RISC Processor Design Specification

Project Name: RISC Processor Design
Version: 1.0
Date: September 2023
Author: Akor Michael

Table of Contents

---

1. Introduction

Overview

This document provides the detailed design specification for a 16-bit Reduced Instruction Set Computer (RISC) processor. The processor is designed to be simple yet efficient, making it ideal for educational purposes and small-scale embedded systems. It is implemented in Verilog and can be synthesized for FPGA deployment.

Objectives

- **Functional:** Implement a 16-bit RISC processor capable of executing a set of basic instructions, including arithmetic, logic, memory access, and control flow.
- **Performance:** Optimize the design for minimal resource usage while maintaining acceptable processing speed.
- **Flexibility:** Ensure the processor can be easily modified or extended to support additional instructions or features.

---

2. Architecture

Processor Block Diagram

The processor is composed of the following key components:

- **Instruction Memory:** Stores the program instructions.
- **General Purpose Registers (GPRs):** Provide temporary storage for data and instructions.
- **ALU (Arithmetic Logic Unit):** Performs arithmetic and logic operations.
- **Control Unit:** Decodes instructions and generates control signals.
- **Data Memory:** Stores data that the processor can read from or write to.
- **Datapath Unit:** Routes data between the processor’s components.

Key Components

- **Instruction Memory:** 16-bit wide, capable of storing 15 instructions, but can be extended to store up to 4K instructions.
- **GPRs:** Eight 16-bit registers for general-purpose use.
- **ALU:** Supports basic arithmetic operations (addition, subtraction) and logic operations (AND, OR, NOT).
- **Control Unit:** Generates control signals based on the opcode and function bits of the instruction.

---

3. Instruction Set Architecture (ISA)

Supported Instructions

The processor supports the following instructions:

Opcode	Mnemonic	Operation	Description
0000	LW	Load Word	Load data from memory

Opcode	Mnemonic	Operation	Description
0001	SW	Store Word	Store data to memory
0010	ADD	Add	Add two registers
0011	SUB	Subtract	Subtract two registers
0100	AND	AND	Logical AND
0101	OR	OR	Logical OR
0110	NOT	NOT	Logical NOT
0111	XOR	XOR	Logical XOR
1000	BEQ	Branch if Equal	Branch if two registers are equal
1001	BNE	Branch if Not Equal	Branch if two registers are not equal
1101	J	Jump	Jump to an address

Instruction Format

Each instruction is 16 bits wide and follows a consistent format:

15-12	11-9	8-6	5-3	2-0
Opcode	rs	rt	rd	Func

- **Opcode:** The operation code, determining the instruction type.
- **rs, rt, rd:** Source and destination registers.
- **Func:** Additional function bits for more complex instructions.

---

4. Detailed Design

Datapath Design

The datapath is responsible for the flow of data within the processor. It consists of:

- **Program Counter (PC):** Keeps track of the current instruction address.
- **ALU:** Executes arithmetic and logic operations.
- **Register File:** Stores the values of registers rs, rt, and rd.
- **Memory Access:** Manages reading from and writing to data memory.
- **Multiplexers:** Control the flow of data between different components.

Control Unit Design

The control unit interprets the opcode of each instruction and generates the necessary control signals to drive the datapath. It consists of:

- **Opcode Decoder:** Decodes the 4-bit opcode.
- **Control Logic:** Generates control signals based on the decoded opcode.

ALU Design

The ALU performs operations based on the ALU control signals. It supports:

- **Arithmetic Operations:** ADD, SUB
- **Logic Operations:** AND, OR, NOT, XOR
- **Comparison:** Produces a zero flag for branch instructions.

Memory Design

- **Instruction Memory:** A read-only memory that stores the instructions to be executed.
- **Data Memory:** A read/write memory for storing data.

---

5. Interface Specifications

Input/Output Signals

- **clk (input):** The clock signal driving the processor.
- **reset (input):** Resets the processor to its initial state.
- **opcode (output):** The current instruction’s opcode.
- **zero_flag (output):** Indicates whether the ALU result is zero.

Timing Diagrams

Timing diagrams illustrate the signal transitions during instruction execution. For example:

- **Fetch-Execute Cycle:** Shows the progression from instruction fetch to execution.

---

6. Simulation and Testing

Testbench Design

The testbench is designed to verify the correctness of the processor:

- **Clock Generation:** Provides a clock signal to the processor.
- **Instruction Loading:** Loads a sequence of instructions into the instruction memory.
- **Assertions:** Checks the correctness of the processor’s outputs.

Simulation Strategy

- **Instruction Testing:** Simulate each instruction to ensure it functions correctly.
- **Integration Testing:** Simulate a program that uses multiple instructions to verify overall processor operation.

---

7. Future Work and Extensions

- **Pipeline Implementation:** Introduce pipelining to improve instruction throughput.
- **Additional Instructions:** Extend the ISA to support more complex operations.
- **Interrupt Handling:** Implement interrupt mechanisms for more complex control flow.
