Design and Implementation of a Single-Cycle RISC-V Processor

1. Introduction
2. Processor Architecture
   2.1 Instruction Execution Flow
3. Implementation
4. Verification
5. Software Workflow
   5.1 Example C Program
   5.2 Compilation and Hex File Generation
6. GitHub Repository Setup
7. Conclusion

1. Introduction

This project presents the design, implementation, and verification of a **Single-Cycle RISC-V RV32I Processor**. The processor is developed using **Verilog HDL** and verified using **Icarus Verilog (iverilog) and GTKWave**. It is also tested for compatibility with **GCC**, allowing execution of compiled C programs.

2. Processor Architecture

The processor follows the **RISC-V RV32I** instruction set and is implemented as a **single-cycle processor**. The key components include:

- **Program Counter (PC)**: Keeps track of the current instruction.
- **Instruction Memory**: Stores program instructions.
- **Register File**: Contains 32 registers for computation.
- **ALU (Arithmetic Logic Unit)**: Performs arithmetic and logic operations.
- **Control Unit**: Decodes instructions and generates control signals.
- **Immediate Generator**: Extracts immediate values from instructions.
- **Data Memory**: Stores and retrieves data.

2.1 Instruction Execution Flow

1. **Fetch**: The instruction at the address specified by PC is fetched.

2. **Decode**: The control unit deciphers the instruction and determines the operation.
3. **Execute**: The ALU performs computations based on control signals.
4. **Memory Access**: Loads from or stores to data memory if applicable.
5. **Write-back**: Results are stored back in registers.

3. Implementation

The implementation consists of multiple Verilog modules:

- **riscv_core.v**: The top module integrating all components.
- **instruction_memory.v**: Stores RISC-V machine code.
- **register_file.v**: Implements the 32-register architecture.
- **alu.v**: Performs all arithmetic and logical operations.
- **control_unit.v**: Generates control signals based on instruction decoding.
- **immediate_gen.v**: Extracts immediate values for I-type and branch instructions.
- **data_memory.v**: Implements memory read/write operations.

4. Verification

The processor has been verified using:

1. **Testbench Simulation**: Implemented in riscv_core_tb.v.
2. **Icarus Verilog Simulation**: iverilog used for compilation and vvp for execution.
3. **Waveform Analysis**: GTKWave used to visualize signal transitions.
4. **GCC Compatibility**: The processor successfully executes compiled assembly and hex files.

5. Software Workflow

To test the processor, a simple C program is compiled and converted into a hex file:

5.1 Example C Program

```
#include <stdint.h>
```

```
int main() {
  volatile int a = 5;
  volatile int b = 3;
  volatile int c = a + b;
  return c;
}
```

5.2 Compilation and Hex File Generation

Using a **RISC-V GCC toolchain**, the program is converted to hex:

```
riscv32-unknown-elf-gcc -march=rv32i -mabi=ilp32 -S program.c -o program.s
riscv32-unknown-elf-gcc -march=rv32i -mabi=ilp32 -o program.elf program.c
riscv32-unknown-elf-objcopy -O verilog program.elf program.hex
```

The generated program.hex is loaded into instruction memory.

6. GitHub Repository Setup

The following files are prepared for GitHub:

- riscv_core.v: Processor source code.
- riscv_core_tb.v: Testbench.
- program.hex: Test program in machine code.
- Makefile: Automates compilation and simulation.
- README.md: Documentation for setup and execution.

7. Conclusion

The **Single-Cycle RISC-V Processor** has been successfully designed, implemented, and verified. It supports basic **RV32I** instructions and is compatible with GCC-compiled binaries. Future work includes optimizing performance, adding pipelining, and implementing additional instructions.