**Computer Architecture and Organization**

Assignment 2

**Implementation of RV32I Core**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



|  |  |
| --- | --- |
| Ali Aqdas | 285050 |
| Farheen Gul | 290352 |

**National University of Science & Technology**

June 3, 2022

# Objective

The objective of this assignment/semester project is to implement RISC-V Integer Core

# Tools Used

1. Quartus Prime Pro

# Implementation Overview

Our implementation of pipelined RISC-V 32-Bit Integer Core is implemented to include all branches, arithmetic, load/store word operations. The core includes a forwarding and hazard detection unit to overcome Data Hazards and Control Hazards. The Verilog code is attached along this report in LMS Portal.

# Block Diagram

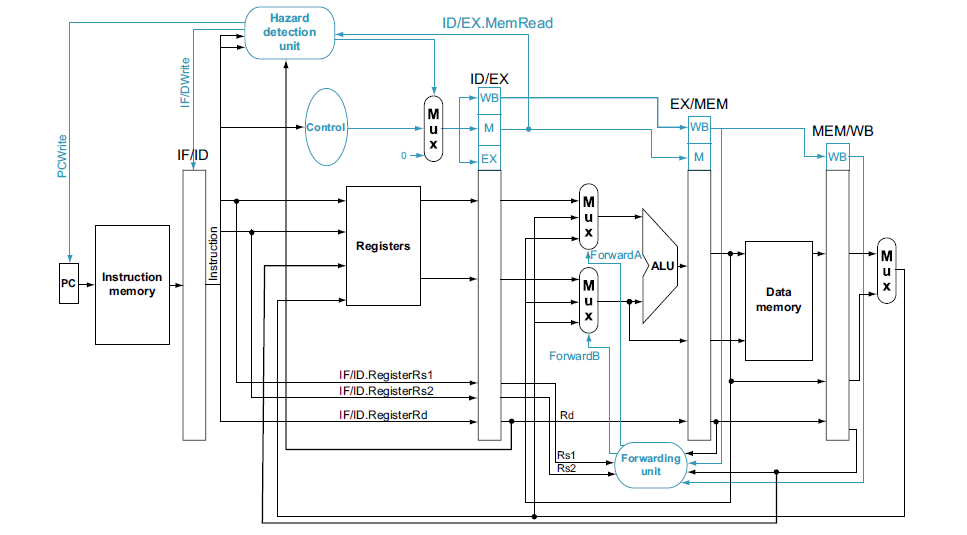


Figure Courtesy: Computer Organization and Design, David Patterson and John L. Hennessy

# Instruction Implemented

|  |  |  |
| --- | --- | --- |
| **Arithmetic** | **Immediate** | **Branches** |
| ADD | ADDI | BEQ |
| SUB | ORI | *BNE* |
| OR | ANDI | *BGE* |
| AND | XORI | *BLT* |
| XOR | SLTI | *BGEU* |
| SLT | *SLTIU* | *BLTU* |
| *SLTU* | *SLLI* | **Upper Immediate** |
| *SLL* | *SRLI* | *LUI* |
| *SRL* | *SRAI* | **Memory Accesses** |
| *SRA* |  | LW/SW |

# ALU

ALU incorporates arithmetic operations which include addition, subtraction, logical operations etc. It outputs a zero bit and ALU Result. The zero bit is used to control branch instructions. The ALU Result is sent to be written back to memory or is used as a memory address depending on the control signal.

# Register File

A register file is implemented with 32 elements, each 32-bit wide. The register file is implemented to first write and then read to overcome structural hazards.

# Instruction/Data Memory

An instruction memory and a data memory has been implemented in a test bench. Instruction Memory loads instruction from a **dat** file. The data memory receives addressing and input data from main core and outputs the data back to MEM Stage in the core.

# Control Unit

A control unit uses instruction OPCODE to generate control signals. A brief list of control signals is as follows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PCSrc | ALUCtrl | ALUSrc | MemWrite | MemRead | RegWrite |

# Immediate Generation Unit

An immediate generation unit uses a case chain to check OPCODE and generate immediate value based on instruction encoding. The immediate value generated is sign-extended 32-bit to be used in ALU and PC Logic.

# Hazard Detection

A hazard detection unit generates stalls when a load instruction is succeeded by an arithmetic or a branch using the loaded variable in a register. To generate a stall IFID Pipeline and PC is frozen and Control Signals are flushed.

A stall is also generated when a branch instruction is loaded from Instruction Memory. This is done by testing instruction in ID Stage. The PC is frozen, and IFID Pipeline Register is flushed. In the next cycle, the PCSrc remains frozen and IFID Pipeline is not updated. Following this cycle everything is resumed to normal after the branch decision has been made. PC is then updated depending on the decision.

# Forwarding Unit

A forwarding unit forwards data from EXMEM and MEMWB pipeline to overcome any data dependency. In case data dependency is occurring with both the pipeline registers satisfying the conditions data is taken from EXMEM Stage to keep up with the most recent data.

# FYP Progress

We have been surveying literature to get ourselves familiar with the current version of RISC-V Vector Extension. We have previously studied related chapters of two books, namely **Computer Organization and Design** and **Computer Architecture: A Quantitative Approach** both by David Patterson and John L. Hennessy. Additionally, we have read some papers related to vector processing to get a big picture of current advancements in Vector Processing Technologies and its history. We are currently reviewing RISCV-V 1.0 Specification sheet and preparing a presentation, gathering all the material we have learned so far at one place. It would in itself provide an overview of what vector processors are, how they are working and details of working of RISC-V Vector ISA.

# Conclusion

We have prepared a version of RV32I Core that would be used in collaboration with our Vector Extension as a part of our FYP. Designing a Hazard Detection unit was in our opinion the most tiresome thing as we have to keep in mind many working parts and fuse together control signals from multiple stages to generate stalls.