DESIGNING OF RISC V ARCHITECTURE

A DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & EMBEDDED SYSTEMS

Submitted By:

VANSHIKA KUMAWAT (2K21/VLS/22)

Under The Supervision Of:

PROF. O.P. VERMA

(PROFESSOR & HOD, ECE, DTU) &

DR. ROHIT KUMAR

(ASSISTANT PROFESSOR, ECE, DTU)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY, DELHI

(Formerly Delhi College of Engineering)

Bawana Road, Delhi- 110042

MAY 2023

DEPARTMENT OF ELECTRONICS &

COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi- 110042

UNDERTAKING

I, **Vanshika Kumawat**, **Roll No. 2K21/VLS/22** student of MTech (VLSI Design & Embedded Systems), hereby declare that the project Dissertation titled "**Designing of RISC V Architecture**", that is submitted by me to the **Department of Electronics & Communication Engineering, Delhi Technological University, Delhi** in the partial fulfillment of the requirement for award of the degree of **Master of Technology** is my original work. I have not plagiarized or submitted the work for the award of any other course. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

Date: Vanshika Kumawat

Place: Delhi (2K21/VLS/22)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Delhi Technological University, Delhi

(Formerly Delhi College of Engineering)

Delhi- 110042

CERTIFICATE

I hereby certify that the work contained in the project Dissertation titled "**Designing of RISC V Architecture",** submitted by **Vanshika Kumawat, Roll No. 2K21/VLS/22** in the partial fulfilment of the requirement for awarding degree of Master of Technology in VLSI Design and Embedded System to the **Department of Electronics and Communication Engineering**, Delhi Technological University, Delhi, is a record of project work done by the student under my supervision. To the best of my knowledge the work has not been presented in part or full for other Degree of Diploma in this university or anywhere else.

[illegible] 30.05.23

Prof. O.P. Verma
(Professor & HOD)
ECE Department
DTU, Delhi

Dr. Rohit Kumar
(Assistant Professor)
ECE Department
DTU, Delhi

Date:
Place: Delhi

ABSTRACT

The University of California, Berkeley created instruction set architecture known as RISC V. Because the majority of CPU instructions were not used by most computer programmes, the RISC architecture was developed. The RISC V processor was introduced to reduce instruction set increase registers resource investment. Due to its total open source and free nature, this technology caught the attention of numerous IT conglomerates and start-ups. The word "RISC" refers to the "reduced instruction set computer" in the RISC V Processor. which only executes a small number of computer instructions, and "V" refers to the fifth generation. It is an ISA (instruction set architecture) for hardware that is open- source and is based on the well-established RISC idea. We designed processor that supports instructions each of 32 bit that propagate via five stages of pipelined registers for fetch, decoding, execution, memory access, and writeback. We have also demonstrated how instruction and data memory interconnected with the CPU core forming a whole processor. Pipeline facilitates simultaneous execution of many instructions and displays the working of an instruction within processor using data from the source and destination registers. Further, overcoming data hazard issue while executing instruction as upcoming instructions need updated value of register from previous one's destination register through stalling by giving an extra cycle to wait and Forwarding unit where we can pass the data directly to next instructions in sequence without taking pause for a cycle to fasten the process and get correct values. Additionally, we used the Fibonacci program to evaluate the functioning of our processor and confirmed it by successfully having this series into memory. As a result, Various manufacturing firms have announced also offered RISC-V with open-source operating systems. This is a brand-new architecture that is offered under open, liberal, and cost-free licences. The chip and device manufacturing industries provided this CPU with marked support. Therefore, primarily made to be flexibly expandable and configurable for variety of applications. Professor David Patterson at the University of California, Berkeley created the RISC sometime in the 1980s. In two volumes titled "Computer Organization and Design" and "Computer Architecture at Stanford University," professors John Hennessy and David contributed their work. As a result, theywere giventhe ACM A.M. Turing Prize in 2017.

ACKNOWLEDGEMENT

It gives me immense pleasure to present this project work for partial fulfillment for awarding degree of Master of Technology in the VLSI Design . I owe special gratitude to my supervisor **Prof. OP Verma, Professor & HOD and Dr. Rohit Kumar**, **Assistant Professor** in Department of Electronics & Communication Engineering, Delhi Technological University, Delhi for their constant support and continuous guidance throughout course of my work. Their sincerity, and perseverance has been a constant source ofinspiration for me in this work.

I would also like to thank all other faculty members of Electronics and Communication Engineering Department, DTU for their valuable suggestions and co-operation at every step in this project. I would like to thank you my parents and batchmates for their motivation throughout the work. Last but not the least I like to thank Almighty god for invisible presence and constant support.

Date: Vanshika Kumawat

Place: Delhi (2K21/VLS/22)

TABLE OF CONTENT

v

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In electronics hardware that comprises a processor uses either architecture of RISC. It contains instructions that vary from more superficial to elaborate and take a shorter execution time. It has a shorter length of codes, so it doesn't require larger memory space for storing. RISC (Reduced Instruction Set Computing Architecture) processors have an elementary addressing mode, so the fixed-in length instructions are simpler to implement and have a very customized set. The execution speed of the RISC processors is much higher due to their limited and simplified instruction sets . Instructions in the RISC has a fetching, decoding, and execution unit. RISC architecture comprises main memory supported by two small cache memory (instruction cache and data cache). It has a hardwired control unit to control the processor's components with the data and address bus. The instruction set architecture (ISA) is core technology on which chip is built, and chip is cornerstone of the information industry. x86 and ARM ISAs gradually gained market share and taken the lead inthe growth of chip industryover the past decades. However, x86 and ARM has expensive licence feesand are not open source. Due to its advantages of no license charge, flexibility, scalability, RISC-V anopen source ISA that expanded rapidly since inception. Over time, it has become one of the main forces behind global innovation of chips. For instance, European country started Processor Initiativewith the goal of creating high-performance CPUs based on RISC-V architecture. The Performance ofRISC architecture depends and varies according to the code being executed because any instruction that is executed presently may relyon previous instructions.

RISC is a microprocessor that improves performance by parallelizing the instructions. That means the speed of the machine is much more or faster by increasing a million instructions per second, or the known name is MIPS. If a million instructions per second are much more, the system's performance is very high. It doesn't imply that the processor is fast if a million instructions per second are high. So a million instructions per second are not the sole variable that decides the processor's speed, and it is not a good practice to use this as a parameter to distinguish between processors. Many known processors use this RISC architecture to make the system work fast. Some improvements can be made apart from improving a million instructions per second.

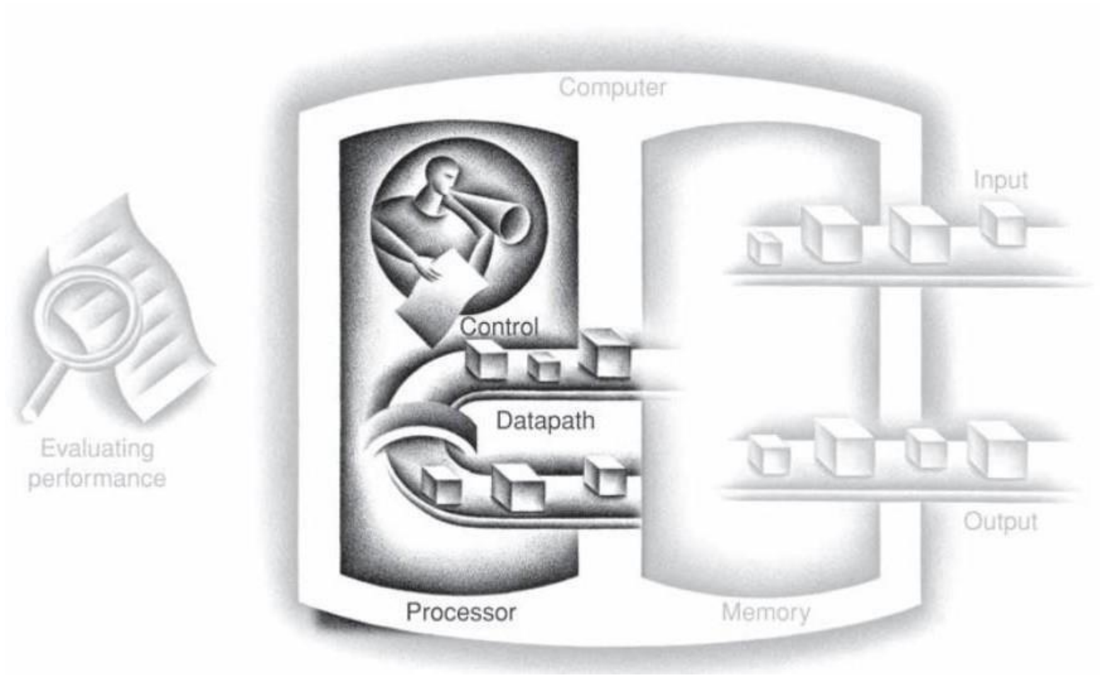


Figure 1.1: Components of computer [1]

CHAPTER 2

LITERATURE REVIEW

The term "reduced instruction set computer" (RISC) refers to a microprocessor design philosophy that favours a more manageable and straightforward set of instructions that all execute fairly quickly. Here we construct datapath of RISCV Instruction set[1]. Researchers from the University of California, Berkeley, created an open ISA based on Reduced Instruction Set Computer (RISC) principles to address the requirements of backward compatibility of previous extensions[2] and Instruction Set Architectures (ISA)s. This architecture, known as RISC-V, aims to help software engineers with a compact and reliable ISA by offering a base ISA and optional application-specific extensions [3]. Using pipelined stages to execute instructions in parallel will increase the processor's total throughput With help of clock we ensure stages are inline properly and used to ensure even if variable delays come our processor should be fully functional and data access performed smoothly[4]. Based on the pipeline schedule, pipeline registers are then inserted between the functional units and produced instruction set to make it work [5]. The load-store concept uses a separate unit that is tailored to handling relatively simple types of data, as opposed to the CPU supporting a range of addressing patterns[6].As there are different instruction with binary code for identification that we got to know from the format of that particular instruction and afterwards our processor process it [7]. After being fetched from memory and passed through pipeline stages, it is then passed on to the decoder unit, execution unit, register unit. The outcome will then be stored by outcome to destination unit[8]. This is the main characteristic, some people have started to call RISC processors "load-store" processors. a crucial element in all such designs and developing techniques for removing hazards and providing improved performance.

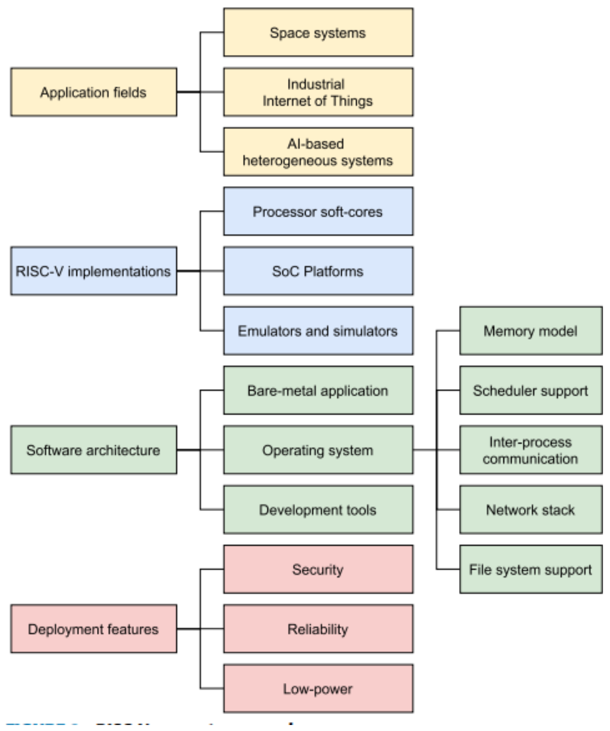


Figure 2.1: System Overview [3]

A comprehensive study on hazard and thus developing technique of Forwarding is quite helpful as we get data in intermediate stages[9].when a pipeline of subsequent instructions contains data dependencies and the succeeding instruction does not receive the desired value. Since the general purpose register set has not yet been updated and the value required by the following instruction is being evaluated in the preceding instruction[10]. Five Stages Pipelining is adopted for the purpose of executing processor working. Branch instructions, memory-reference instructions, and integer arithmetic-logical instructions[11]. Thus, Whole design of RISCV has been developed along with different units in the CPU Core[13]and thus synthesized it whole together handles interconnections between CPU Core and Peripherals, We developed various modules in processor through high description language[14] Utilizing information particular to the instruction set, datapath synthesis populates the combinational functional units that have been pre-defined.

CHAPTER 3

RISC V ARCHITECTURE

The RISC architecture uses a technique called the pipeline to improve processor speed by slightly reducing execution time by retrieving, decoding, and executing instructions in parallel without changing the logic of the instructions code, leading to a lower CPI (number of clocks per instruction). The combinational circuit follows each segment of the pipeline design, which is made up of flip flops that store or hold the data. When the flip-flop is triggered by the clock, the flip-flops send the data to the combinational circuit to conduct operations. When the processor is able to separate the processes of one stage from another and allow for the independent operation of another stage, it is done at each pipeline level. Consequently, it enables the simultaneous execution of several instructions.

3.1 BUILDING DATAPATH: -

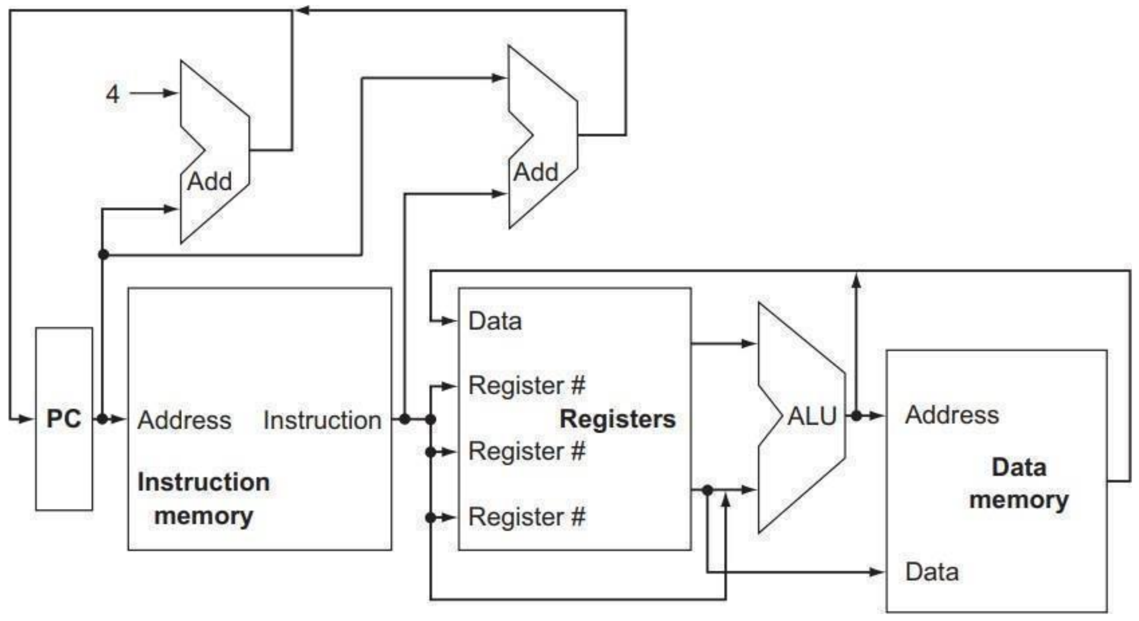


Figure3.1: Abstract view of RISC-V Implementation [1]

Each instruction begins from sending address of instruction to memories using programs counter. The fields included an instruction specifying operand registers that would be in use after instruction has been fetched. Once the register operand been fetched, they will be used to perform an equality check, compute arithmetic result of integer arithmetics logical instruction , or computememory address (for load or storing) (for branch).

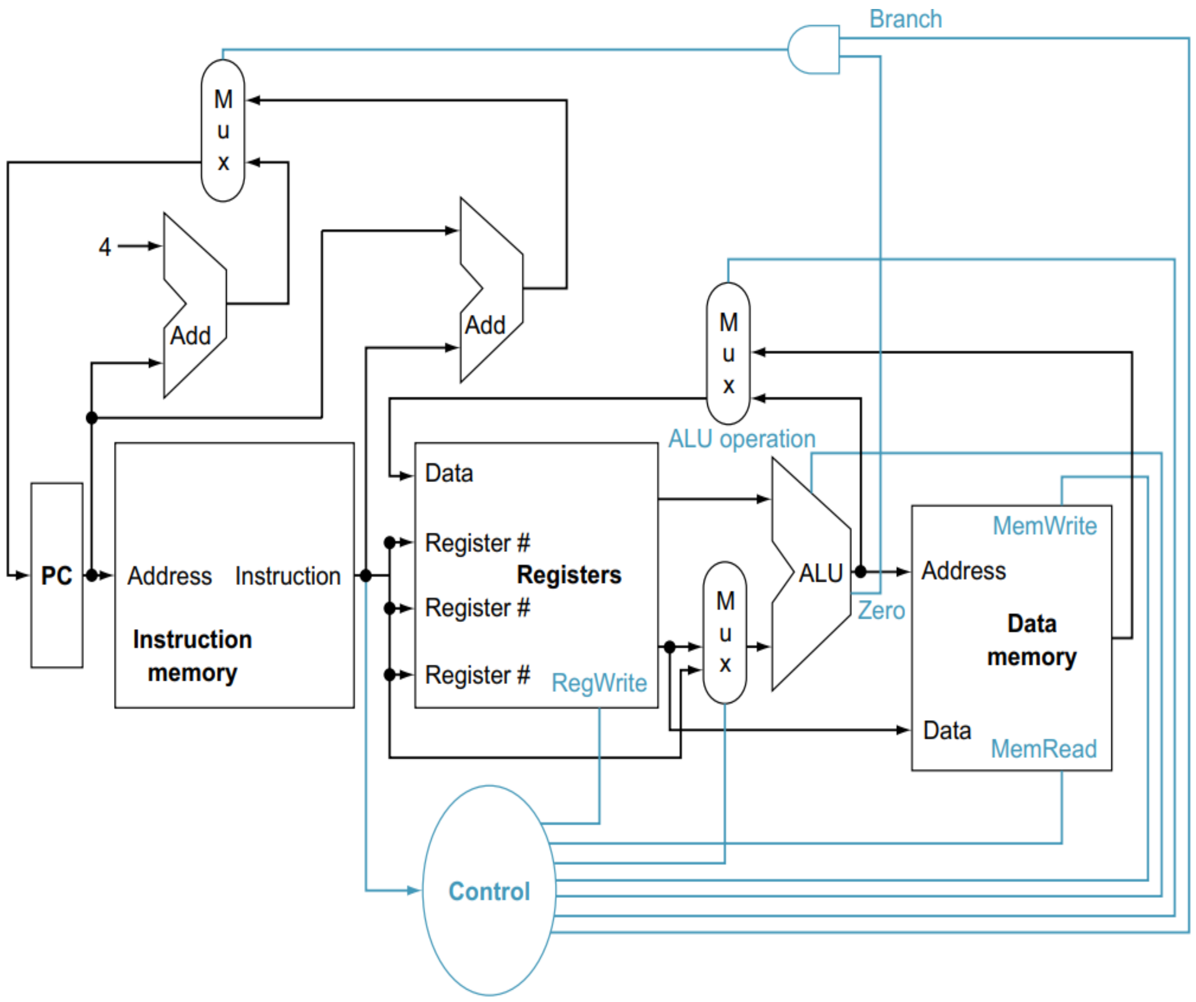


Figure 3.2: Execution including mux and control lines [1]

Every instruction starts by using a programme counter to send the instruction address proximity to instruction memory. The register operands that we will be using once instruction is fetched are specified in an instruction's fields. Post fetching register operands , they possibly be usedfor computing an integer arithmetic-logical instruction's arithmetic result, execute an equality check, compute address related to memory (for loading and storing) (for a branch).

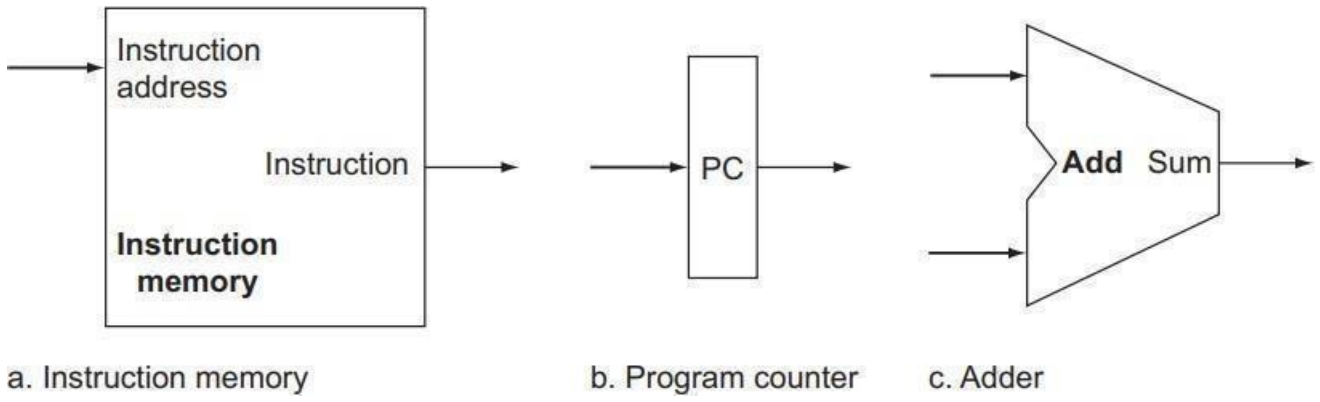


Figure 3.3: Elements need tostoreand retrieve instructions [1]

The first thing we require is a memory unit that can store programme instructions and deliver instructions in response to an address. also displays the PC, or counter, a register contains the address of current instructions. Last but not least, we require an adder for raising the PC to address the following instruction. The combinational ladder would be constructed using the ALU. The 32 registers general purposes of processor within building known as register file. We can write any register or read byspecifying its numbers in a register files, which is the collection of register.

from registers file and enter data word by one per instruction into register file. We needed an input for register file which specify register to read from it and will convey value that is read from registers for single data words to be read from registers.To write a data word, you need two inputs: the data to write inside the register, and the register number to write it.

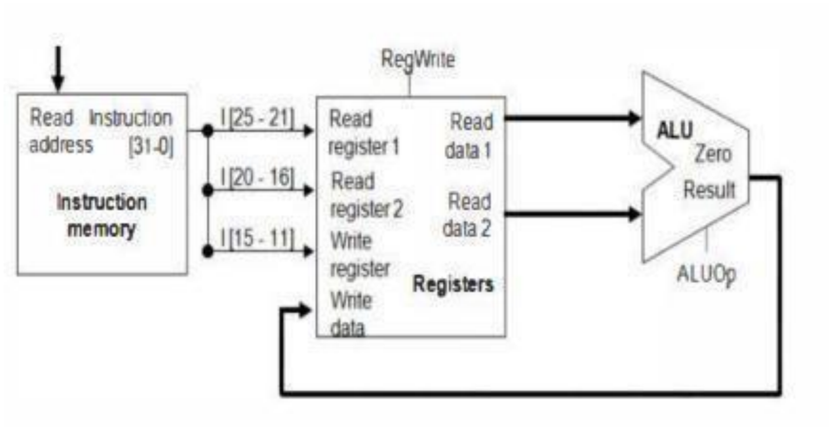


Figure 3.4: Features implementing R- Format ALU Operations [7]

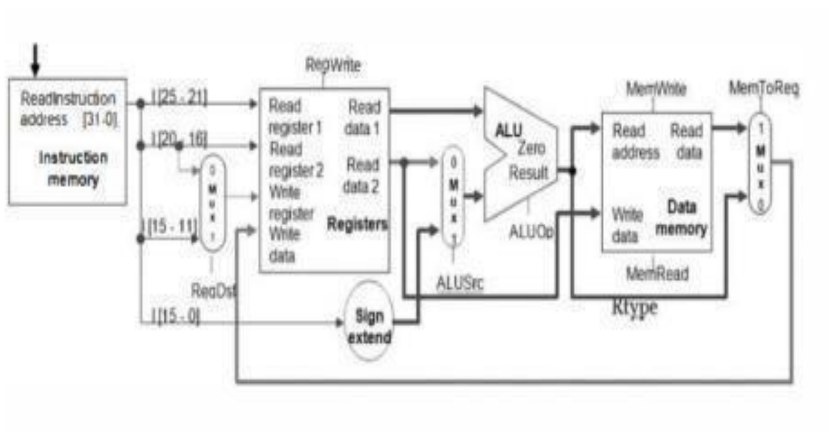


Figure 3.5: Data Path for I-Type Instructions [7]

Arising need of three inputs (double for register number and single for the data) along two outputs . The number of inputs of registers are 5 bit wide to tell one among 32 registers , wherein input at data with two output ofdata bus will be 32 bit wide. Take into account the general structure of the RISC-V load teamup with store registre value, which are x1, offsets (x2). These instructions add base register to 12-bit signed offset field to get the memory address. The value would be stored, if the instruction is a store, must also read from reg files where it is located in register x1. Thevalues reading from memory should written in the register file to designated register. So, the register file and ALU from figure 3.4 are both necessary. We must decide whether the instruction at target branch location or instructionthat follows sequentiallybefore we cancompute the address oftarget branch. The branch is said to be takenwhen the condition ismet, which iswhen two operands will be equal and the address of target becomes new PC. As with any other regular instruction. Whether the operand is not zero, increased PC should be in place of Current PC; Under these circumstance, we say that branch wont't be take. We can consolidate the datapathes elements required for each kind of instruction into a single path of data, then add control to finish our implementation. Theaimofthis straightforward datapath is to complete each instruction ina single clock.

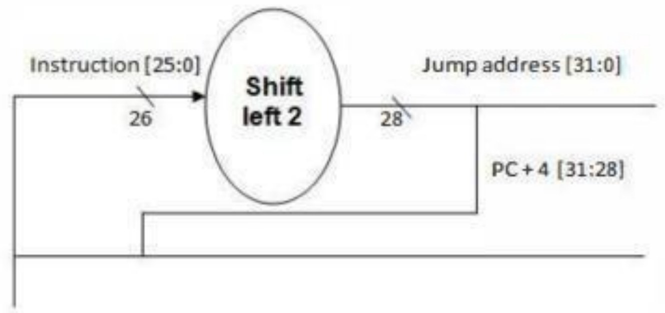


Figure.3.6 : Section for Branching [7]

Based on the ALU's Zero output, control logic determines whether the branch target or increased PC should take the placeof the PC.

We may need to permit severalconnections to an element's input in order to share element of datapaths between two separateclasses of instruction. We would then use a multiplexers and signal of control to select one of the numerous inputs.We needto enable two separate sources for the second Arithmetic unit of logical input along with two different source forthe information stored in the File of registers in order to build a datapath. As aresult, one plus one multiplexors are installed—one will be at data input and other at the ALU input.

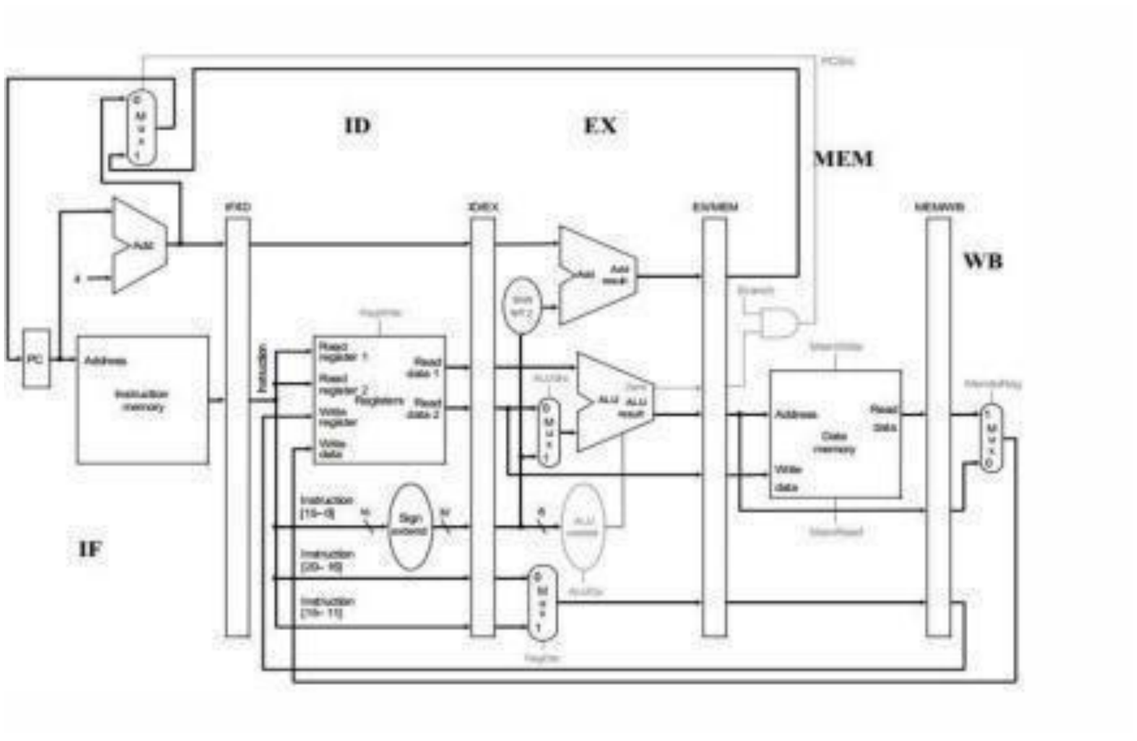


Figure3.7: Cumulative Datapaths [7]

The fundamental instructions (loading and store type register, ALU operations, and branching) can be completed by this datapaths in an individual clock cycle. To merge branches, needed one more multiplexor is required.

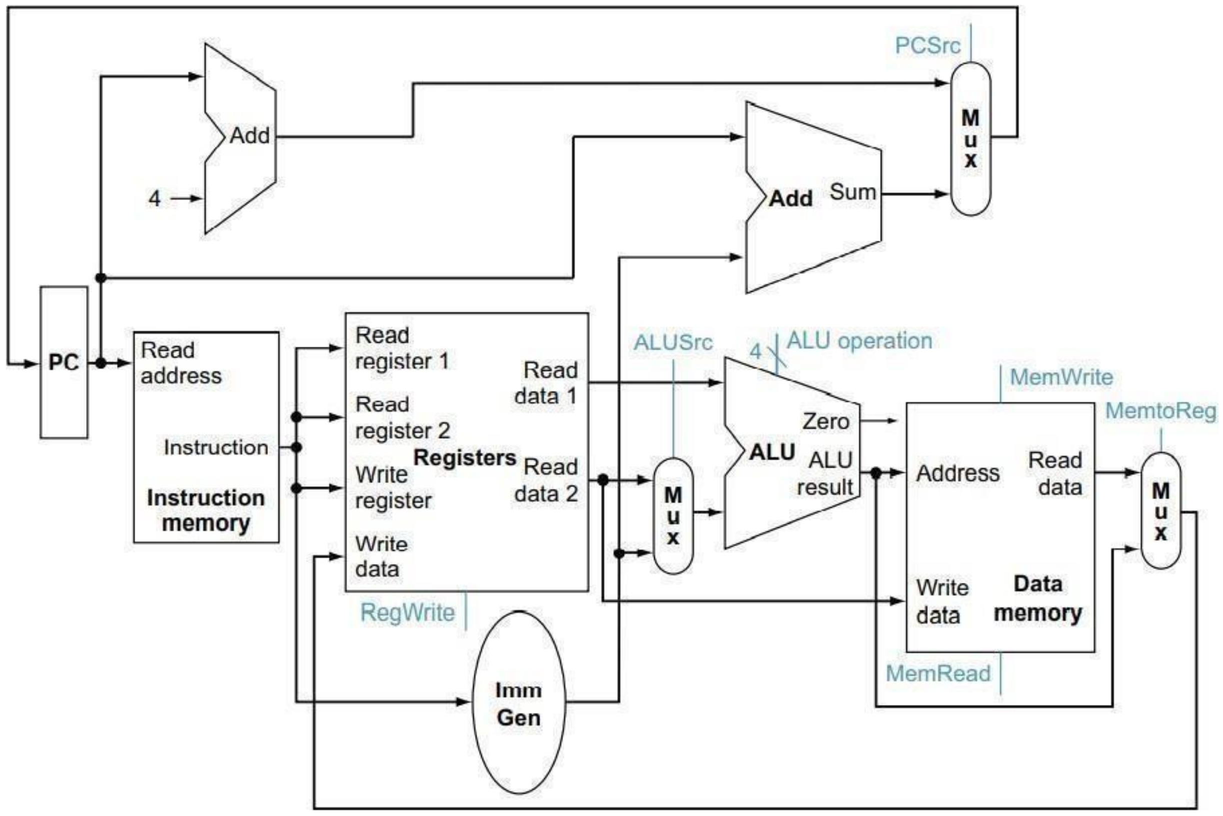


Figure3.8: The components needed by various instruction classes are combined in the straightforward datapath RISC-V cores architecture [1]

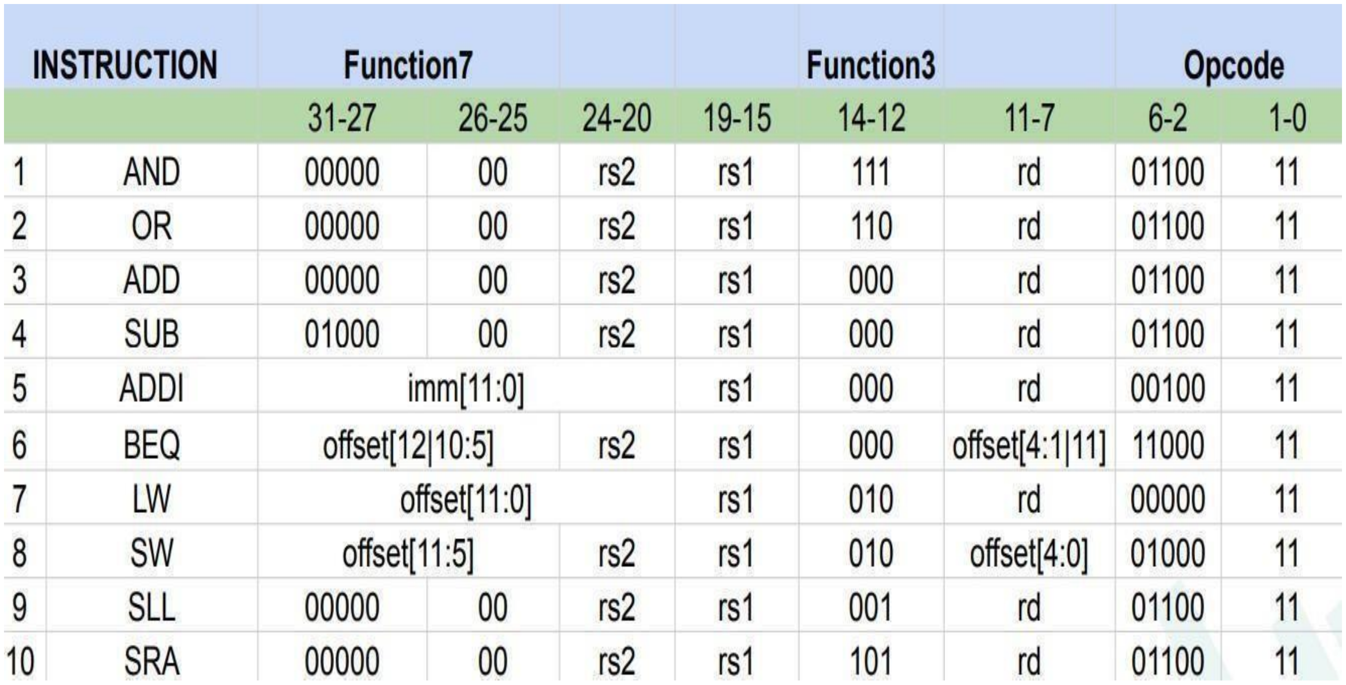
The datapaths into memory instructions and arithmetic-logical (or R-type) instructions both perform comparable tasks. The following are the main variations:

The ALU is used by the arithmetic-logical instructions, and the two registers serve as its inputs. Although the extended sign 12-bit offsets field from instruction serves as the second input, the memory instructions can still calculate addressesusing unit of arithmetic logic.

3.2 STAGES OF OPERATION:

Before going through stages of operation that a particular instruction goes through, we should aboutvarious instructions used in RISC V processor.

Table 3.1: Instruction sets



INSTRUCTION		Function7				Function3		Opcode	
		31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
1	AND	00000	00	rs2	rs1	111	rd	01100	11
2	OR	00000	00	rs2	rs1	110	rd	01100	11
3	ADD	00000	00	rs2	rs1	000	rd	01100	11
4	SUB	01000	00	rs2	rs1	000	rd	01100	11
5	ADDI	imm[11:0]			rs1	000	rd	00100	11
6	BEQ	offset[12|10:5]		rs2	rs1	000	offset[4:1|11]	11000	11
7	LW	offset[11:0]			rs1	010	rd	00000	11
8	SW	offset[11:5]		rs2	rs1	010	offset[4:0]	01000	11
9	SLL	00000	00	rs2	rs1	001	rd	01100	11
10	SRA	00000	00	rs2	rs1	101	rd	01100	11

Pipelined stages for parallel instruction processing will boost the processor's overall throughput but come with some functional hazards. Data risks are those dangers that are created as a result of sharing source and destination resources in later instructions when source for one instruction also the destination of an earlier instruction. The preventive method is by forwarding. Structural risks are created when programme and data memory often use. By structuring the queue within the processor, risks could eliminated.

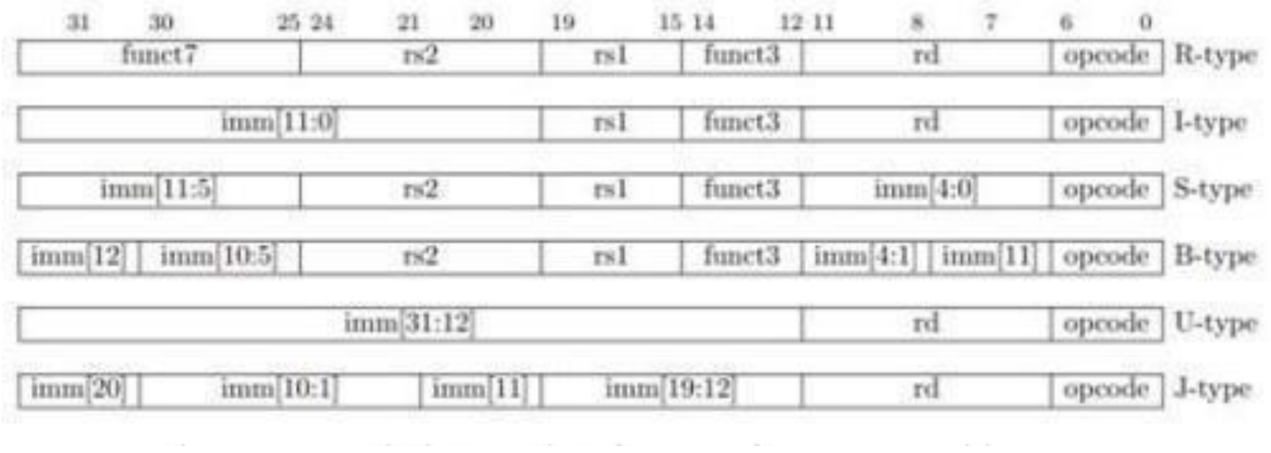

31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 | rs2 | rs1 | funct3 | rd | opcode | R-type
imm[11:0] | rs1 | funct3 | rd | opcode | I-type
imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | S-type
imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type
imm[31:12] | rd | opcode | U-type
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | J-type

Fig 3.9. Format of instructions [5]

3.2.1 <u>Instruction Fetch</u>

Instruction from memory is retrieved from the location where the PC points. If there is a branch instruction, store the current PC to another NPC and increment the program counter value. The staging processor extracts the instructions from the instruction memory in the fetch. And after pulling instructions, the program counter increment byfour asthe processor is 32 bits.

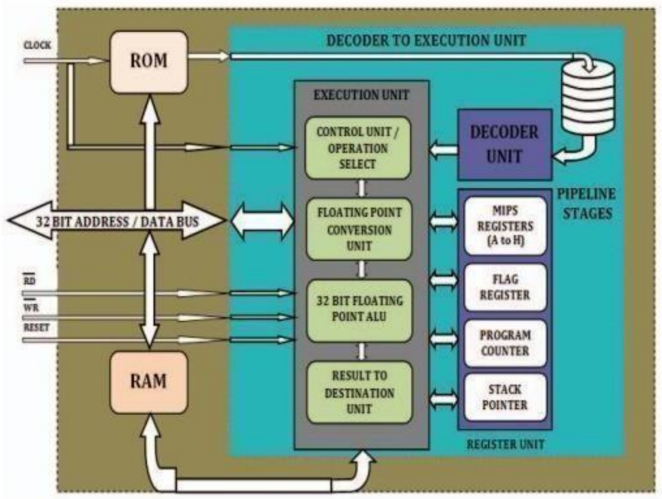

CLOCK
ROM
DECODER TO EXECUTION UNIT
EXECUTION UNIT
CONTROL UNIT / OPERATION SELECT
DECODER UNIT
PIPELINE STAGES
32 BIT ADDRESS / DATA BUS
FLOATING POINT CONVERSION UNIT
MIPS REGISTERS (A to H)
FLAG REGISTER
RD
WR
RESET
32 BIT FLOATING POINT ALU
PROGRAM COUNTER
RAM
RESULT TO DESTINATION UNIT
STACK POINTER
REGISTER UNIT

Fig 3.10 :Internal Architecture [8]

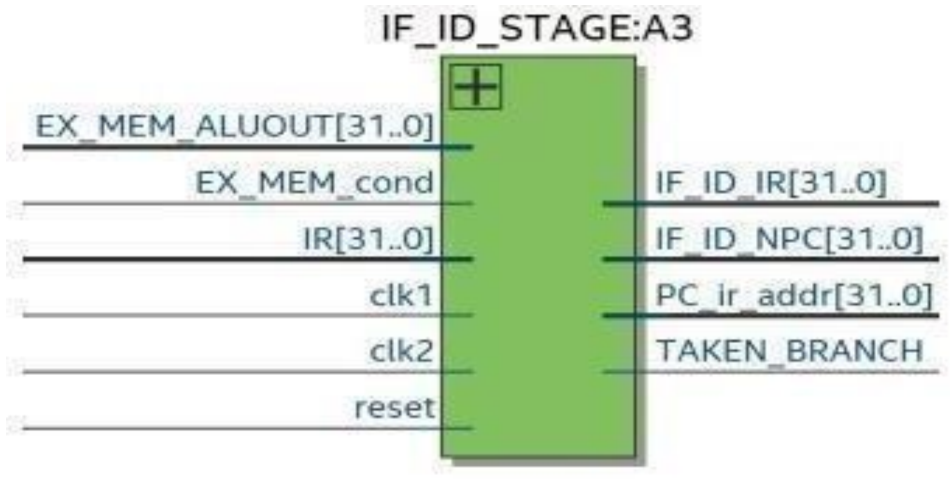


Figure3.11: Fetching and updating programcounter[4]

3.2.2 Instruction Decode

In this stage, instruction decode and register read were performed. Instruction fetched from IF is in the instruction register, which is decoded. According to the instruction format, reading register operands is done in parallel to decode. The values of load/store and arithmetic units in this stage might be skipped to utilize them in the following step. The instruction decoder decodes the instructions whichare fetched. It solves the destination register, source register, and associated paths. They are transmitted to the execution stage to execute.

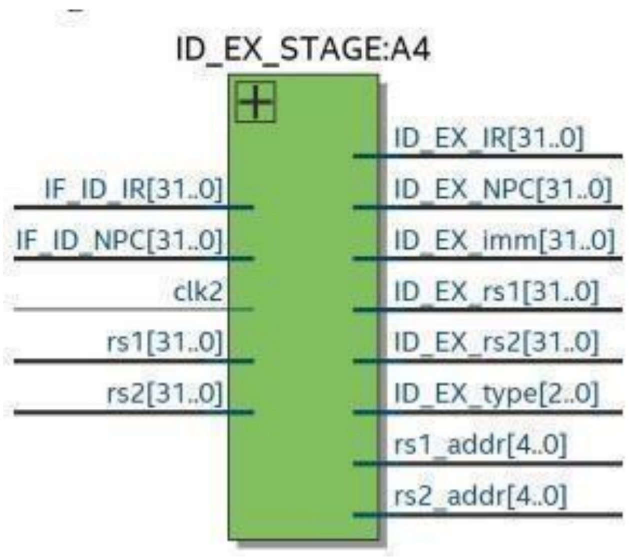


Figure3.12: Instruction Decoding [4]

3.2.3 Instruction Execute:

All instructions are executed in this stage. When required, an effective address for memory access is also calculated here. It is usually using an ALU here. The new program counter value is checked and updated for the branch instruction. The data decoded at the decode stage is received at the execution stage, and at this stage, the data processing takes place. At the execution stage, there are different modules such as arithmetic logic unit, adder, multiplier, etc.

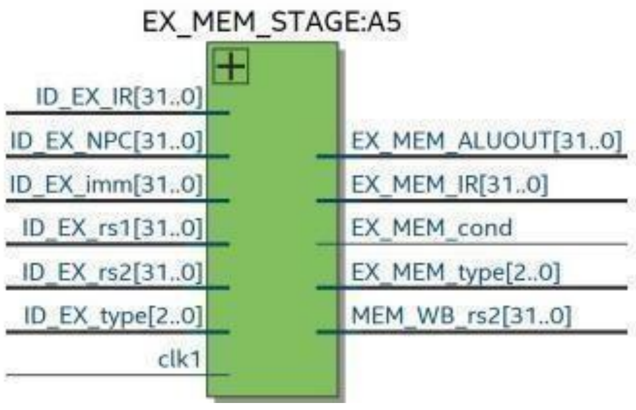


Figure3.13: Execution Stage[4]

3.2.4 <u>**Instruction Memory**</u>:

This step is not used for most of the instructions as it requires memory access, which is used by load/store instructions. For the Branch, the program counter has two addresses. When the flag is one, the program counter jumps that address. Otherwise, the following address is given for decoding.

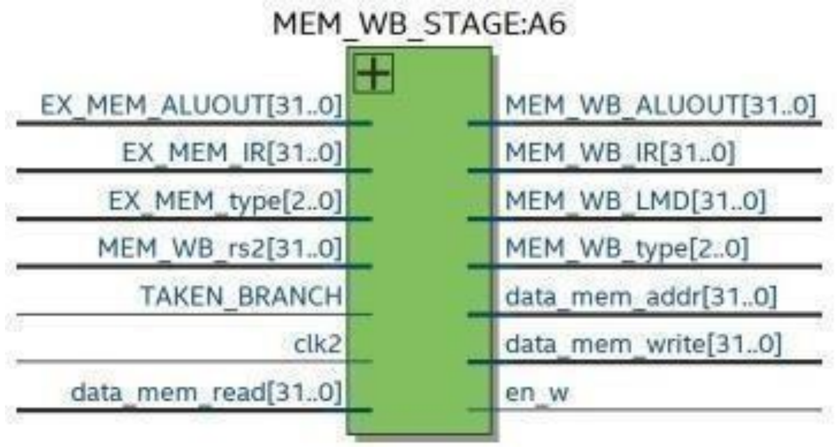


Fig.3.14: Access of Memory stage [4]

3.2.5 <u>**Instruction Writeback**</u>:

The output after this stage goes into the register file updating the value of the register in the Writeback stage. The position of the destination register depends on the instruction, which is known after the decoding has been done.

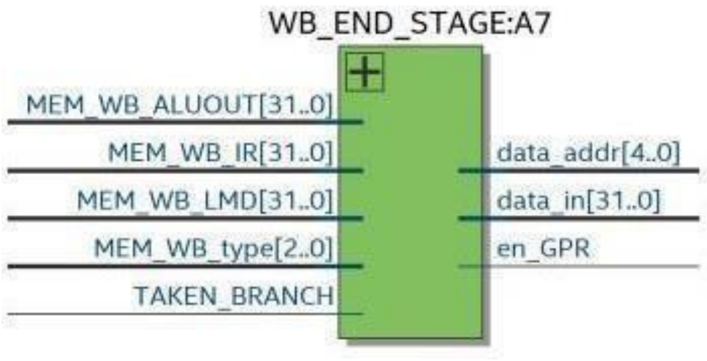


Fig. 3.15: Writes back stages [4]

The most important characteristic of a pipeline technique is that several computations can be in progress in distinct segments .The structure of a pipeline organization can be represented simply by including an input register for each segment followed by a combinational circuit. The pipelined stages used in this processor are fetched, decoded, executed, memory, and written back. Now, if the one instruction reaches the solve step after being brought, the other instruction is available and continues.

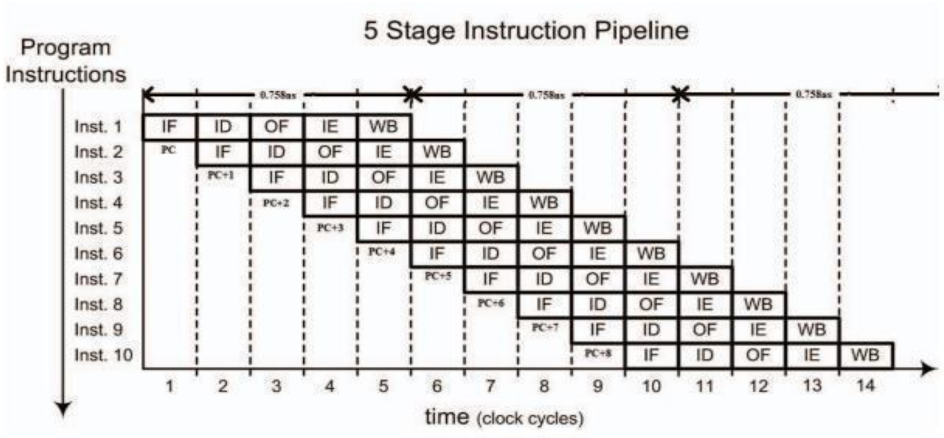


Figure 3.16: Enhanced stages of instruction set [8]

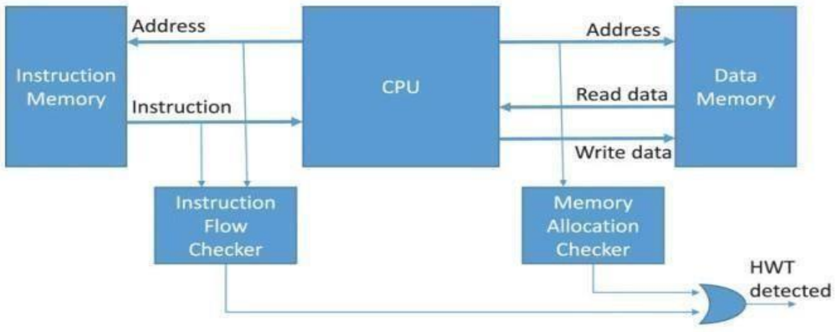


Figure.3.17: Interconnection of processor [3]

CHAPTER 4

INTRODUCTION ABOUT PIPELINING

Pipelining is a method of implementation where several instructions are executed simultaneously.

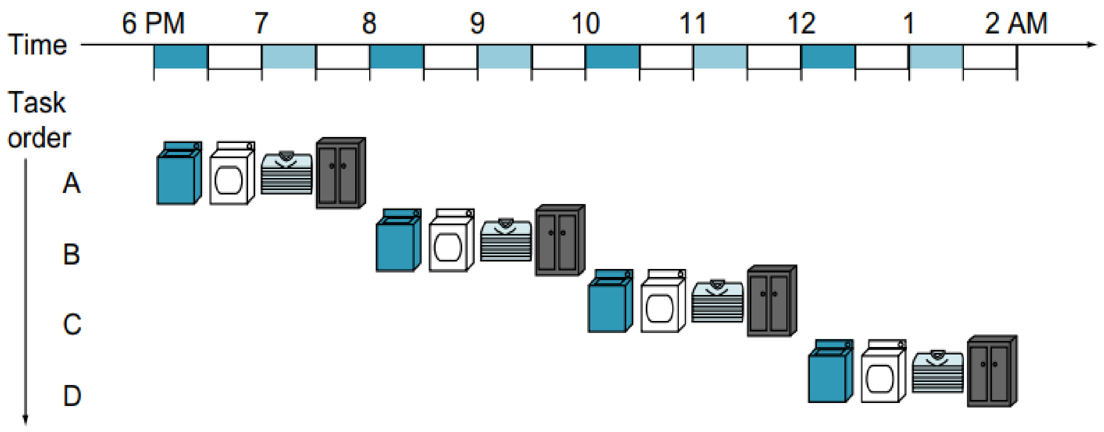
Pipelining is almost always used today.

Anyone who performed lot of laundry have employed pipelining instinctively. Laundry would notbe pipelined in this way, which would be as follows:

1. Fill the washer with one filthy load of laundry.
2. After the washer has completed its cycle, put the wet clothes into drier.
3. After the drier has completed, lay folded dried load out on table.
4. Once the garments have been folded, ask your partner to put them away.

Start a new with the next dirty load once this one is finished. Figure 4.1 illustrates how much faster the pipelined technique is. You put the second filthy load in the washer as soon first load is finished in the washer and put in to dryer. The load accompanied by wet clothes is moved to dryer, the first batch of cloth is placed ontable to begin fold process, and the second filthy cloth of load is placed in washer. The second load of cloth is folded, third is in the dryer, and the fourth is placed in the washing after having your roommate put first pack of cloth away. All steps, or stages as they are known in pipelining, are currentlytaking place simultaneously.

We can pipeline the jobs as long we having different resource for everystage. The pipelining states that while pipelining is faster for many loads since everything is operating in parallel, loads would be completed each hour, time taken by putting dirty cloth within washer till dry, folded.



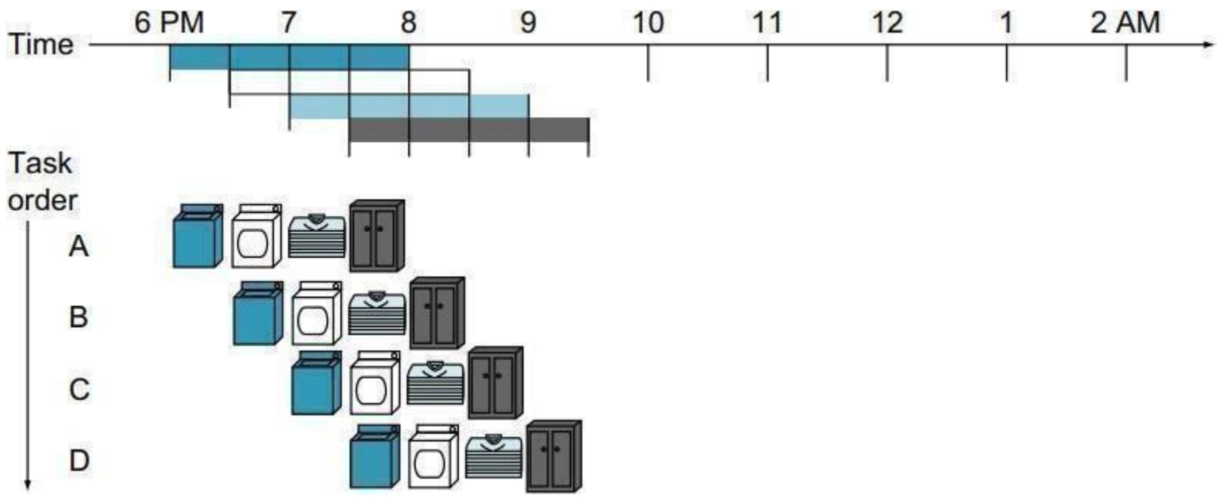


Figure 4.1: The laundry Example[1]

Our washing system's throughput is increased by pipelining. Thus, pipelining will not reduce time needed to doone pack of cloth related to laundry, but we are have several loads there to finish, the increase in throughputs reduces the overall time required finish operation.

Number of steps within pipelines if all stage take roughly the similar amount of the time and there is sufficient work to be done. There are four steps in this instance: washing, dry, folding, and keep away. Laundrythat is pipelined could be 4 time as quick as laundry that is not pipelined. While one load of laundrywould take aroundfive times as long as twenty, twenty loads of laundrywould take twentytimes as long. Only2.3 times as fast.

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

On processors where we pipeline instruction execution, the same rules apply. Traditionally, RISC-V instructions gothrough these steps:

First, get the instruction from memory.

Decode the instruction by reading

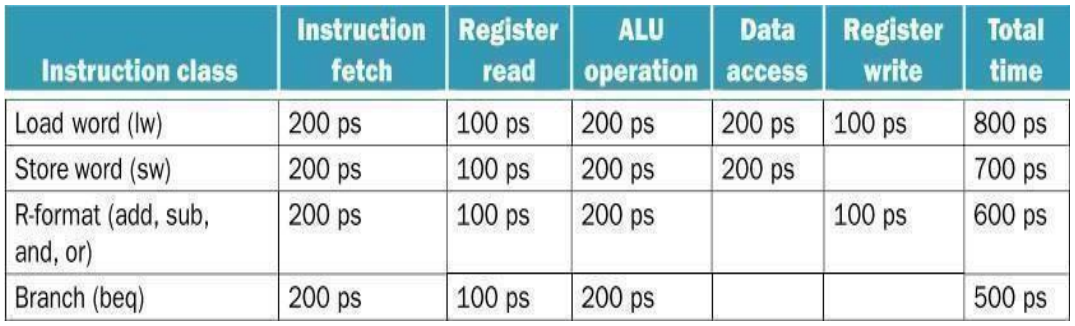
registers.

3. Carryout the operation or determine the address.

4. Use a data memoryoperand .

5. Enter the outcome in a register .

Table 4.1: Total time calculated fromindividual component [1]



Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

4.1 PIPELINED DATAPATH:

Taking a load instruction execution as an example (because it is active in all five stages). Here we have shownthe instruction "lw" , over the five stages that activates . The five stages are :

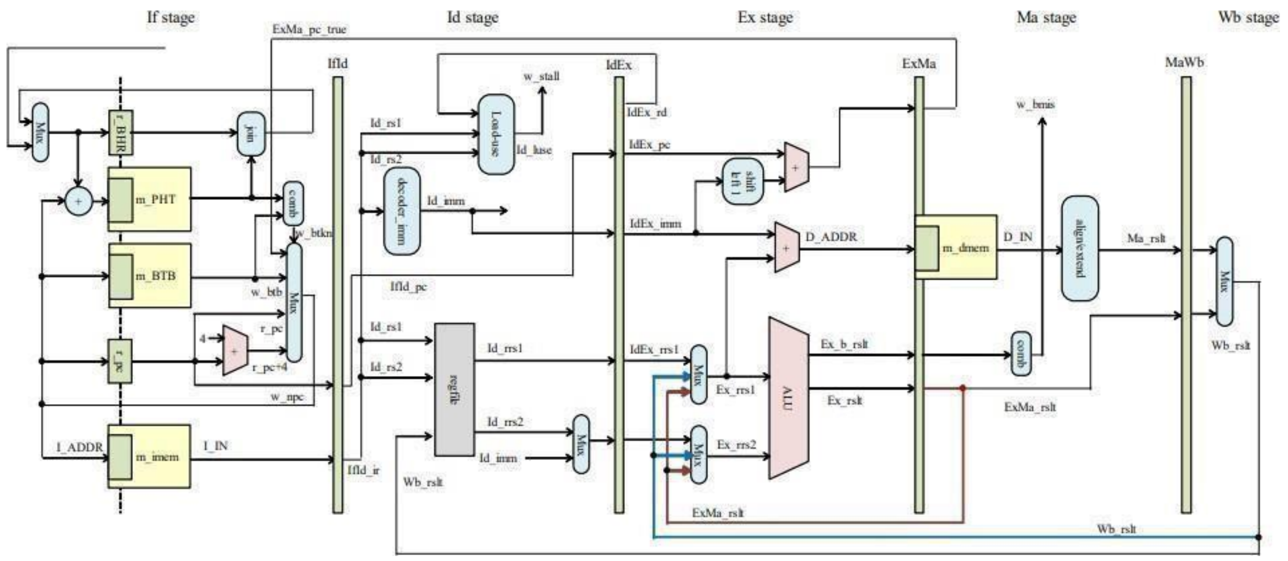


Figure.4.2: Layout of Datapath[11]

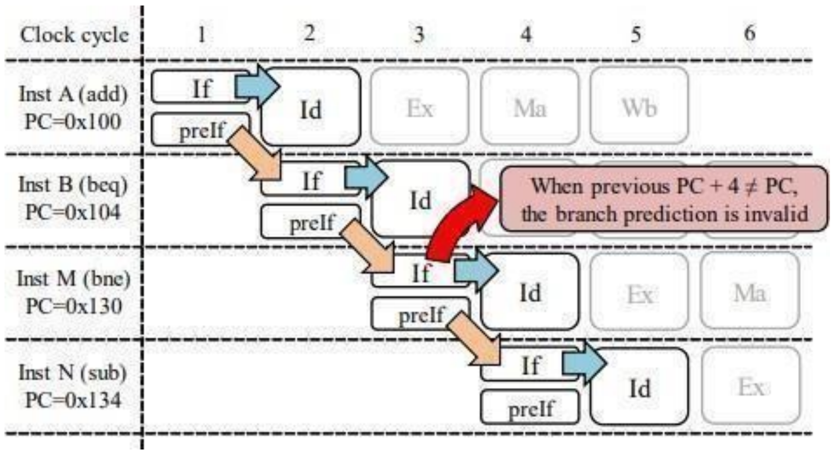


Figure.4.3: Execution Stages[11]

4.1.1 Instruction fetch:

This demonstrates how the instruction keep put in the IF/ID pipelining register afterread from memories using PC's addresses. In order to prepare for the subsequent clock cycle, the address of PC keep increased by4, then write it back into PC. In case it becomes necessaryat a later time for an instruction like beq, this PC also preserving in the IF/ID pipelines registers. Since the computer will not determining the type of instruction we are fetching, it should be be ready for anykind of instruction and pass any informations that maybe requiredalong the pipeline.

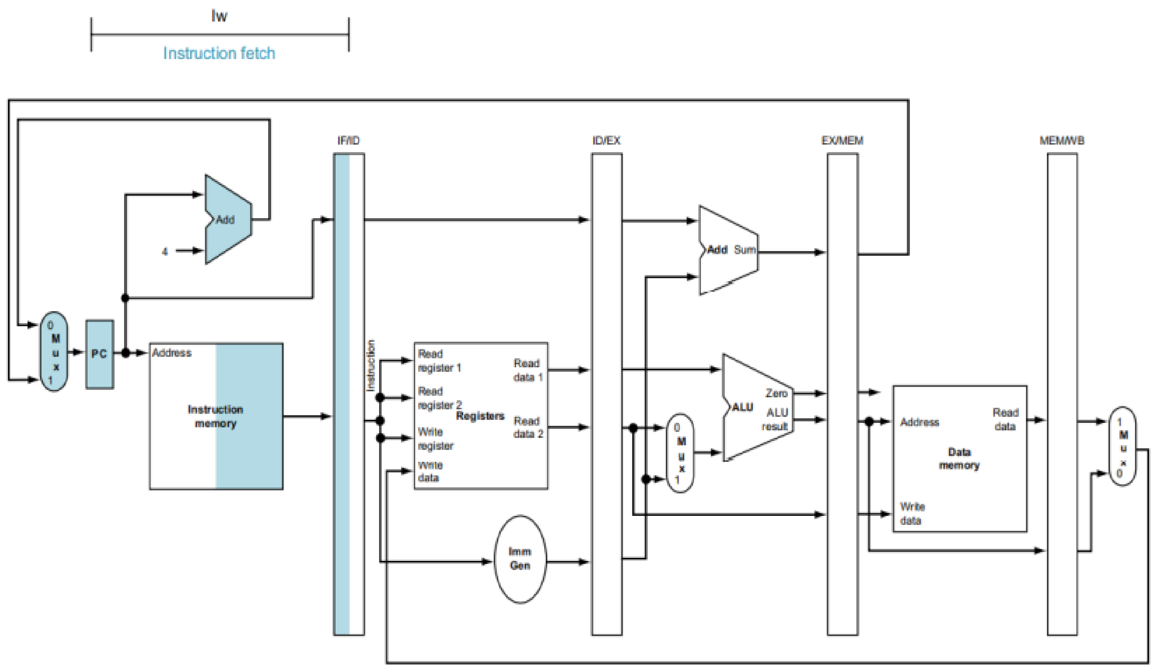


Figure 4.4: Instruction fetch/ First pipelining stages[1]

4.1.2 Instructions decoding and register files:

The fig.4.5 displays the register numbers to be read from two registers as well as their instruction section of IF ID register of pipeline supply field of immediate data 32 bits. Along with PC address, allvalue kept in the ID/EX pipelining register.We once more transfer allthat a future clock cycle's worthof instructions might require.

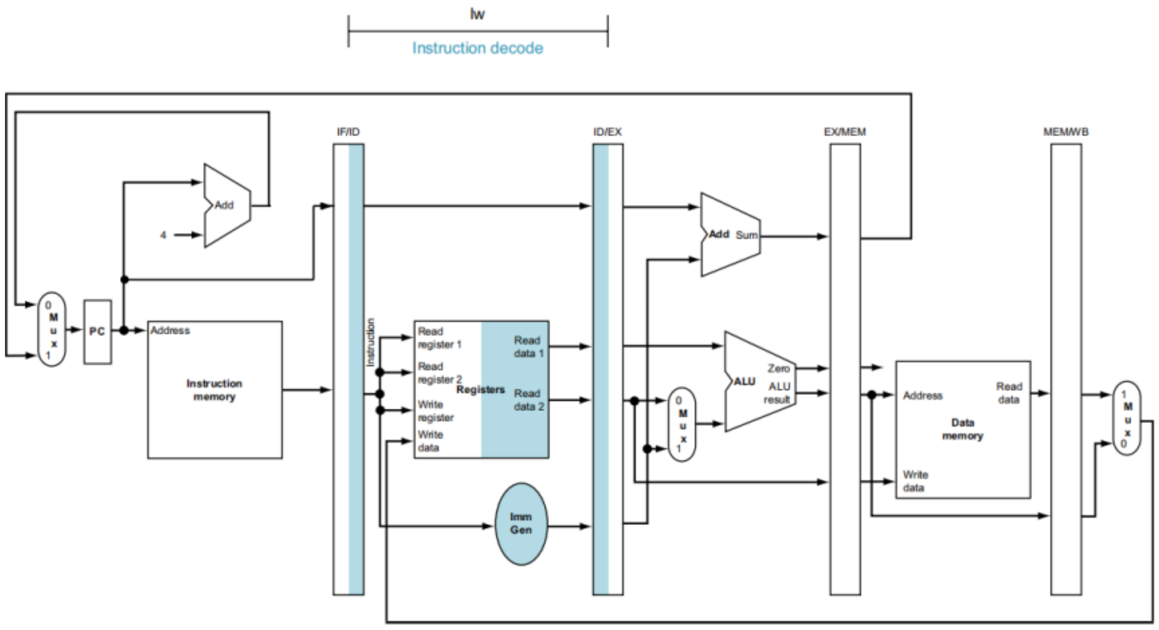


Figure 4.5: Instruction decode/ Second pipeline stage[1]

4.1.3 Execute or address calculation:

Fig 4.6 demonstrates how the load instruction uses the ALU to add the content of register and the sign-extension immediate from the ID/EX pipelining registers. A register for the EX/MEM pipeline contains

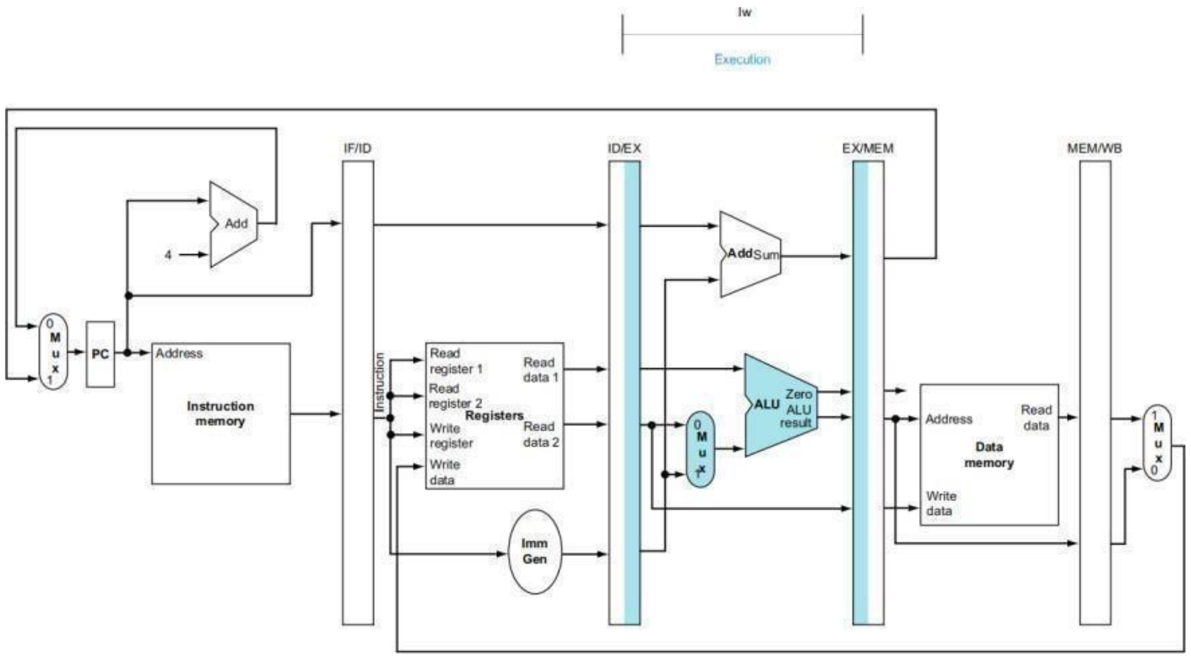


Fig 4.6 : Instruction Execution/ Third pipelinestage[1]

4.1.4 Memory access

The portion at top in Fig 4.7 demonstrates how the load instruction loads data into MEMWb register of pipelining by accessing memory owned by data using address from EXMem register for pipelining.

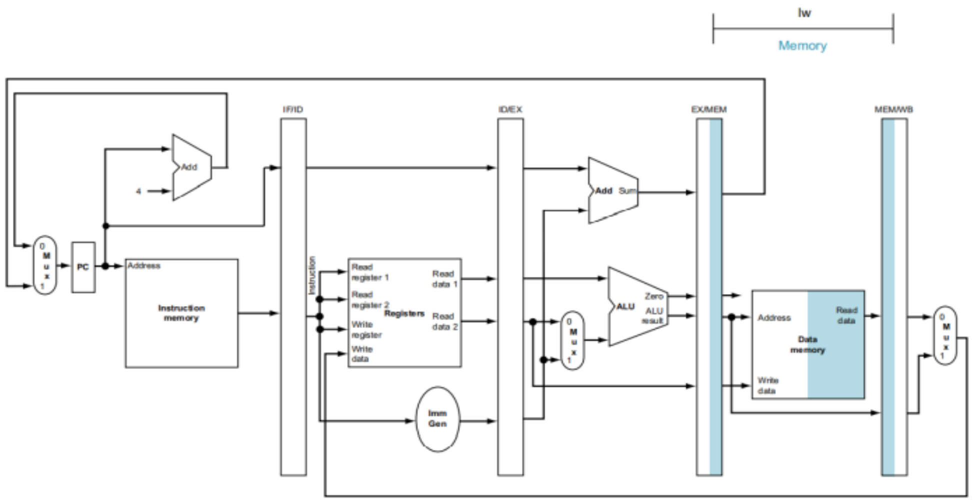


Figure 4.7 Memory access/ Forth pipeline stage[1]

4.1.5 Write-back : The Fig 4.8 depicts the last action: inserting the data in the registerfiles in the centre of figure after reading it fromMemWb pipelines register.

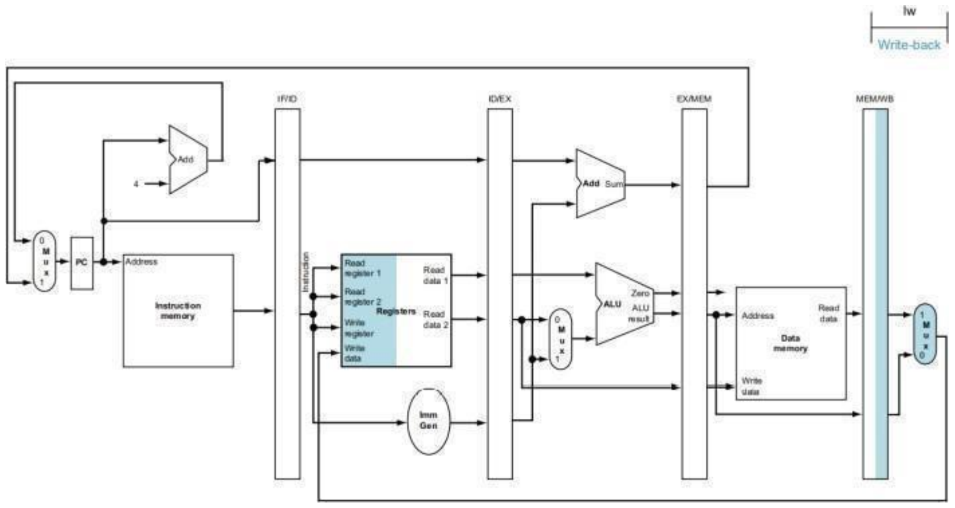


Fig 4.8: Write back/ Fifth pipeline stage[1]

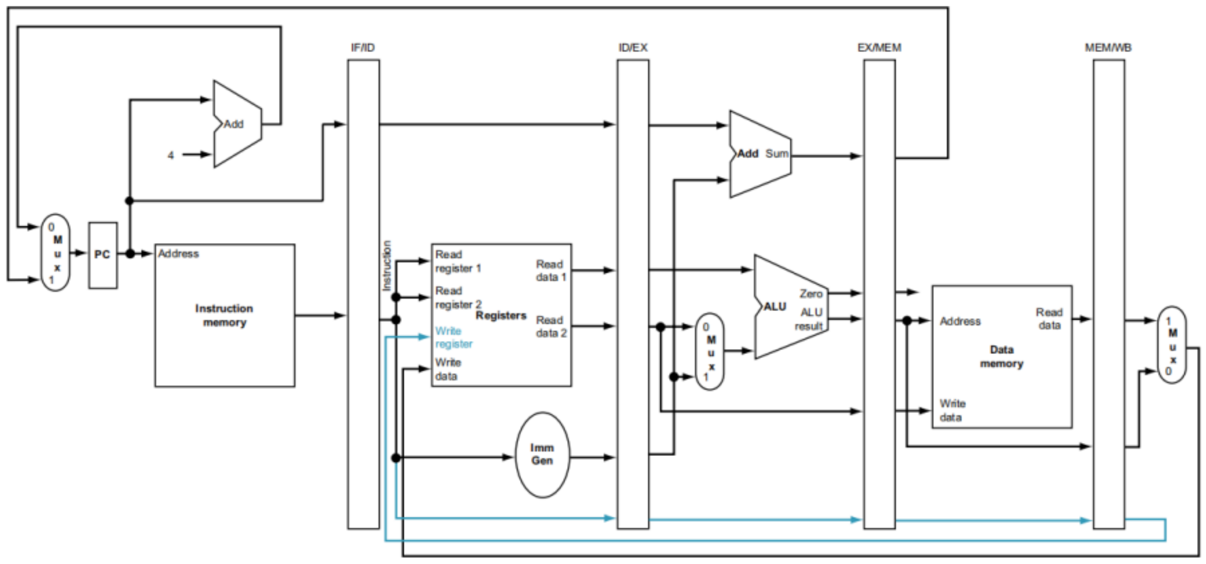


Fig 4.9: Corrected pipelining datapaths tohandle load instructions [1]

4.2 PIPELINED CONTROL:

We just need to set the control settings during each pipeline stage to specify controls for pipeline stage. We may categorise control lines into five groups based on the stage of pipelining because each line of control is linked to component that is only activate during one stageofthe pipeline.Instruction fetch: There is nothing unusual to regulate at this pipeline level because signals of control to read the instruction from memory and write it back to PC will always beasserted. Instruction decoding/register files read, Dual Source registers willalways be in exact location inside RISC-V formatting of instruction,

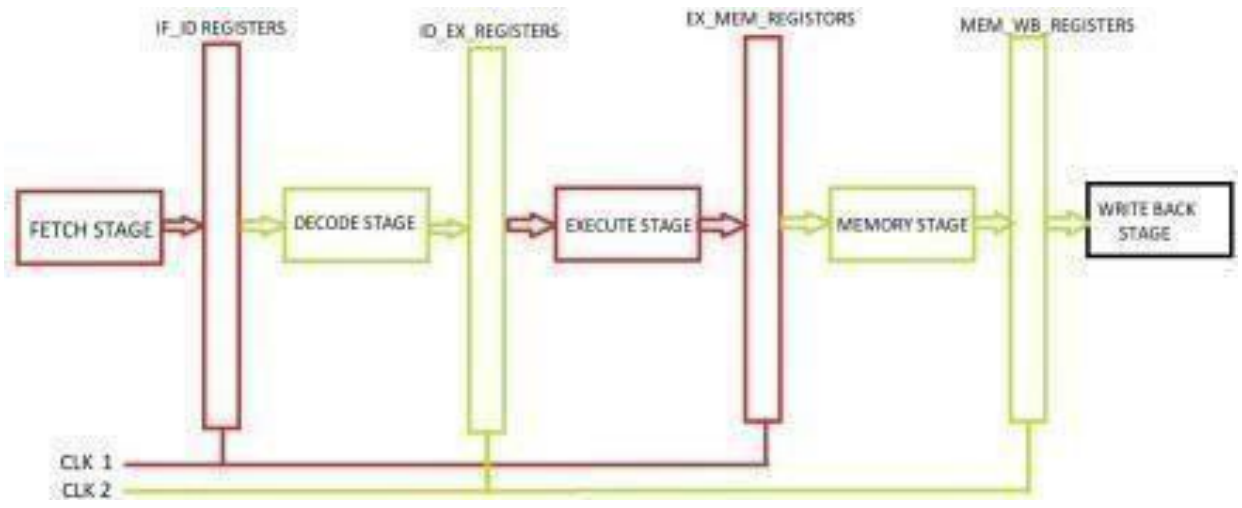


Fig 4.10: Controlling of stages [4]

Execution/calculation ofaddress: ALUOp and ALUSrc arethe signals that need to be set. The signals decide which ALU operationto use and which inputs to give the ALU—Read dataor a sign-extended immediately. Memory access: Branch, MemRead, and MemWrite are the control lines that have been set at this point. These signals will be activated by branch if equal, load, and storing instructions executing, respectively. Until control asserts Branch and Arithmetic unit result was 0, PCSrcchooses following sequential address. Write-back: Mem to Reg and Reg Write are the two control lines that determine whether ALU result or memory values .

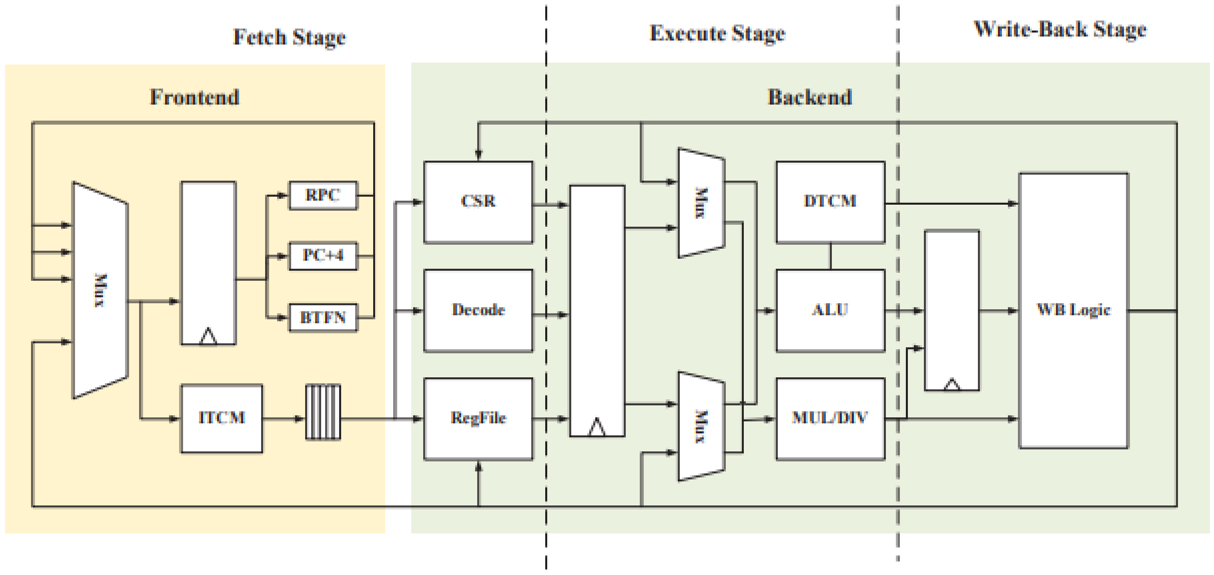


Fig 4.11: Process of Instruction format [5]

4.3 PIPELINE HAZARDS:

In pipelining, there are times when subsequent instructions cannot be executed in subsequent clock cycle.There are tridifferent categories ofthese occurrences, or dangers.

4.3.1 DATA HAZARDS :

The RISC-V instructions are sometime dependent i.e use the results calculated by any of the others. In such cases data hazards obstacle to pipelined executions. Let’s look at procedure along many dependencies:subs x2, x1, x3 // Register z2 written sub

AND x12, x2, x5 // 1st operand of (x2) depending the subOR

x13, x6, x2

add x14, x2, x2

SW X15, 100 (x2) // Base (X2)

The outputs of the first instruction in register x2 is a prerequisite for the last four instructions. It displays the value of register x2, which is updated when the subtraction instruction writing its result in middle of clockcycle 5. It demonstrates that reading values for the registers x not having obtained by instruction without the read. occurring at or after clock cycle of 5. Consequently, the instructions would providing proper value.

When the pipeline must stop while waiting for another step to finish, data dangers occur. Imagine discovering asock at the folding station that had no match. Chase down to room to verifying nearest match.

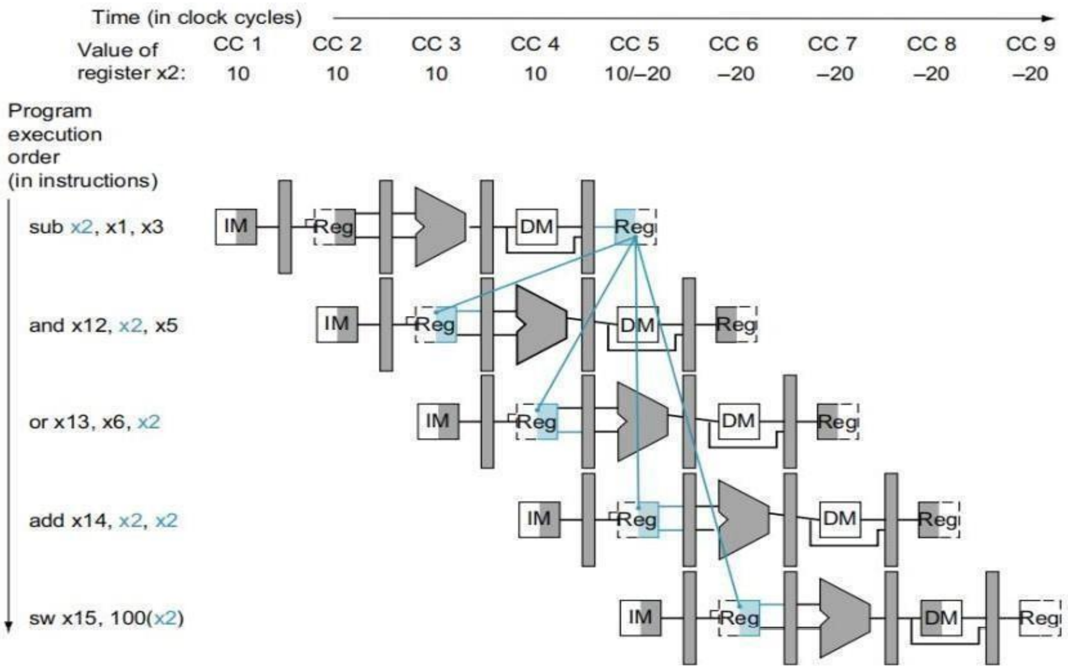


Fig 4.12: Pipelined dependencies in a five-instruction sequences using simplifying datapaths to show dependencies[9]

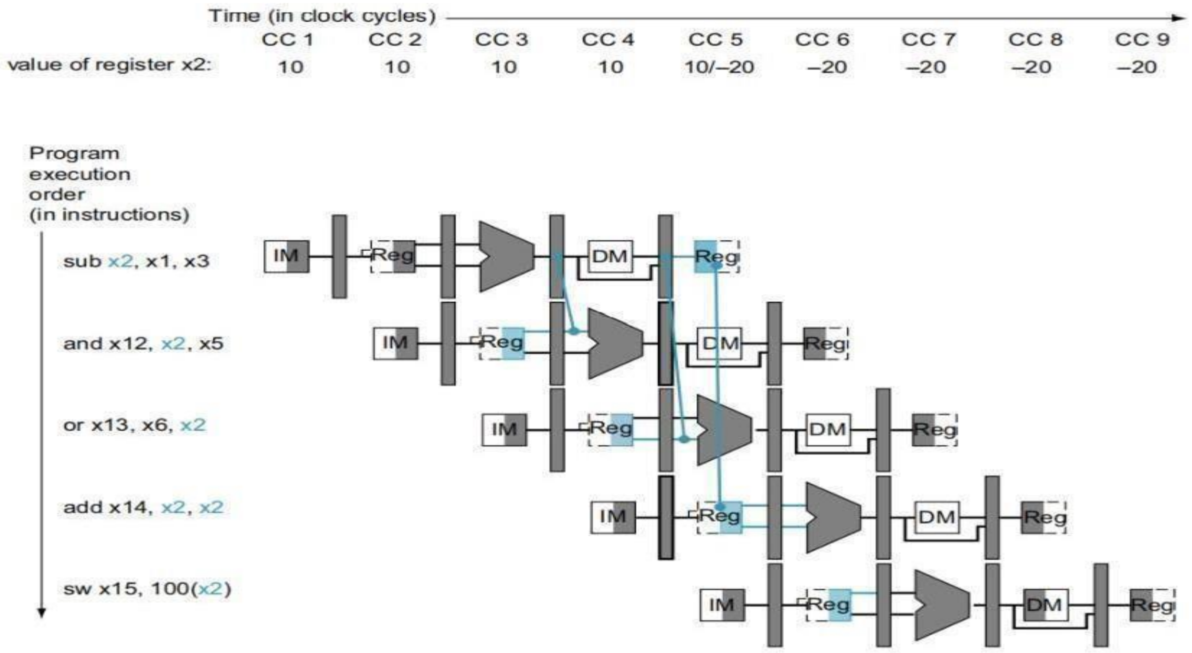


Fig 4.13: Dependency between pipelines registers and inputs to ALU [9]

Instead of waiting for writeback stage to write register file, reliance now starts from a pipeline register. The pipelining registers will hold the data that has transmitted, ensuring that the necessary data is available in timefor subsequent instructions. We can forward the right data if we can feed the inputs from any of given registerrather then simply ID/EX. I can execute the pipelining at full speed even in face of data dangers by attaching multiplexers to the ALU's input and using the appropriate controls.

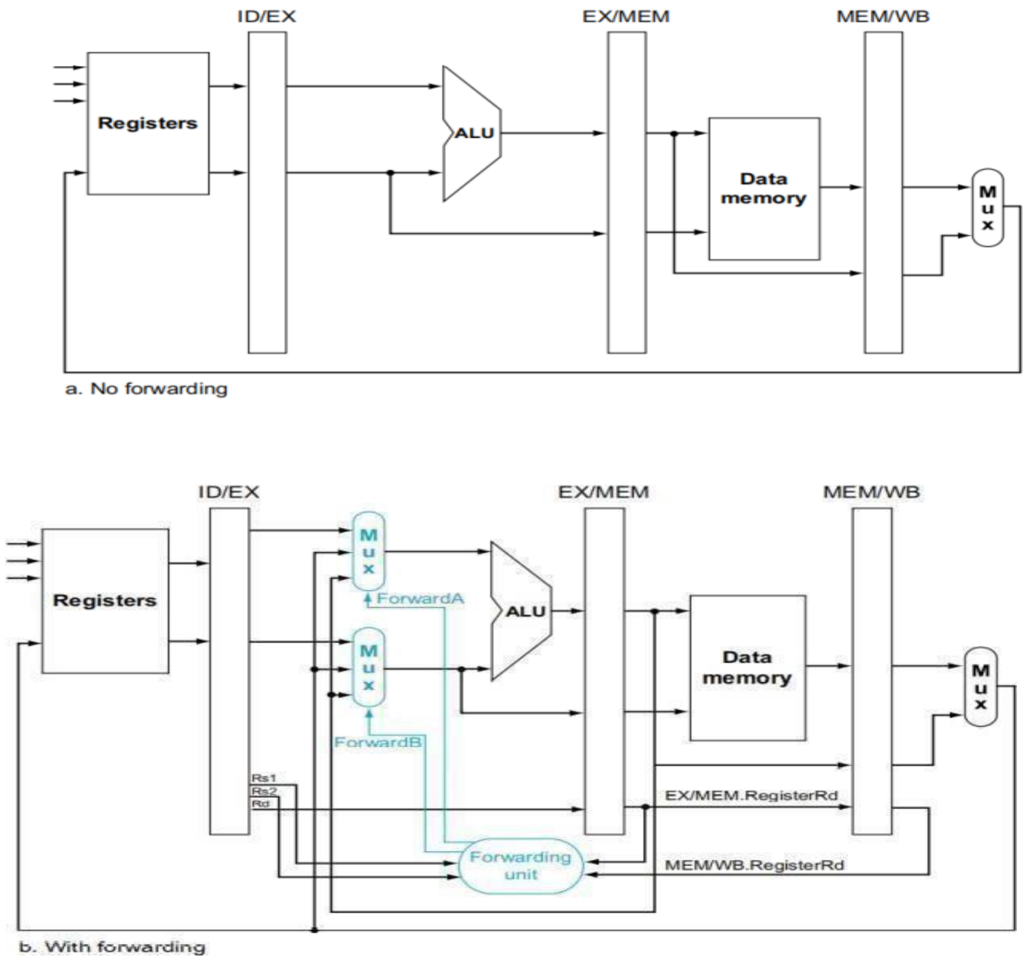
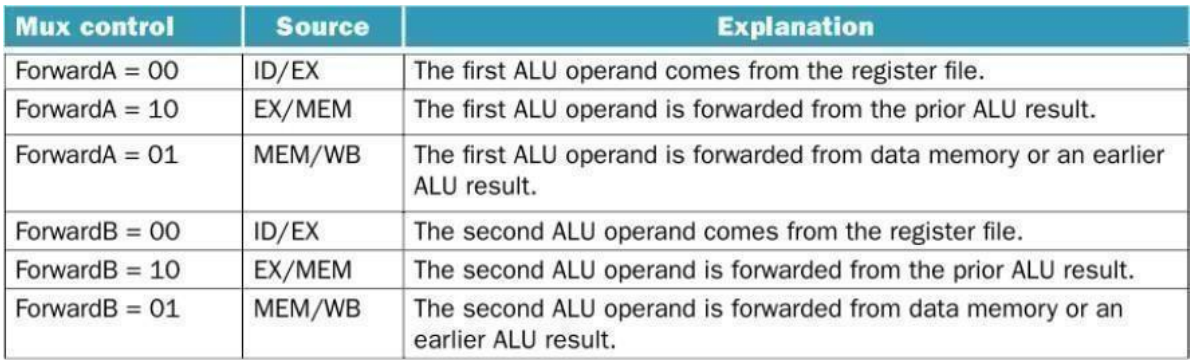


Fig 4.14: Forwarding [9]

Table 4.2: depicting running mechanism[1]



Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

4.3.2 Data Hazards and Stalls :

When instructions trying to reading register after load which consequently writing to sameregister, that is a situation when forwarding will not save your day. The issue is demonstrated in Fig.4.15**.** During clock cycle 4, the ALU is operating on the next instruction while we are reading data from memory. must cause something to halt the pipeline.

We therefore require detection unit for hazard in addition to forwarding .That depends upon it, it acts during the ID stage. The hazard detectingunit's control is this solitary requirement, which is to check for load instructions. The software technique is dependent on the compiler to rearrange the user code or add a delay slot to resolve the combinations of instructions that could generate data hazard, which highly based on how reliable the compiler technology is. Using software to address data hazards is a less sophisticated method. Rearranging the code could not be effective because it depends on how the programme flows. On the other hand, the processor throughput is unaffected by the hardware method based on the forwarding scheme.

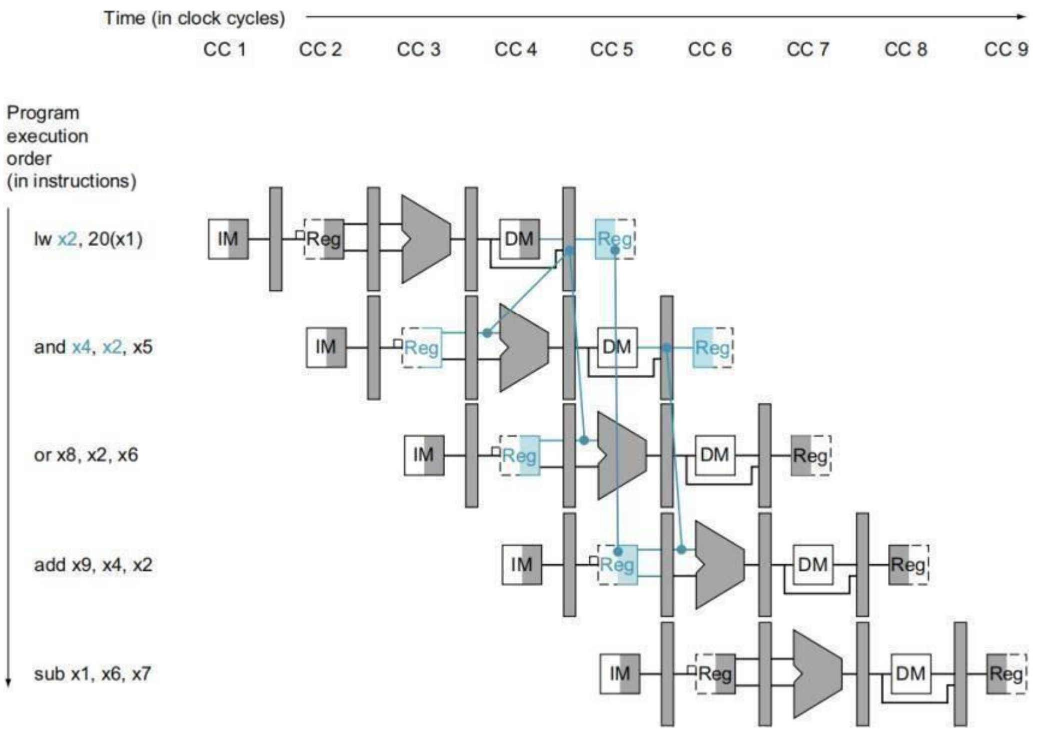


Fig 4.15: Sequence of instructions[9]

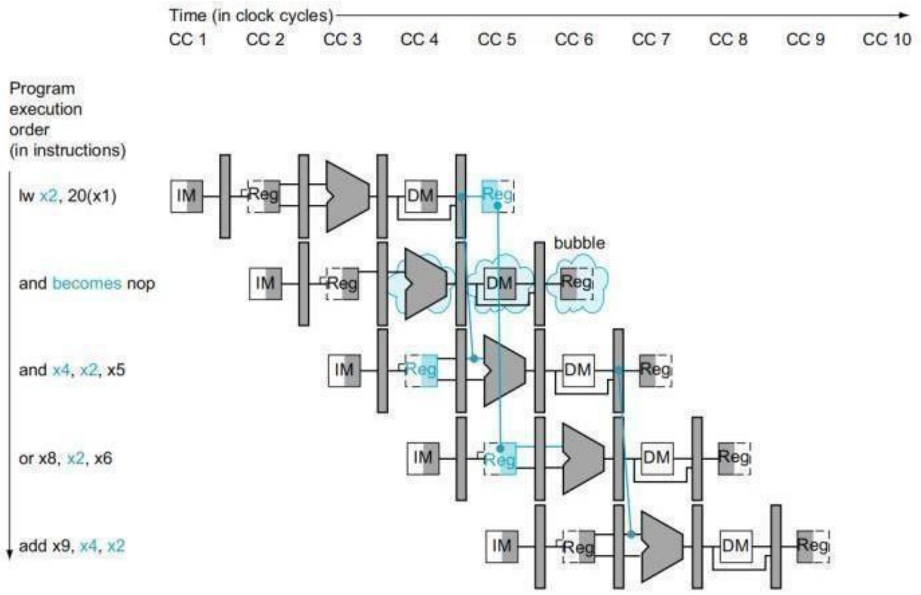
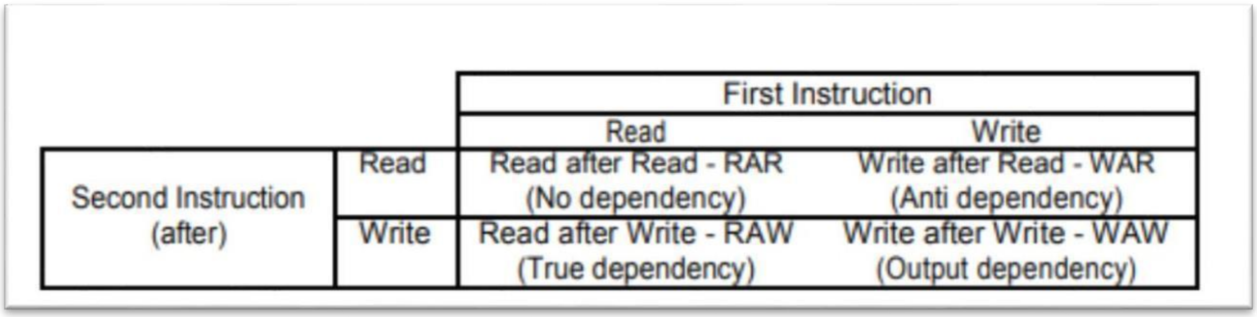


Fig 4.16: Stalls are inserted [9]

Table 4.3 : Data Dependency relationship[9]



		First Instruction	
		Read	Write
Second Instruction (after)	Read	Read after Read - RAR (No dependency)	Write after Read - WAR (Anti dependency)
	Write	Read after Write - RAW (True dependency)	Write after Write - WAW (Output dependency)

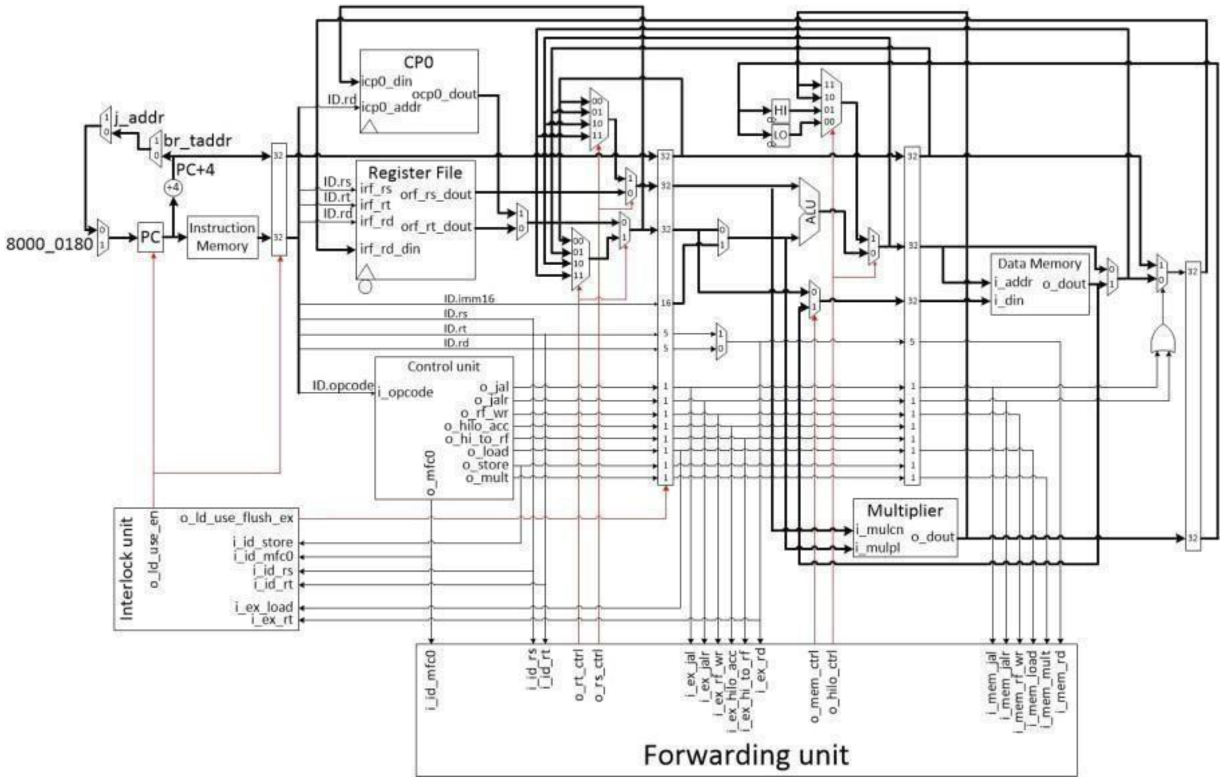


Fig 4.17 : Control overviews muxes in continuation with unit of forwarding [9]

4.3.3 CONTROL HAZARDS:

These conditional branch pipeline hazards exist. In our system, the choice to branch is made only after the MEMpipeline step. Control hazards or branch hazards .Introduced a new 2 bit branch prediction technique that alters its flow only after two consecutively incorrect predictions. This plan's accuracy ranges from 85% to 90%. These duo bit values saved for branch and used to make predictions.

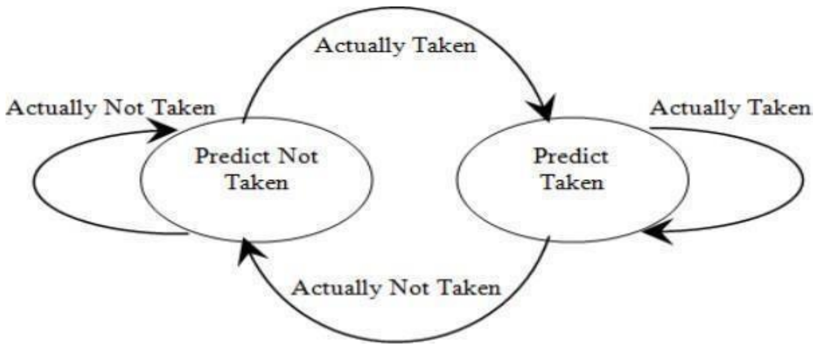


Fig 4.18: The states in 2-bit scheme of prediction[10]

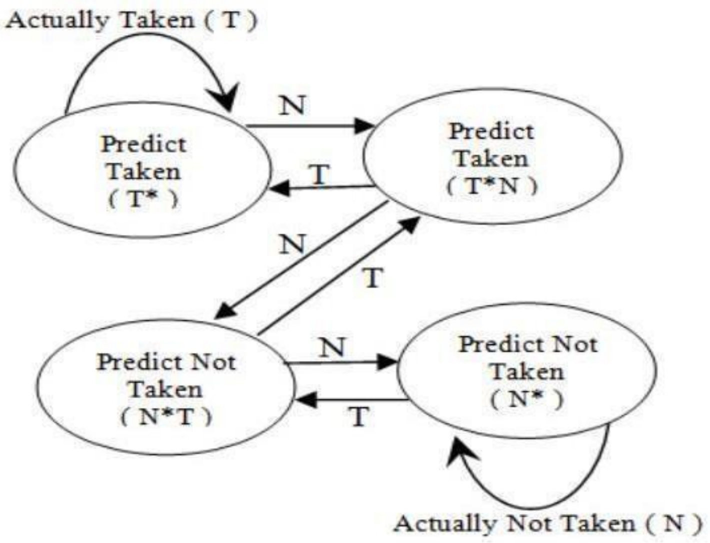


Fig. 4.19 : 2 bit branch prediction scheme [10]

CHAPTER 5

IMPLEMENTED RESULTS

Utilizing the datapath from the previous part and a straightforward control function, we create this straightforward implementation. The load word (lw), store word (Sw), branching if equal (BEQ), and arithmetic commands addition , subtract, AND, and doing OR are all covered by this straightforward implementation.

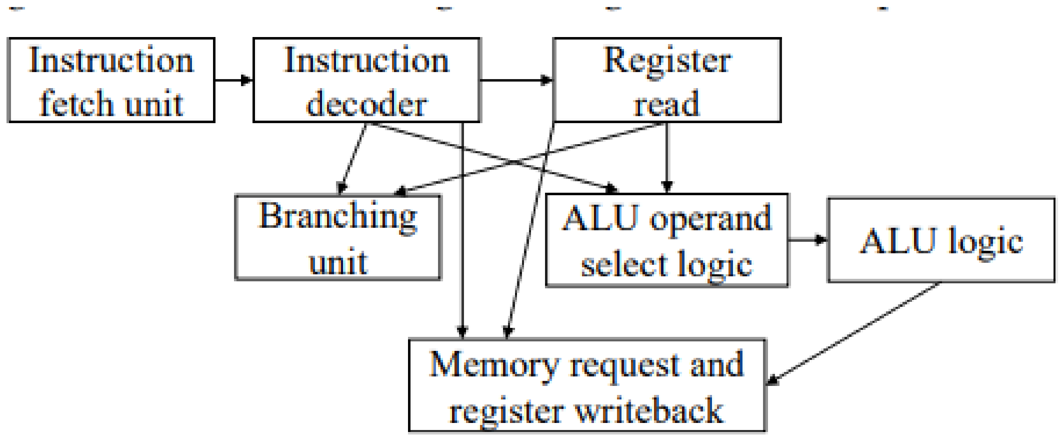
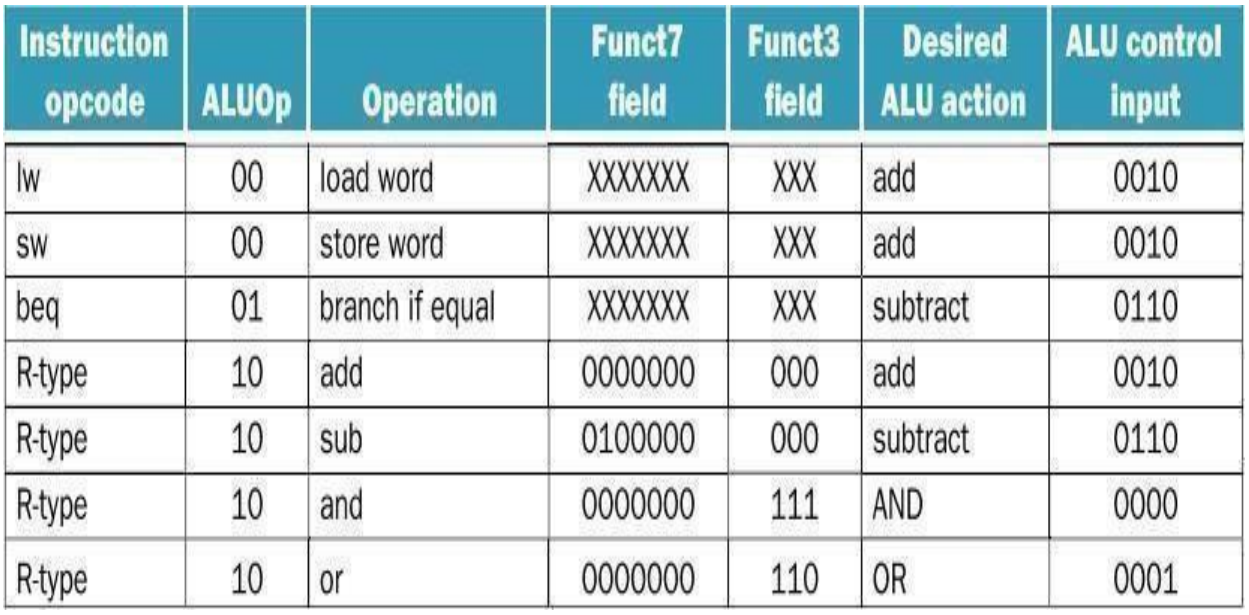


Fig 5.1: Correlation moduling [13]

The funct7 and function3 field as well as a 2 bit controlling field can be used as inputs to a small unit of controlthat we refer to as ALUOp in order to generate the 4 bits logicalarithmetic unit of control input. ALUOp specifies whether the operation is carried out should beaddition (00) for loading and storeing, a subtraction and a test of zero (01), or it should be determined by operation encoding. A four bit signal that directly controlling ALU producing single handed previously illustrated 4-bit combinationes is result of controlunit by ALU.

employing a tiny control unit, or ALUOp, that takes the function 7,3 fields for instruction asinputs along two bit control field. ALU Operation specifies whether operation to be carried out should an addition (00) in load and stores, a subtraction a test of zero (01) else it should be determining by the operations encoded in designated place.

Table 5.1: ALU depend on op bits[1]



Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

By producing the output of the ALU unit of control. Based on the 2-bit ALUOp control.we demonstrate how to setup ALU controls. The size of primary controlunits can be decreasedbyusing numerous levels of control. Using a number of minor controls.

The mapping from 2-bit ALUOps. And the function fields to four ALU operations control bit will be implemented in number of ways. Because only a small subset of the possible values for funct fields are of interest and because they are only used when the ALUOps bit equals 10.

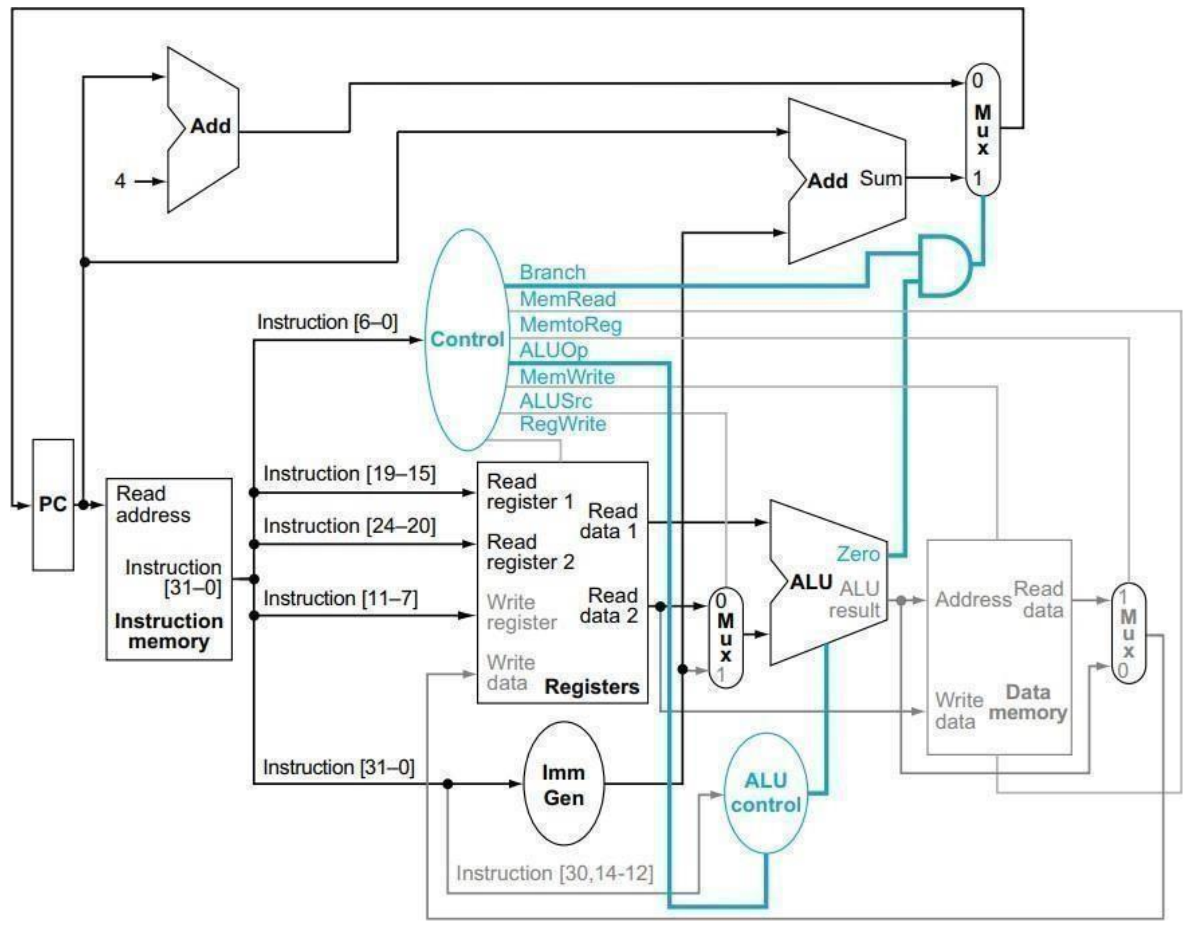


Fig 5.2: Basic Units of Processor [1]

5.1 Architecture Schematics

This Design showing communication between Core CPU, Instruction memory and data Memory accessing instructions from Instruction memory and approaching towards CPU for efficient execution through pipelined registers.

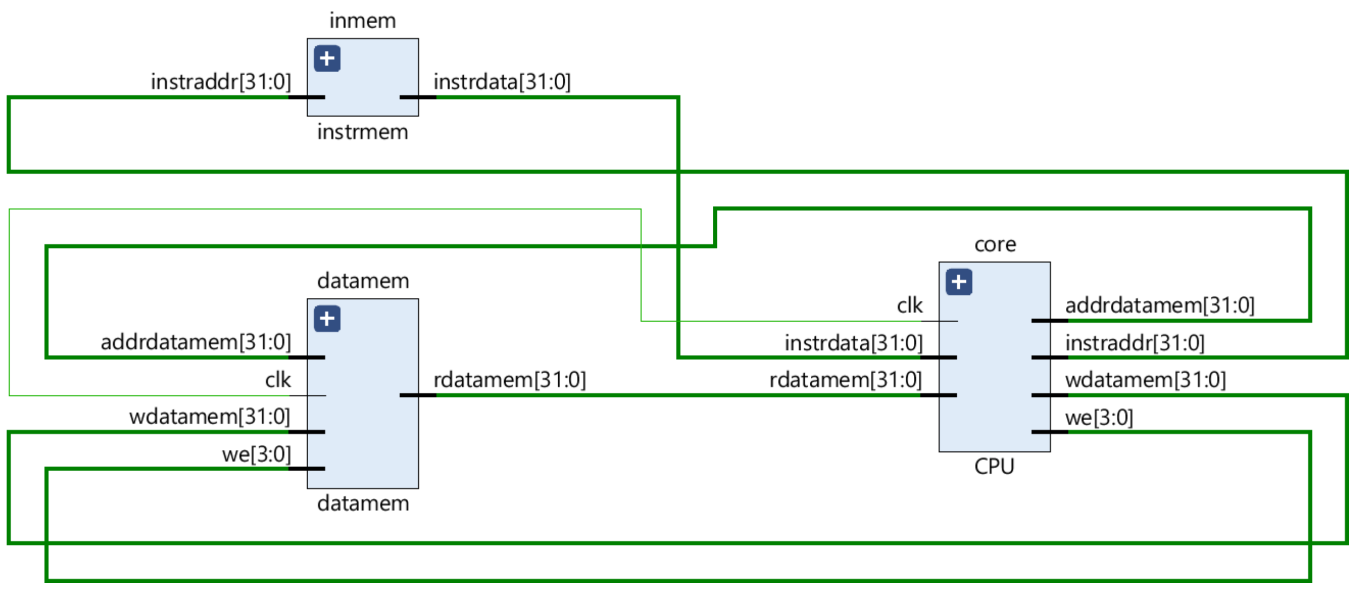


Fig 5.3: Interconnection between different units

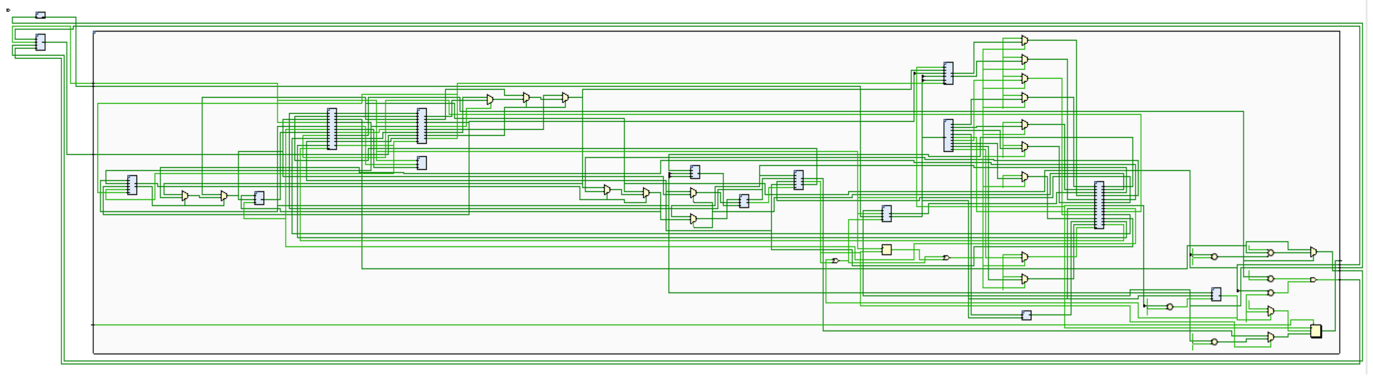


Fig 5.4: Integrated CPU Schematic

A. Instruction Memory:

Instruction memory used in fetch stage to store the instructions to be executed. It has single read port which take instruction address input as instraddr, and read data from that address and give output as instrdata.

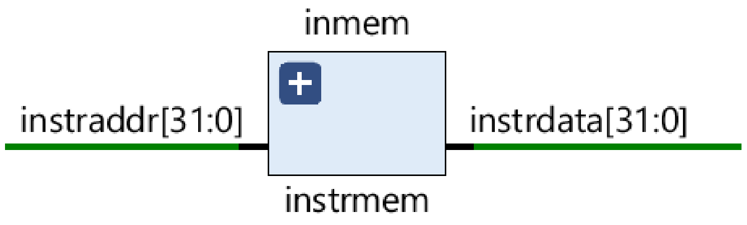


Fig 5.5: Schematic of instruction memory

B. Program Counter

Program Counter is a register in computer processor that store the address of the instruction that is being executed at present. And as the instruction being fetched the program counter store the the next address of instruction in the sequence.

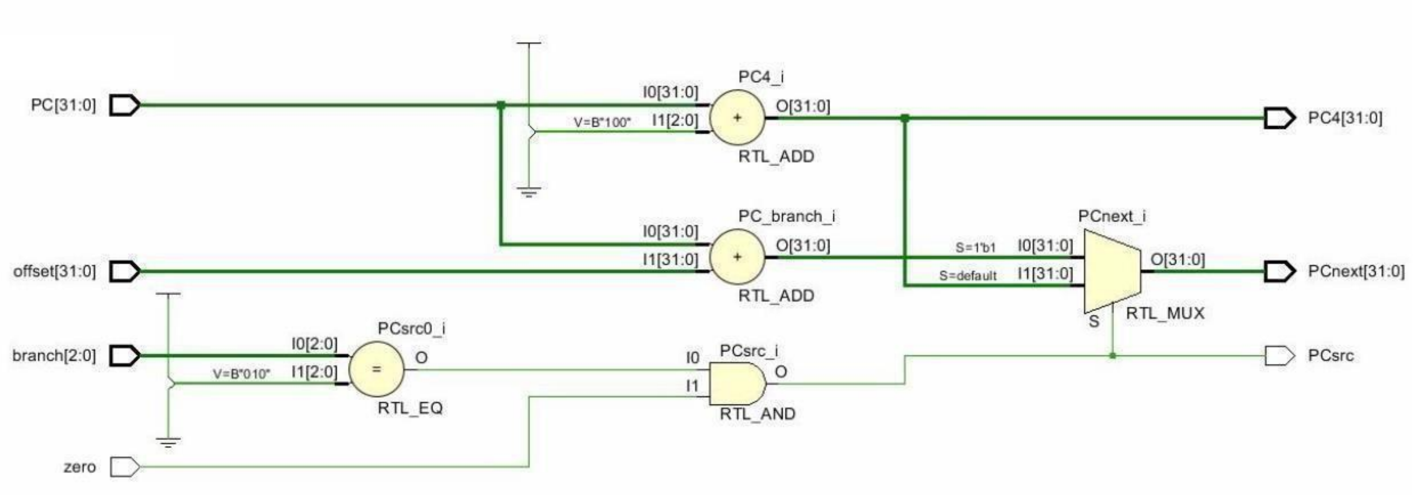


Fig 5.6: Schematic of Program Counter

C. IF-ID

We need program counter to fetch the instruction and this register stores about kind of instruction need to be executed given into it in the form of binary number each of which is 32bits.It is helpful in fetching multiple instructions.

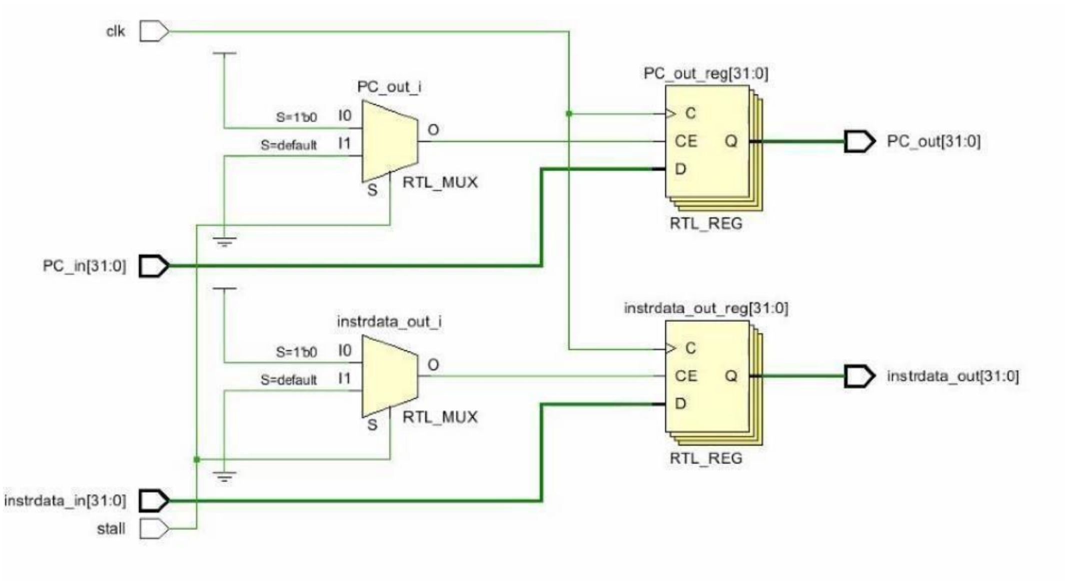


Fig.5.7: Schematic of IF-ID

D. ID-EX

This decodes the instruction as per binary format afterwards we got to what kind of instructions whether arithmetic or shift instructions need to be performed and also get to know from which source registers to read and write to destination Register. Thus, Parallel register reading are performed.

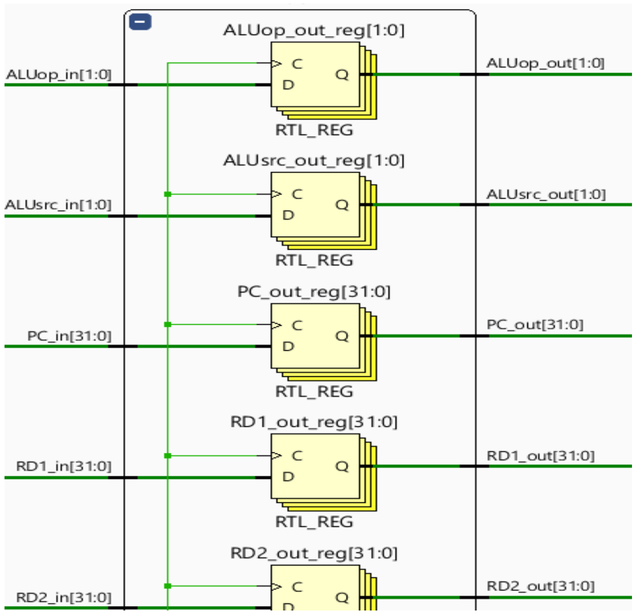


Fig. 5.8: Schematic of ID -Ex

E. EX-Mem

As per the input from decoding stage we do the execution of instruction and it is majorly done in ALU modules and as per execution we forward data onto registers else it passes on to memory.

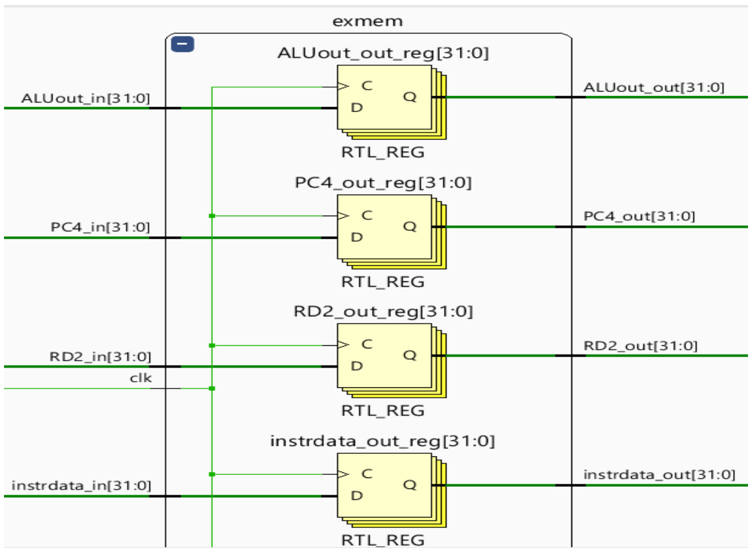


Fig.5.9: Schematic of Ex-Mem

F. Mem-Wb:

Now as per the need and operation of instruction we load data address into the memory and stores the respective on to it especially if it is load /store type instructions.

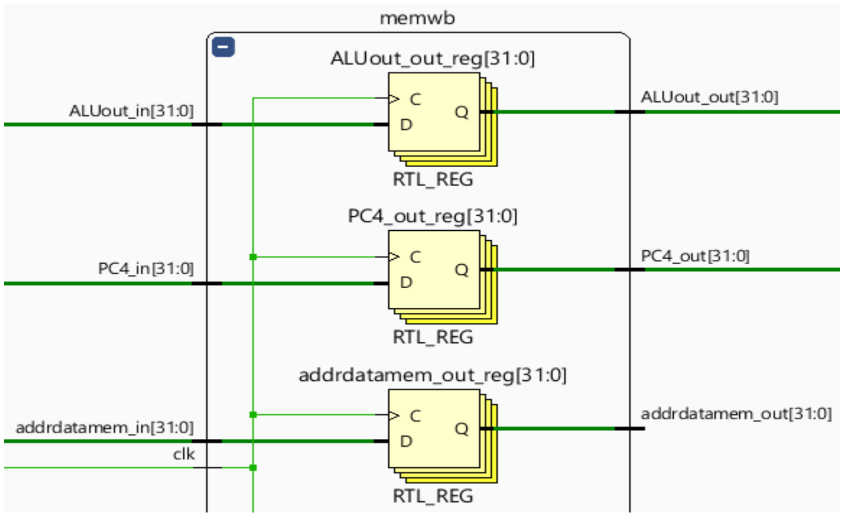


Fig.5.10: Schematic of Mem-Wb

G. REGISTER FILE

There are two source registers rs1 and rs2 to evaluate data as per your instruction format

and storing its value into respective destination register rd.

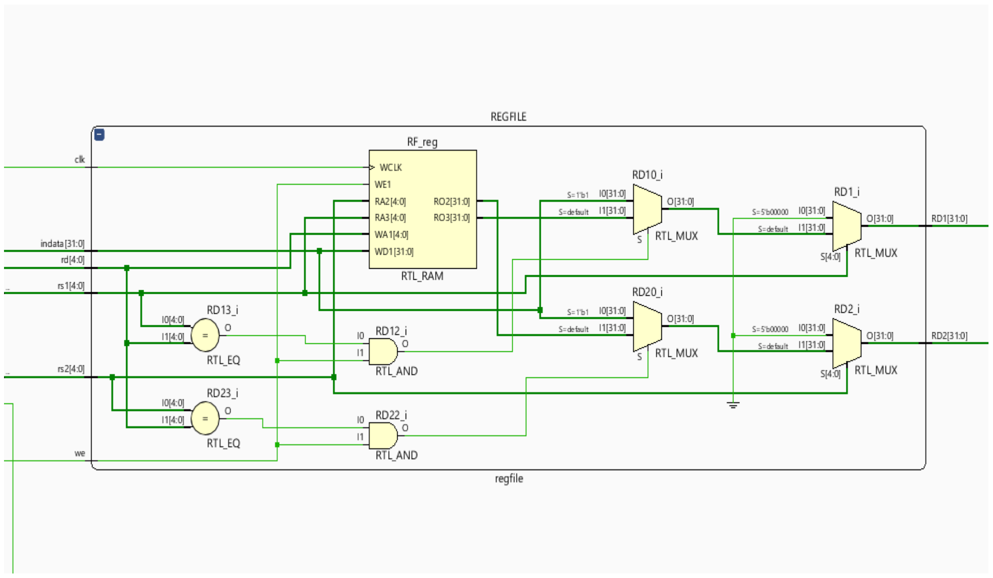


Fig.5.11: Schematic of Register File

H. ALU Control

As per opcode of the instruction you get knowledge about alu, load/word or branch type of instructions to perform within the processing units.

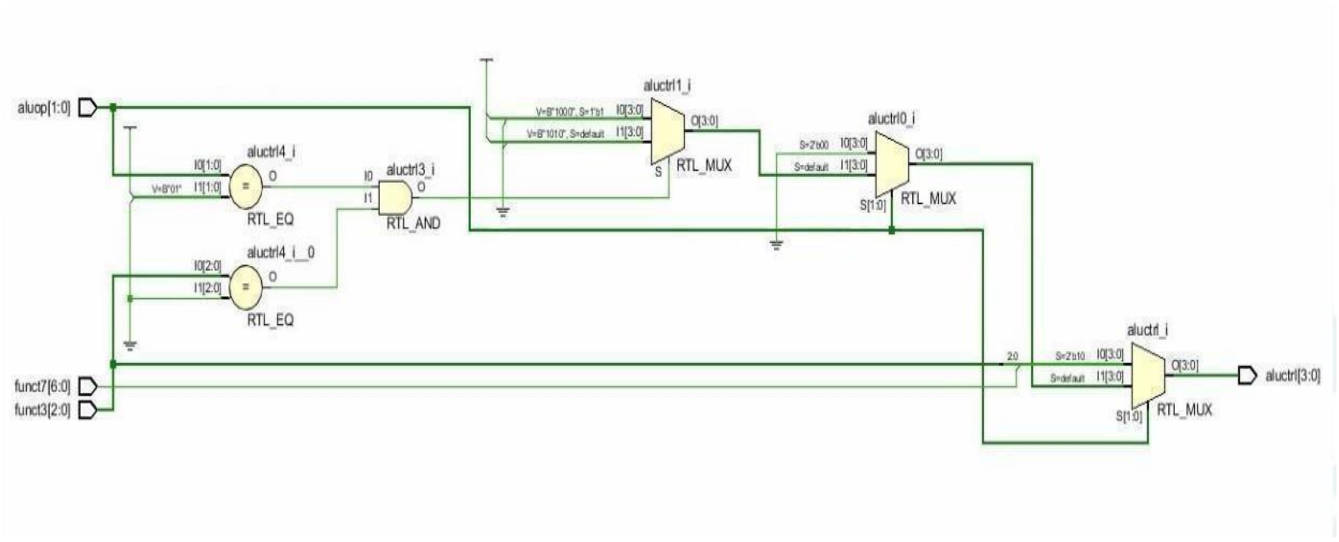


Fig.5.12: Schematic of ALU Control

I. ALU

ALU (Arithmetic logic unit) is a combinational circuit which takes two 32 bit data words A and B as input and perform logical and arithmetic operation on A and B and generate 32 bit output. Execution of add, sub type instructions take place in it.

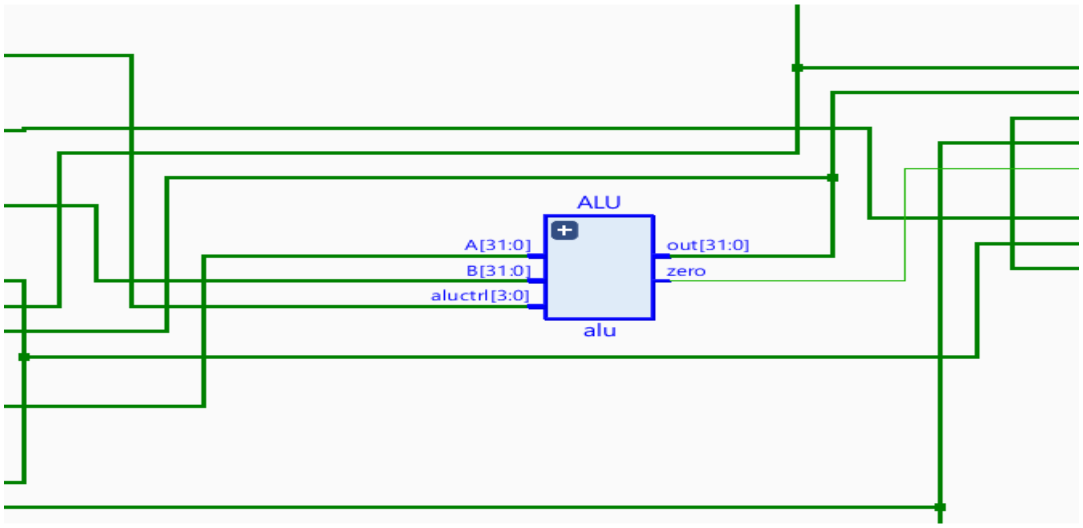


Fig.5.13: ALU Module

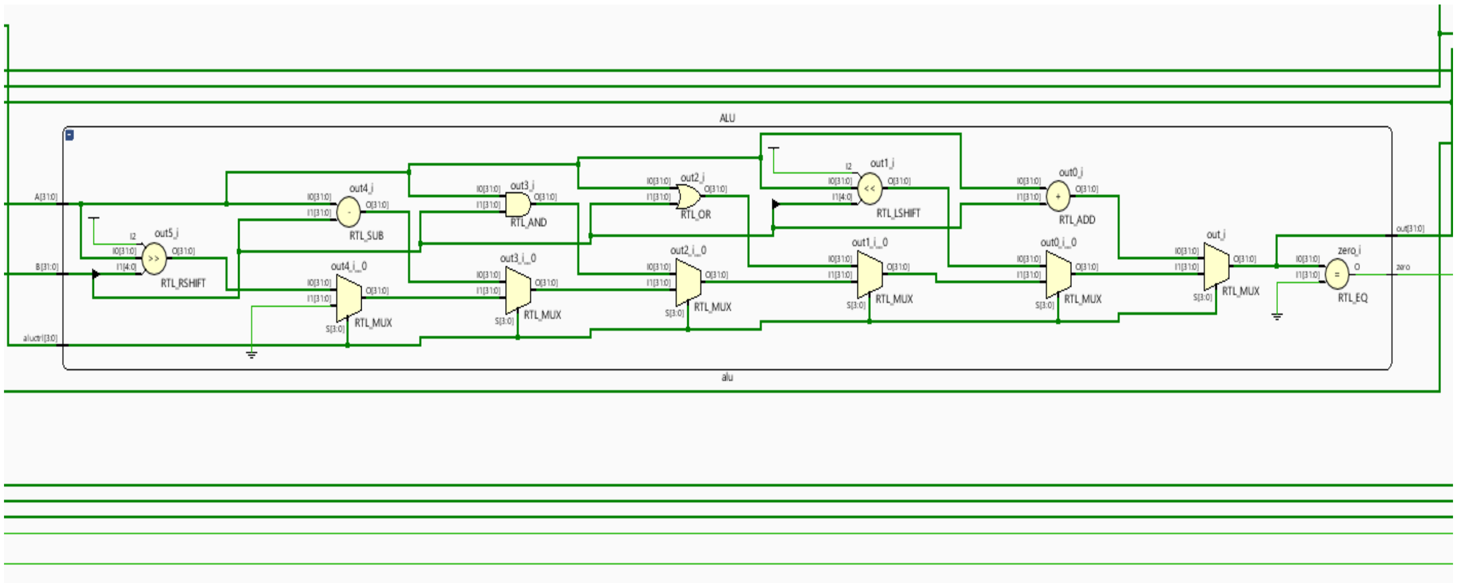


Fig.5.14: Schematic of ALU

J. <u>**Control Unit**</u>

A control unit functions by accepting input data, processing it into control signals, and sending those signals to the central processor. The attached Unit is subsequently instructed to conduct certain tasks by the computer's processor. It enables the logic unit, memory, and input and output devices of the computer to understand how to react to commands received from a programme.

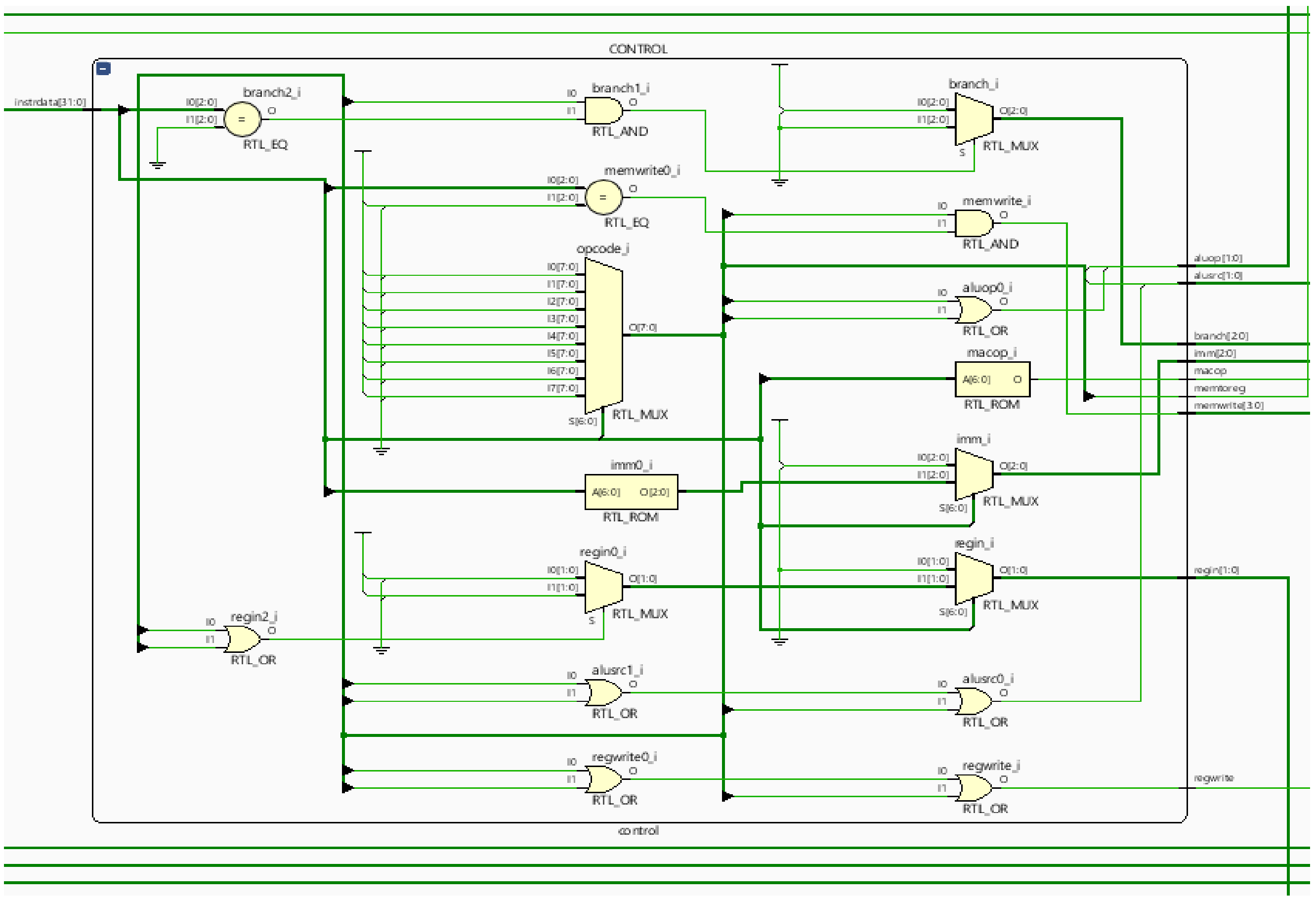


Fig.5.15: Schematic of Control unit

K. Stalling

This unit developed as per the need to resolve conflict of data hazard as when register data needed from Previous instruction.So, there is requirement of stalling a cycle by giving stall signal for efficient operation.

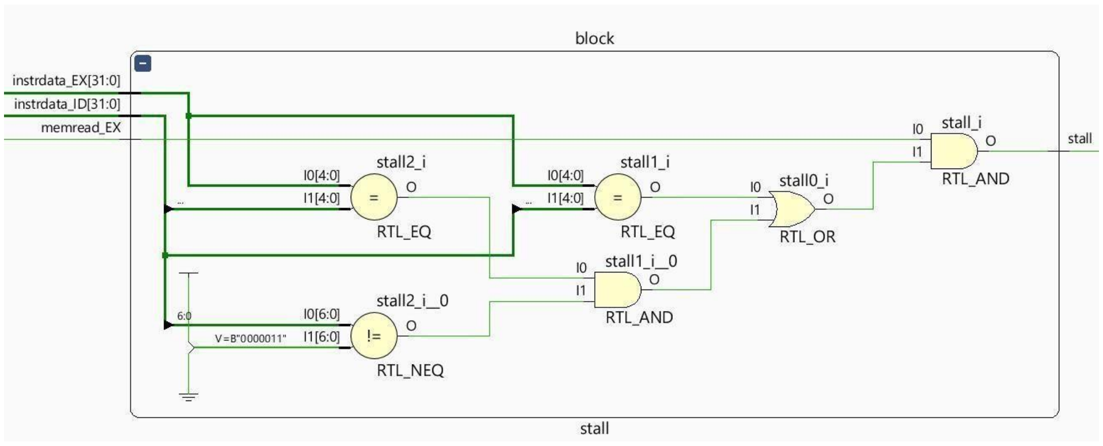


Fig 5.16: Schematic of Stalling Unit

L. FORWARDING

This module is developed to avoid the cycle stall and save your time to execute further instruction executions by temporarily storing data into some temporary block and thus forwarding it whenever needed.

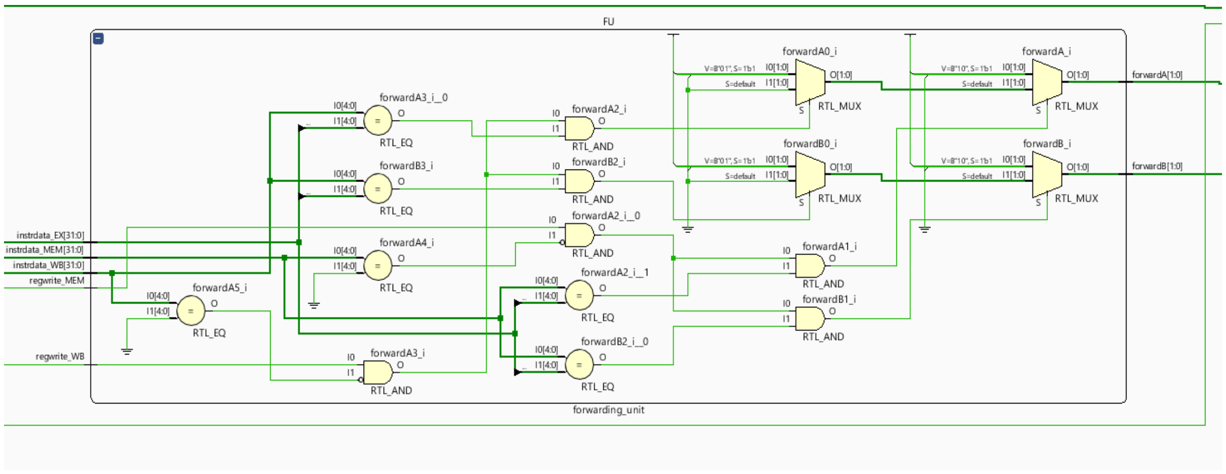
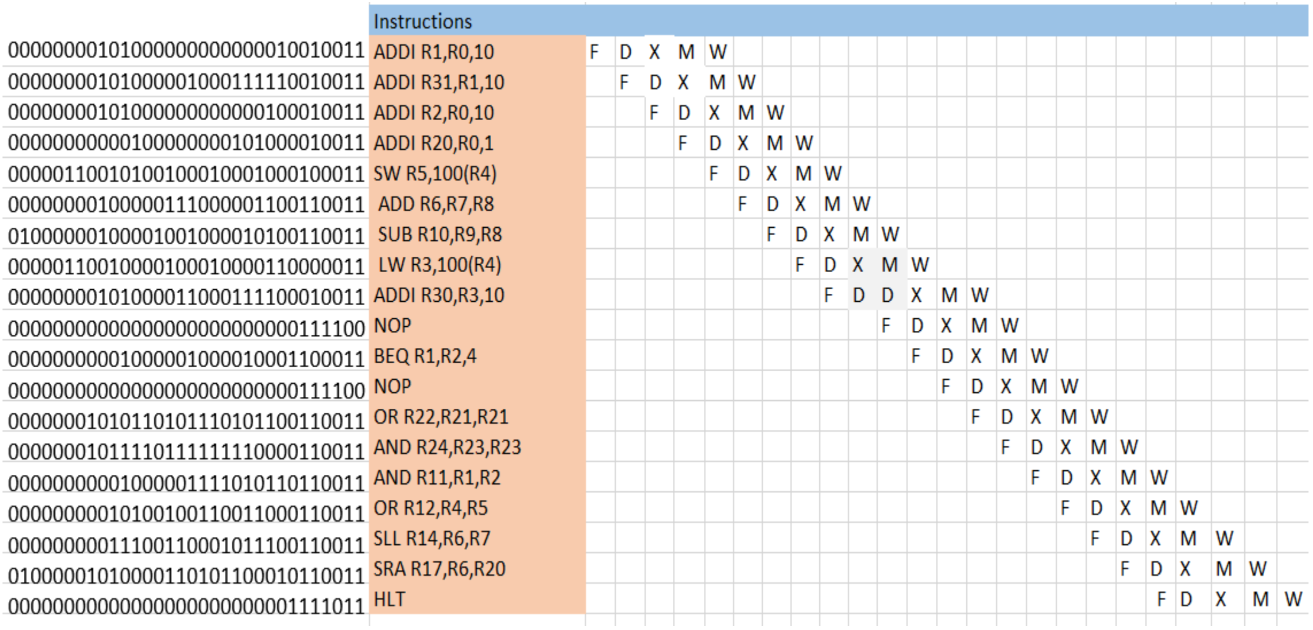


Fig 5.17: Schematic of Forwarding

5.2 A. SIMULATIONS:

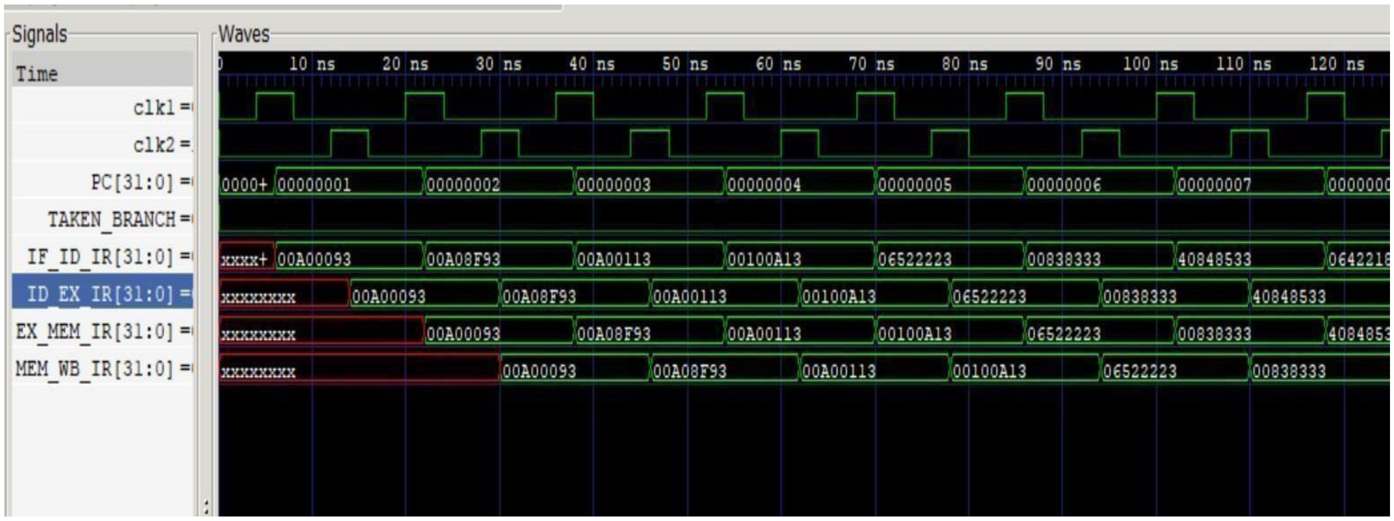
These are the generalized testcases for instructions that we have shown below and they arefollowing pipeline execution.



	Instructions																								
00000000101000000000000010010011	ADDI R1,R0,10	F	D	X	M	W																			
00000000101000001000111110010011	ADDI R31,R1,10		F	D	X	M	W																		
00000000101000000000000100010011	ADDI R2,R0,10			F	D	X	M	W																	
00000000000100000000101000010011	ADDI R20,R0,1				F	D	X	M	W																
00000110010100100010001000100011	SW R5,100(R4)					F	D	X	M	W															
00000000100000111000001100110011	ADD R6,R7,R8						F	D	X	M	W														
01000000100001001000010100110011	SUB R10,R9,R8							F	D	X	M	W													
00000110010000100010000110000011	LW R3,100(R4)								F	D	X	M	W												
00000000101000011000111100010011	ADDI R30,R3,10									F	D	D	X	M	W										
00000000000000000000000000111100	NOP											F	D	X	M	W									
00000000001000001000010001100011	BEQ R1,R2,4												F	D	X	M	W								
00000000000000000000000000111100	NOP													F	D	X	M	W							
00000001010110101110101100110011	OR R22,R21,R21														F	D	X	M	W						
00000001011110111111110000110011	AND R24,R23,R23															F	D	X	M	W					
00000000001000001111010110110011	AND R11,R1,R2																F	D	X	M	W				
00000000010100100110011000110011	OR R12,R4,R5																	F	D	X	M	W			
00000000011100110001011100110011	SLL R14,R6,R7																		F	D	X	M	W		
01000001010000110101100010110011	SRA R17,R6,R20																			F	D	X	M	W	
00000000000000000000000001111011	HLT																				F	D	X	M	W

Fig.5.18 : Pipelined instructions

Below waveforms shows that as program counter keep on updating, we get new instruction and that too passing through multiple pipelining registers thereby performing operation.

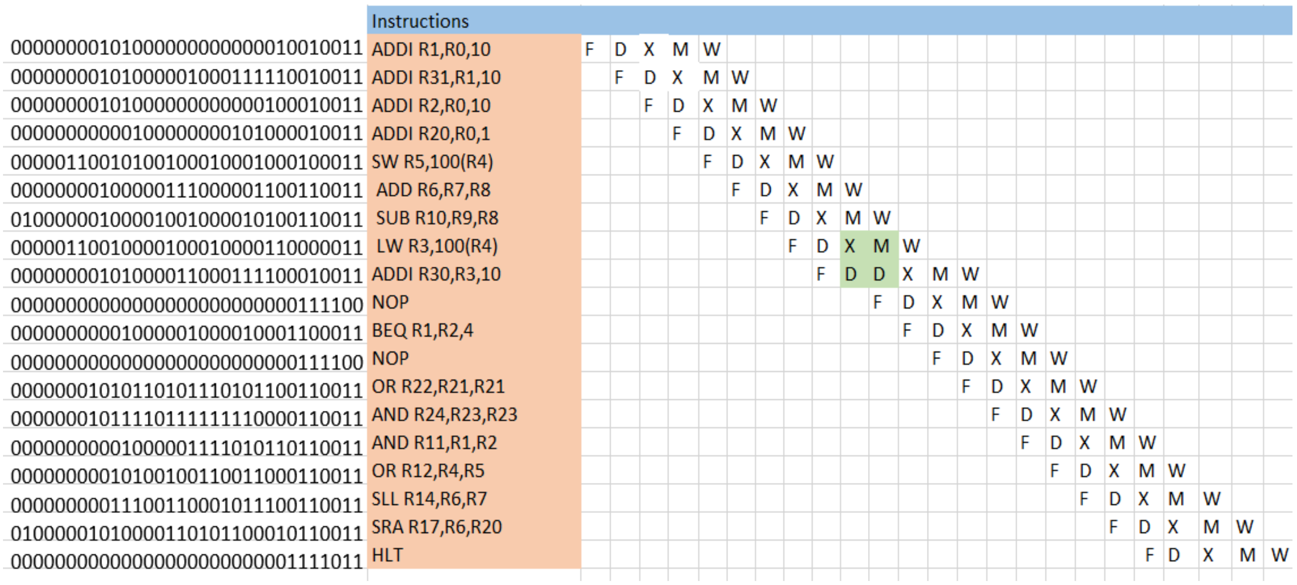


```
Signals          Waves
Time             10 ns    20 ns    30 ns    40 ns    50 ns    60 ns    70 ns    80 ns    90 ns    100 ns   110 ns   120 ns
clk1 =
clk2 =
PC[31:0] =       0000+ 00000001  00000002  00000003  00000004  00000005  00000006  00000007  0000000
TAKEN_BRANCH =
IF_ID_IR[31:0] = xxxx+ 00A00093  00A08F93  00A00113  00100A13  06522223  00838333  40848533  064221
ID_EX_IR[31:0] = xxxxxxxx  00A00093  00A08F93  00A00113  00100A13  06522223  00838333  40848533
EX_MEM_IR[31:0] = xxxxxxxx  00A00093  00A08F93  00A00113  00100A13  06522223  00838333  408485
MEM_WB_IR[31:0] = xxxxxxxx  00A00093  00A08F93  00A00113  00100A13  06522223  00838333
```

Fig 5.19: Waveform

B. STALLING:

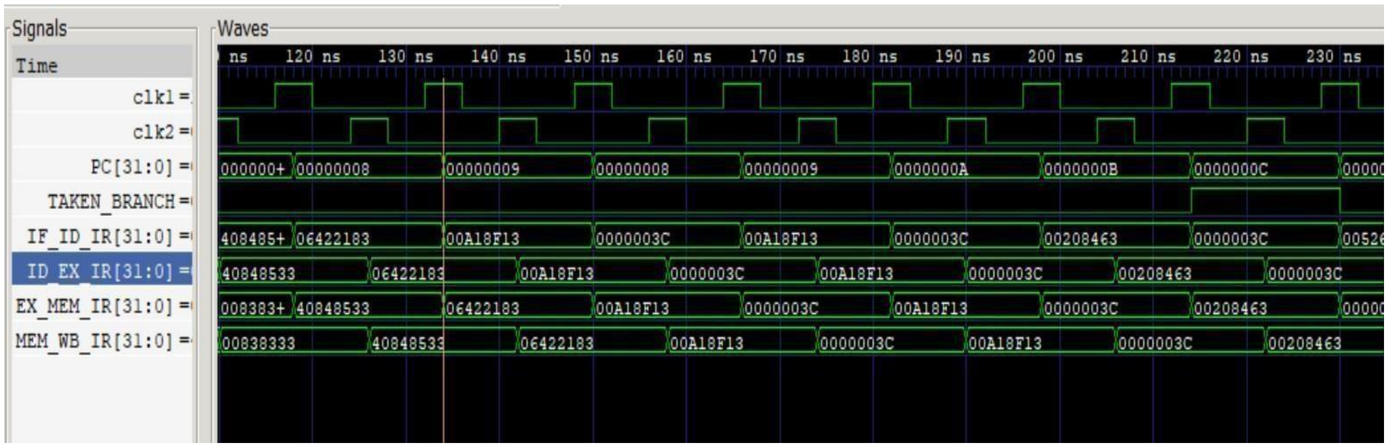
Stalling is used to prevent data risks brought on by pipelining. However, using the stall mechanism lengthens the processor's latency; as a result, we use bypassing/forwarding to prevent stalling as much as possible**.** Highlighted part shows for getting correct value for R3 we need to wait for a cycle from back instruction as once it writes data on to memory then only, we can have correct output.



	Instructions																								
00000000101000000000000010010011	ADDI R1,R0,10	F	D	X	M	W																			
00000000101000001000111110010011	ADDI R31,R1,10		F	D	X	M	W																		
00000000101000000000000100010011	ADDI R2,R0,10			F	D	X	M	W																	
00000000000100000000101000010011	ADDI R20,R0,1				F	D	X	M	W																
00000110010100100010001000100011	SW R5,100(R4)					F	D	X	M	W															
00000000100000111000001100110011	ADD R6,R7,R8						F	D	X	M	W														
01000000100001001000010100110011	SUB R10,R9,R8							F	D	X	M	W													
00000110010000100010000110000011	LW R3,100(R4)								F	D	X	M	W												
00000000101000011000111100010011	ADDI R30,R3,10									F	D	D	X	M	W										
00000000000000000000000000111100	NOP											F	D	X	M	W									
00000000001000001000010001100011	BEQ R1,R2,4												F	D	X	M	W								
00000000000000000000000000111100	NOP													F	D	X	M	W							
00000001010110101110101100110011	OR R22,R21,R21														F	D	X	M	W						
00000001011110111111110000110011	AND R24,R23,R23															F	D	X	M	W					
00000000001000001111010110110011	AND R11,R1,R2																F	D	X	M	W				
00000000010100100110011000110011	OR R12,R4,R5																	F	D	X	M	W			
00000000011100110001011100110011	SLL R14,R6,R7																		F	D	X	M	W		
01000001010000110101100010110011	SRA R17,R6,R20																			F	D	X	M	W	
00000000000000000000000001111011	HLT																				F	D	X	M	W

Fig. 5.20 : Stall Operation

Below Part Showing as ADDI instruction having code 00A18F13 comes in, we take no operation for a cycle so by that time back instruction can properly load data into memory and now the data that our present instruction need can be properly estimated and then get executed normally.

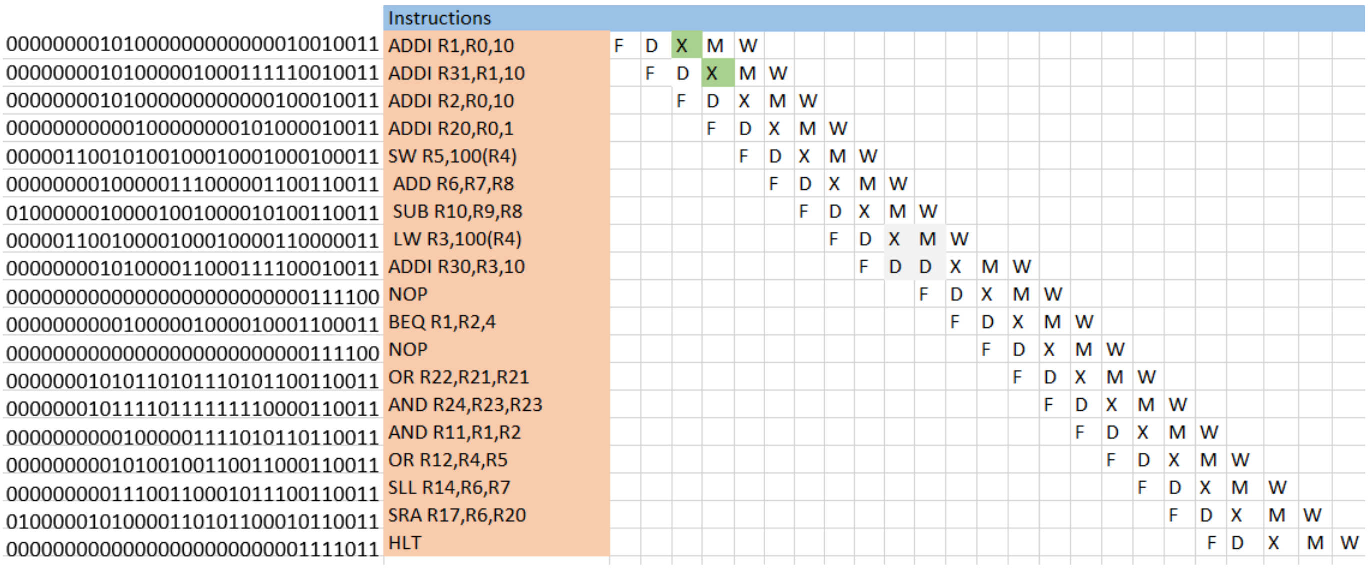


Signals
Waves
Time
clk1
clk2
PC[31:0]
TAKEN_BRANCH
IF_ID_IR[31:0]
ID_EX_IR[31:0]
EX_MEM_IR[31:0]
MEM_WB_IR[31:0]
ns 120 ns 130 ns 140 ns 150 ns 160 ns 170 ns 180 ns 190 ns 200 ns 210 ns 220 ns 230 ns
000000+ 00000008 00000009 00000008 00000009 0000000A 0000000B 0000000C
408485+ 06422183 00A18F13 0000003C 00A18F13 0000003C 00208463 0000003C
40848533 06422183 00A18F13 0000003C 00A18F13 0000003C 00208463 0000003C
008383+ 40848533 06422183 00A18F13 0000003C 00A18F13 0000003C 00208463
00838333 40848533 06422183 00A18F13 0000003C 00A18F13 0000003C 00208463

Fig. 5.21: Stall Waveforms

C. FORWARDING:

When the outcome is computed in the Execute stage of an instruction, forwarding is sufficient to address RAW data dangers since its result can then be transmitted to the Execute stage of the following instruction.



	Instructions																								
00000000101000000000000010010011	ADDI R1,R0,10	F	D	X	M	W																			
00000000101000001000111110010011	ADDI R31,R1,10		F	D	X	M	W																		
00000000101000000000000100010011	ADDI R2,R0,10			F	D	X	M	W																	
00000000000100000000101000010011	ADDI R20,R0,1				F	D	X	M	W																
00000110010100100010001000100011	SW R5,100(R4)					F	D	X	M	W															
00000000100000111000001100110011	ADD R6,R7,R8						F	D	X	M	W														
01000000100001001000010100110011	SUB R10,R9,R8							F	D	X	M	W													
00000110010000100010000110000011	LW R3,100(R4)								F	D	X	M	W												
00000000101000011000111100010011	ADDI R30,R3,10									F	D	D	X	M	W										
00000000000000000000000000111100	NOP											F	D	X	M	W									
00000000001000001000010001100011	BEQ R1,R2,4												F	D	X	M	W								
00000000000000000000000000111100	NOP													F	D	X	M	W							
00000001010110101110101100110011	OR R22,R21,R21														F	D	X	M	W						
00000001011110111111110000110011	AND R24,R23,R23															F	D	X	M	W					
00000000001000001111010110110011	AND R11,R1,R2																F	D	X	M	W				
00000000010100100110011000110011	OR R12,R4,R5																	F	D	X	M	W			
00000000011100110001011100110011	SLL R14,R6,R7																		F	D	X	M	W		
01000001010000110101100010110011	SRA R17,R6,R20																			F	D	X	M	W	
00000000000000000000000001111011	HLT																				F	D	X	M	W

Fig. 5.22 : Operation during data forward

It is showing as your ADDI instruction comes it needs data from previous instruction's register. So, as per above way there is no need to take break and we can directly forward this data further as instruction result, we are getting within same stage. So, it continuously propagates across pipelined registers.

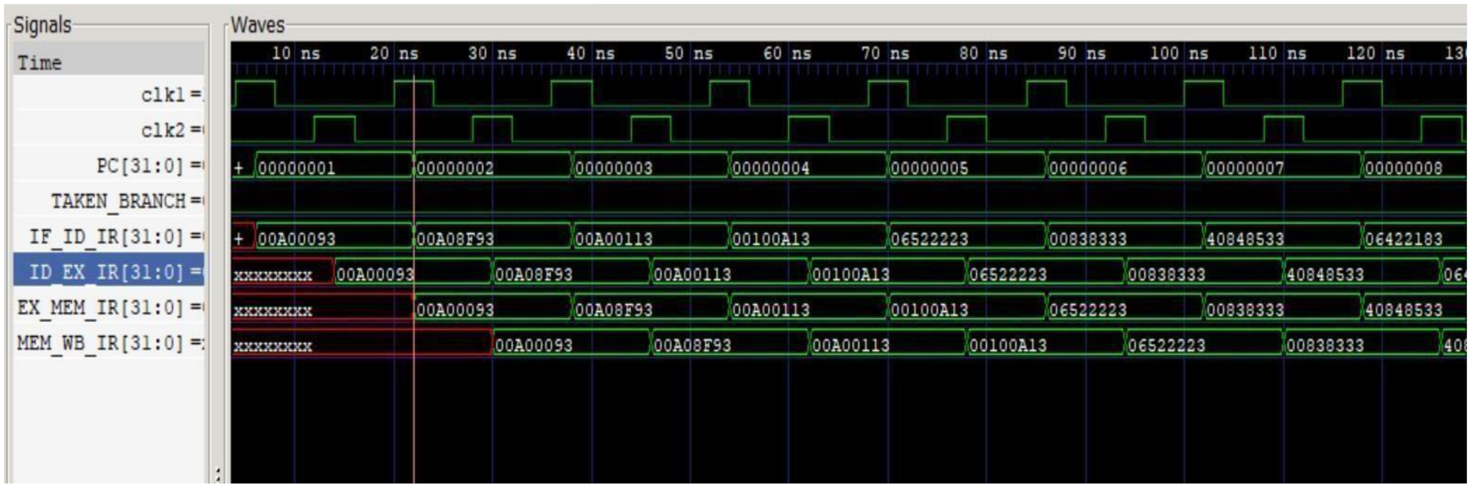
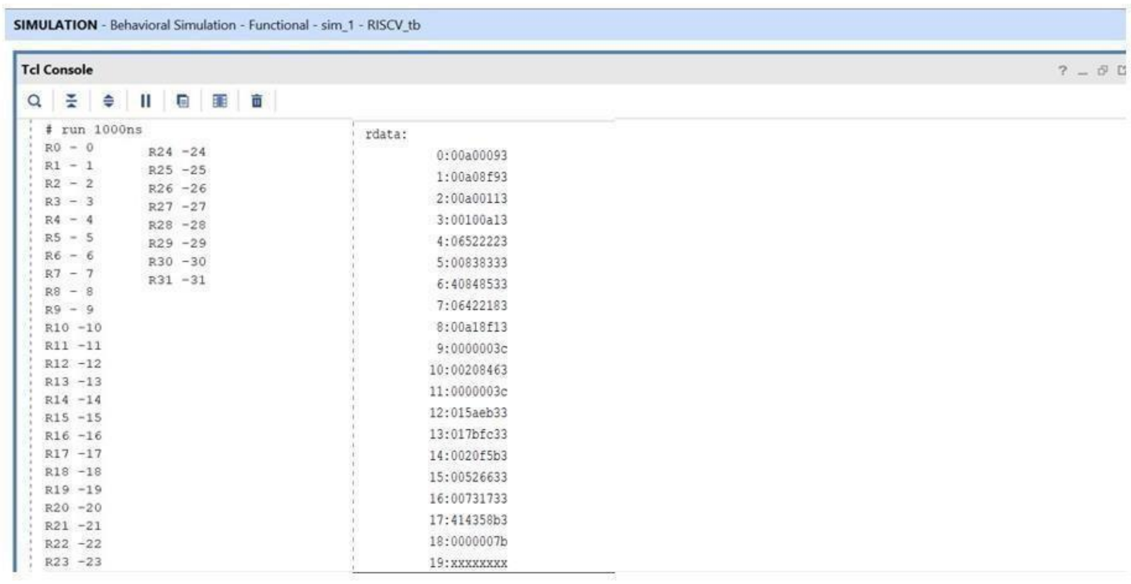


Fig.5.23: Forwards Waveform

D. Register Values before simulation

This Shows the fact that as per sequence of instructions that we are executing initially we don't observe any working of instruction and techniques to improve data accuracy.


SIMULATION - Behavioral Simulation - Functional - sim_1 - RISCV_tb

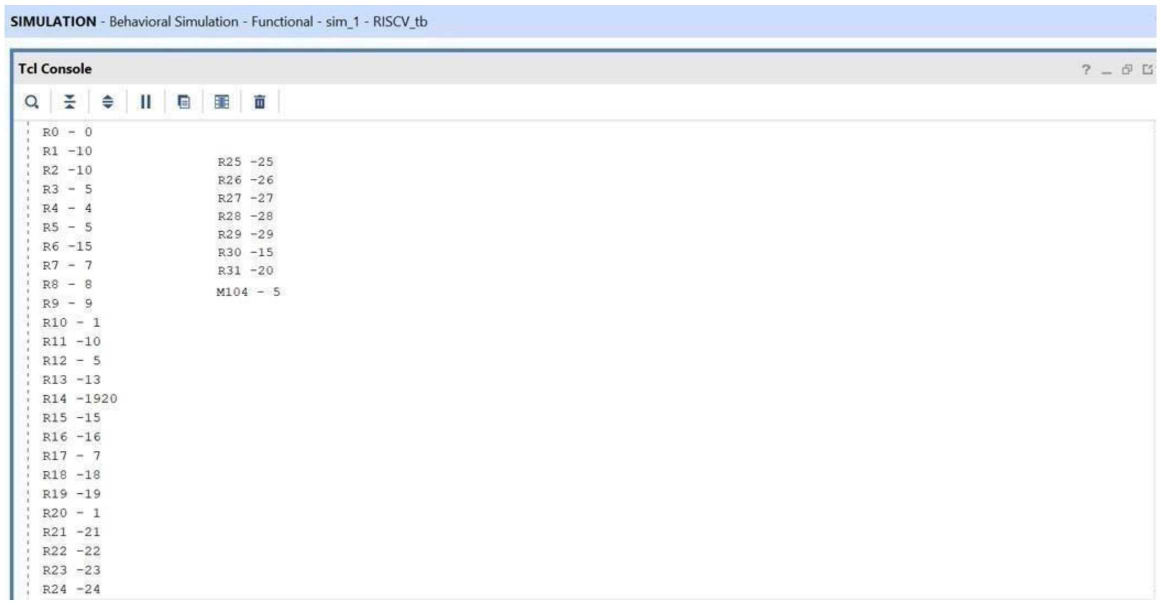
Tcl Console

```
# run 1000ns
R0 - 0          R24 -24        rdata:
R1 - 1          R25 -25                   0:00a00093
R2 - 2          R26 -26                   1:00a08f93
R3 - 3          R27 -27                   2:00a00113
R4 - 4          R28 -28                   3:00100a13
R5 - 5          R29 -29                   4:06522223
R6 - 6          R30 -30                   5:00838333
R7 - 7          R31 -31                   6:40848533
R8 - 8                                    7:06422183
R9 - 9                                    8:00a18f13
R10 -10                                   9:0000003c
R11 -11                                  10:00208463
R12 -12                                  11:0000003c
R13 -13                                  12:015aeb33
R14 -14                                  13:017bfc33
R15 -15                                  14:0020f5b3
R16 -16                                  15:00526633
R17 -17                                  16:00731733
R18 -18                                  17:414358b3
R19 -19                                  18:0000007b
R20 -20                                  19:xxxxxxxx
R21 -21
R22 -22
R23 -23
```

Fig. 5.24: Initial registers Values

E. Register Values after simulation

It depicts that as our processor runs, the above sequence of instructions are working and Register values getting updated showing correct functionality of processor.


SIMULATION - Behavioral Simulation - Functional - sim_1 - RISCV_tb

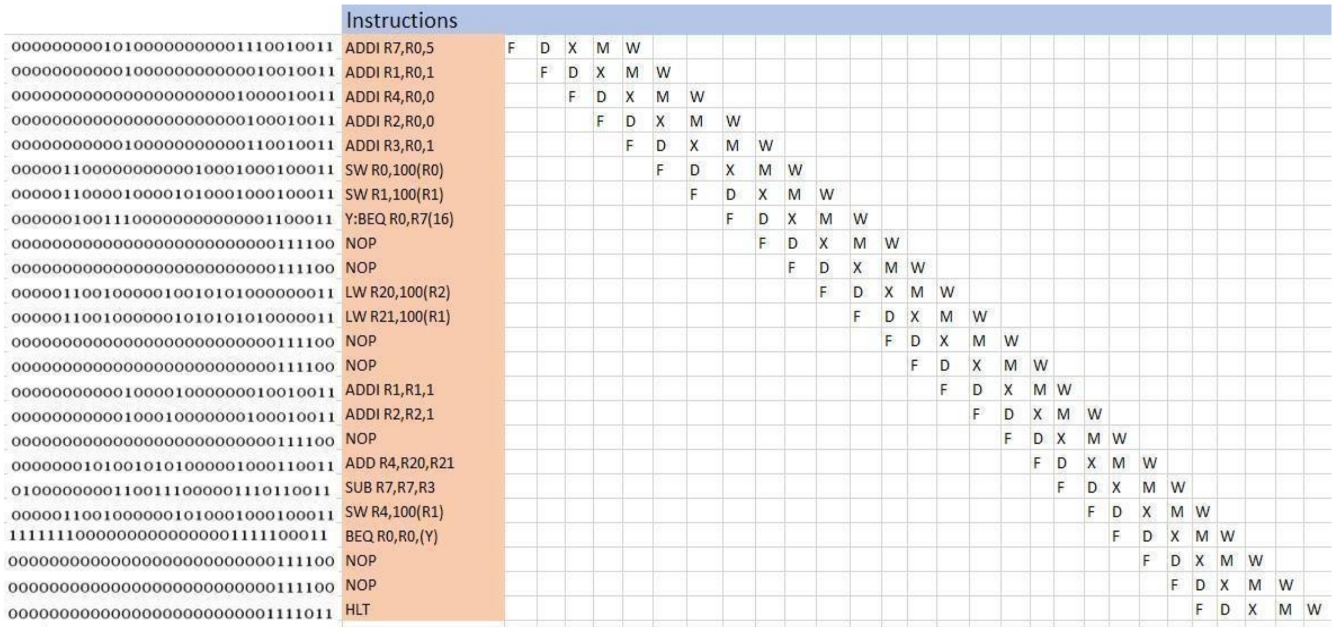
Tcl Console

```
R0 - 0
R1 -10                     R25 -25
R2 -10                     R26 -26
R3 - 5                     R27 -27
R4 - 4                     R28 -28
R5 - 5                     R29 -29
R6 -15                     R30 -15
R7 - 7                     R31 -20
R8 - 8                     M104 - 5
R9 - 9
R10 - 1
R11 -10
R12 - 5
R13 -13
R14 -1920
R15 -15
R16 -16
R17 - 7
R18 -18
R19 -19
R20 - 1
R21 -21
R22 -22
R23 -23
R24 -24
```

Figure 5.25: Executed Register Value

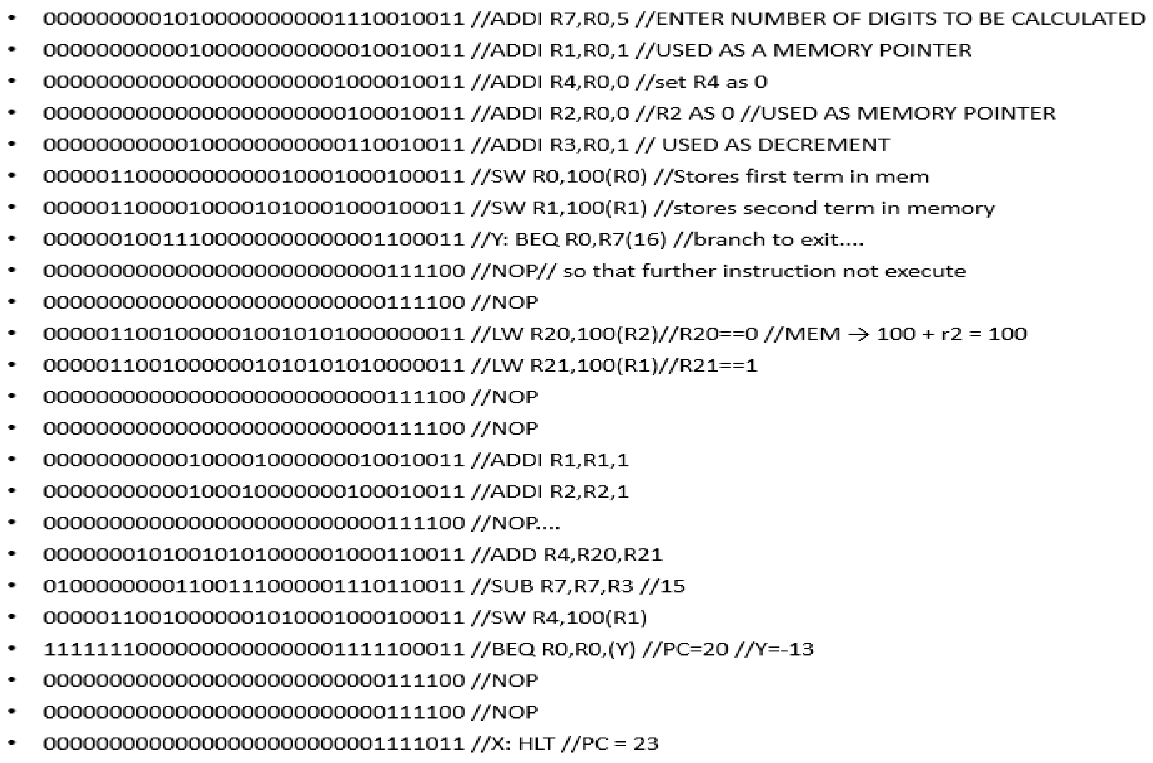
F. Fibonacci Run

This shows the sequence of instructions needed for Fibonacci series program to run on our processor with their respective binary format and thereby showing processing through pipelined stages.



	Instructions																												
00000000010100000000001110010011	ADDI R7,R0,5	F	D	X	M	W																							
00000000000100000000000010010011	ADDI R1,R0,1		F	D	X	M	W																						
00000000000000000000001000010011	ADDI R4,R0,0			F	D	X	M	W																					
00000000000000000000000100010011	ADDI R2,R0,0				F	D	X	M	W																				
00000000000100000000000110010011	ADDI R3,R0,1					F	D	X	M	W																			
00000110000000000010001000100011	SW R0,100(R0)						F	D	X	M	W																		
00000110000100001010001000100011	SW R1,100(R1)							F	D	X	M	W																	
00000010011100000000000001100011	Y:BEQ R0,R7(16)								F	D	X	M	W																
00000000000000000000000000111100	NOP									F	D	X	M	W															
00000000000000000000000000111100	NOP										F	D	X	M	W														
00000110010000010010101000000011	LW R20,100(R2)											F	D	X	M	W													
00000110010000001010101010000011	LW R21,100(R1)												F	D	X	M	W												
00000000000000000000000000111100	NOP													F	D	X	M	W											
00000000000000000000000000111100	NOP														F	D	X	M	W										
00000000000100001000000010010011	ADDI R1,R1,1															F	D	X	M	W									
00000000000100010000000100010011	ADDI R2,R2,1																F	D	X	M	W								
00000000000000000000000000111100	NOP																	F	D	X	M	W							
00000001010010101000001000110011	ADD R4,R20,R21																		F	D	X	M	W						
01000000001100111000001110110011	SUB R7,R7,R3																			F	D	X	M	W					
00000110010000001010001000100011	SW R4,100(R1)																				F	D	X	M	W				
11111110000000000000001111100011	BEQ R0,R0,(Y)																					F	D	X	M	W			
00000000000000000000000000111100	NOP																						F	D	X	M	W		
00000000000000000000000000111100	NOP																							F	D	X	M	W	
00000000000000000000000001111011	HLT																								F	D	X	M	W

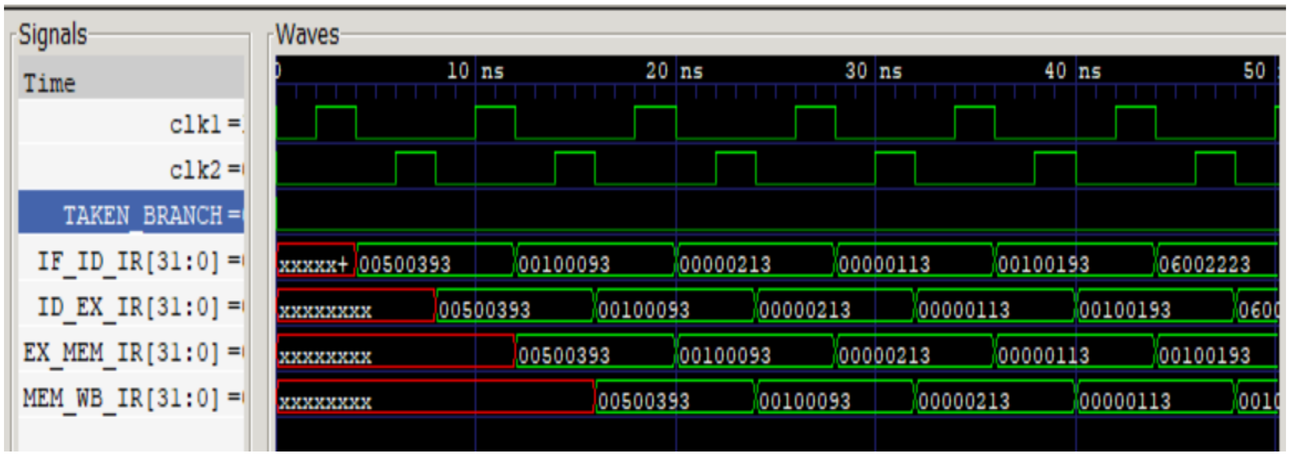
Figure 5.26: Operations chart



- 00000000010100000000001110010011 //ADDI R7,R0,5 //ENTER NUMBER OF DIGITS TO BE CALCULATED
- 00000000000100000000000010010011 //ADDI R1,R0,1 //USED AS A MEMORY POINTER
- 00000000000000000000001000010011 //ADDI R4,R0,0 //set R4 as 0
- 00000000000000000000000100010011 //ADDI R2,R0,0 //R2 AS 0 //USED AS MEMORY POINTER
- 00000000000100000000000110010011 //ADDI R3,R0,1 // USED AS DECREMENT
- 00000110000000000010001000100011 //SW R0,100(R0) //Stores first term in mem
- 00000110000100001010001000100011 //SW R1,100(R1) //stores second term in memory
- 00000010011100000000000001100011 //Y: BEQ R0,R7(16) //branch to exit....
- 00000000000000000000000000111100 //NOP// so that further instruction not execute
- 00000000000000000000000000111100 //NOP
- 00000110010000010010101000000011 //LW R20,100(R2)//R20==0 //MEM → 100 + r2 = 100
- 00000110010000001010101010000011 //LW R21,100(R1)//R21==1
- 00000000000000000000000000111100 //NOP
- 00000000000000000000000000111100 //NOP
- 00000000000100001000000010010011 //ADDI R1,R1,1
- 00000000000100010000000100010011 //ADDI R2,R2,1
- 00000000000000000000000000111100 //NOP....
- 00000001010010101000001000110011 //ADD R4,R20,R21
- 01000000001100111000001110110011 //SUB R7,R7,R3 //15
- 00000110010000001010001000100011 //SW R4,100(R1)
- 11111110000000000000001111100011 //BEQ R0,R0,(Y) //PC=20 //Y=-13
- 00000000000000000000000000111100 //NOP
- 00000000000000000000000000111100 //NOP
- 00000000000000000000000001111011 //X: HLT //PC = 23

Figure 5.27: Snippet showing way of Fibonacci

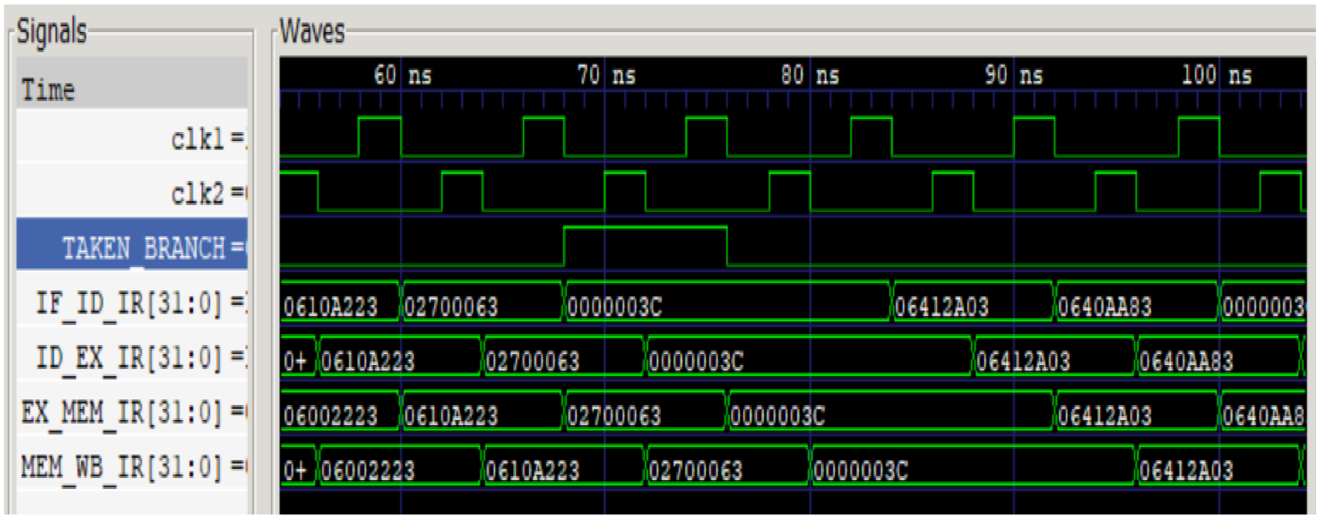
Below waveforms shows that our Fibonacci program adapted itself according to processor and smoothly passing each instruction through various stages.



Signals
Waves
Time
clk1 =
clk2 =
TAKEN_BRANCH =
IF_ID_IR[31:0] =
ID_EX_IR[31:0] =
EX_MEM_IR[31:0] =
MEM_WB_IR[31:0] =
10 ns
20 ns
30 ns
40 ns
50
xxxxx+ 00500393 00100093 00000213 00000113 00100193 06002223
xxxxxxxx 00500393 00100093 00000213 00000113 00100193 0600
xxxxxxxx 00500393 00100093 00000213 00000113 00100193
xxxxxxxx 00500393 00100093 00000213 00000113 0010

Figure 5.28: Instructions in fibonacci

This simulation as branch is taken on instruction '02700063' after having no operation for two cycles we finally got the next required value as per program.

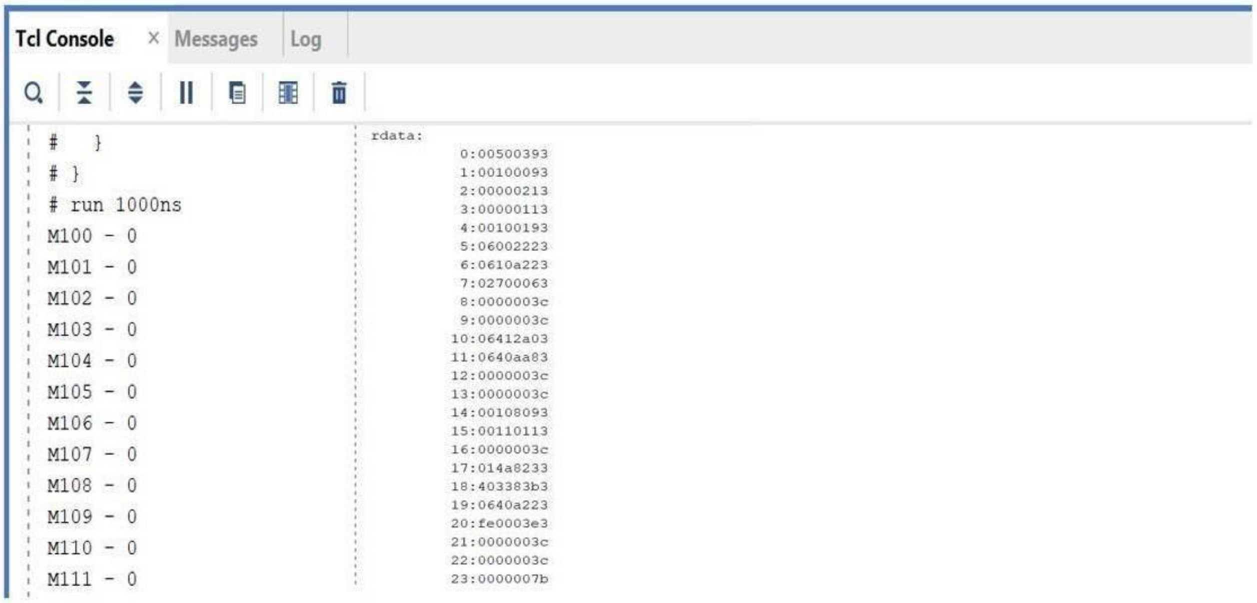


Signals
Waves
Time
clk1 =
clk2 =
TAKEN_BRANCH =
IF_ID_IR[31:0] =
ID_EX_IR[31:0] =
EX_MEM_IR[31:0] =
MEM_WB_IR[31:0] =
60 ns
70 ns
80 ns
90 ns
100 ns
0610A223 02700063 0000003C 06412A03 0640AA83 0000003
0+ 0610A223 02700063 0000003C 06412A03 0640AA83
06002223 0610A223 02700063 0000003C 06412A03 0640AA8
0+ 06002223 0610A223 02700063 0000003C 06412A03

Figure 5.29: Fibo Waveform

G. <u>Memory Values</u>

These are initial memory values within processor before running Fibonacci programme as all are Having its resetted values.

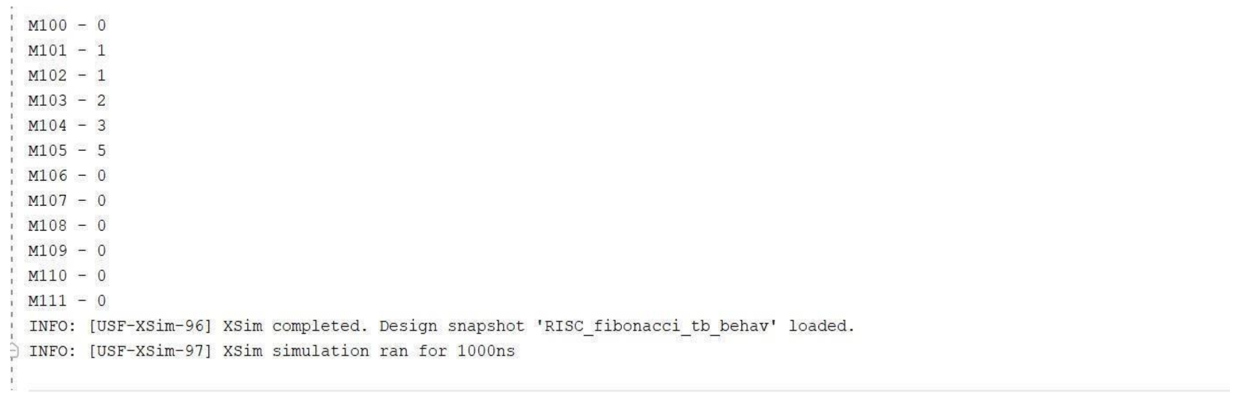


```
#   }
# }
# run 1000ns
M100 - 0
M101 - 0
M102 - 0
M103 - 0
M104 - 0
M105 - 0
M106 - 0
M107 - 0
M108 - 0
M109 - 0
M110 - 0
M111 - 0
```

```
rdata:
          0:00500393
          1:00100093
          2:00000213
          3:00000113
          4:00100193
          5:06002223
          6:0610a223
          7:02700063
          8:0000003c
          9:0000003c
         10:06412a03
         11:0640aa83
         12:0000003c
         13:0000003c
         14:00108093
         15:00110113
         16:0000003c
         17:014a8233
         18:403383b3
         19:0640a223
         20:fe0003e3
         21:0000003c
         22:0000003c
         23:0000007b
```

Figure 5.30: Initial Memory data

Below Result showing that after we successfully run program, we are successful in getting a series of Fibonacci till 5 digit as desired in which to get present value we need addition of previous two values.



```
M100 - 0
M101 - 1
M102 - 1
M103 - 2
M104 - 3
M105 - 5
M106 - 0
M107 - 0
M108 - 0
M109 - 0
M110 - 0
M111 - 0
INFO: [USF-XSim-96] XSim completed. Design snapshot 'RISC_fibonacci_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 5.31: Updated Memory

H. **Tabular Depictation**:

This table shows that a clear idea on working of instructions in the processor and updating registers Data verifying that processor is in running stage.

Table 5.2: Simulated Reg values

Registers	Value before simulation	Values after Simulation
R0	0	0
R1	1	10
R2	2	10
R3	3	5
R4	4	4
R5	5	5
R6	6	15
R7	7	7
R8	8	8
R9	9	9
R10	10	1
R11	11	10
R12	12	5
R13	13	13
R14	14	1920
R15	15	15
R16	16	16
R17	17	7
R18	18	18
R19	19	19
R20	20	1
R21	21	21
R22	22	22
R23	23	23
R24	24	24
R25	25	25
R26	26	26
R27	27	27
R28	28	28
R29	29	29
R30	30	15
R31	31	20
M104	5	5

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

- The RISC-V ISA stands out from other ISAs for its easy to understand and use qualities, as well as the fact that all of the instructions are simple to schedule and implement hazard detection.
- The main goal is to contrast the machine's performance when running the application with and without bypassing. Imagine that the same line of code is run without being bypassed. The number of clocks needed for each instruction will then be higher since the stall will last for a few cycles and wait for the execution of the preceding instructions to be finished before writing the instruction back to the register file.
- Bypassing directly from the execute stage or the memory stage for arithmetic bypassing or memory bypassing is possible with the forwarding logic. As a result, fewer clock cycles are needed, which accelerates the execution ofthe instruction onour system.
- We are Successful in developing a whole Schematic designing with interconnections between varietyof domains.
- Today's world requires high-speed processors that can carry out our commands with the least amount of Cycles per instructions, therefore bypassing offers a substantial advantage to meet that need.
- As it is open source, anyone can create original designs and work as a developer. In addition, IIT Madras launched the SHAKTI Processor effort to improve its utility.

REFRENCES

[1] Computer Organization and Design, The Hardware and software interface (RISC-V edition) by David A. Patterson, John L. Hennessy, The Morgan Kaufmann Series in Computer Architecture And Design , UC Berkeley,2020.

[2] E. Cui, T. Li and Q. Wei, "RISC-V Instruction Set Architecture Extensions: A Survey," in IEEE Access, vol. 11, pp. 24696-24711, 2023.

[3] B. W. Mezger, D. A. Santos, L. Dilillo, C. A. Zeferino and D. R. Melo, "A Survey of the RISC-V Architecture Software Support," in IEEE Access, vol.10, pp. 51394-51411, 2022.

[4] S. Prabhakaran, M. N and V. Vedanarayanan, "Design and Analysis of a Multi Clocked Pipelined Processor Based on RISC-V,"International Conference on Communication, Computing and Internetof Things (IC3IoT), Chennai, India, 2022, pp. 1-5.

[5] Zhang, Yunrui, Zichao Guo, Jian Li, Fan Cai, and Jianyang Zhou. "AnnikaCore: RISC-V Architecture Processor Design and Implementation for IoT." 15th International Conference on Anti-counterfeiting, Security, and Identification (ASID*)*, pp. 200-203. IEEE, 2021.

[6] A. Dörflinger, M. Albers, B. Kleinbeck, Y. Guan, H. Michalik, R. Klink, C. Blochwitz, A. Nechi, and M. Berekovic, ‘‘A comparative survey of open-source application-class RISC-V processor implementations,’’ in Proc. 18th ACM International Conference on computing Frontiers, pp. 12–20, May 2021.

[7] M. N. Topiwala and N. Saraswathi, "Implementation of a 32-bit MIPS based RISC processor using Cadence,"IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, India, pp. 979-983,2014.

[8] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, "Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL," International Conference on Advanced Computingand Communication Systems, Coimbatore, India, pp. 1-6,2015.

[9] W. P. Kiat, K. M. Mok, W. K. Lee, H. G. Goh and I. Andonovic, "A Comprehensive Analysis on Data Hazard for RISC32 5-Stage Pipeline Processor,"IEEE, 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA*)*, Taipei, Taiwan, 2017, pp. 154-159,2017.

[10] A. Pandey, "Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor," International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, pp. 1-4,2016.

[11] Miyazaki, H., Kanamori, T., Islam, M.A. and Kise, K., RVCoreP: An optimized RISC-Vsoft processor of five-stage pipelining. IEICE TRANSACTIONS on Information and Systems, 103(12), pp.2494-2503,2020.

[12] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer and M. Linauer, "Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation," 8th Mediterranean Conference on EmbeddedComputing (MECO), Budva, Montenegro, pp. 1-6, 2019.

[13] G. Liu, J. Primmer, and Z. Zhang, "Rapid Generation of High-Quality RISC-V Processors from Functional Instruction Set Specifications," in Proceedings of the 56th Annual Design Automation Conference, pp. 1–6,Jun 2019..

[14] C. Palmiero, G. Di Guglielmo, L. Lavagno and L. P. Carloni, "Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications," IEEE High Performance extreme Computing Conference (HPEC*)*, Waltham,MA, USA, pp. 1-7,2018.

turnitin **Similarity Report ID**: oid:27535:36494262

PAPER NAME

FINAL_REPORT-12-65.pdf

WORD COUNT	CHARACTER COUNT
5763 Words	30264 Characters
PAGE COUNT	FILE SIZE
54 Pages	3.1MB
SUBMISSION DATE	REPORT DATE
May 30, 2023 12:24 PM GMT+5:30	May 30, 2023 12:24 PM GMT+5:30

● 6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 1% Internet database
- 1% Publications database
- Crossref database
- Crossref Posted Content database
- 5% Submitted Works database

[illegible] 30.05.23

● 6% Overall Similarity

Top sources found in the following databases:

- 1% Internet database
- 1% Publications database
- Crossref database
- Crossref Posted Content database
- 5% Submitted Works database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Texas A&M University, College Station on 2022-09-13 Submitted works	2%
2	Chaithanya D J, Ibrahim Rashad, Mohammed Thouqeer, Mohd Taha W... Crossref	<1%
3	Technological Institute of the Philippines on 2022-10-06 Submitted works	<1%
4	Manipal University on 2021-08-18 Submitted works	<1%
5	CSU, San Jose State University on 2014-03-09 Submitted works	<1%
6	Coventry University on 2022-04-19 Submitted works	<1%
7	Virginia Commonwealth University on 2021-12-14 Submitted works	<1%
8	University of Wales Institute, Cardiff on 2022-09-04 Submitted works	<1%

9	coursehero.com Internet	<1%
10	projects.tuni.fi Internet	<1%
11	University of London External System on 2020-09-16 Submitted works	<1%