Python-based RISC-V Simulator Project Report

Team Details : Final Project Group 3

1. Anurag Ranga – PSU ID 965691727
2. Yashaswi Katne – PSU ID 963584934
3. Koushik Seggari – PSU ID 984570188
4. Prasanna Kumar Panchada – PSU ID 944709842

GIT HUB LINK : https://github.com/sashakatne/Python-based-RISC-V-Simulator.git

ABSTRACT

The Python-based RISC-V Simulator project, developed by Yashaswi Katne, aims to simulate the execution of machine code in the RISC-V Instruction Set Architecture (ISA). The simulator provides a detailed view of the changes in memory and registers, as well as the flow of data during runtime. The project supports both pipelined and non-pipelined execution, with features such as cache statistics and register updates. The main findings include the successful execution of various RISC-V instructions and the ability to visualize the internal workings of the CPU. This project was completed as part of the ECE508: Python Workshop course during the Spring 2024 term.

INTRODUCTION

BACKGROUND

RISC-V is an open standard ISA based on reduced instruction set computer principles. It is gaining popularity due to its simplicity and flexibility, making it suitable for a wide range of applications from embedded systems to supercomputers. The RISC-V ISA is designed to be scalable and is provided under open-source licenses, which do not require fees to use.

OBJECTIVE

The main objective of this project is to develop a Python application that can simulate the execution of RISC-V machine code, providing insights into the internal operations of the CPU. The simulator aims to help students and researchers understand the RISC-V architecture and the execution of machine code instructions.

PROBLEM STATEMENT

The project addresses the need for a tool that can simulate RISC-V machine code execution, allowing users to observe changes in memory, registers, and data flow during runtime. This tool is essential for educational purposes and for researchers working on RISC-V architecture.

METHODOLOGY

DESIGN AND APPROACH

The project is designed to simulate the execution of RISC-V instructions by breaking down each instruction into five stages: Fetch, Decode, Execute, Memory Access, and Register Update. The approach includes both pipelined and non-pipelined execution modes. The simulator also provides detailed output for each stage, including register values, control signals, and cache statistics.

Tools and Libraries

The project uses Python as the primary programming language. Key libraries include `argparse` for command-line argument parsing. The modular structure of the project allows for easy extension and maintenance.

Data Collection

The project uses predefined machine code files (`.mc` files) as input data for simulation. These files contain RISC-V machine code instructions that are executed by the simulator.

Implementation Details

The project is implemented with a modular structure, separating the components for pipelined and non-pipelined execution. Key modules include:
- `ALU.py`: Contains the Arithmetic Logic Unit (ALU) class.
- `IAG.py`: Contains the Instruction Address Generator (IAG) class.
- `control.py`: Contains the Control class for generating control signals.
- `memory.py`: Contains the Memory class for handling memory operations.
- `register.py`: Contains the Register class for managing CPU registers.
- `buffer.py`: Contains the Buffer class for pipelined execution.

The main execution logic is handled in `main.py`, which parses command-line arguments and coordinates the execution of instructions.

RESULTS

Output and Findings

The simulator successfully executes RISC-V machine code, providing detailed output for each stage of execution. The results include register values, cache statistics, and cycle counts. For example, the execution of the `factorial.mc` test case in pipelined mode resulted in 162 cycles, with detailed register updates and cache statistics.

Analysis

The results demonstrate the simulator's ability to accurately simulate RISC-V instructions and provide insights into the internal operations of the CPU. The detailed output helps users understand the execution flow and the impact of each instruction on the CPU state.

DISCUSSION

Interpretation

The results indicate that the simulator can effectively model the execution of RISC-V instructions, providing valuable insights for educational and research purposes. The detailed output for each stage of execution helps users understand the internal workings of the CPU.

Challenges

One of the main challenges was ensuring accurate simulation of pipelined execution, which was addressed by carefully managing data hazards and control hazards. Another challenge was providing detailed output for each stage without affecting the performance of the simulator.

Limitations

The current version supports only 32-bit RISC-V instructions and does not include advanced features such as branch prediction. The simulator also does not support multi-core execution or advanced memory hierarchies.

Future Work

Future enhancements could include support for 64-bit instructions, branch prediction, and a graphical user interface for better visualization. Additionally, support for multi-core execution and advanced memory hierarchies could be added to make the simulator more comprehensive.

Conclusion

The Python-based RISC-V Simulator project successfully simulates the execution of RISC-V machine code, providing detailed insights into the internal operations of the CPU. The project demonstrates the effectiveness of Python for developing educational

tools and highlights the potential for future enhancements. The simulator is a valuable tool for students and researchers working on RISC-V architecture.

References

- RISC-V Foundation. (n.d.). Retrieved from https://riscv.org/
- Python Software Foundation. (n.d.). Python. Retrieved from https://www.python.org/

- Satsuma Day Light (youtube.com)

- AntonLydike/riscemu: RISC-V emulator in python (github.com)

- Peer Discussion

- Generative AI for better understanding of concepts.