Implentation of 32-bit RISC Processor using Verilog HDL

# Raj Nirmal

*Dept. of Electronics and Communication Engineering Nirma University, Ahmedabad, India* [20bec075@nirmauni.ac.in](mailto:20bec075@nirmauni.ac.in)

***Abstract*—This paper describes the development of a 32-bit, 5-stage pipelined, MIPS-based RISC core. MIPS is an acronym for “Microprocessor without Interlocked Pipeline Stages”. RISC architecture (Reduced Instruction Set Computer) is a type of microprocessor that was created to carry out a condensed set of instructions in order to boost the processor’s overall speed. Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB) modules make up the five phases of the MIPS pipeline. Instruction Memory, Data Memory, ALU, Registers, and other modules are employed. The goal of this study is to implement the pipeline efficiently by incorporating the Hazard detecting unit and Data forwarding unit. Verilog-HDL is used in the design’s development.**

***Index Terms*—Introduction, Instruction Set, Architecture, Sim- ulation Results**

1. INTRODUCTION

Two philosophies—Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer—have been used to create microprocessors and microcontrollers (RISC). The CISC concept is a method of designing the Instruction Set Architecture (ISA) that stresses getting more done with each instruction while utilising a wide range of addressing modes and varied numbers of operands in different places across the Instruction Set. Because of this, the instructions have wildly different lengths and execution times, necessitating the use of an extremely complicated Control Unit that takes up a lot of space on the chip. In contrast, the RISC processor has a smaller number of instructions, a fixed instruction length, more general-purpose registers, a load-store architecture, and simpler addressing modes than the CISC processor, which results in faster execution of individual instructions, a net improvement in performance, and a simpler overall design with less silicon consumption. The RISC design is perfectly suited to take part in ”system-on-a-chip,” a significant trend in the embedded processor market thanks to the capabilities listed above. The most popular RISC processors are IBM’s PowerPC, ARM, SP ARC, and MIPS. Many semiconductor firms use RISC processors as part of ”systems on chips,” like Atmel AVR and Micro Blaze, which are popular in embedded and DSP applications. The RISC design paradigm, which stresses load/store architecture, is the foundation for MIPS processor design. It is significantly faster to carry out operations in on-chip registers than in memory because of the difference in access time between a register and a memory

address. The architecture remains the same for all MIPS-based processors while the implementations may differ like a single- cycle, multi-cycle and pipelined implementation.

1. INSTRUCTION SET

Three different types of instruction sets make up the MIPS design. Types include Register, Immediate, and Jump. The following figure depicts the instruction format, respectively.

1. *Register Type (R-Type):*

The R-Type instruction format is shown in Figure 1. The opcode is represented by these last 6 bits. The next 15 bits stand for the three registers Rs, Rt, and Rd, which are used for operations. Source registers are Rs, Rt, and destination registers are Rd, respectively.

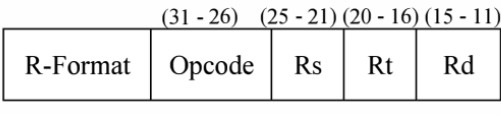


Fig. 1. R-Type Instructions Format

Figure 2 shows the data route for the instruction of the R-Type type. The primary operations it performs are ADD, SUB, and OR. Like adding Rd, Rs, and Rt Here, the signed addition’s (Rs) + (Rt) contents are saved into Rd.

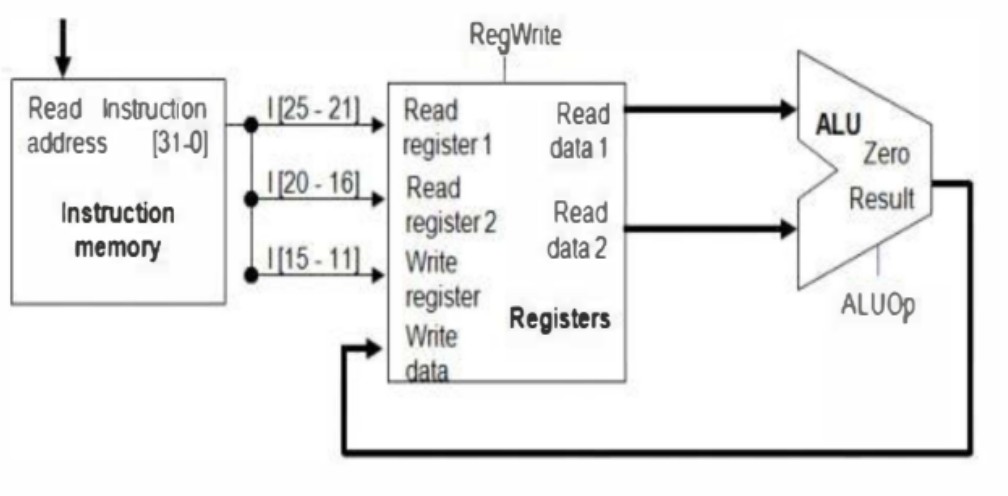


Fig. 2. Data Path for R-Type Instructions

1. *Immediate Type (I-Type):*

I-Type instruction format is seen in Figure 3. Similar to R- type, the first 6 bits reflect the opcode while the following

10 bits, respectively, represent Rs and Rt. Note that Rt is the source register for the store operation and that Rs is the destination register for the load operation. The final 16 bits are used to indicate instantaneous values, which are a component of instructions but not of memory.

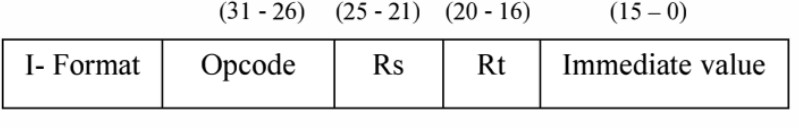


Fig. 3. I-Type Instructions Format

Figure 4 can be used to illustrate the data route for an I- Type instruction. The last six bits are the instantaneous value supplied to the sign extend and then to the ALU, respectively, in order to complete the necessary function, demonstrating that the Rt register may be used as both a source and destination properly. For ADD!, AND!, and OR! operation, it is utilised. As in addi Rt, Rs, 5, where 5 is an instant value and (Rs) + 5 is placed in the destination register Rt

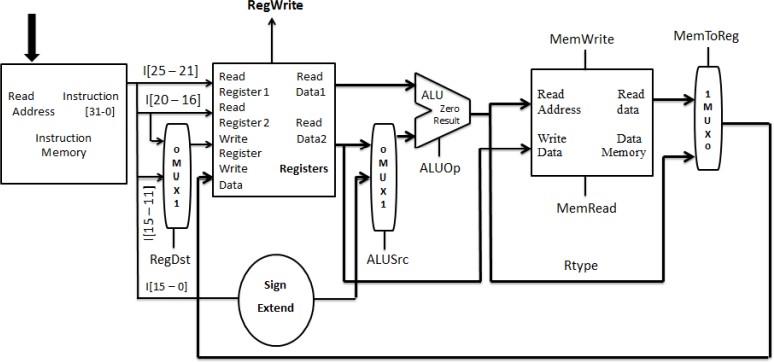


Fig. 4. Data Path for I-Type Instructions

1. *Jump Type (J-Type):*

J-Type instruction format is seen in Figure 5. The type of branch operation to be conducted is represented by the first 5 bits of this instruction format. In 2’s complement format, the branch offset is represented by the final 26 bits. This number is added to the value of the PC to obtain the branch target address.

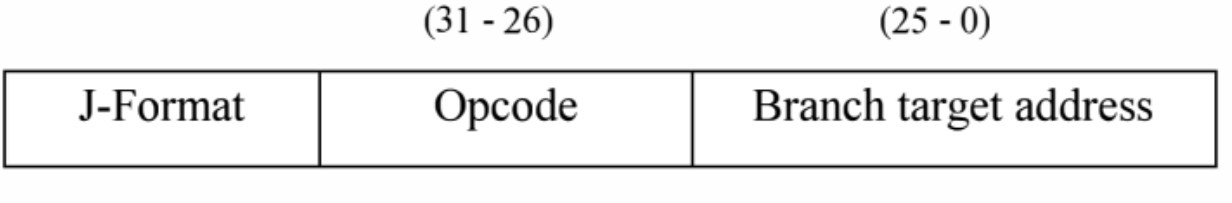


Fig. 5. J-Type Instructions Format

The J-type instruction’s functionality is shown in figure 6. It demonstrates that to obtain the 32-bit jump address, the final four bits of PC+4 are added to the 26-bit instruction’s shift left by 2-bit value. Example: j trgt. Here, j is jump instruction word and trgt is target. It skips the other instructions and jumps to the trgt.

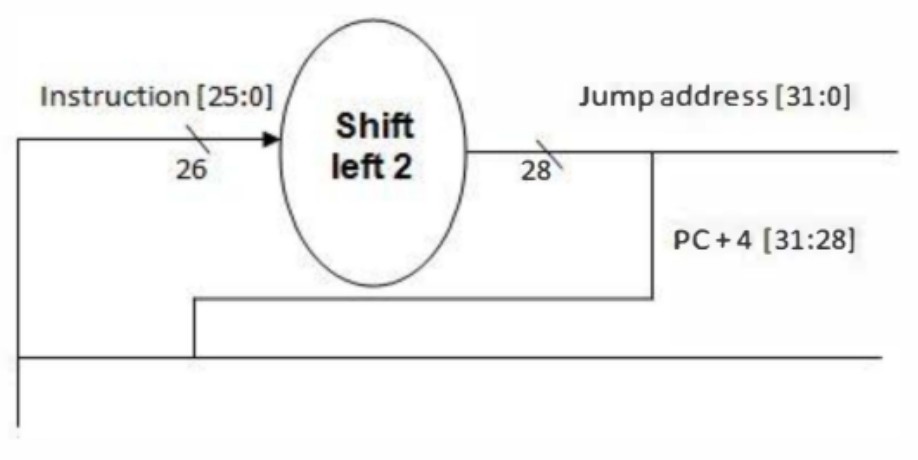


Fig. 6. Data Path for J-Type Instructions

1. ARCHITECTURE

The implementation of the pipe lined architecture on a MIPS-based RISC CPU. The MIPS architecture featured a five-stage pipeline. The act of performing many operations along a single data route is known as pipelining. A method for enhancing RISC processor performance overall is pipelining. Multiple processes are present on a multicycle CPU. For instance, although load may require up to 5 clock cycles, beq only needs 3 clock cycles. Therefore, if a process is already underway, start a new process in the same data stream at the same time without interfering with the active process. To make this possible, each step in the process is broken down into a number of pipelined steps.In order to allow another operation to begin in that stage without interfering with the current process, the process is stored into the following pipe- lined stage after every clock. Therefore, the entire path’s steps can be utilised at once. This in turn might boost the MIPS design’s throughput.

Five pipeline stages have been used to implement the MIPS processor architecture. Instruction Fetch (IF), Instruction De- code (ID), Execution (EX), Memory Access (MEM), and Write Back are the stages of this pipeline (WB). Pipeline reg- isters are specialised registers that serve as a barrier between these stages. These registers serve to separate the phases of the instructions so that data conflicts brought on by many instructions being executed at once are avoided. They are called the IF/ID Register, ID/EX Register, EX/MEM Register, and MEM/WB Register after the stages they are positioned between. The datapath for a MIPS pipelined CPU is shown in figure 7.

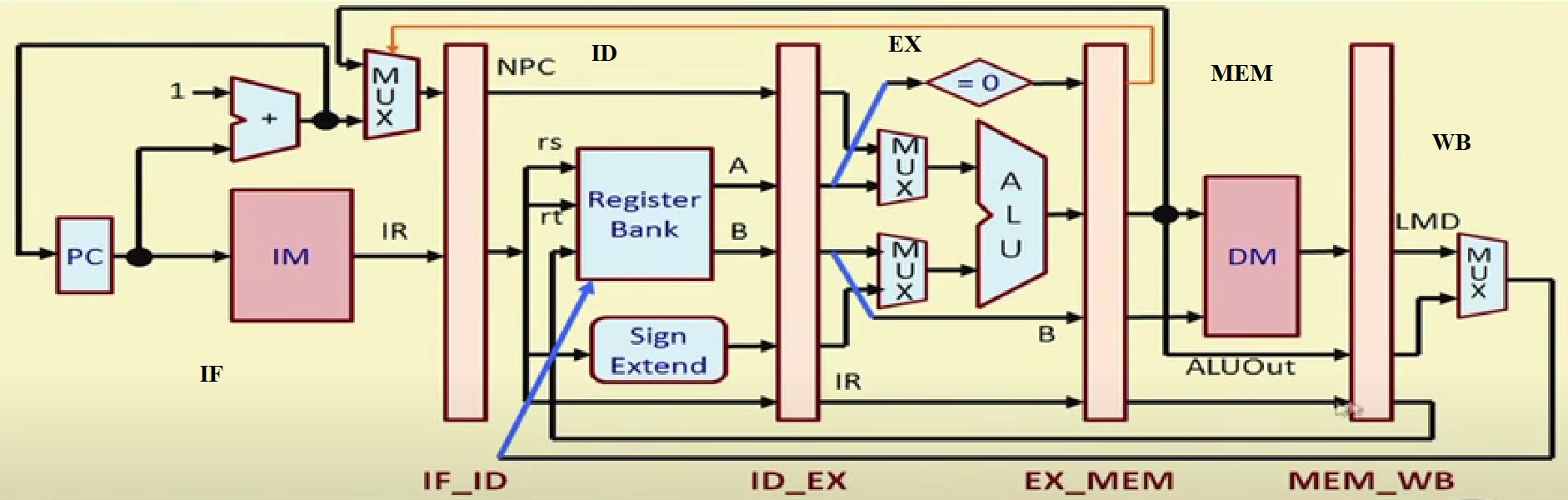


Fig. 7. Data Path for MIPS Processor

1. *Instruction Fetch:*

The instruction is pulled from the Instruction Memory by the Program Counter (PC) and put in the Instruction Register (IF/ID) at the following positive clock. The necessary instructions are stored in modules like Instruction Memory at this stage. The address of the currently running instruction on the computer serves as the address for the instruction memory. The Instruction Register, a component of the IF/ID stage, houses the instructions that have been read from the Instruction memory.

1. *Instruction Decode:*

Decodes the instructions supplied from the Instruction Reg- ister at this level. It reads the operands needed for the register file based on the instructions. 16 of the 32 bits are used for sign extension, which increases those 16 bits to 32 bits. The value of two registers, which are passed to the ALU through the ID/EX stage, is provided by the register file module.

1. *Execution:*

At this point, all instructions are carried out. This level is where all ALU operations, including logical and mathematical operations, happen. It manipulates the data that was sent from the ID/EX stage. For beq operation, this stage also includes a left shift by 2 and an adder. The ALUout register, which is in the EX/MEM stage, receives the ALU output result.

1. *Memory Access:*

To read from and write to the data memory is what the memory access stage is for in this phase. To choose which operation to do, control signals were sent to the EX/MEM register. The MEM/WB register receives the WB control from the EX/MEM register and writes the memory’s output along with it.

1. *Writeback:*

Essentially, at this stage, the result is written back into the register file. Additionally, it is in charge of removing data from the EX/MEM register and writing it to one of the registers in the register file after it has been computed or loaded from memory.

1. HAZARDS IN THE STRUCTURE
2. *Structural Hazards*

Structural hazards mean two instructions which are already there in the pipeline, in two different stages but trying to use the same hardware resource. If there is a single copy of the hardware resource then they cannot access it together, so one of them has to wait and the other can proceed. This is an example where we may have to insert a stall cycle in a pipeline which means one clock cycle may get wasted. For example, if you are fetching instructions from memory and in mem for load and store instructions also you are accessing memory supposing you had a single memory then you could not do these two things together.

In order to address this, we have separated instruction memory

and data memory. As a result, instructions will be kept in one memory and data will be stored in a different memory, eliminating this type of hazard.

1. *Data Hazards*

Data hazards occur when an instruction is exposed by the overlap of the instructions in the pipeline depending on the outcome of a preceding instruction in a way that makes the results unavailable. This causes the pipeline to stall until the results are made available.

Let us take an example of the group of successive instructions and understand the issue of data hazard, E.g.

SUB R2,R1,R3 (Register R2 written by sub)

AND R12,R2,R5 (1st operand(R2) depends on sub) OR R13,R6,R2 (2nd operand(R2) depends on sub)

Above all instructions are dependent on sub instruction. R2 stores resulting subtraction of R1 and R3. Figure 9 illustrates how these instructions are interdependent. It is evident that R2 only updates its value at clock cycle 5, and that before to that time the written value is unavailable. However, all succeeding instructions followed by sub instructions read the value from R2. In essence, they require the modified value within the following clock cycle. This is called data hazard.

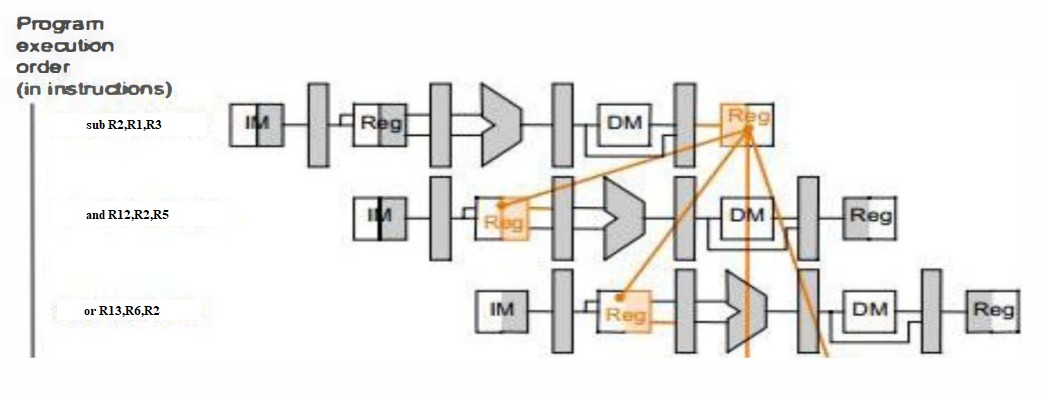


Fig. 8. Pipelined Data Dependencies

1. *Control Hazards*

Control Hazards arise from branch instructions. For exam- ple, if a branch instruction has already entered the pipe and you decide to branch only after the EX stage and there are already two other instructions in the pipe when the branch instruction enters the EX stage, so those two instructions must be discarded.

1. SIMULATION RESULTS

The verilog code for 32-bit MIPS based RISC Processor compiled using Quartus II 13.0sp launch and simulated using Modelsim tool to check their outputs. Simulation waveform is shown in figure 10.

We have simulated the code for the set of instructions mentioned below:

ADDI R1, R0, 10 ADDI R2, R0, 20

ADDI R3, R0, 30 ADD R4, R1, R2 ADD R5, R4, R3 HLT

Which basically initializes R1 with 10, R2 with 20 and R3 with 30. Add the numbers in the R1 and R2 registers and store the sum in R4. In a similar fashion, it adds R3 and R4 registers and stores the result in R5. But due to the presence of Data Hazards we have to add dummy instructions in between for proper execution.

ADDI R1, R0, 10—–(i)

ADDI R2, R0, 20—–(ii)

ADDI R3, R0, 30—–(iii)

OR R7,R7,R7 (Dummy)—–(iv)

ADD R4, R1, R2—–(v)

OR R7,R7,R7 (Dummy)—–(vi)

ADD R5, R4, R3—–(vii)

HLT—–(viii)

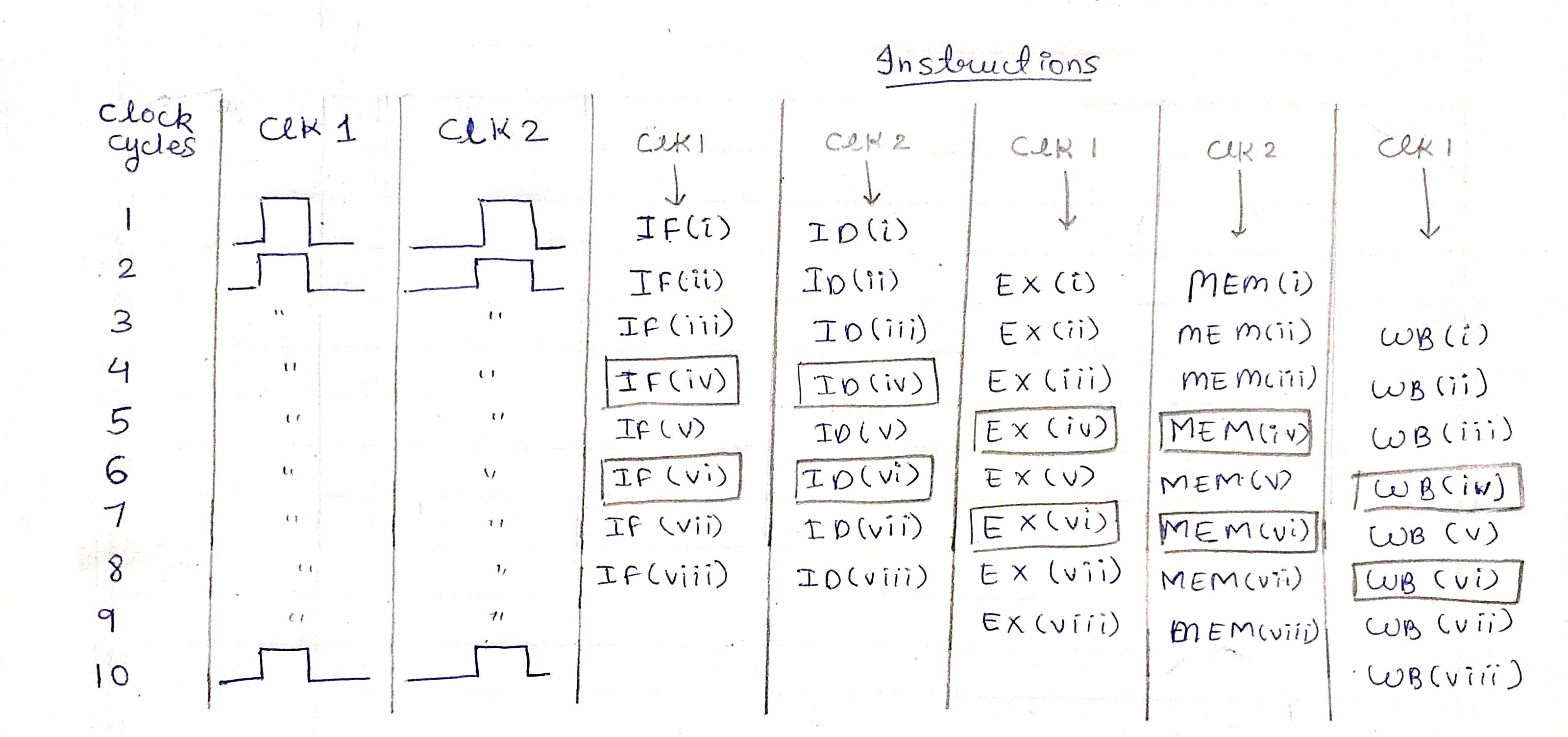


Fig. 9. Timing Diagram

Here Clock 2 is with some delay w.r.t Clock 1 and the instructions represented in the box are referred to as dummy instructions.

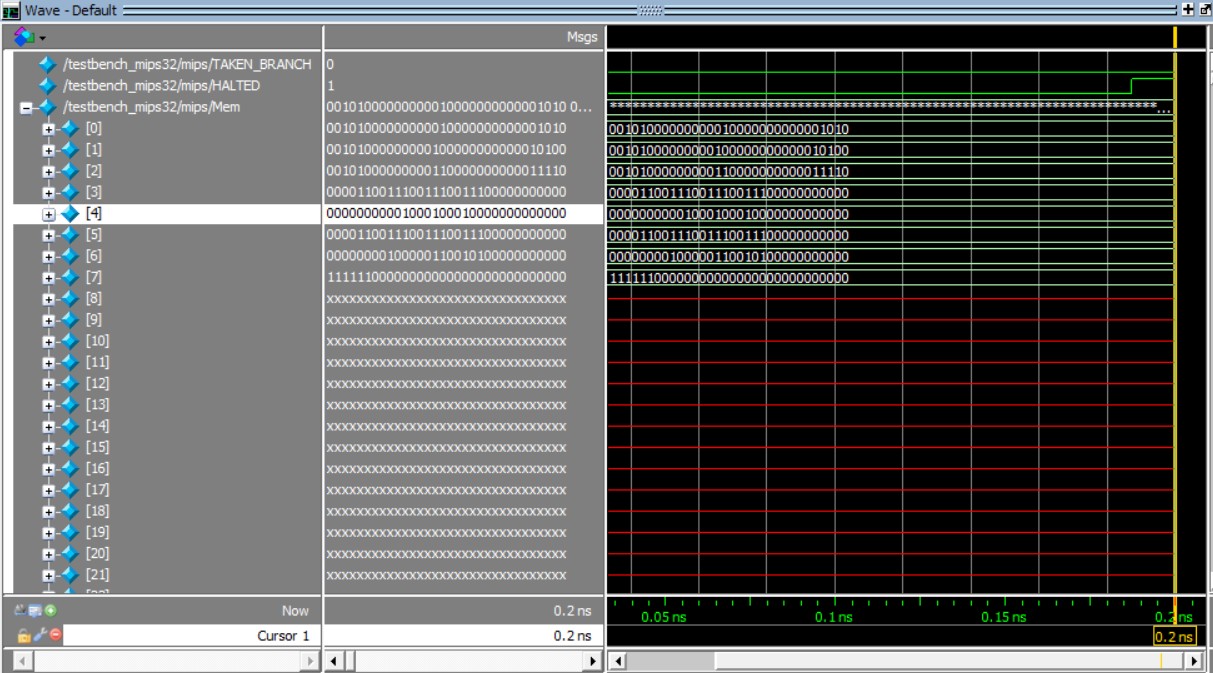


Fig. 10. Simulation Waveform

Let us simulate it for another set of instructions which contains load and store instructions also:

ADDI R1, R0, 120 LW R2, 0(R1) ADDI R2, R2, 45 SW R2, 1(R1) HLT

Which basically loads a word from memory location 120, adds 45 to it and then stores the result in memory location 121. In Figure 11 you can see the contents of all the registers mentioned in the proposed structure and in Figure 12 you can the end result for the above set of instructions.

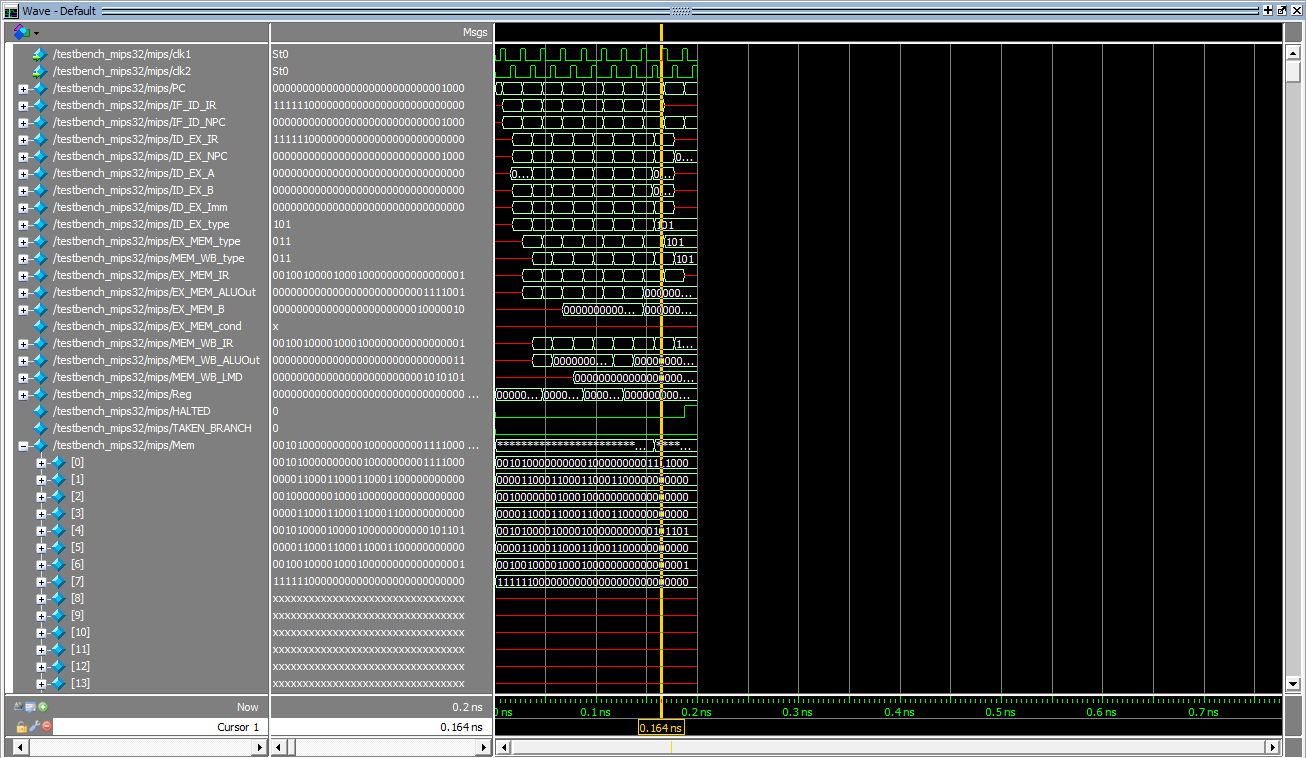


Fig. 11. Simulation Waveform for another set of instructions(a)

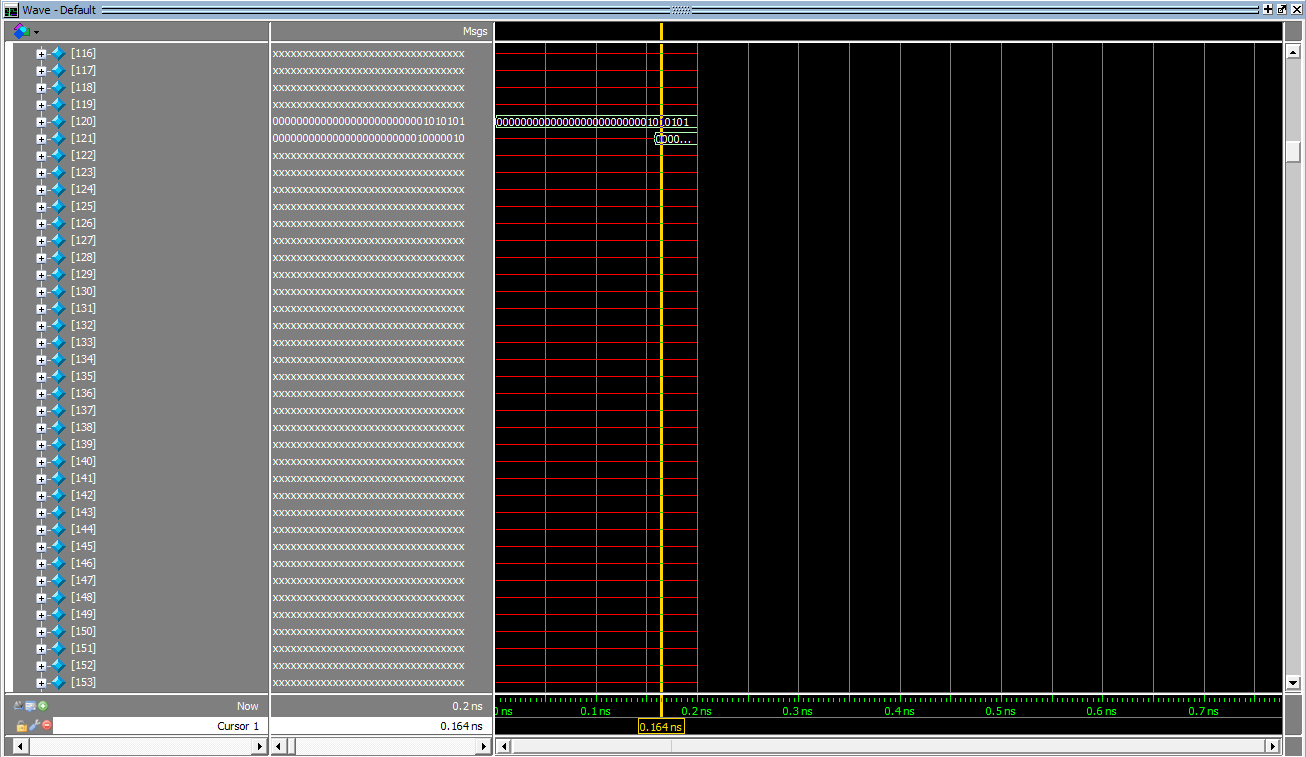


Fig. 12. Simulation Waveform for another set of instructions(b)

1. CONCLUSION

This work effectively implements the design of a 32-bit MIPS-based RISC processor with pipeline functions. With 5- stage pipe lining, every instruction is carried out in a single clock cycle. This design demonstrates the creation of a MIPS- based CPU that can handle a variety of R-type, J-type, and I- type instructions, each of which has a unique format. Through testbench, these instructions have been properly validated. Designing a forwarding unit and a hazard detecting unit was a crucial effort that was successfully carried out in order to overcome data dependencies. The design is implemented using

Verilog-HDL and synthesized using Quartus compiler.The design of the MIPS processor is optimized both in timing and area.

1. ACKNOWLEDGEMENT

Prof. Hardik Joshi is to be commended for providing us with the opportunity to complete this assignment within the time span allotted. We learned about the Implementation of 32-bit RISC Processor using Verilog HDL in this research. Furthermore, we would like to express our gratitude to all of the teachers who assisted us.

REFERENCES

1. “Computer Architecture: From Microprocessors to Supercomputers” book by Behrooz Parhami.
2. Neenu Joseph. Sabarinath.S. ”FPGA based Implementation of High Performance Architectural level Low Power 32-bit RISC Core”, 2009 IEEE.
3. Priyavrat Bhardwaj, Siddharth Murugesan. ”Design and Simulation Of a 32-bit RISC Based Mips Processor Using Verilog”