# RISC Processor User Manual

---

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

---

### Table of Contents



---

## 1. Introduction

### Overview

This manual provides instructions on how to use the 16-bit RISC processor implemented in Verilog. The processor is designed to be used in educational settings or small-scale embedded systems. It supports a basic set of instructions and is implemented in a modular fashion to facilitate understanding and customization.

### System Requirements

- **Hardware**: FPGA development board (optional for hardware implementation)
- **Software**:
    - Verilog HDL simulator (e.g., ModelSim, Vivado)

- Text editor (e.g., VSCode, Sublime Text)
- Command-line tools (for compiling and running simulations)

---

## 2. Getting Started

### Installation

1. **Clone the Repository**: Download the RISC processor project files from the repository.

   ```
   https://github.com/Akor-Michael/RISC-Processor-16-Bit
   ```

2. **Set Up the Environment**: Ensure you have a Verilog simulator installed. Follow the simulator's installation guide if needed.

### Folder Structure

The project files are organized as follows:

```
risc-processor/
├── src/
│   ├── Risc_16_bit.v           # Top-level module
│   ├── Datapath_Unit.v         # Datapath module
│   ├── Control_Unit.v          # Control Unit module
│   ├── ALU.v                   # ALU module
│   ├── Instruction_Memory.v    # Instruction Memory module
│   ├── Data_Memory.v           # Data Memory module
│   └── GPRs.v                  # General Purpose Registers
module
├── testbench/
│   ├── test_Risc_16_bit.v      # Testbench for the processor
│   └── simulation_time.v       # Simulation timing parameters
├── docs/
│   ├── Design_Specification.pdf # Design specification document
│   └── User_Manual.pdf         # User manual (this document)
└── README.md
```

---

## 3. Using the Processor

### Writing Assembly Code

Write your program in assembly language using the instruction set supported by the processor. An example program might look like this:

```
LW R1, 0x0004   // Load word from memory address 0x0004 into R1
ADD R2, R1, R3  // Add R1 and R3, store the result in R2
```

```
SW R2, 0x0008   // Store word from R2 into memory address 0x0008
J 0x0000        // Jump to address 0x0000
```

### Compiling and Loading Programs

1. **Compile the Assembly Code**: Convert your assembly code into machine code using an assembler that targets the processor's instruction format.
2. **Load the Program**: Write the machine code into the `Instruction_Memory.v` module or load it into the memory using a testbench script.

### Running Simulations

1. **Open the Testbench**: Open the `test_Risc_16_bit.v` file in your Verilog simulator.
2. **Run the Simulation**: Use the simulator's command to run the simulation.

   ```
   vsim -c -do "run -all" test_Risc_16_bit
   ```

3. **View the Results**: Analyze the output to verify the correct operation of your program.

---

## 4. Understanding the Processor

### Processor Architecture

The processor consists of a datapath and a control unit:

- **Datapath**: Handles the flow of data within the processor, including instruction fetching, decoding, execution, and memory access.
- **Control Unit**: Generates control signals based on the opcode of the current instruction to control the datapath.

### Instruction Set Overview

The processor supports a basic set of instructions, including:

- **LW (Load Word)**
- **SW (Store Word)**
- **ADD, SUB (Arithmetic Operations)**
- **AND, OR, NOT, XOR (Logical Operations)**
- **BEQ (Branch if Equal)**
- **BNE (Branch if Not Equal)**
- **J (Jump)**

---

## 5. Troubleshooting

### Common Issues

- **Simulation Doesn't Start**: Check if the clock signal in the testbench is toggling correctly.
- **Unexpected Output**: Verify that the program was loaded correctly into the instruction memory.

### Debugging Tips

- **Use Waveforms**: Most Verilog simulators allow you to view waveforms. Use this feature to observe the behavior of signals and identify issues.
- **Check Control Signals**: Ensure that the control signals generated by the Control Unit match the expected behavior for the given opcode.

---

## 6. Extending the Processor

### Adding New Instructions

1. **Modify the Instruction Set**: Define the new instruction in the `Control_Unit.v` module.
2. **Update the ALU**: Implement the logic for the new instruction in the `ALU.v` module.
3. **Test the Instruction**: Write test cases to ensure the new instruction operates correctly.

### Modifying the Datapath

- To change the processor's functionality, you may need to modify the `Datapath_Unit.v` module. For example, adding support for a new addressing mode or pipeline stage would involve updating the relevant components.

---