Concordia University
Engineering and Computer Science

COEN 6741: Computer Architecture and Design

Project Report

On

"MINI-MIPS PROCESSOR"

Team Members

Cesar Augusto Pifaia Cipelli **(40070751)**
Bibin Eugine **(40075231)**
Neerav Bhatia (**40090577)**
Vivek Bansal (**40091987)**

TABLE OF CONTENTS

List of Figures

Table Contents

ABSTRACT

The aim of this project is to design a simple pipelined processor i.e. 32-bit five stage processor based on RISC architecture. The Mini-MIPS uses 3 instruction formats (R, I and J-types) The Mini-MIPS processor is designed to perform 10 MIPS instructions.

In this project, knowledge about various MIPS instructions, their instruction format and addressing, instruction data path, control module function and design theory based on RISC CPU instruction set is gained. Moreover, pipeline design process has been simulated successfully, which involves 5 stages i.e. instruction fetch (IF), instruction decode (ID), execution (EX), data memory (MEM) and write back (WB). Further, to handle the pipeline hazard, separate instruction-memory and data-memory as well as some hardware related to the forwarding technique have been designed.

The project is implemented (i.e. simulated) using ModelSIM. The block diagram is designed using an online software application “Creatly”.

1. INTRODUCTION

The MIPS architecture is based on a design by John Hennessy. It is a processor whose architecture would represent the lowering of the compiler to the hardware level, as opposed to the raising of hardware to the software level. Thus, the MIPS processor implemented a smaller, simpler instruction set. Each of the instructions included in the chip design ran in a single clock cycle. The processor used a technique called pipelining to more efficiently process instructions. Today, it is widely used as it power many consumer electronics and other devices. This report describes the " Design of Mini-MIPS processor".

1.1 Objective

The idea of the project is to design a simple pipelined processor, Mini-MIPS, which is a subset of the 32- bit MIPS architecture. It uses 3 instruction formats i.e. R, I and J types which is used to implement 10 instructions: AND, SUB, XOR, ANDI, SUBI, ADD, BEQ, LW, SW, and JR. Moreover, it is made sure that the pipeline design is hazard free by solving data and control hazards while stalling.

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems (an American company that is now called MIPS Technologies).

The key concepts of the MIPS architecture according to this project are:

- Uses general-purpose registers with a load-store architecture.
- Addressing Mode: -Displacement, register and immediate addressing.
- Support these data sizes and types: 32-bit IEEE integers.
- Support of simple instructions, load, store, add, subtract, register-register, register-memory and shift.
- Compare equal and branch (with a PC-relative address at least 8 bits long) and jump register (JR)
- In parameter like performance, it uses fixed instruction encoding and in code size parameter, it uses variable instruction encoding.
- Provides 16 GPR, Addressing Modes, Instructions and aim for a minimum instruction set.

MIPS with a cache controller and five-stage execution pipeline i.e. IF (Instruction Fetch), Instruction decode (ID), Execution/ effective address cycle (EX), Memory access/branch completion cycle (MEM), Write-back cycle (WB) could be integrated onto a single chip improving performance over non-pipelined designs. Pipelining is the execution of micro-operations (e.g., IF, ID, EX, MEM, WB) simultaneously which give us an increase in the throughput. In ideal pipelining the clock cycle per instruction (CPI) is equal to one but in reality, we can’t have ideal CPI because of Hazard.

The project design is structural hazard free. Data hazards are resolved with the help of bypassing or forwarding, whereas Branch hazards are solved using “Delayed Branch Concept and Prediction Schemes".

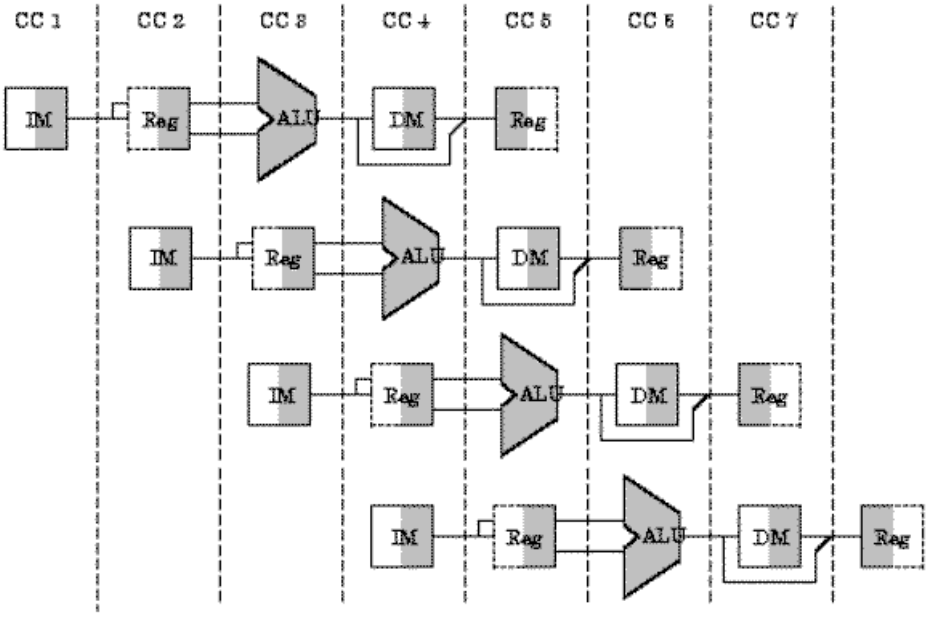

Fig. 1.1 MIPS - 5 Stage Pipeline

1.2 Software Environment

For large and complicated ASIC designs, it is difficult to read and understand the circuits based on schematic drawings alone; as a result, a hardware description language is much needed for a succinct, descriptive, and human-readable summarization of the circuit.

In this project, we explore VHDL, a hardware description language popular in educational environment. Through the design of a 32-bits pipelined CPU, several VHDL compilers, simulators, and synthesizers are present that are readily accessible for academic environments.

Usually, the easiest way to understand a simple circuit is to study its schematics. From the schematics, we can derive the gate structures, wiring structures, critical paths, and more. However, as the circuits become more complicated, their schematics also become harder to understand due to the heavy amount of wires and gates crossing each other. As a result, it is imperative to find a new method to accurately describe the circuits. The solution for this problem lies in hardware description language (HDL). There are two main varieties of HDL: VHDL (Very high-speed IC Hardware Description Language) and Verilog. Both of them are IEEE standards; however, Verilog is more common in production environment while VHDL is mostly utilized by educational institutions. In this project, VHDL is being used.

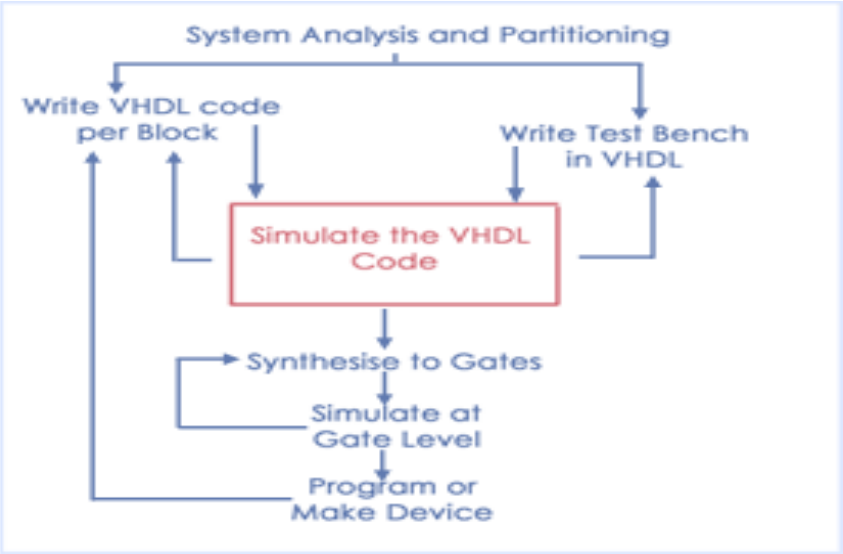

Fig. 1.2 Design-Flow

1.3 Design Methodology

For designing MIPS 32-bit processor, a methodology has been followed which is shown below in the flowchart.

Before starting the project, an intensive knowledge of VHDL has been gained which took several weeks. After that, the objective was followed in which it's further segmented into different parts among four team members, hence every team member has implemented and analyzed different part. Next step is to determine the ISA which gives an overview(schematic) of the whole 32-bit MIPS processor. Further, control unit has been designed in which basic 5-stages has been designed using VHDL. The core part is the execution unit in which instruction coding of every component is done especially in execution stage. Hazard Unit takes place in the next step and how to overcome data hazards using bypassing, whereas for structural hazards "Static prediction and delay branch" concepts overcome the branching problem. The result was simulated and, in the end, wave forms have been generated, if the results or waveforms are incorrect, it is loop back to "Instruction Encoding" stage. The whole concept is merged into one, hence forming a MIPS processor.

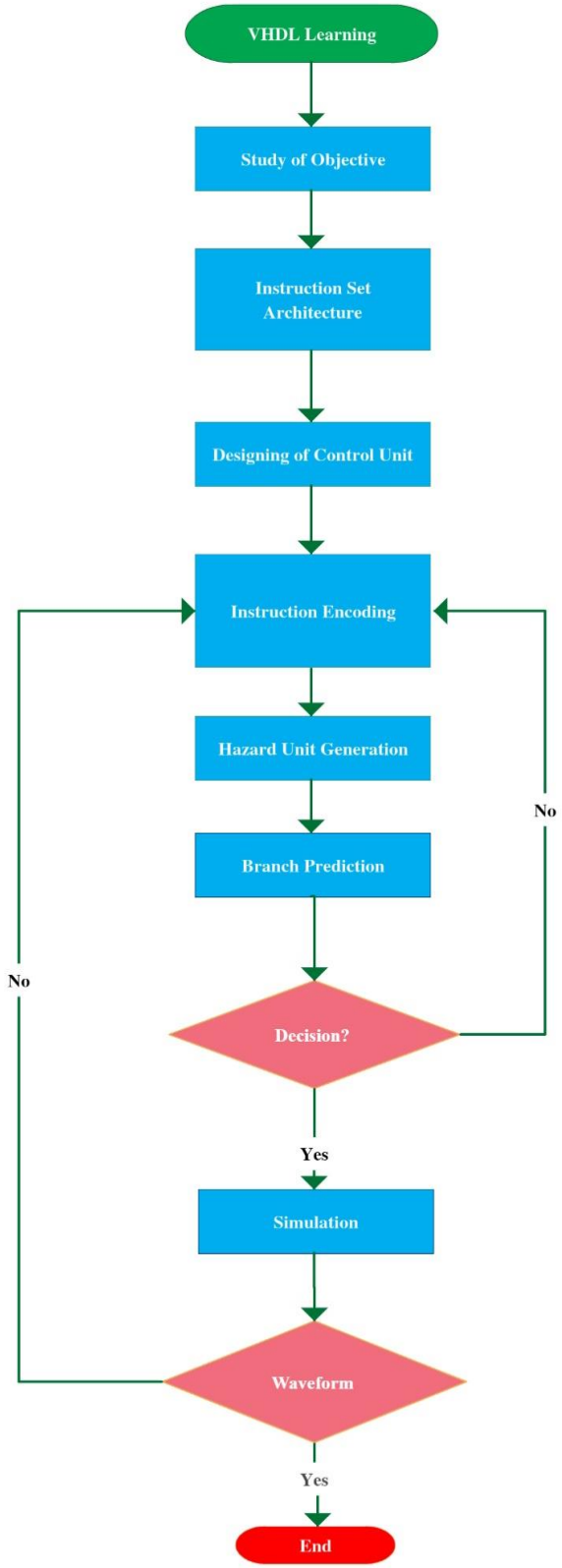

Fig. 1.3 Design Methodology (Flowchart)

1.4 Block Diagram

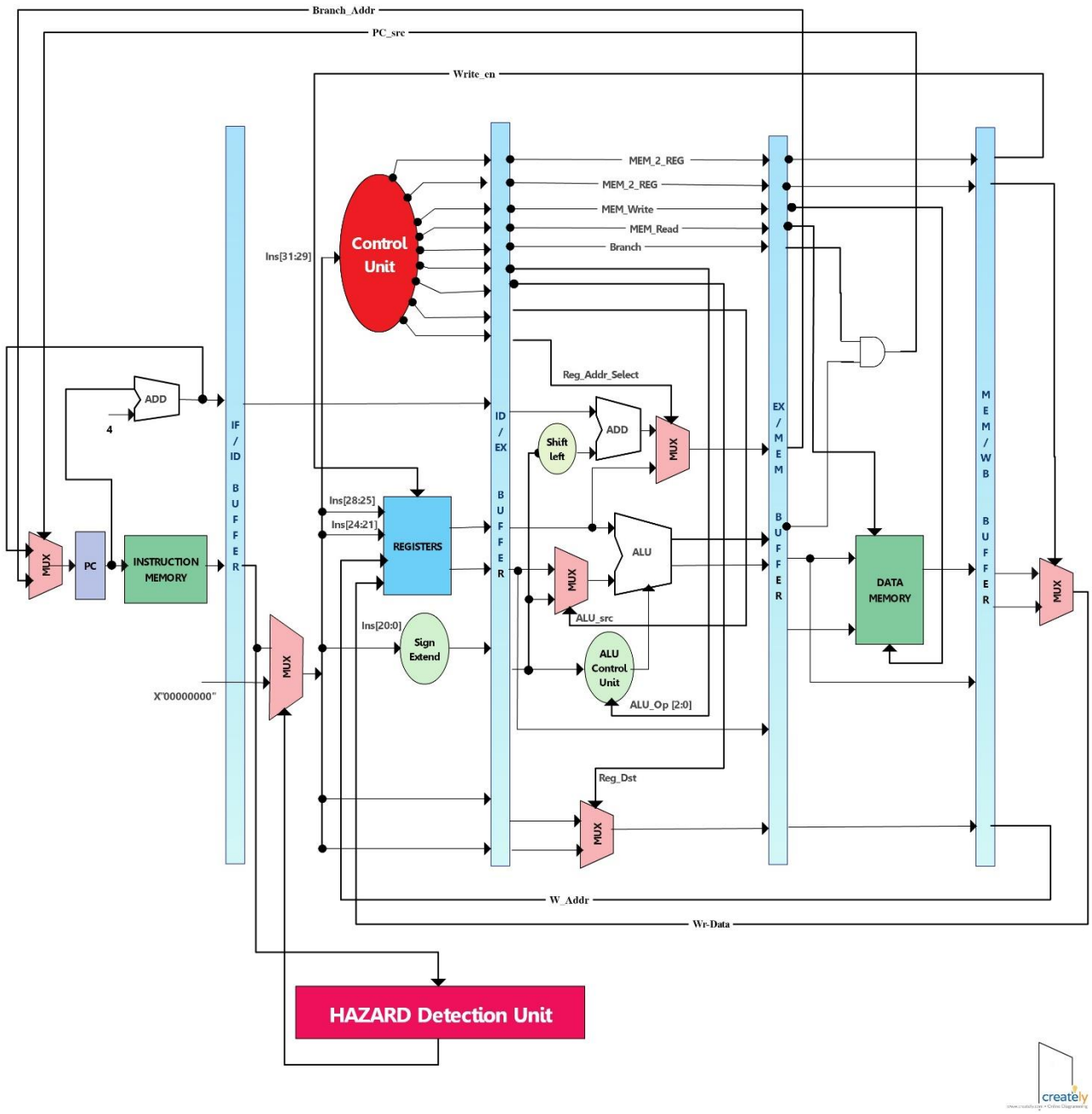

Fig. 1.4 Block Diagram

2. INSTRUCTION SET ARCHITECTURE (ISA)

2.1 MIPS Instruction Format

MIPS consists of three instruction types such as R-type, I-type and J-type as shown in figure 2.1:

Type	Format [31-0]					
R - Type	opcode (3)	rs (4)	rt (4)	rd (4)	shamt (14)	funct (3)
I - Type	opcode (3)	rs (4)	rt (4)	immediate (21)		
J - Type	opcode (3)	Address (29)				

Table 2.1: Instruction Format

Each instruction type has 3-bit opcode, apart from that R – type has three registers with 4 bits each along with 14-bit shift amount and 3-bit function field. While I – type instruction has two registers with 4-bits each and one 21-bit immediate field. Whereas J – type have one 29-bit address field. The meaning of each fields in the instruction are shown in the figure 2.2:

Field	Meaning
Opcode	Basic operation of instruction
rs	Source register operand 1
rt	Source register operand 2
shamt	Register destination operand
funct	Pick the right variant of operation in opcode
immediate	Immediate branch or address displacement
Address	Jump target address

Table 2.2: Meaning of Instruction Fields

2.2 MIPS instruction set definition

The instruction definition of the set is shown in the figure 2.3.

Category	Instruction	Example	Meaning	Type
Arithmetic	ADD	add $s1,$s2,$s3	$s1 = $s2 + $s3	R
	Subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	R
Logical	AND	and $s1,$s2,$s3	$s1 = $s2 & $s3	R
	XOR	xor $s1,$s2,$s3	$s1 = $s2 ^ $s3	R
	AND Immediate	andi $s1,$s2,30	$s1 = $s2 & 30	I
	SUB Immediate	subi $s1,$s2,30	$s1 = $s2 - 30	I

Data Transfer	Load Word	lw $s1,30($s2)	$s1 = Mem[$s2 + 30]	I
	Store Word	sw $s1,30($s2)	Mem[$s2 + 30] = $s1	I
Conditional Branch	Branch on equal	bne $s1,$s2,25	if ($s1== $s2) go to PC + 4 + 100	I
Uncond. Jump	Jump register	jr $ra	go to $ra	R

Table 2.3: Instruction Set Definitions

The MIPS instruction set consists of different categories such as arithmetic, logical, data transfer, conditional branch and unconditional branch.

1. **Arithmetic Instructions and Logical Instructions:** It uses either one sign-extend value and one source register or two source registers.
2. **Data Transfer Instructions:** It uses the values in the source register and offset value to calculate the effective address (EA) of the memory.
3. **Conditional Branch Instructions:** It uses the values of the two source registers and check the necessary condition whether it meet or not for the branch.
4. **Unconditional Branch instruction:** Irrespective of the condition, control will directly move on to the new address, which is the value in the register.

2.3 MIPS Addressing Modes

The different ways by which addressing the memory are generally called addressing modes. Standard MIPS has 5 different addressing modes such as immediate addressing, register addressing, base or displacement registering, PC – relative addressing, pseudo-direct addressing. In this project, only immediate addressing, register addressing and displacement addressing.

- **Immediate Addressing:** Operand is the constant value given in the instruction.
- **Displacement Addressing:** Operand is taken from the memory location whose address is the sum of the constant and register value in the given instruction.
- **Register Addressing:** Operand is the register.

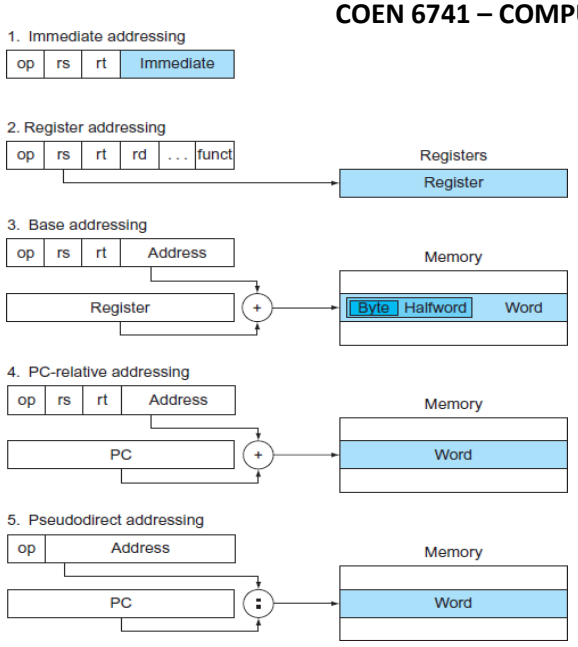

Figure 2.1: Illustration of five MIPS addressing modes

2.4 MIPS Instruction Encoding

Machines are unable to read human languages, so in-order to communicates with machines the respective system should convert the code to machine level language which is understandable by machines. Each instruction consists of operations and operands, but by encoding the entire instruction can be represented as binary values. Instruction encoding is done by assembler, here each instruction is encoded into 0's and 1's. The figure 2.5 shows the value of opcode and function corresponding to each instruction.

Name	Format	Opcode	Function
ADD	R	000	010
SUB	R	000	001
AND	R	000	000
XOR	R	000	011
ANDI	I	010	XXX
SUBI	I	110	XXX
LOAD	I	100	XXX
STORE	I	101	XXX
BEQ	I	011	XXX
JR	R	000	100

Table 2.4: Instruction Encoding

3. DESIGN OF CONTROL UNIT AND DATAPATH

3.1 Components of Datapath

In order to design data path, many components have to be examined like IF(PC), R-type, Load/Store, etc. The description of these components is given below:

3.1.1 Instruction Fetch (IF)

For an instruction to be executed, it first must be read out of memory. MIPS has a special program counter register (PC) that holds the address of the current instruction being executed. To execute the next instruction, PC is updated to PC+4 because it is assumed that each instruction takes one clock cycle. The components in this stage are: - Instruction Memory, which stores the instructions and supply instructions given at a given address pointed out by PC; (2) Program Counter (PC), a register that holds the address of the current instruction; (3) Adder, which is to increment the PC by 4 to the address of the next instruction.

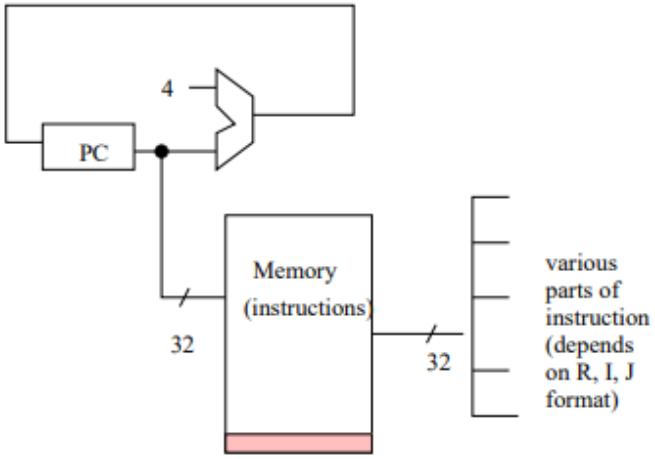

Figure 3.1 Datapath- Instruction Fetch, PC and Instruction Memory

3.1.2 R-type Instructions (ADD/SUB/AND/XOR)

For addition, AND and other various instructions, R-Type instruction format is used. There are three register fields in which one is the resulting field and other two are the operating which are performing the particular operation. The opcode field are used to control the as an input and the resulting output written into register file i.e. WriteReg.

So, the elements needed are: (1) Register file, the processor's 32 general-purpose registers which can be read or written by specifying the number of the register in the file; (2) Arithmetic logic unit (ALU), to operate on the values read from the registers.

Figure 3.2 shows the ALU, which takes two 32-bit inputs and produces a 32-bit result.

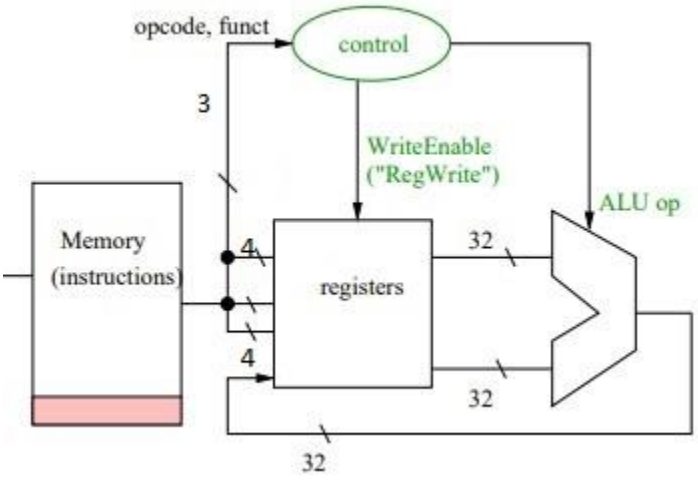

Figure 3.2 R type components- Register file and ALU

3.1.3 Load/store Instructions (LW/SW)

Now, next is load and store instructions which access the memory. Let's discuss below:

a) Datapath for LW

In addition to R-type, here base address (for memory) is read from register and is fed to ALU. Moreover, 21-bit signed offset field is also fed to ALU, also control signals are setup for the operation and ALU computes the effective address and is sent to memory from where it is loaded or written to register file.

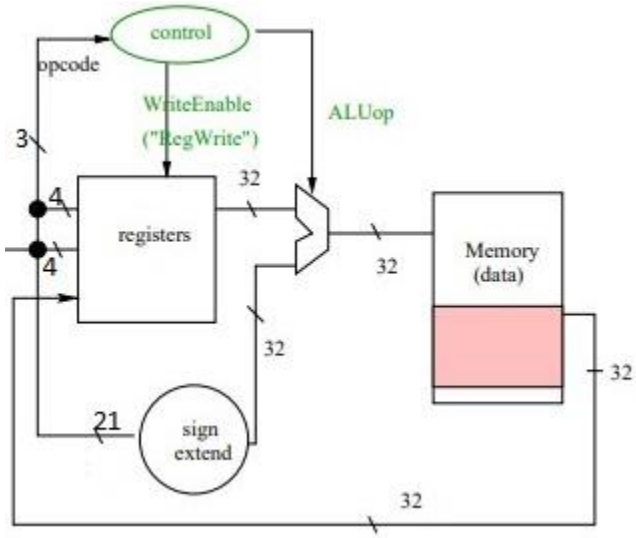

Figure 3.3 Load Instruction- ALU,21-bit offset field and Data Memory

b) Datapath for SW

In this, ALU computes the base address and 21-bit offset is added to it, result is sent to memory location from where it loads the corresponding data from the particular location and is loaded to register file and then it is stored into Data Memory.

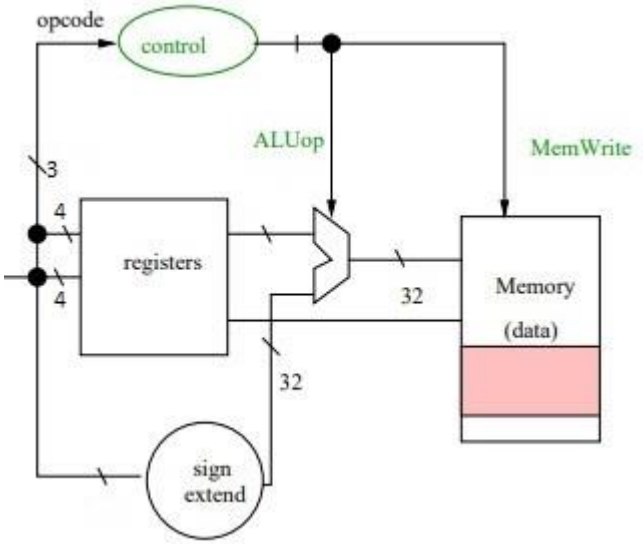

Figure 3.4 Store Instruction- Register File, ALU and Data Memory

3.1.4 Immediate Instructions (ANDI/SUBI)

In immediate instruction, there are two inputs in which one is taken from Reg file and other is a 21-bit offset field which is sign extended to 32-bit (for a 32-bit processor) and these are fed to ALU, hence the result is stored into Reg file.

In this, essential components are ALU and 21-bit offset field.

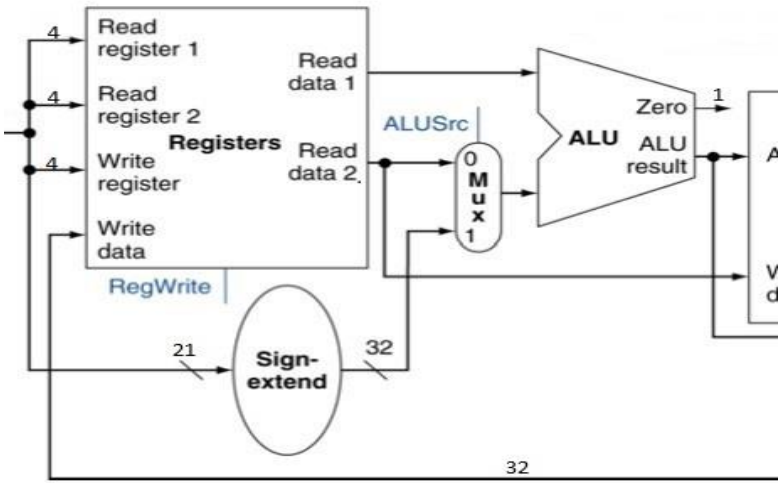

Figure 3.5 Immediate Instruction- ALU and 21-bit offset field

3.1.5 Control Instructions (BEQZ/JR)

The MIPS instruction BEQZ rs, rt, will branch to the target address (PC + 4) + offset if rs is equal to rt. The offset is 21 -bit which computes the branch target address.

Opcode (3-bit)	Rs (4 bit)	Rt (4 bit)	Immediate (21 bit)

Table 3.1 BEQZ Format

R - Type	opcode (3)	rs (4)	rt (4)	rd (4)	shamt (14)	funct (3)

Table 3.2 JR Format

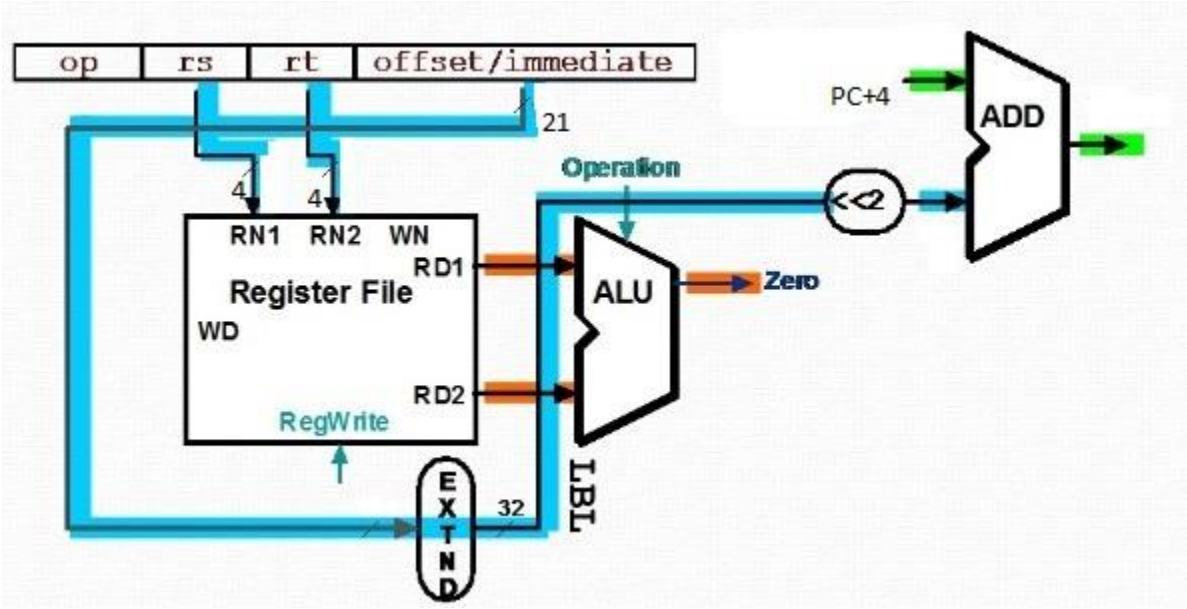

Figure 3.6 BEQ Data Path

The JR instruction only has an operand which is a 29-bit target address.

To implement BEQZ instruction, the branch data path must do two operations: (1) compute the effective branch target address using the offset label and (2) compare the register contents.

To perform the compare, we need to use the register file to supply the register operand rs and r0 (which is always 0). In addition, the comparison can be done using the ALU. Since that ALU provides an output signal that indicates whether the result was 0, we can send the two register operands (rs and rt) to the ALU with the control set to do a subtract. If the Zero signal out of the ALU unit is asserted, the two values are equal.

3.2 The Basic Control

A control unit is responsible for generating the correct control signals for this data path, based on the program code so that each instruction is executed properly. The control unit's input is the 32-bit instruction word, while the outputs are values for the blue control signals in the data path. The control signals are generated after the instruction is fetched.

For the ALU instruction format in Figure 2.1, ALU needs to perform one action depending on the value of the 3-bit functional field in the low-order bits of the instruction. For BNEZ instruction, the ALU must perform a subtraction. For another control signals, the similar processes are implemented to gain the control unit.

3.3 Pipeline and Hazards Control

Pipeline implementation and hazards control is an important concept. The MIPS processors have five pipeline stages: - Instruction Fetch, Instruction Decode, Execute, Memory and Write back.

Hazards are the circumstances that would cause incorrect execution. It can be characterized into three categories: Structural hazard, data hazard and control hazard.

In this project, we aim for structural hazards which should be hazard free, also for data hazard, it can be resolved by forwarding technique; whereas for branch, we are considering a one delayed slot.

Obviously, to avoid the structural hazard, we need to design separate instruction-memory and data-memory and to handle the data hazard, we design the processor with forwarding technique. To implement the forwarding technique, we need to design elements as follows: buffer between stages, multiplexers, and control unit which control pipeline as well as forward control signals and data through the pipeline.

3.4 Putting it all together

In the end, we are merging every component including MUXes, adders, etc. into one. For this we have to create different pipeline stages, every component, Control Unit, etc. Through structural modelling in VHDL, we can integrate all the component into one and in the end, we made a "MAIN VHDL SOURCE CODE". The main code is simulated with different test signals and waveforms have been plotted.

4. SIMULATION AND TEST WAVEFORMS

The simulation results and waveforms of different hazards are given below: -

```
ARCHITECTURE arch1 of InstructionMemory is

--memory should have 4 times less words than bytes (1 word = 4 bytes);
--for a 2^8 bits address memory we have 256 byte addresses and 64 word addresses.
-- In reality, this memory is not byte addressable

TYPE MEM is array(0 to 64) of std_logic_vector(31 downto 0);

CONSTANT rom_arr: MEM := (
0=>"10000000001000000000000000000000",
1=>"10000000010000000000000000000001",
2=>"11000010111111111111111110111001",
3=>"00000100001001100000000000000010",
4=>"00000100001010000000000000000000",
5=>"00000100001010100000000000000001",
6=>"00000100001011000000000000000011",
7=>"10100000011000000000000000000101",
8=>"10100000100000000000000000000010",
9=>"10100000101000000000000000000011",
10=>"10100000110000000000000000000100",
11=>"01101110011111111111111111110110",
12=>"00000100001011000000000000000011",
13=>"00000100001011000000000000000011",
OTHERS=>"00000000000000000000000000000000");
```

Fig. 4.1 A glimpse of Instruction Memory Source Code

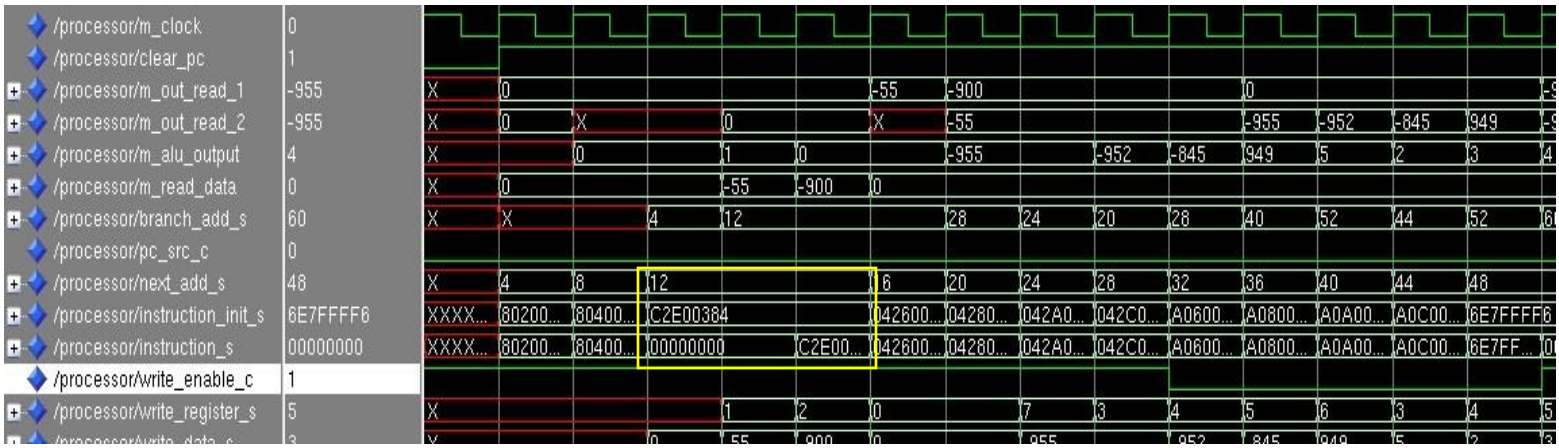

Figure 4.2 Simulation Output: Data hazard

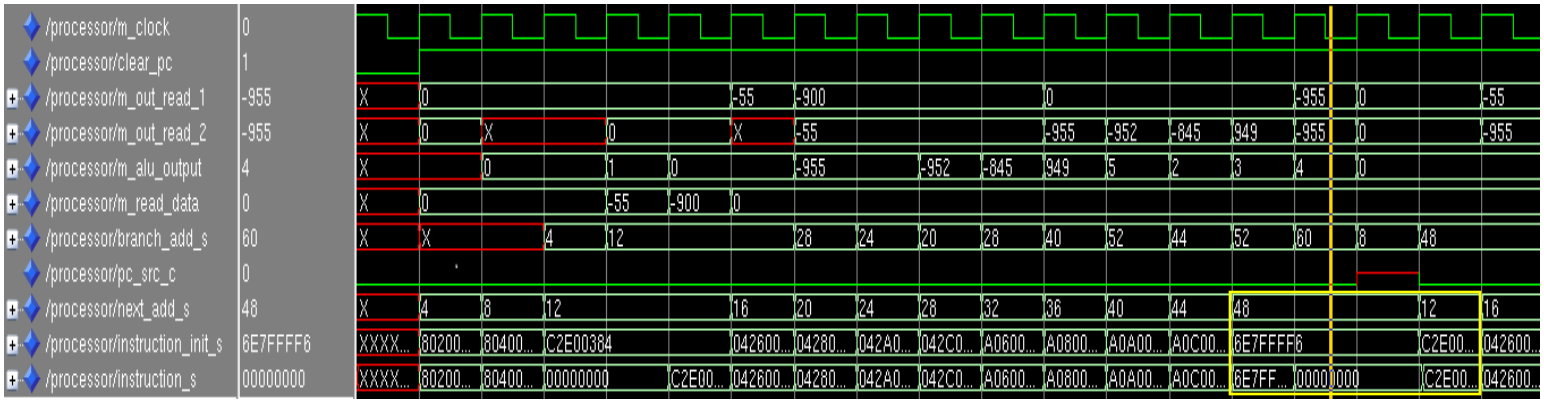

Figure 4.3 Simulation Output: BEQZ: - Taken Case

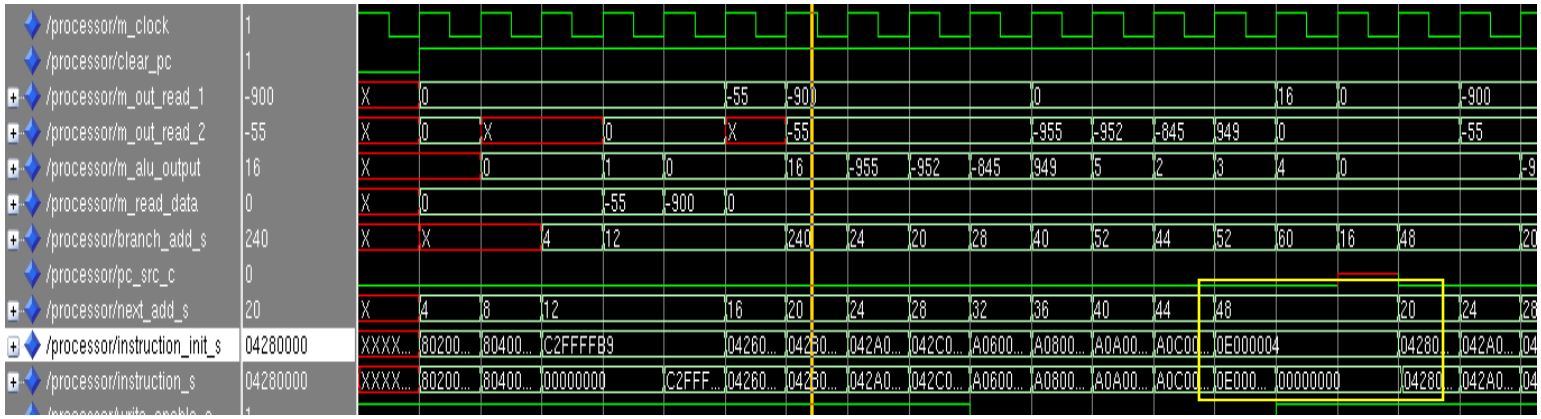

Figure 4.4 Simulation Output: BEQZ: JR Instruction

5. CONCLUSION

In this project, knowledge about 32 bit Mini-MIPS processor has been gained and implemented successfully. The aim is to implement 10 instruction provided that, structure will be hazard free and to reduce data and control hazards by forwarding, branch delay techniques, etc. This project is implemented in ModelSIM and VHDL is chosen to be software environment.

Furthermore, apart from MIPS, this project gives intense knowledge about VHDL which is beneficial for beginners. Through VHDL modelling, it has set a hand on experience in ModelSIM (VHDL), though facing difficulties in hazard section, it is managed successfully with the help of our team, our coordination and dedication.

6. REFERENCES

[1] "Computer Architecture: A Quantitative Approach 6th Edition", John L. Hennessy and David A Patterson, Publisher- Morgan Kaufmann, Edition 2017, ISBN 978-0128119051.

[2] "Review of Datapath Components"

http://www.cim.mcgill.ca/~langer/273/13-datapath1.pdf

[3] "Introduction to MIPS Architecture"

http://web.engr.oregonstate.edu/~walkiner/cs271-wi13/slides/02-MIPSArchitecture.pdf

[4] "The Control Unit"

http://www.pitt.edu/~kmram/CoE0147/lectures/datapath3.pdf

[5] "Computer Organization and Design: The Hardware/Software Interface, John L. Hennessy and David A Patterson, Publisher- Morgan Kaufmann, Edition 2015, ISBN 978-0-12-407726-3.

[6] "Computer Architecture and Design: Lecture Notes" (Dr. Sofiène Tahar) 2016.

[7] "RTL Hardware Design Using VHDL: - Coding for Efficiency, Portability and Scalability", Pong P.Chu, A John Wiley and Sons, Inc, Publication, 2006, ISBN 978047172092.

[8] "RISC Architecture"

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/mips/index.html