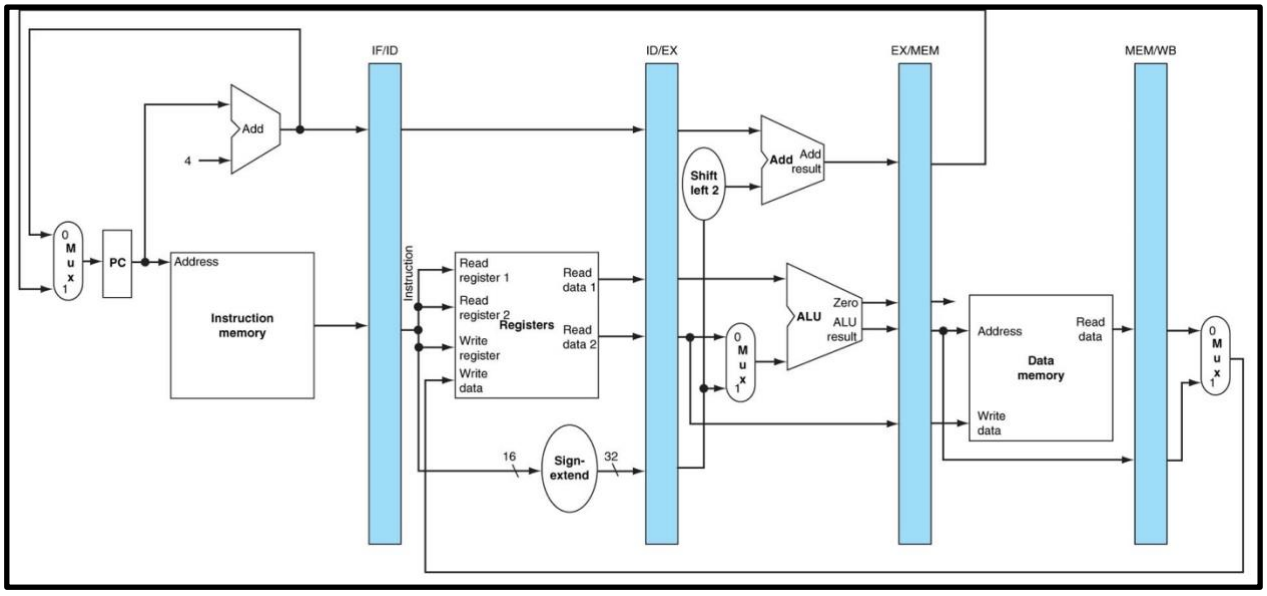

Figure 2: MIPS Architecture w/ Pipelined

HAZARDS IN PIPELINE

Pipelining in processors come with its own challenges known as hazards. A hazard occurs when an instruction has to stall before continuing execution due to the behaviour of other instructions in the pipeline and therefore could result in incorrect execution. Performance gained through pipelining is reduced when stalls are introduced in it.

TYPES OF HAZARDS

There are 3 standard types of hazards that are often to tackle in pipelined MIPS processors.

STRUCTURAL HAZARD

Structural hazards occur when multiple instructions try to use the same hardware unit in the same clock cycle. This often occurs when the pipelining is not done properly in the hardware thereby resulting in the consecutive use of resources by instructions requiring more than one clock cycle. This type of hazards could also occur if different instructions try to access the same resource in the same clock cycle, for example two instructions attempting to read data from the same register in the same cycle.

Structural hazards can be avoided by adding more hardware which could also lead to increased cost and unit latency.

DATA HAZARD

Data hazards occur when the data needed for the completion of an instruction is dependent on the data produced by another instruction in the pipeline and as a result, the instruction has to stall because all instructions must be completed in-order. This type of hazards could occur if a subsequent instruction tries to read from memory before a previous instruction writes to it, tries to write to memory before a previous instruction reads from it or tries to write to memory before a previous instruction writes to it.

By forwarding data, data hazards are usually avoided from an instruction to instructions needing it almost immediately as it is available. Another simple technique that can be used to avoid data hazards is writing instructions in the first half of a cycle and reading in the second half of the cycle.

CONTROL HAZARD

Control hazards are hazards that occur due to branching instructions. This usually happens when the processor has to determine whether or not to execute a subsequent instruction based on the outcome of a previous branching instruction. This outcome of a branch can't be determined until the second stage of the instruction execution known as instructions decode stage and by then, the subsequent instruction would have already been issued. Control hazards have big impact on performance as all programs are essentially loops and also stalling a fully pipelined machine, that is a machine with a cycle per instruction of one, on encountering of any control hazards would result in three-cycle stalls for each of all the loops in the program.

This type of hazard is usually avoided through branch prediction and a compiled time mechanism called loop unrolling.

EXCEPTION

An exception occurs when the execution flow is changed unexpectedly which could result from an attempt to execute an invalid operation. What makes exceptions different from control hazards resulting from branching is that exceptions are usually determined late in instruction execution with multiple other subsequent instructions already issued unlike branching hazards.

Exceptions are usually handled in the hardware by flushing all subsequent instructions, allowing preceding ones to complete and writing the exception information to special type of registers.

INSTRUCTION SET ARCHITECTURE

The first step in understanding computer architecture is learning about language of computer. In this project, a subset of the 32- bit MIPS architecture has been designed. The idea was to extend the non- pipelined MIPS processor to a pipelined processor which is hazard free and be capable of executing at least following 10 instructions provided by the instructor as per course project requirements. Those instructions follow the below mentioned formatting.

ISA DESIGN FORMAT

As per project requirements, below mentioned design formatting has been chosen in order to design Mini-Mips processor for provided instruction sets.

Table 2: ISA Design

Instruction Format	Context	Instruction Sets		
		Instruction	**Type**	**Description**
Arithmetic	Involves ALU computation. Takes two registers as operands or one register and one sign extended immediate value.	ADD	R	Logical Addition
		SUB	R	Logical Subtraction
		ADDI	I	Logical Addition Immediate
		SUBI	I	Logical Subtraction Immediate
		AND	R	Logical AND
		XOR	R	Logical XOR
Memory Access	Involves the movement of data to or from the memory. Takes register and offset values as operands.	LW	I	Load Word
		SW	I	Store Word
Control	Involves the movements of jump and branch of registers. Computed by adding a sign extended immediate value to the program counter.	JR	R	Jump to a register
		BEQ	I	Branch if equal to zero

ISA ENCODING FORMAT

In this report, Mini-Mips follow the below mentioned encoding format for both I-Type and R-Type instructions.

Table 3: R-Type Instruction Encoding Format

Type	Instruction	OpCode	Rs	Rt	Rd	Function
		3Bits	4Bits	4Bits	4Bits	17Bits
R-Type	AND	000	SSSS	TTTT	DDDD	00000000000100100
	SUB	000	SSSS	TTTT	DDDD	00000000000100010
	XOR	000	SSSS	TTTT	DDDD	00000000000100110
	ADD	000	SSSS	TTTT	DDDD	00000000000100000
	JR	000	SSSS	TTTT	DDDD	00000000000001000

Table 4: I-Type Instruction Encoding Format

Type	Instruction	OpCode	Rs	Rt	Function
		3Bits	4Bits	4Bits	21Bits
I-Type	ANDI	001	SSSS	TTTT	000000000000000101011
	SUBI	010	SSSS	TTTT	000000000000000101011
	LW	011	SSSS	TTTT	IMM
	SW	100	SSSS	TTTT	IMM
	BEQ	101	SSSS	TTTT	IMM

Table 5: Legends of Encoding Format

Qty.	Code	Context
1	OpCode	Operational Code
2	Rs	Identifier of First Source Register
3	Rt	Identifier of Second Target Register
4	Rd	Identifier of Third Destination Register
5	Function	Distinguishes among R-Type and I-Type instructions
6	IMM	Immediate Values

DATAPATH DESIGN

Datapath is a collection of functional units, such as arithmetic logic units or multipliers, which perform data processing operations, registers, and buses. Mini-MIPS is divided into five pipelined stages namely instruction fetch, instruction decode, execute, memory access and write back. The design has separate instruction memory unit and data memory unit. 32-bit datapath is used as it is a 32-bit architecture. Execution of instructions in datapath is controlled by the control unit that monitors the flow of instructions to prevent any potential hazards in the processor. (Khan)

DATAPATH IN FIVE STAGES

For five different stages, the following is a brief explanation about the datapath among them.

INSTRUCTION FETCH STAGE (IF)

- The IF stage calculates the next instruction address for program counter register (PC) by simply adding 4 bytes to the current PC. The output of the adder is connected to the one input of the Multiplexer.
- The Multiplexer's selection line determines whether the next program counter has to be PC + 4 or a branch/jump address, which is computed in ID stage.
- The second Input of the Mux is connected to an adder in the ID stage, which calculates the Target Address for Branch or Jump Instruction.
- PC signal/Mux selection for is coming from the ID stage. Mux Output is connected to the program counter register (PC).
- Read the instruction from the memory according to the address given by current PC.
- Update the IF/ID Pipeline register which includes Instruction Register and PCPlus4 buffer.

INSTRUCTION DECODE STAGE (ID)

- The decoding part of the instruction is completed in this stage. In addition, ID stage handles hazards that require stalls.
- Operations related to validating the branch condition are computed in this stage by making use of an adder.
- The source, target and the register operands are fetched from the R-file in this stage.
- The control signals required for the instruction in the subsequent stages are also generated in this stage.

- The sign extension unit extends the sign from 16 to 32 bit. The output of this unit is fed to a 2-bit shift register (required to compute branch address) and also to the ID/EX buffer, which will later be used by the ALU in the execute stage.
- The Comparator, which is a key unit in validating the branch condition, compares the register contents and if the outcome is zero, the output of the comparator is fed to the Hazard Detection unit as well as the mux in the IF stage.

EXECUTE STAGE (EX)

- The arithmetic and logical unit (ALU) performs arithmetic/logic operations.
- The effective address for Load or Store Instruction is calculated in this stage.
- Forwarding units are implemented in this stage, to handle data hazards.
- The input line of ALU comes from the sign extension unit in the instruction decode stage or from RT in the register file.

MEMORY ACCESS STAGE (MEM)

- Give various memory access control signals and finish the memory access task where contents are loaded from the memory to a register and stored in a memory location.
- As part of design implementation, the data memory was implemented as an array of 256 entries each with a width of 32 bit

WRITE BACK STAGE (WB)

- This stage of MIPS pipeline, the data from the buffer is written back into the register.
- The multiplexer in the write back stage is used to select the appropriate data whether it was from the ALU or the data loaded from the data memory.

DATAPATH FOR INSTRUCTION TYPES

Datapath is different for different types of instruction. Since all instructions require Program Counter (PC) to be incremented, each instruction will require different types of datapath. From different RTL, it can be achieved easily. Also, determining the control signals is also important along with checking the elements in the appropriate sequence for each instruction is also necessary. (MIPS ISA)

DATAPATH FOR R-TYPE INSTRUCTIONS

R[rd] ← R[rs] op R[rt] ➔ Example: add rd, rs, rt

Recalling from Table-3, it is understandable that the datapath contains the 32 bit register file and ALU capable of performing all the required arithmetic and logic functions. The register operands in the instruction field determine the registers which are read from and written to, and the function field of the instruction determine which particular ALU operation is executed. Referring to the Table-6, below are the control unit for ALU used in this project.

Table 6: Datapath for R-Type Instructions

ARITHMETIC UNIT	
Control Unit	**Function**
000	AND
001	XOR
010	ADD
110	SUBSTRACT
010	JR

DATAPATH FOR IMMEDIATE ARITHMETIC AND LOGICAL INSTRUCTIONS

Instruction: addi rt, rs, imm16

R[rt] ← R[rs] op imm16

The main difference between this and an r-type instruction is that here one operand is taken from the instruction, and sign extended (for signed data) or zero extended (for logical and unsigned operations.)

DATAPATH FOR THE LOAD AND STORE INSTRUCTION

Instruction: Lw rt, rs, imm16

Addr ← R[rs] + SignExt(imm16)

➔ Here memory address are being calculated

R[rt] ← Mem[Addr]

➔ Here, loading the data into register rt

Instruction: sw rt, rs, imm16

Addr ← R[rs] + SignExt(imm16)

➔ Calculate the memory address

Mem[Addr] ← R[rt]

➔ Store the data from register rt to memory

DATAPATH FOR BRANCH INSTRUCTION

Instruction: beq rt, rs, imm16

Cond ← R[rs] - R[rt] ➔ Calculate the branch condition

if (Cond eq 0) {

PC ← PC + 4 + (SignExt(imm16) × 4)

elsePC ← PC+4 } ➔ calculate the address of the next instruction

DATAPATH FOR JUMP INSTRUCTION

Instruction: jr r12

Here, the PC will be jumped to R12 and show the value of $r12.

DATAPATH AS A WHOLE

Combining the datapath elements is rather straightforward, since the datapath by adding functionality to accommodate the different instruction types has already been built. Below mentioned figure gives an overview of datapath for Mini-Mips as a whole.

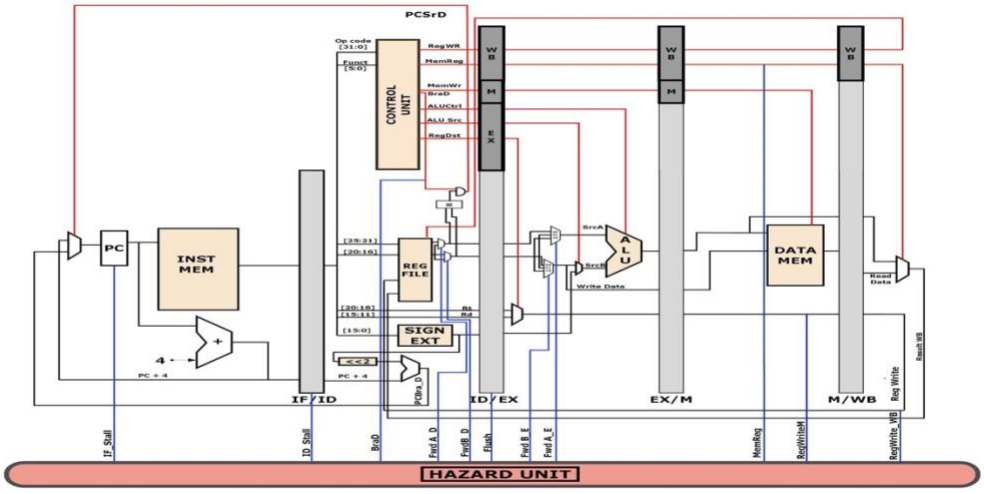

Figure 3: Datapath of Mini-Mips

CONTROL UNIT

The purpose of the control unit is to preserve and maintain the flow of instructions in the pipeline stages so that every stage knows what to do and what to operate on. The control part is implemented in the the control unit controls everything happening in a pipeline. It generates the control signals for all the components in the system. The control unit ensures that every stage in the pipeline knows what to do and what to operate on. It is implemented in the decode stage of the pipeline. The opcode and function field are extracted from the instruction set and are fed to the control unit which generates various control signals for the proper execution and completion of instruction. For each instruction control unit sends particular control signals. Control unit differentiate each instruction from its opcode. The following table shows the controls for each instruction.

Table 7: Control Unit for each instruction

Type	Inst	Branch	Aluop	Regdst	Alusrc	MemWrite	RegWrite	Mem to reg
R-Type	ADD	0	010	1	0	0	1	0
	SUB	0	010	1	0	0	1	0
	AND	0	010	1	0	0	1	0
	JR	0	110	1	0	0	1	0
	XOR	0	010	1	0	0	1	0
I-Type	ANDI	0	011	0	1	0	1	0
	SUBI	0	001	0	1	0	1	0
	LW	0	000	0	1	0	1	1
	SW	0	000	X	1	1	0	X
	BEQ	1	001	X	0	0	0	X

HAZARD HANDLING

MIPS is based on pipeline architecture that faces a number of pipeline hazards. These hazards can lead to faulty results based on wrong data operands or data to/ from memory. Below mentioned precautions are being taken into action in order to handle hazards in this project report.

STRUCTURAL HAZARDS

- When a resource is required by multiple instructions at a time, resource conflict occurs between those instructions.
- Using single memory doesn't solve structural hazard because it is not possible to access the memory twice during a clock cycle.
- To prevent this problem, two memories-Instruction memory and Data memory are used.
- The register file logic has been implemented in such a way that writes to register happen at the positive edge of clock and reads happen at the negative edge of the clock.

DATA HAZARD

- It is caused by the data dependency.
- The data hazard has been resolved using forwarding technique.
- Two forwarding units have been used to resolve the data hazards
- One forwarding unit has been used in the decode stage and other forwarding unit has been placed in the execution stage.

CONTROL HAZARD

- When a branch instruction or a jump instruction is fetched and is found to be already taken.
- This results in a control hazard.
- It generally happens in the execution stage, after the ALU computes the new address and writes it back to the Program Counter (PC).
- Because of this hazard we have to flush away all the instruction until the branch is known.
- The solution to this hazard is to decode the branch or jump instruction in the decode stage so that it does not cause much damage.

SIMULATION TEST ANALYSIS

After all the design units were implemented separately, they were tested using test benches. After their successful individual behavior, all the units were integrated and whole design were tested based on following process instructions.

INSTRUCTION MEMORY

Following instructions were coded into the instruction memory.

	Op	Rs	Rt	Rd	Funct	32-bit instruction	(HEX)	Instruction	
1	000	0010	0011	0001	00000000000100000	00000100011000100000000000100000	04620020	R2+R3=R1	add
2	000	0111	0110	0000000000000010000		00001110110000000000000000010000	4EC00032	R6=R7-0x32	subi
3	000	0111	0001	0110	00000000000100110	00001110001011000000000000100110	0E2C0026	R6=R7 Xor R1	xor
4	000	0110	1101	0110	00000000000100010	00001101101011000000000000100010	0DAC0022	R6=R6-13	sub
5	001	1001	1000	0000000000000101011		00110011000000000000000000101011	3300002b	R8=R9& ox2b	andi
6	000	1100	1101	0100	00000000000100100	00011001101010000000000000100100	19A80024	R4=R12&R13	and
7	100	0101	1100	0000000000000000000		10001011100000000000000000000000	6B800000	SW R12, 0(R5)	Store
8	011	1001	1100	0000000000000000000		01110011100000000000000000000000	99600000	LW R11, 0(R12)	Load
9	000	1101	00000000000000000001000			00011010000000000000000000001000	1A000008	JR R13	Jump
10	101	0011	1101	0000000000000000111		10100111101000000000000000000111	A7A00007	BEQ R13 R3 Imm7	Branch

REGISTER MEMORY

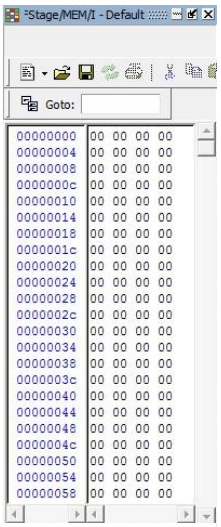

Figure 5: before

Results were compared after the simulation and following contents of register file were obtained that match our theoretically proposed values. Both the images are before and after images of memory values accordingly.

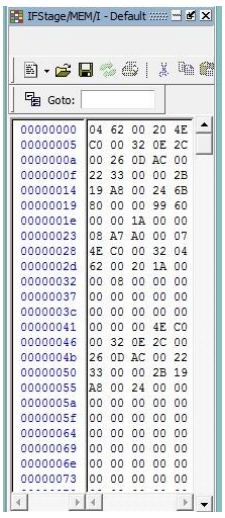

Figure 4:After

Figure 6: Simulation Output

SIMULATION RESULTS IN TIMING DIAGRAM

Simulation results of each stage bearing all the signals involved in corresponding stage are shown below:

IF STAGE

Figure 7: IF stage

ID STAGE

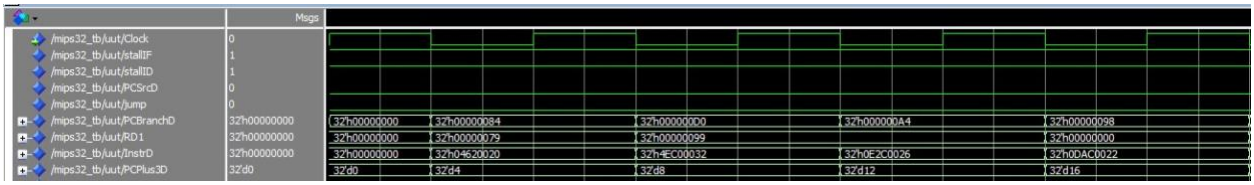

Figure 8: ID Stage

EX STAGE

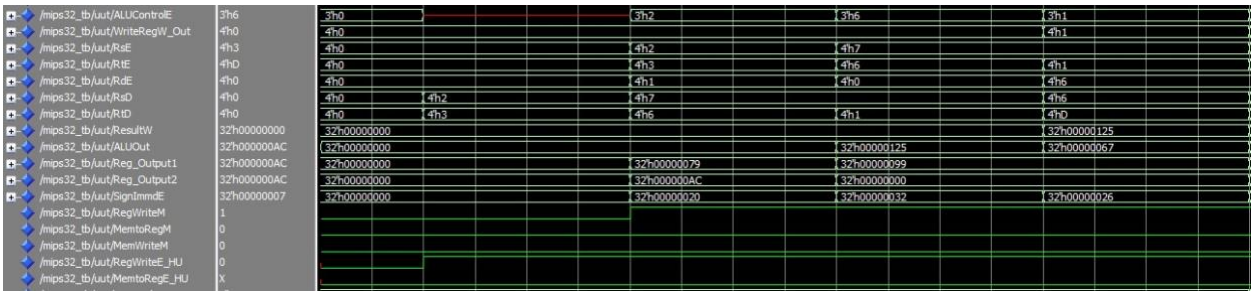

Figure 9: EX Stage

WB STAGE

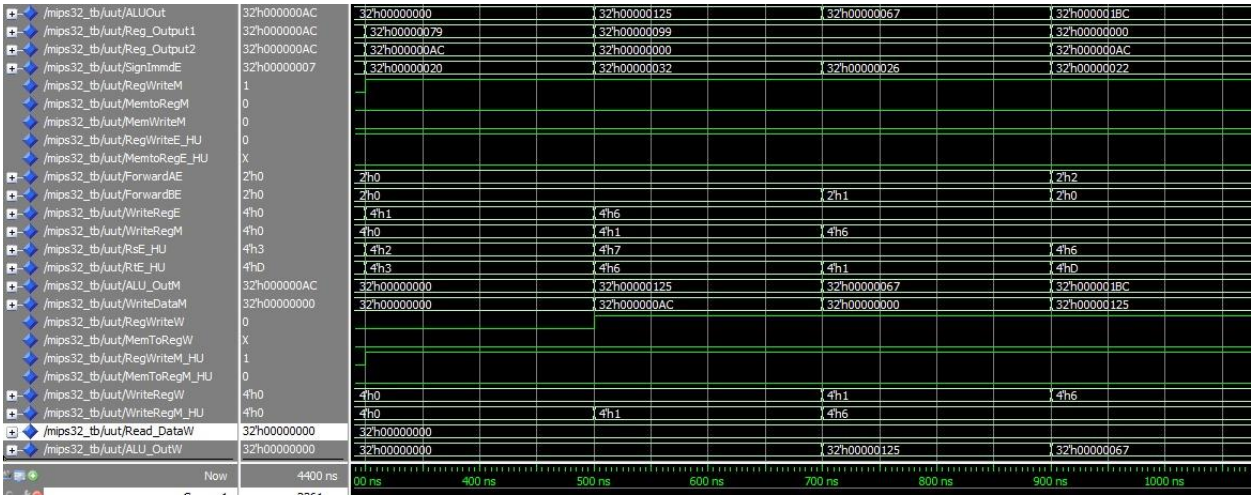

Figure 10: WB Stage

CONCLUSION

It is generally known that in today's world MIPS 32 is basically considered as a yardstick for the development in various computer architectures at present as well as in the future. In our project, the MIPS 32 has been implemented with pipelining to increase the architecture's speed which simultaneously gives rise to throughput/efficiency. What is obtained with the help of pipelining is high CPI.

Implementing this architecture is not an easy task as we faced various types of hazards. These hazards were overcome with the help of hazards control techniques and implementing them we obtained a hazard free pipelined MIPS architecture. Stalls were introduced wherever necessary and finally we were able to implement various MIPS instructions in pipelined fashion without any hazards.

WORKS CITED

(n.d.). Retrieved from MIPS Architecture Overview: https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/mips.html

Khan, D. G. (n.d.). MIPS-Lite Processor Datapath Design. Retrieved from https://www.ee.ryerson.ca/~courses/coe608/lectures/MIPS-Lite-Scycle-DP.pdf

MIPS ISA. (n.d.). Retrieved from http://web.cs.mun.ca/~paul/cs3725/material/review.pdf

Strukov, D. (n.d.). Pipelined MIPS Processor. Retrieved from https://www.ece.ucsb.edu/~strukov/ece154aFall2013/viewgraphs/pipelinedMIPS.pdf

Strukov, D. (n.d.). Pipelined MIPS Processor. Retrieved from https://www.ece.ucsb.edu/~strukov/ece154aFall2013/viewgraphs/pipelinedMIPS.pdf

David Money Harris and Sarah L. Harris, “Digital Design and Computer Architecture second edition.

COEN 6741 -”Computer Architecture and Design” (Dr. Sofiène Tahar) lecture notes (2018) and peer references.