GROUP MEMBERS:

HUNAIN ABBASI

YOUSHA AMIR

MARVI NEK

COURSE: Digital System Design

Complex Engineering problem

Title :Design and Implementation of a 32-bit Pipelined Custom Processor Using Verilog HDL

Abstract

This project presents the design and implementation of a custom 32-bit processor architecture developed from scratch using Verilog HDL. The processor supports arithmetic, logical, and memory operations through a custom Instruction Set Architecture (ISA). A 5-stage pipelined datapath is implemented to improve performance and instruction throughput. The processor is verified using ModelSim simulation and organized as a modular and synthesizable design. This project demonstrates practical understanding of computer architecture, digital system design, and hardware implementation.

Introduction

Modern processors use pipelining and parallel execution to increase computational speed. The purpose of this project is to design a simplified yet fully functional processor architecture that demonstrates core concepts of CPU design such as datapath construction, control logic, instruction execution, and hardware verification.

This processor is designed as a 32-bit RISC architecture and implemented entirely in Verilog HDL. The design is modular to ensure clarity, maintainability, and synthesis capability.

Project Objectives

The main objectives are:

• Design a 32-bit processor architecture
• Develop a custom ISA
• Implement ALU and Register File
• Create datapath and control unit
• Interface instruction and data memory
• Implement pipelining
• Verify using ModelSim
• Upload project to GitHub

Processor Architecture Overview

The processor follows a RISC-based architecture. Each instruction completes through five stages:

IF → ID → EX → MEM → WB

The system consists of:

• Program Counter (PC)
• Instruction Memory
• Register File (32 registers)
• Arithmetic Logic Unit (ALU)
• Data Memory
• Control Unit
• Pipeline Registers

Instruction Set Architecture (ISA)

The processor uses 32-bit instructions.

Instruction Formats

R-Type

Used for arithmetic and logical operations.

Fields:
Opcode | rs | rt | rd | funct

I-Type

Used for memory and immediate instructions.

Fields:
Opcode | rs | rt | immediate

Supported Instructions

Instruction	Description
ADD	Addition
SUB	Subtraction
AND	Bitwise AND
OR	Bitwise OR
ADDI	Add immediate
LW	Load word
SW	Store word

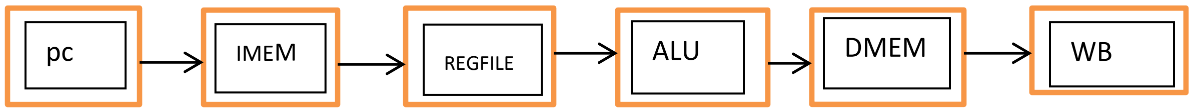
Datapath Design

The datapath controls the flow of data between components.

Steps:

1. Fetch instruction from memory
2. Decode instruction
3. Read operands

4. Perform ALU operation
5. Access memory if required
6. Write result back



Module Descriptions

ALU

Performs arithmetic and logical operations. Controlled by ALU_Sel signals.

Register File

Contains 32 general-purpose registers. Supports two reads and one write.

Program Counter

Holds address of current instruction.

Instruction Memory

Stores program instructions.

Data Memory

Stores load/store data.

Control Unit

Generates control signals such as:
RegWrite, MemRead, MemWrite, ALUSrc.

Pipeline Implementation

To enhance performance, a 5-stage pipeline is introduced.

Pipeline Stages

IF – Instruction Fetch

Instruction fetched using PC.

ID – Instruction Decode

Registers read and control signals generated.

EX – Execute

ALU performs operation.

MEM – Memory Access

Load/store operations performed.

WB – Write Back

Result written to register.

Pipeline Registers

- IF/ID
- ID/EX
- EX/MEM
- MEM/WB

These registers allow parallel instruction execution.

Control Signals

The control unit generates signals to manage datapath operation:

- RegWrite – enables register write
- MemRead – enables memory read
- MemWrite – enables memory write

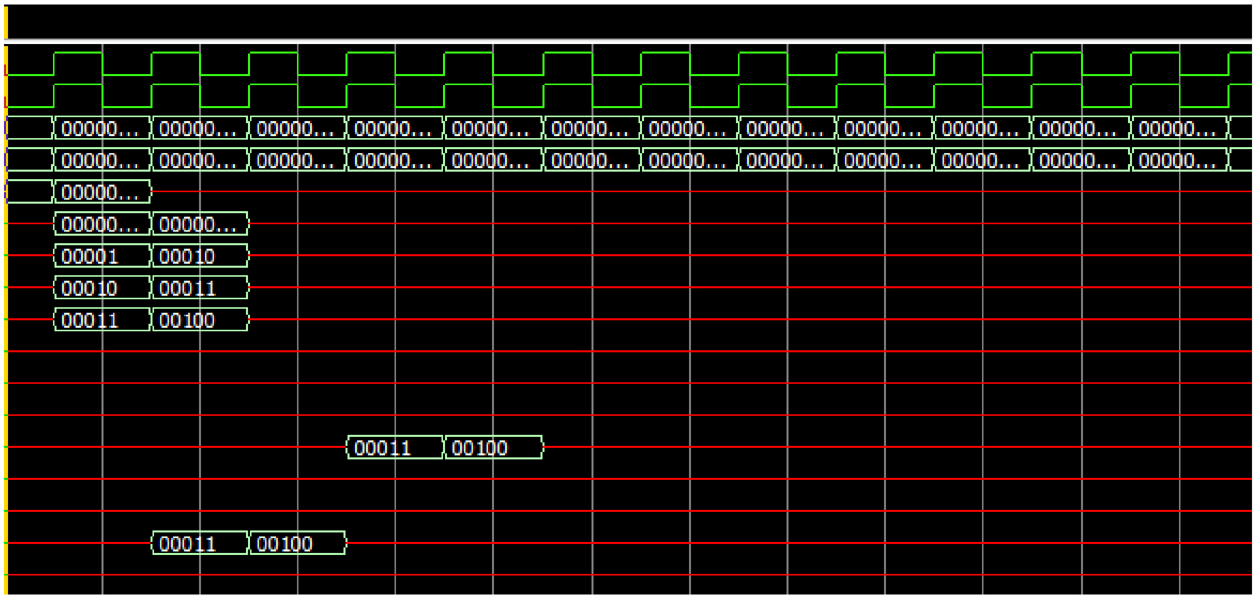
• ALUSrc – selects immediate or register operand
• Branch – controls branching
• Jump – controls jump instruction

Verilog Implementation

The complete processor is implemented using modular Verilog files:

• ALU.v
• RegFile.v
• PC.v
• Instruction Memory.v
• Data Memory.v
• Pipeline Registers.v
• CPU Top Module.v
• Testbench.v

The modular approach improves readability and debugging.



GitHub Repository

The project is uploaded to GitHub for version control and documentation.

GitHub provides:

• Code management
• Collaboration
• Professional portfolio
• Easy submission

Repository includes source code, README, and report.

Results

The processor successfully:

• Executes arithmetic instructions
• Performs memory operations
• Supports 32-bit data
• Executes multiple instructions simultaneously via pipelining

Performance improvement observed compared to single-cycle design.

Applications

This processor design can be used in:

• Embedded systems
• FPGA projects
• Educational CPU design
• RISC architecture studies

Conclusion

A complete 32-bit pipelined processor architecture was successfully designed and implemented in Verilog. The design demonstrates key concepts of digital system design and computer architecture. The pipelined implementation improves throughput and performance. The project fulfills all requirements of the Complex Engineering Problem.

Future Improvements

Possible enhancements:

• Hazard detection
• Forwarding
• Branch prediction
• Cache memory
• FPGA implementation
• Custom assembler

References

• Digital System Design lectures
• Computer Architecture textbooks
• ModelSim documentation
• Verilog HDL reference