



Web-Based Simulator of Superscalar RISC-V Processors

Jiri Jaros, Michal Majer, Jakub Horky and Jan Vavra



Brno University of Technology, Faculty of Information Technology, Brno, CZ

Code Editor

The Code Editor interface includes a C code editor with a 'Compile' button, an assembly view, and a console for error messages. A red error message states: "riscv undeclared (first use in this function)".

Processor Architecture

The Processor Architecture settings allow users to configure cache size, associativity, and replacement policy. The Main Memory section shows a table of memory locations:

Label	Address
Array	1440
a	640
b	1040
L2	16
main	0
LCD	1456

The Memory Inspector shows a hex dump of memory up to the highest touched address.

Runtime Statistics

The Runtime Statistics dashboard displays key performance indicators:

- IPC: 1.12
- Clocks: 60
- Branch Prediction Accuracy: 75.00%

Additional charts include Static Instruction Mix, Dynamic Instruction Mix, and Functional Units utilization.

Instruction Details

The Instruction Details view for the instruction `beq tg1,x0,loopEnd` shows the instruction type, operands, branch prediction accuracy (100.00%), and a detailed pipeline timing diagram.

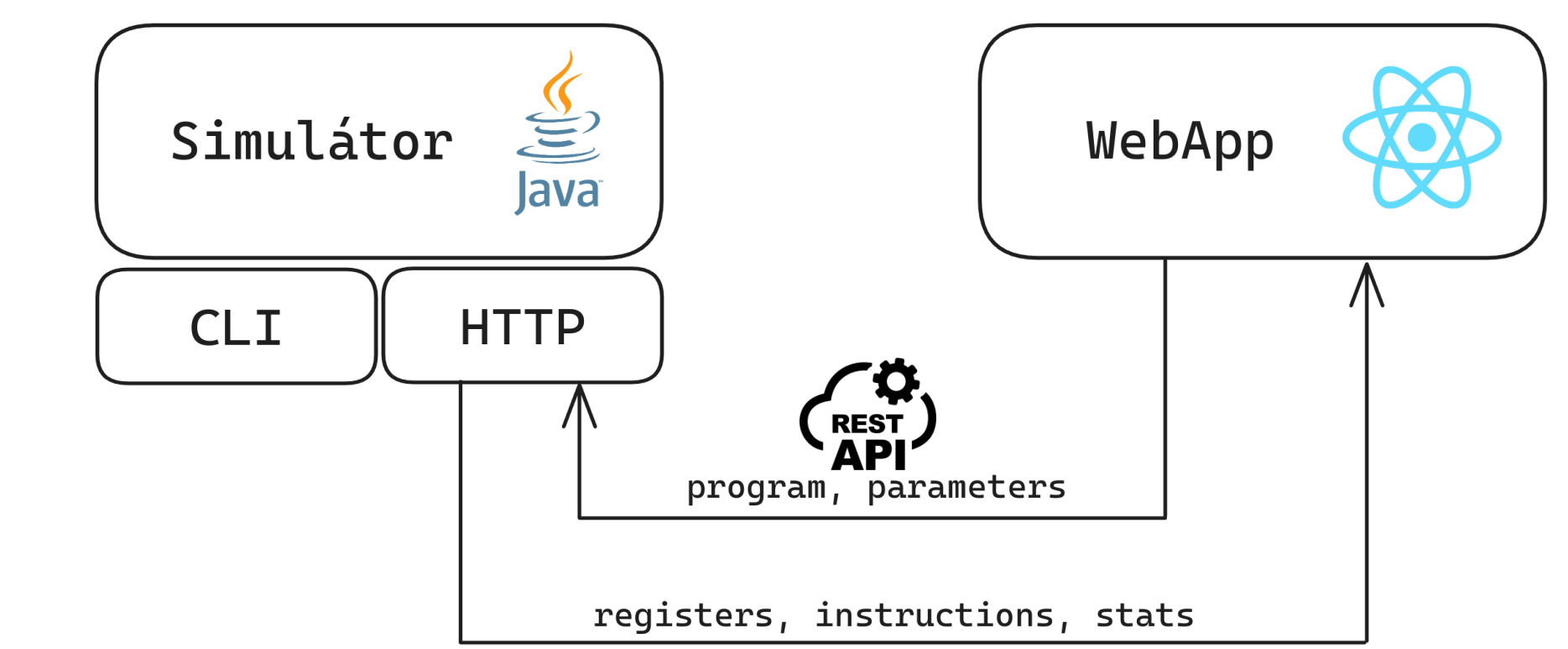
- ### 1 Motivation and Goals

Mastering computational architectures is essential for developing fast and power-efficient programs. Our advanced simulator empowers both IT students and professionals to grasp the fundamentals of superscalar processors and HW-SW co-design. With customizable processor architecture, full C compiler support, and detailed performance statistics, this tool offers a comprehensive learning experience. Enjoy the convenience of a modern, web-based GUI to enhance your understanding and skills.
- ### 2 Feature Highlights

 - User-Friendly Interface:** Simple and illustrative web presentation with detailed information on each block and instruction.
 - Fully Configurable Processors:** Customize issue width, register fields, reorder, load and store buffers, branch predictors, functional and memory units, along with cache memory settings including size, associativity, cache line size, and replacement strategy.
 - Forward and Backward Simulation:** Flexibility to simulate in both directions for thorough analysis.
 - GCC Compiler Interface:** Build C code into assembly using various optimization levels with syntax highlighting and pairing between C and assembly code.
 - Comprehensive Performance Statistics:** Access static and dynamic metrics such as FLOPs, IPC, branch prediction accuracy, unit utilization and cache hit rate.
 - Benchmark CLI:** Command-line interface for benchmarking complex programs.

- ### 3 Implementation, Testing and Deployment

 - Fully Containerized Solution:** Implemented in Docker for seamless deployment and scalability.
 - Extensive Static Unit Testing:** Achieves 83% code coverage.
 - Robust Dynamic Testing:** Supports 100 concurrent users with a median latency under 1.2 seconds on an Intel i5-8300 laptop and 16GB RAM.



- ### 4 Video Tutorial, Source Codes and Live Demo

 - [Short Video Tutorial](#)
 - [Source Codes](#)
 - [Live Demo](#)

The main simulator interface displays a program being executed, with various execution blocks visible:

- Program:** Shows the current instruction stream.
- Fetch Block:** Shows the current instruction being fetched.
- Reorder Buffer:** Shows instructions waiting to be issued.
- ALU Issue Window:** Shows instructions being issued to the ALU.
- FP Issue Window:** Shows instructions being issued to the floating-point unit.
- Store Buffer:** Shows instructions waiting to be written back to memory.
- Load Buffer:** Shows instructions waiting to be read from memory.
- L_S:** Shows the current state of the load/store queue.
- Cache:** Shows the current state of the cache.
- Memory:** Shows the current state of the memory.

