Implementation of 32-bit 5-stage pipelined RISC-V processor

by

Ali Imran
2018-EE-062

Advisor:
Mr. Umer Shahid

FALL 2021

Department of Electrical Engineering
University of Engineering and Technology, Lahore

Course Code and Title: EE-475: Computer Architecture
Semester: Fall 2021 (7th Semester)
Instructor: Dr. Muhammad Tahir & Mr. Umer Shahid
Total Marks: 50 (in Lab)
Deadline: End of Semester

CLOs and PLOs for Complex Engineering Problem

Below stated are the CLOs and PLOs addressed in the complex engineering problem along with domain and level. These are the CLOs from the theory/lab course which are already defined.

CLOs		Description	Domains & Levels	PLOs, Levels
CLO1	Theory	Demonstrate the skills to design datapath of single-cycle, multi-cycle, and pipeline microprocessor architecture using a variety of techniques	Cognitive, 2. Understand	PLO1
CLO2	Lab	Formulate the control portion of a processor, and its data path consisting of the general-purpose register file, ALU, the memory interface, internal registers between stages, and multiplexers.	Psychomotor, 4. Articulate	PLO3
CLO3	Lab	Design and emulate a pipelined CPU by given specifications using Verilog	Psychomotor, 5. Naturalization	PLO3

Problem Statement

In this open-ended design lab, students will investigate and design a 32-bit RISC-V processor with five stage pipeline.

- Implement a processor that supports a subset of the RISC-V ISA. The designed processor does not support all RISC-V ISA instruction, however it will have most parts that a real processor has: A fetch unit, decode logic, functional units, a register file, I/O support, access to memory, interrupt handling, hazard elimination protocols, and CSR Support.
- Students will be implementing the datapath while designing a pipelined RISC-V processor which will work with five stages mainly, It must contain a file or block RAM of FPGA that will be used as Random Access Memory. It will have 32 Registers working as General purpose Register. While some special purpose registers (Program Counter, CSR registers) will be used to hold the address of the instruction and will handle the interrupts. Each Register must be 32 bit wide and clock edge triggered.

ABSTRACT

Pipelined processors allow us to increase processing thoroughput which allows us to have a cycles per instruction of 1 and higher clock frequency than single cycle processors. As the number of stages increases, time period decreases and hence processor can work on greater clock frequency as compared to a single cycle processor. In this complex engineering problem, RISC-V instruction set architecture based 5 stage pipelined processor has been implemented. As the name suggest, instruction was divided into 5 different stages for its complete execution. We implemented six base instructions named as R-type, S-type, I-type, B-type, J-type and U-type and we also implemented some CSRs instructions to configure the CSR registers. We have also removed data dependency and control hazards in our implementation.

ACKNOWLEDGMENTS

It is a genuine pleasure to express our deep sense of thanks and gratitude to our course (Computer Architecture) instructor **Dr. Muhammad Tahir**, Chairman Electrical Engineering Department, University of engineering and technology, Lahore. His timely and scholarly advice and scholarly approach have helped us to a very great extent to accomplish this task.

We owe a deep sense of gratitude to **Mr. Umer Shahid.** His prompt inspirations, timely suggestions with kindness, enthusiasm and dynamism have enabled us to complete this work.

STATEMENT OF ORGINALITY

It is stated that this CEP presented in this course consists of my own ideas and hands-on working. The contributions and ideas from others have been duly acknowledged and cited in the dissertation. This complete report and code is written by me.

[Ali Imran]

LIST OF FIGURES

Table of Contents

1. INTRODUCTION

1.1. Design Description

Microarchitecture is a connection between logic and architecture. For implementation of RISC-V ISA, different microarchitectures are created to convert the ISA logic into a hardware architecture. Microarchitecture is the particular plan of registers, number juggling rationale units (ALUs), limited state machines (FSMs), recollections, and other rationale building blocks expected to execute a design. A specific design, like RISC-V, may have a wide range of microarchitectures, each with various compromises of execution, cost, and intricacy. They all run similar projects, yet their interior plans fluctuate broadly. The three distinct designs incorporate single cycle, multicycle and pipeline engineering.

Microarchitectures are partitioned into two communicating parts: the data path and the control unit. The datapath works on expressions of information. It contains designs like recollections, registers, ALUs, and multiplexers. We are carrying out the 32-bit RISC-V (RV32I) design, so we use a 32-digit datapath. The control unit gets the current guidance from the datapath and tells the datapath how to execute that guidance. In particular, the control unit produces multiplexer select, register enable, and memory write signals to control the operation of the datapath.

Execution time of a code is highly dependent on the cycles per instruction(CPI) and the frequency of the clock. And single cycle architecture donot allow us to work on high frequencies because of the propogation delay. The multicycle processors are used for having higher frequency but the CPI increases. So for increasing program execution

performance, pipelined processors are used because they work at high frequency and their throughput increases which decreases the CPI as compared to multicycle processor.

1.2. Datapath Explanation

We are implementing pipelined processor with five stages which include Fetch, Decode, Execute, Memory, and Writeback. In the Fetch stage, the processor reads the instruc tion from instruction memory. In the Decode stage, the processor reads the source operands from the register file and decodes the instruction to produce the control signals. In the Execute stage, the processor performs a computation with the ALU. In the Memory stage, the processor reads or writes data memory, if applicable. Finally, in the Writeback stage, the processor writes the result to the register file, if applicable.

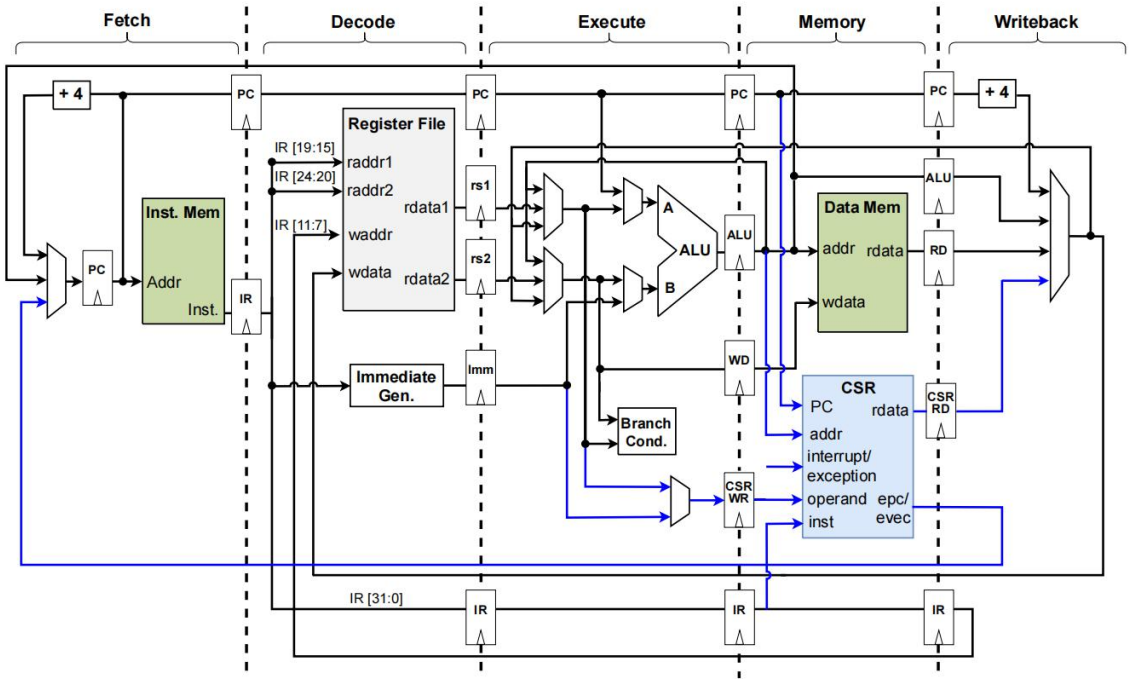


Figure 1 Pipelined RISC V architecture

1.2.1. Highlighted Datapath for R-type Instructions

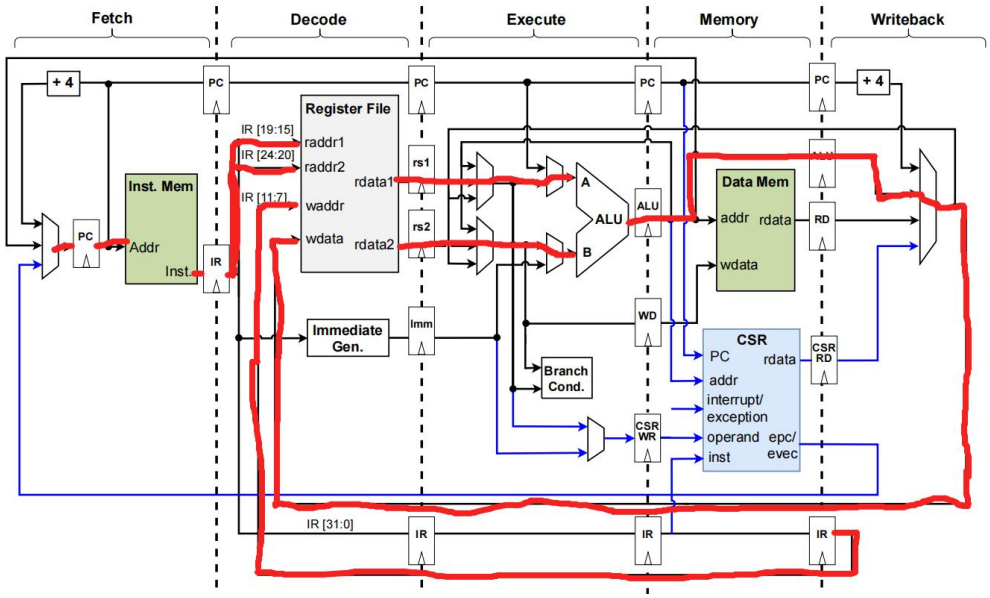


Figure 2 R Type Instruction Path Highlighted

1.2.2. Highlighted Datapath for I-type Instructions

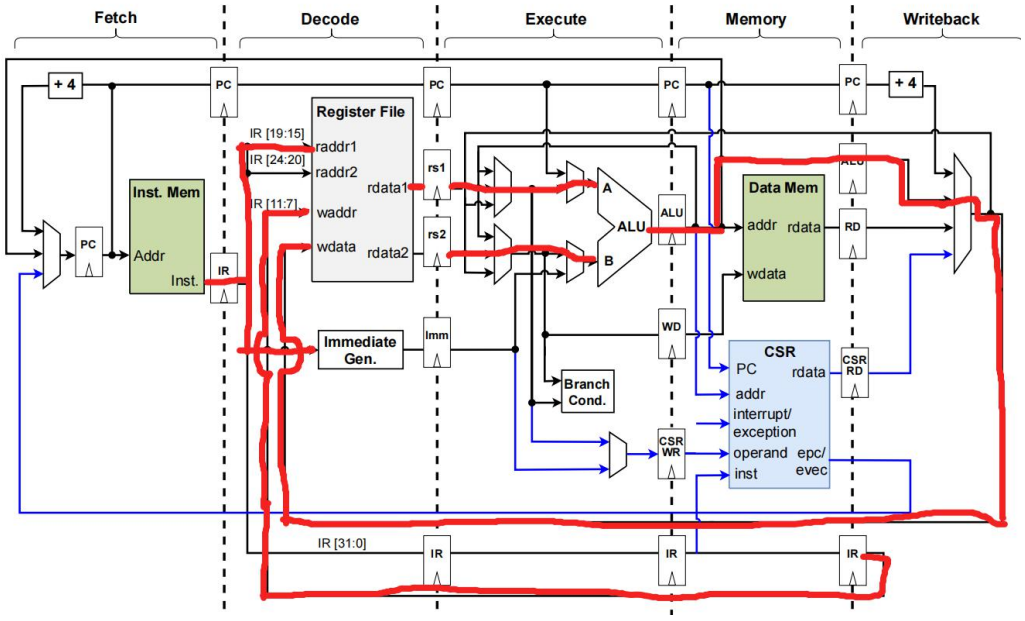


Figure 3 I Type Instruction Path Highlighted

1.2.3. Highlighted Datapath for S-type Instructions

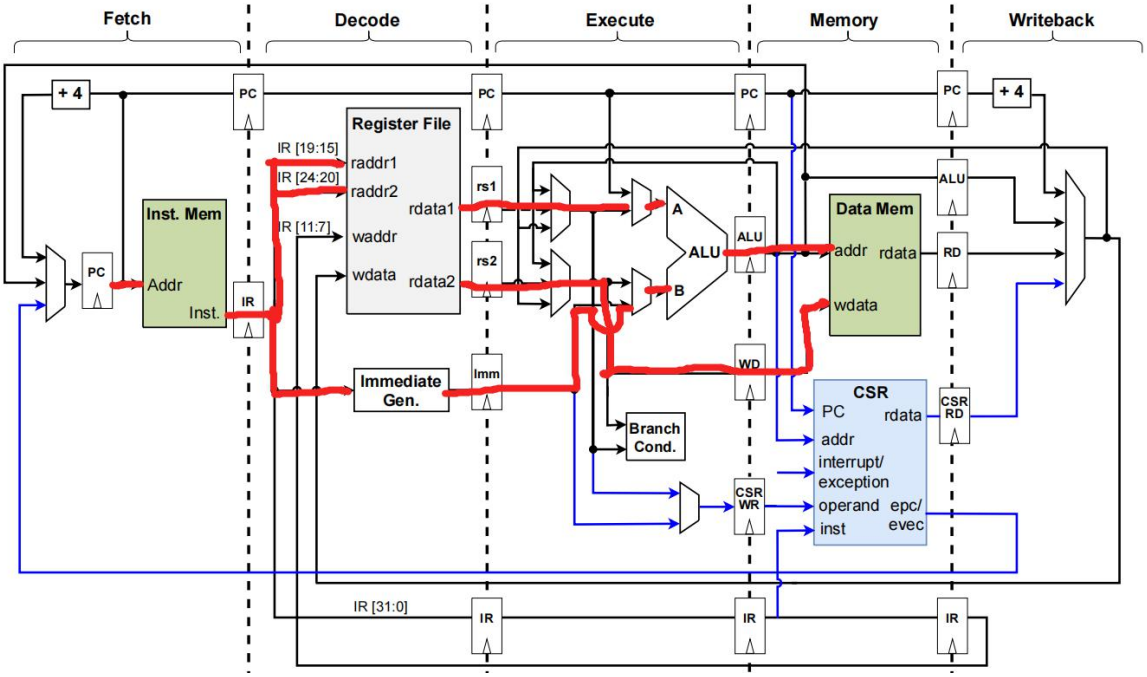


Figure 4 S Type Instruction Path Highlighted

1.2.4. Highlighted Datapath for B-type Instructions

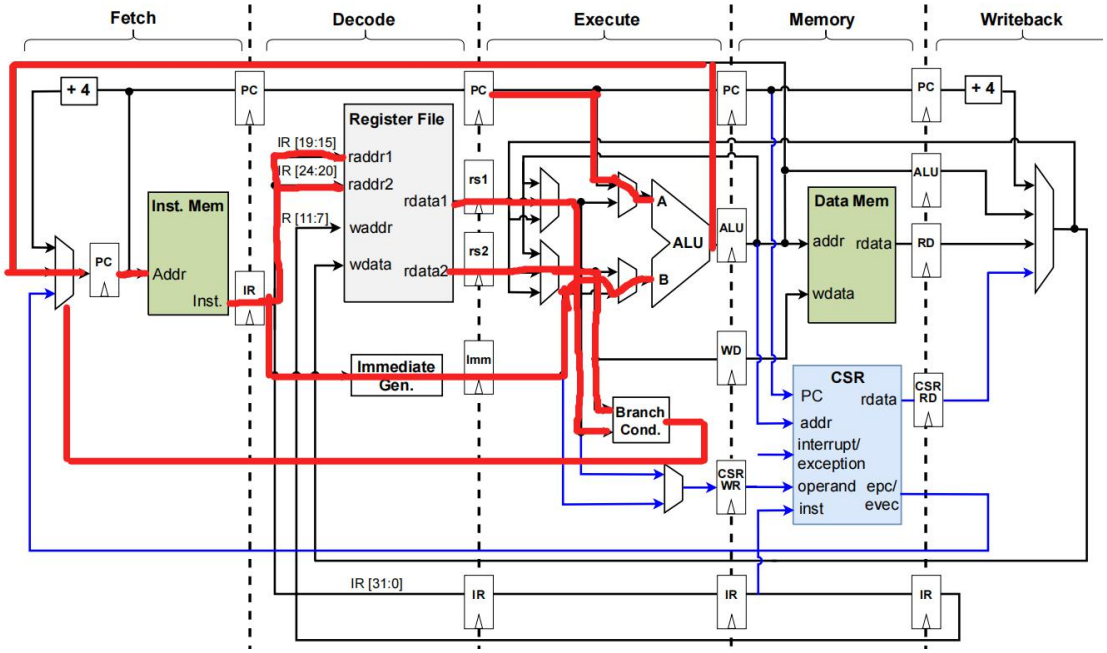


Figure 5 B Type Instruction Path Highlighted

1.2.5. Highlighted Datapath for J-type Instructions

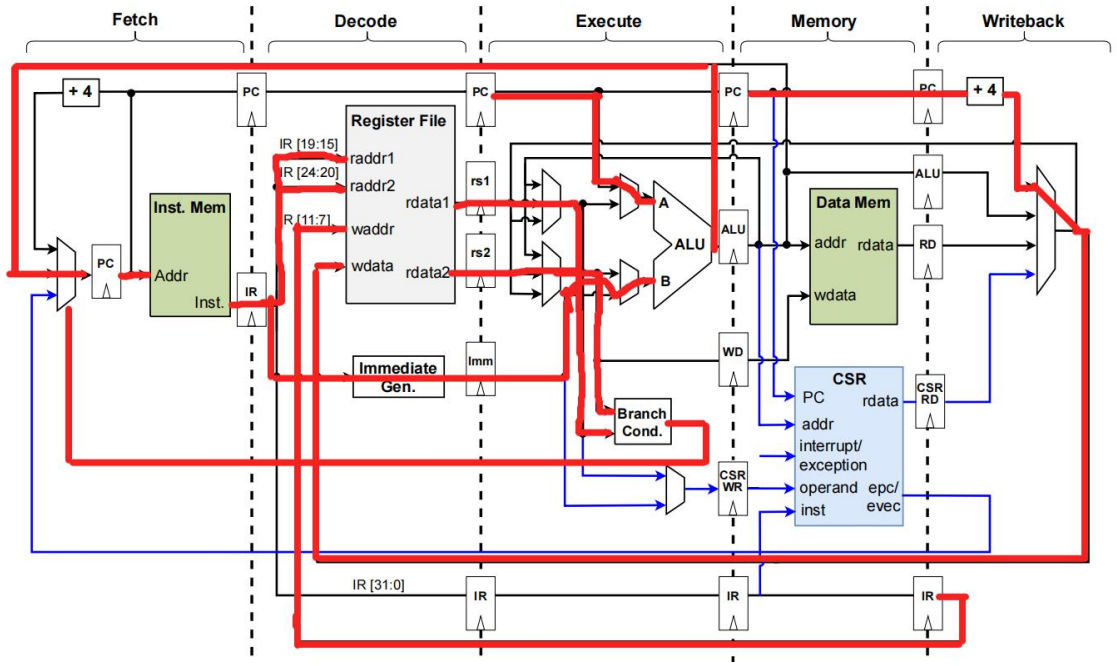


Figure 6 J Type Instruction Path Highlighted

1.2.6. Highlighted Datapath for Forwarding Hazards

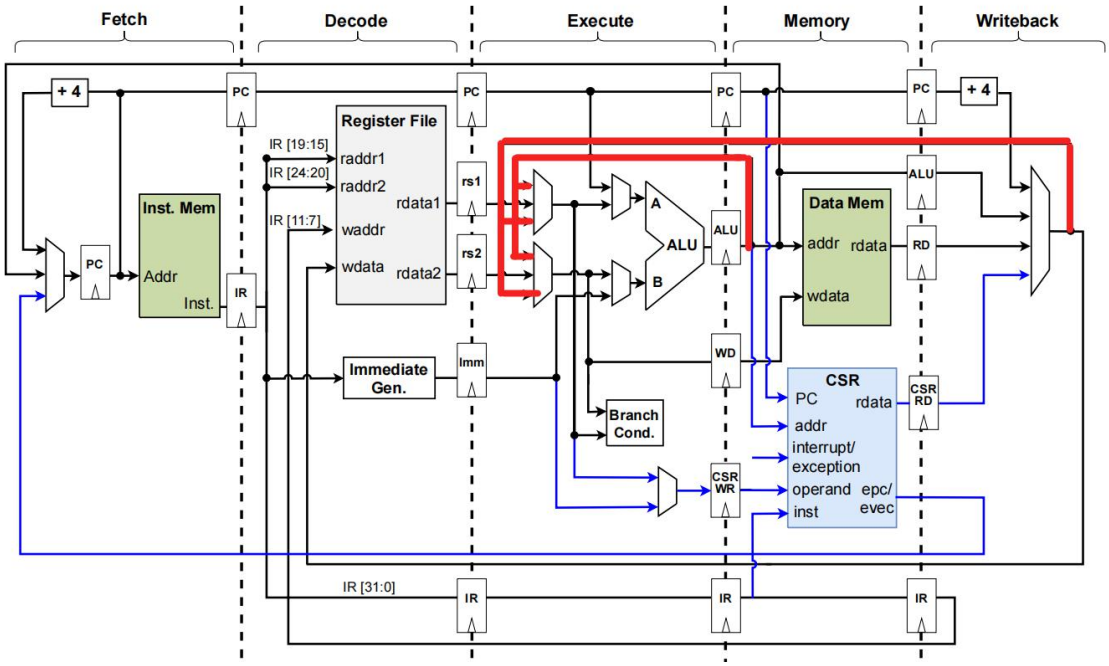


Figure 7 Datapath Hazards Forwarding Highlighted

1.2.7. Highlighted Datapath for CSR Instructions

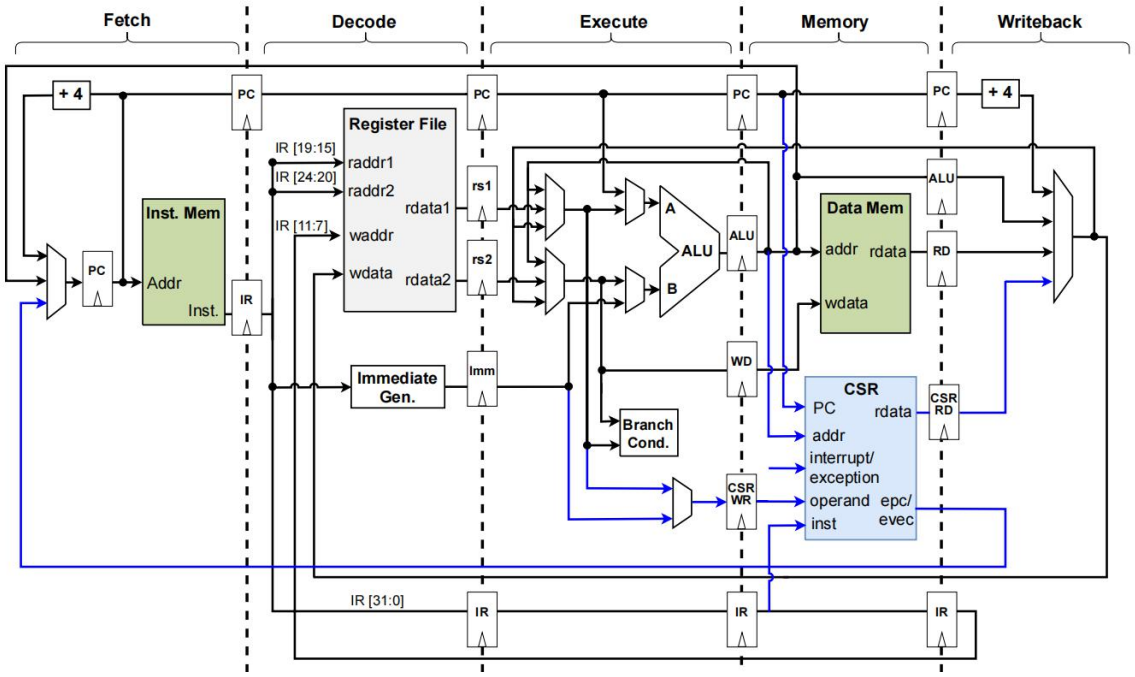


Figure 8 CSR instructions Path Highlighted

1.3. Potential Hazards

In a pipelined architecture, multiple instructions are handled concurrently. When one instruction is dependent on the results of another that has not yet completed, a hazard occurs. In case of our pipeline we had potential hazard of read after write data dependency hazard and control hazards which have been removed from our pipeline using forwarding and stalling for removing read after write hazards and for removing control hazards flushing has been performed.

2. Modules

2.1. The Top module

The top module of our design contains multiple modules including the control unit, register file, data memory, ALU, hazard controller, forwarding muxes, CSR module and a timer interrupt module used to create interrupts for testing the processor.

2.2. CPU Control Unit

For this datapath, all the RV32I instructions have been implemented. The Controller generates different control signals(alu_op, reg_wr, sel_A, sel_B, wr_en, rd_en, unsign, br_type, wb_sel) for the different datapath modules based on the Instruction available at the input of the controller.

2.3. Memory Module

The Data_Memory module creates the data memory from which data can be accessed or can be written onto based on the control signals(wr_en and rd_en).

2.4. Register File

For register file, the data available at two specified addresses(raddr1 and addr2) will be available at the output(rdata1 and rdata2). And data(wdata) will be written at the specified address(waddr) when the register write control signal(reg_wr) is set. The 31 general purpose registers have been declared as the output for debugging purposes.

2.5. ALU

The ALU will perform different operations on the two inputs(A and B) based on the control signal(alu_op) and the Branch Condition module performs different comparisons for the B-type based on the br_type control signal to generate the control signal(br_taken) for the B-type and J-type instructions which will be the input to the Program_Counter module.

2.6. Hazard Controller and Forwarding Unit

The forwarding unit compares the destination register of the previous and the second previous instruction with the source registers of the current instruction and forwards the ALU output from the memory and the writeback stage to the ALU inputs at the execution stage based on the comparison and the register write signals at the memory and writeback stage. For the lw instruction, we need to stall the pipeline. We can identify the hazard by the wb_selE signal and we compare rd_E with rs1_D and rs2_D. We add stall signal to the Program Counter(Fetch Stage) and the Decode pipeline register. And the a Flush signal is added to the Execute pipeline register to clear the garbage values. For the branch and jump hazards, we are going to flush the Decode and fetch stage incase the br_taken signal becomes true.

2.7. CSR

Three csr instructions csrrs, csrrw and mret are added. Whenever an interrupt arrives the PC value at writeback stage is stored in mepc and the Decode, Execute and Writeback stages will be flushed after which the PC will be updated based on values of register

mtvec and mcause when timer arrives the mcause[30:0] becomes 1 and the value of 'PC' will become mtvec+ mcause[30:0] * 4 at that position the jump instruction to the handler will be available. So the handler will be executed then at the end we will return from handler to normal execution using mret. In this case on the execution of handler the x3 register values will be toggled.

3. Tests

3.1. Simulation Tool Description

For simulation purposes two different simulator tools have been used which include *iverilog* and *Cadence Xcelium*. The first one is an open source tool and the second one is a commercial tool which gives accurate results as compared the the open source tools. *Cadence Xcelium* not only allows us to test our code but it also allows us to check the coverage of our tests which means that by increasing coverage we can verfiy our processor thoroughly and we can remove errors before moving into the manufacturing stage of the processor.

3.2. Steps to run a basic modular test

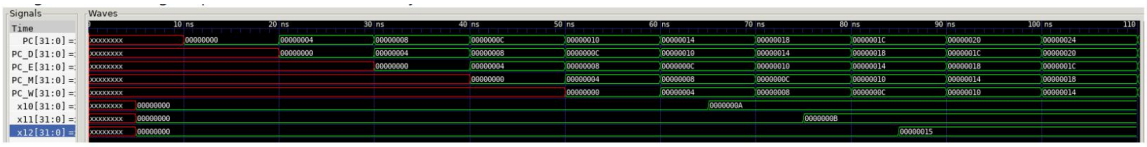
For running the tests, we need to provide the clock and the reset signal to the top module and then by changing the memory file (.mem) we can test different machine codes for verifying the functionality of the data. When using *xcelium*, we compile our testbench and design file which checks the code for synthesis errors. After that, we start the simulation and we can obesrve the behaviour of signals in the waveform and then by using the IMC coverage module we check the coverage of our tests.

3.3. Test performed and Correctness

Following code have been executed with their results for pipelined processor.

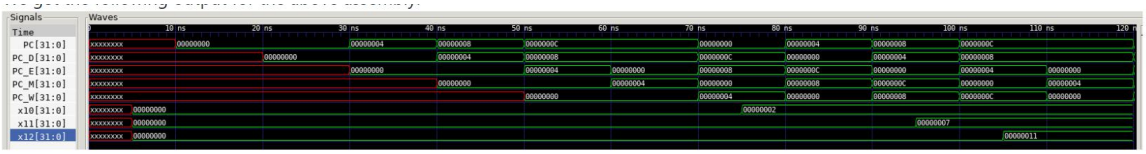
```
addi x10,x0,10
addi x11,x0,11
add x12,x10,x11
```

We get the following results for above code.



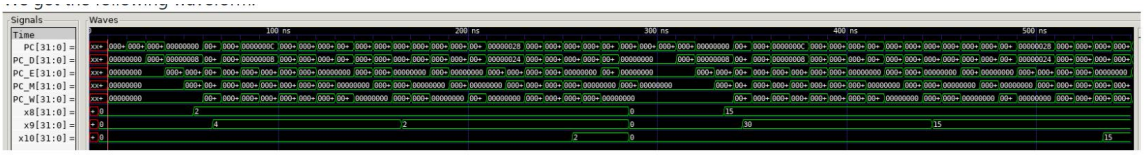
```
lw x10,0(x0)
addi x11,x10,5
addi x12,x11,10
```

We get the following results for above code.



```
lw x8, 0(x0)
lw x9, 4(x0)
gcd:
beq x8, x9, stop
blt x8, x9, less
sub x8, x8, x9
j gcd
less:
sub x9, x9, x8
j gcd
stop:
sw x8,8(x0)
lw x10,8(x0)
end:
j end
```

We get the following results for above code.



3.4. Coverage Report

For coverage different codes have been testes. One of the sample code is given below which has been configured with interrupt such that the bits of register are toggled when an interrupt comes.

```
j reset
j handler
handler:
xori x12,x12,0xFFFFFFFF
mret
reset:
addi x10,x0,0x80
addi x13,x0,1
nop
nop
nop
nop
nop
csrrw x0,mtvec,x13
csrrw x0,mie,x10
main:
addi x11,x11,1
j main
```

We get the following coverage for our processor.

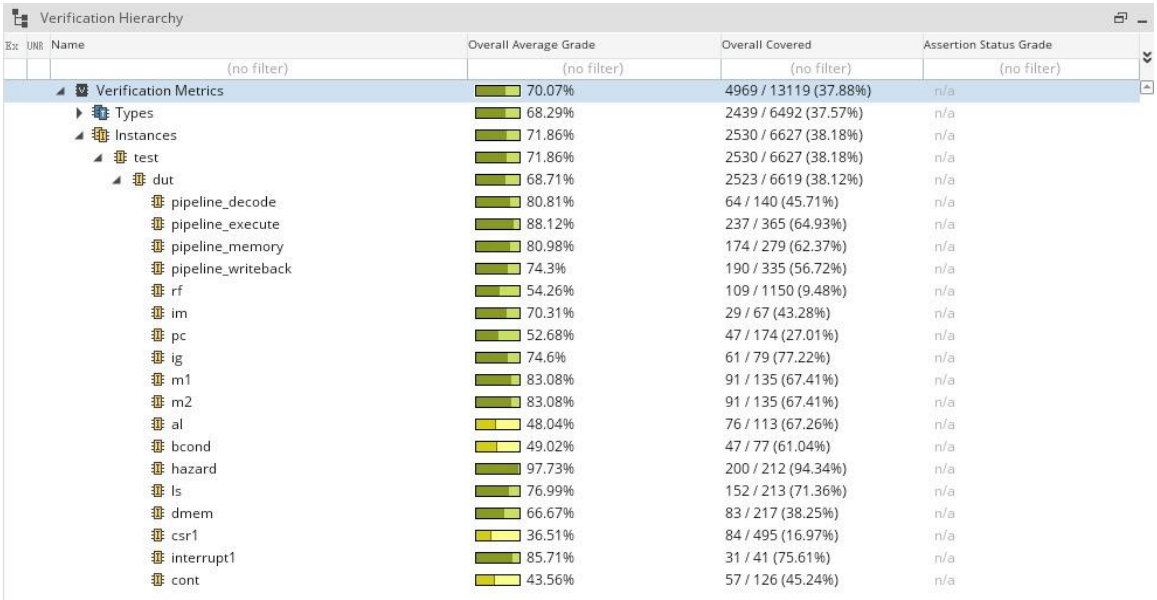


Figure 9 Code Coverage Results

3.5. Challenges and Limitations

The low code coverage in the above figure comes because of the low toggle coverage. The overall coverage can be increased by increasing the toggle coverage.

3.5.1. Challenges

The main challenges in this CEP arrived in the case of increasing the code coverage because of time limitations the extensive tests could not be written which might give us the maximum coverage.

3.5.2. Limitations

The major limitation for our design can observed in the recent machine code that we have added bubbles in the code in the form of nop because forwarding has not been implemented for csr instruction.

BIBLIOGRAPHY

[1] Harris, David, and Sarah L. Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2021.

APPENDIX

Complex Engineering Problem Attributes

<table>
<tr><td rowspan="6">WP1: Depth of knowledge
WP2: Range of conflicting requirements
WP3: Depth of analysis
WP4: Familiarity of issues
WP5: Extent of applicable codes
WP6: Extent of stakeholders
WP7: Interdependence</td><td colspan="2">WP1: Depth of Knowledge Requires knowledge of Digital Systems and Computer Architecture (WK4), design of Datapath and Controller for pipelined architecture (WK5), use of Modern Tools (Xilinx Vivado 2020.1) to program and test the code on FPGA (WK6) and engagement in research literature (WK8)
WP3: Depth of analysis Numerous approaches can be adopted. Choice of the selected algorithm requires in-detail analysis
WP5: Extent of applicable codes Requires going beyond the use of standardized features of Standard Verilog modules.</td></tr>
<tr><td colspan="2">Rubrics</td></tr>
<tr><td>Methodically investigates different design approaches and techniques to design datapath of pipelined microprocessor architecture. (CLO1)</td><td>WP1, WP3</td></tr>
<tr><td>Selects an optimal methodology to design the control portion of a processor, and its data path consisting of the general-purpose register file, ALU, the memory interface, internal registers between stages, and multiplexers with proper justification (CLO2)</td><td>WP3, WP5</td></tr>
<tr><td>Designs a comprehensive testbench system to test and verify the designed processor requirements subject to the given constraints (CLO3)</td><td>WP1, WP5</td></tr>
<tr><td>Identifies limitations and implications of the proposed solution (CLO3)</td><td>WP1</td></tr>
</table>

EA1: Range of resources EA2: Level of interaction EA3: Innovation	• ***EA1: Range of resources*** The design involves internet resources, information related to software usage and technology (modern end tools including Cadence).

EA4: Consequences for society and environment EA5: Familiarity	• ***EA3: Innovation*** Addressing sustainability and optimization of the design based on engineering principles and knowledge.	
	Rubrics	
	Methodically investigates different design approaches and techniques to design datapath of pipelined microprocessor architecture. (CLO1)	**EA1, EA3**
	Selects an optimal methodology to design the control portion of a processor, and its data path consisting of the general-purpose register file, ALU, the memory interface, internal registers between stages, and multiplexers with proper justification (CLO2)	**EA1, EA3**
	Designs a comprehensive testbench system to test and verify the designed processor requirements subject to the given constraints (CLO3)	**EA1, EA3**
	Identifies limitations and implications of the proposed solution (CLO3)	**EA3**

Complex Engineering Problem Rubrics Distribution

	Assessment Tool	**CLOs**	**WP**	**EA**	**Marks Distribution**
Grading Policy	Phase – I: Basic Constraint Design (Software Evaluation)	CLO1	**WP1, WP3**	**EA1, EA3**	5 (In Lab)
	Phase – II: Adding Hazards (Software Evaluation & Viva)	CLO2	**WP3, WP5**	**EA1, EA3**	5 (In Lab)
	Phase – III: Adding CSR Support (Software Evaluation)	CLO3	**WP1, WP5**	**EA1, EA3**	5 (In Lab)
	Final Hardware Evaluation on FPGA with External Viva	CLO3	**WP1**	**EA3**	25 (In Lab)
	Comprehensive Report	CLO1, CLO2	**WP1, WP3, WP5**	**EA1, EA3**	10 (In Lab)

Complex Engineering Problem Rubrics

	Good (8-10)	Average (4-7)	Bad (0-3)
Report (10 Marks)	References properly added in IEEE Format Template has been followed with all required sections.	References properly added in IEEE Format Template has not been followed and formatting is not properly done but all sections have been covered	If any section is missing.
Hardware Viva (10 Marks)	100% answer rate.	60-90% answer rate	0-50% answer rate
Software Viva (10 Marks)	100% Understanding.	60-90% Understanding	0-50% Understanding
Software Evaluation (15 Marks)	All Tests passed	RTL submitted with no errors but missing some tests	RTL has errors or is not complete
Hardware Working (5 Marks)	Perfectly/Minor issues in working with all constraints fulfilled	Working with not able to fulfill few constraints	Not Working/Working with not able to fulfill many constraints

VITA

My Name is Ali Imran and currently I'm a student of Electrical engineering (7^{th} semester) at EED, UET Lahore. I completed my matriculation from Crescent Model Higher Secondary School in 2016 and I completed Fsc (Pre-engineering) securing 95% marks from Government College University. Lahore, in 2017.

My final year specialization subjects include Machine Learning and Computer Architecture.

Finally, leave a blank page at the end of your report.