



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

Božetěchova 1/2, 612 66 Brno - Královo Pole

jiri.jaros@vut.cz



- **Bridge the Educational Gap**

Provide HPC developers with an accessible, hands-on tool to explore and understand superscalar RISC-V processor architectures.

- **Empower Developers**

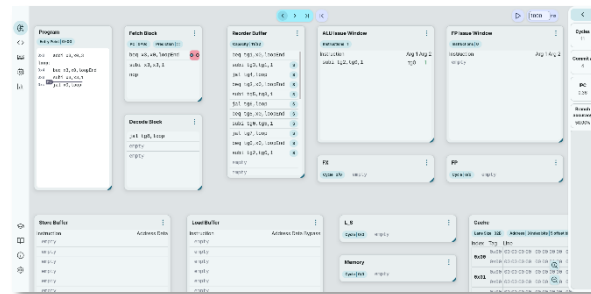
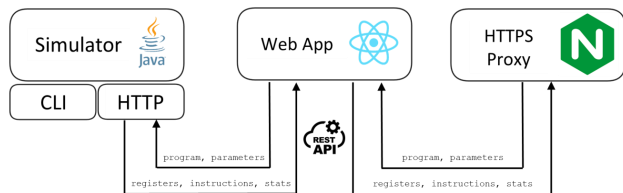
Equip users with skills to design processors and optimize code for performance, cost efficiency, and power consumption.

- **User-Friendly Customization**

Enhance the learning experience with a customizable interface and robust tools for performance analysis to address modern computing challenges.



Live Demo



- **User-Friendly Interface**

Simple and illustrative web presentation with detailed information on each block and instruction.

- **Fully Configurable Processors**

Customize issue width, register files, 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.

Web-Based User Interface

Graphical User Interface – Simulation View

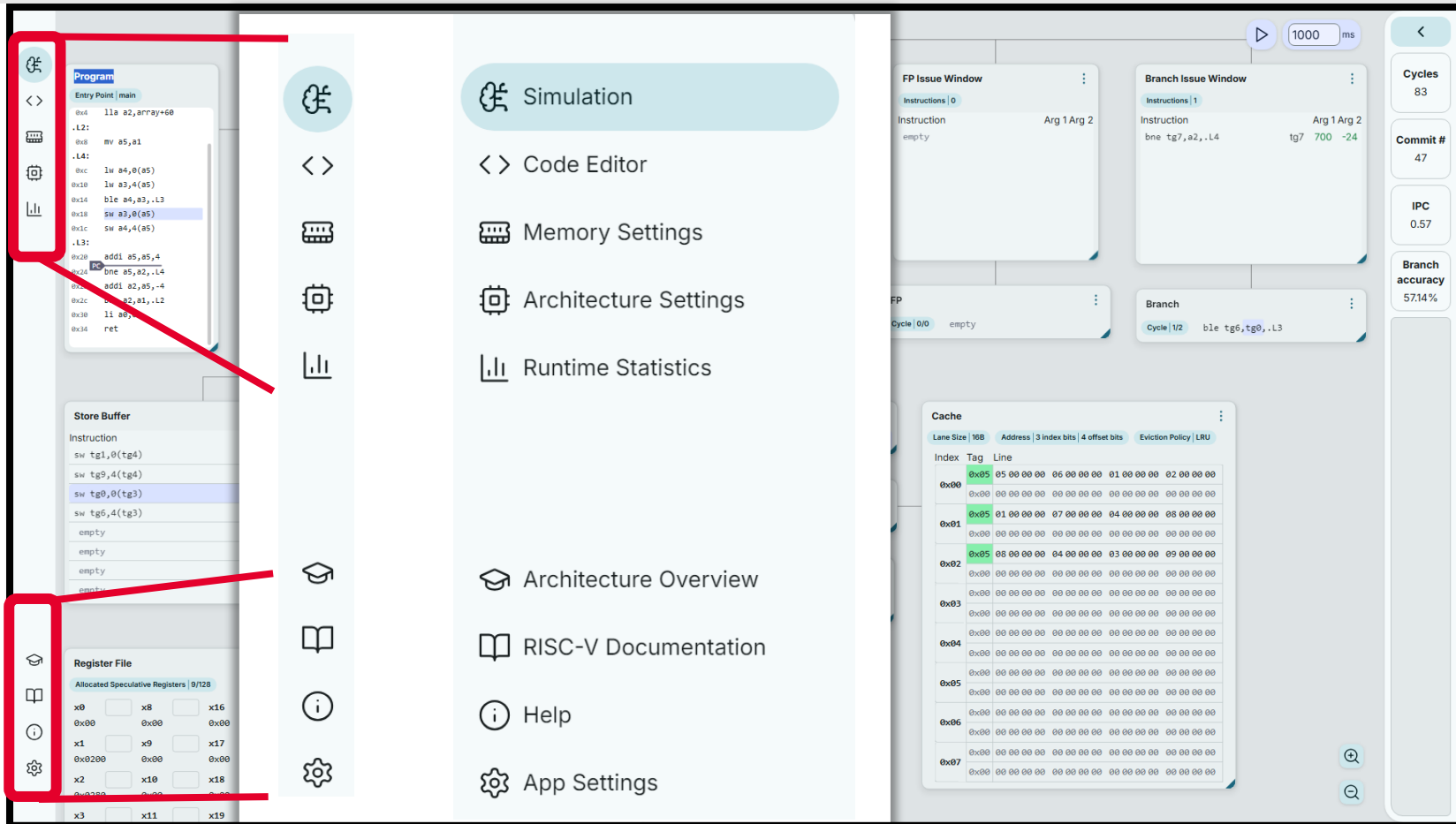
The simulation view displays the internal state of a superscalar RISC-V processor. Key components include:

- Program:** A list of instructions with their addresses and opcodes. The current instruction is highlighted.
- Fetch Block:** Shows the current PC (0x24) and the instructions being fetched.
- Decode Block:** Shows the instructions being decoded.
- Reorder Buffer:** A buffer for instructions, showing their capacity (13/16) and the current state of the buffer.
- ALU Issue Window:** Shows the instructions being issued to the ALU.
- FP Issue Window:** Shows the instructions being issued to the FP unit.
- Branch Issue Window:** Shows the instructions being issued to the branch unit.
- Store Buffer:** A buffer for store instructions, showing their addresses and data.
- Load Buffer:** A buffer for load instructions, showing their addresses and data.
- L2 Cache:** A cache for instructions, showing its state and the current instruction being fetched.
- Register File:** A table of registers, showing their current values.

On the right side, a sidebar displays performance metrics:

- Cycles: 83
- Commit #: 47
- IPC: 0.57
- Branch accuracy: 57.14%

Graphical User Interface – Menu Panel

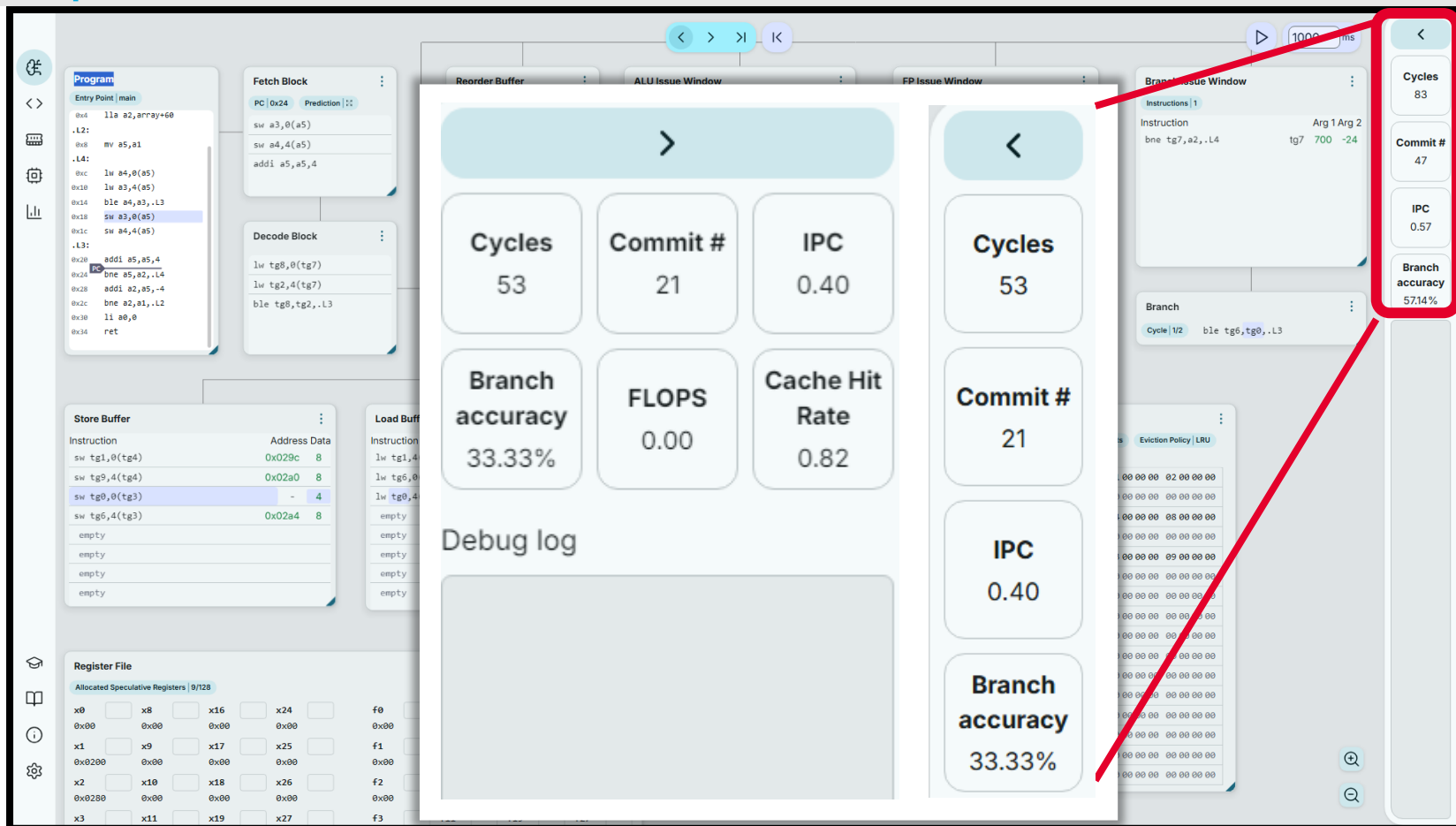


The screenshot displays the graphical user interface of the Jiri simulator, which is a web-based simulator of Superscalar RISC-V processors. The interface is divided into several panels:

- Menu Panel (Left):** A vertical sidebar containing icons for various simulation components. A red box highlights the top four icons: a gear (Simulation), a code editor (Code Editor), a memory icon (Memory Settings), and an architecture icon (Architecture Settings). Another red box highlights the bottom four icons: a graduation cap (Architecture Overview), a book (RISC-V Documentation), an information icon (Help), and a settings gear (App Settings).
- Simulation Panel (Top Center):** A light blue header with the word "Simulation" and a gear icon.
- Code Editor (Center):** A panel with a light blue header "Code Editor" and a code editor icon. It displays assembly code for a program, including instructions like `llw a2,array+60`, `mv a5,a1`, `lw a4,0(a5)`, `ble a4,a3,.L3`, `sw a3,0(a5)`, `sw a4,4(a5)`, `addi a5,a5,4`, `lwe a5,a2,.L4`, `addi a2,a5,-4`, `ble a2,a1,.L2`, `li a0`, and `ret`.
- Memory Settings (Center):** A panel with a light blue header "Memory Settings" and a memory icon. It shows a "Store Buffer" with instructions like `sw tg1,0(tg4)`, `sw tg9,4(tg4)`, `sw tg0,0(tg3)`, and `sw tg6,4(tg3)`.
- Architecture Settings (Center):** A panel with a light blue header "Architecture Settings" and an architecture icon. It shows a "Register File" with allocated speculative registers (9/128) and a table of registers (x0 to x19) with their values.
- Runtime Statistics (Center):** A panel with a light blue header "Runtime Statistics" and a bar chart icon. It displays various statistics: Cycles (83), Commit # (47), IPC (0.57), and Branch accuracy (57.14%).
- Simulation Interface (Right):** A panel showing the execution of instructions. It includes a "Branch Issue Window" with instructions like `bne tg7,a2,.L4` and `lg7 700 -24`. It also shows a "Cache" window with a table of cache entries (Index, Tag, Line) and a "Branch" window with instructions like `ble tg6,tg0,.L3`.

The screenshot displays the Jiri Jaros Web-Based Simulator GUI. A central navigation bar contains several controls: a set of four buttons (left arrow, right arrow, step right, and stop) highlighted by a red box with arrows pointing to them; a play button, a text input field set to '1000', and a 'ms' unit label, all highlighted by another red box with arrows pointing to them; and a zoom control with '+' and '-' buttons highlighted by a third red box with arrows pointing to them. The main interface is divided into several panels: 'Program' (showing assembly code), 'Fetch Block', 'Reorder Buffer', 'ALU Issue Window', 'FP Issue Window', 'Branch Issue Window', 'Store Buffer', 'Load Buffer', 'L_S', 'Memory', 'Main Memory', 'Cache', and 'Register File'. The right sidebar displays simulation statistics: Cycles (83), Commit # (47), IPC (0.57), and Branch accuracy (57.14%).

Graphical User Interface – Simulation Status



The screenshot displays the simulation status interface of the Jiri Jaros Web-Based Simulator of Superscalar RISC-V Processors. The interface is divided into several panels:

- Program Panel:** Shows the entry point 'main' and a list of instructions. The instruction 'sw a3,0(a5)' is highlighted.
- Fetch Block Panel:** Displays the PC (0x24) and the instruction 'sw a3,0(a5)'.
- Decode Block Panel:** Shows the instruction 'sw a3,0(a5)' and its decoded form 'sw a3,0(a5)'.
- Store Buffer Panel:** Lists instructions and their addresses. The instruction 'sw tg0,0(tg3)' is highlighted.
- Register File Panel:** Shows the state of registers. The register 'x0' is highlighted.
- Simulation Status Panel (Right):** Displays key performance indicators (KPIs) for the simulation. A red box highlights the 'Cycles', 'Commit #', 'IPC', and 'Branch accuracy' metrics.

The Simulation Status Panel (Right) displays the following metrics:

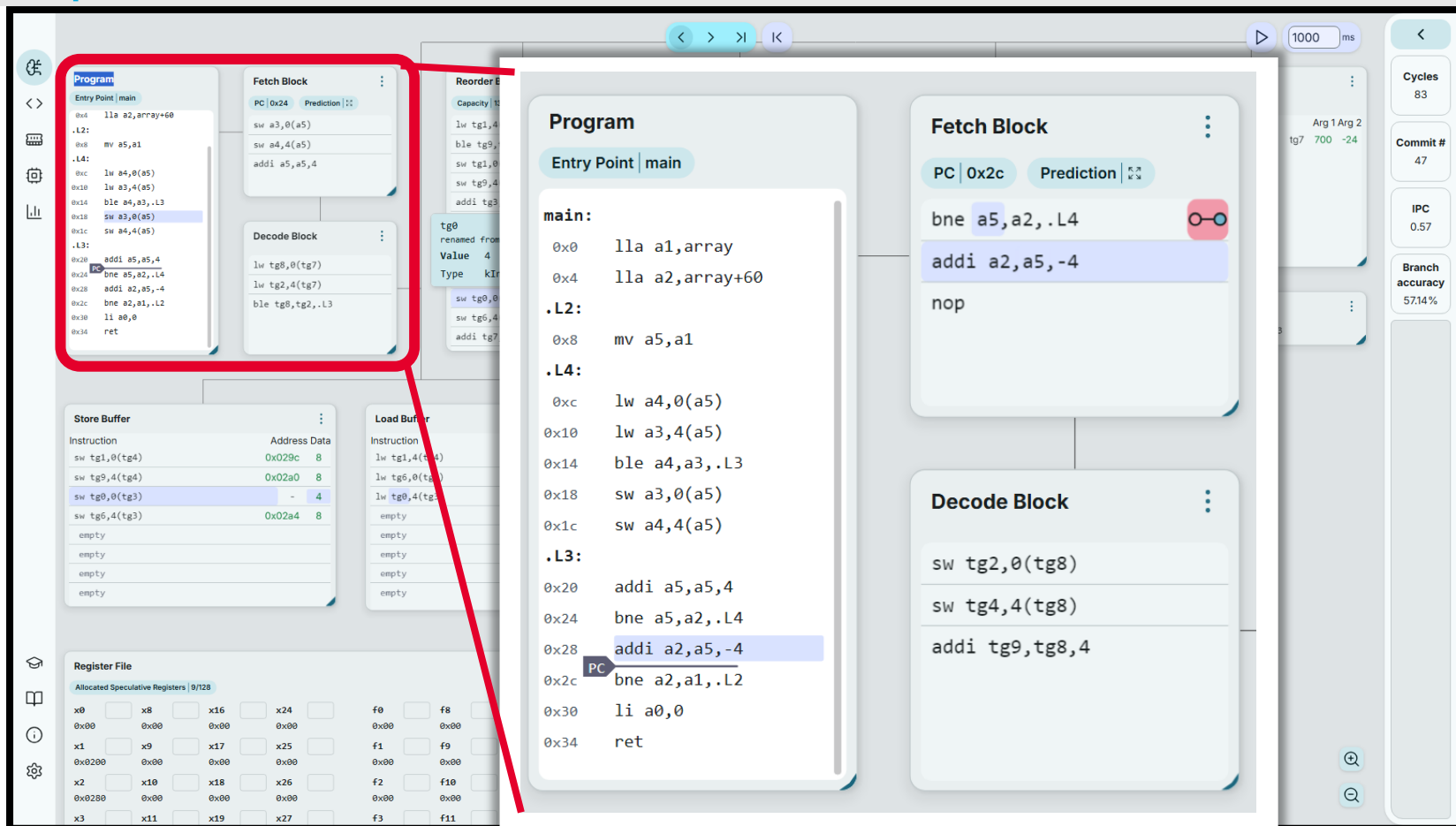
- Cycles: 83
- Commit #: 47
- IPC: 0.57
- Branch accuracy: 57.14%

The Simulation Status Panel (Left) displays the following metrics:

- Cycles: 53
- Commit #: 21
- IPC: 0.40
- Branch accuracy: 33.33%
- FLOPS: 0.00
- Cache Hit Rate: 0.82

The Debug log panel shows a list of instructions and their addresses, with the instruction 'sw tg0,0(tg3)' highlighted.

Simulation View



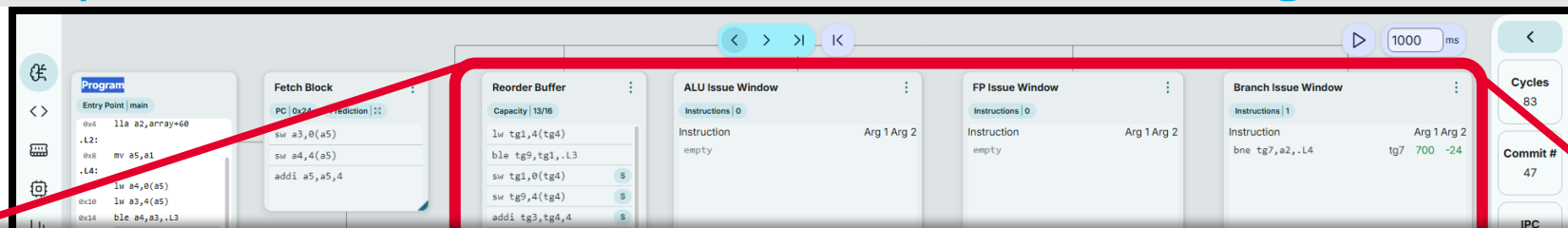
The screenshot displays the Jiri Jaros Web-Based Simulator of Superscalar RISC-V Processors GUI. The interface is divided into several panels:

- Program View:** Shows the entry point 'main' with instructions. A red box highlights the program view and the fetch block, with a red arrow pointing to the program view.
- Fetch Block:** Shows the current instruction being fetched, with PC 0x2c and Prediction. The instruction is `bne a5,a2,.L4` followed by `addi a2,a5,-4` and `nop`.
- Decode Block:** Shows the instructions being decoded, including `sw tg2,0(tg8)`, `sw tg4,4(tg8)`, and `addi tg9,tg8,4`.
- Store Buffer:** A table showing instructions and their addresses. The instruction `sw tg0,0(tg3)` is highlighted.
- Load Buffer:** A table showing instructions and their addresses. The instruction `lw tg0,4(tg3)` is highlighted.
- Register File:** A table showing allocated speculative registers (x0 to x31) and their values. The register `x0` is highlighted.

On the right side, there are performance metrics:

- Cycles: 83
- Commit #: 47
- IPC: 0.57
- Branch accuracy: 57.14%

Graphical User Interface – Out-of-Order Engine



Program

```
Entry Point | main
0x4 11a a2,array+60
.L2: mv a5,a1
.L4: lw a4,0(a5)
0x10 lw a3,4(a5)
0x14 ble a4,a3,.L3
```

Fetch Block

PC | 0x20 | Prediction |

```
sw a3,0(a5)
sw a4,4(a5)
addi a5,a5,4
```

Reorder Buffer

Capacity | 13/16

```
lw tg1,4(tg4)
ble tg9,tg1,.L3
sw tg1,0(tg4)
sw tg9,4(tg4)
addi tg3,tg4,4
```

ALU Issue Window

Instructions | 0

Instruction empty

Arg 1 Arg 2

FP Issue Window

Instructions | 0

Instruction empty

Arg 1 Arg 2

Branch Issue Window

Instructions | 1

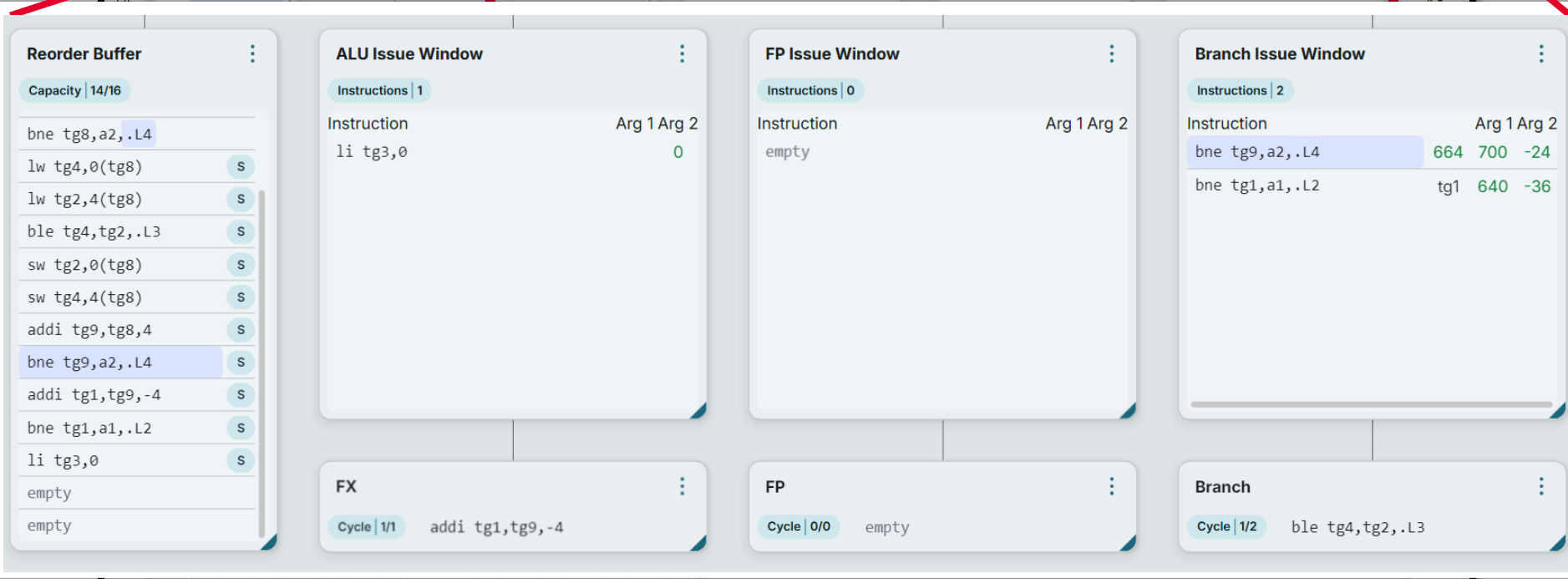
Instruction bne tg7,a2,.L4

Arg 1 Arg 2 tg7 700 -24

Cycles 83

Commit # 47

IPC



Reorder Buffer

Capacity | 14/16

```
bne tg8,a2,.L4
lw tg4,0(tg8)
lw tg2,4(tg8)
ble tg4,tg2,.L3
sw tg2,0(tg8)
sw tg4,4(tg8)
addi tg9,tg8,4
bne tg9,a2,.L4
addi tg1,tg9,-4
bne tg1,a1,.L2
li tg3,0
empty
empty
```

ALU Issue Window

Instructions | 1

Instruction li tg3,0

Arg 1 Arg 2 0

FP Issue Window

Instructions | 0

Instruction empty

Arg 1 Arg 2

Branch Issue Window

Instructions | 2

Instruction bne tg9,a2,.L4

Arg 1 Arg 2 664 700 -24

bne tg1,a1,.L2 tg1 640 -36

FX

Cycle | 1/1 addi tg1,tg9,-4

FP

Cycle | 0/0 empty

Branch

Cycle | 1/2 ble tg4,tg2,.L3

Graphical User Interface – Load and Store Units

The screenshot displays the graphical user interface of the Jiri Jaros Web-Based Simulator of Superscalar RISC-V Processors. The interface is divided into several panels:

- Program Panel:** Shows the entry point 'main' with assembly code. The instruction `sw a3,0(a5)` at address `0xc1c` is highlighted.
- Fetch Block Panel:** Shows the current instruction being fetched, `sw a3,0(a5)`, with address `0xc1c`.
- Decode Block Panel:** Shows the instruction being decoded, `sw a3,0(a5)`, with address `0xc1c`.
- Store Buffer Panel:** A table showing the state of the store buffer. The instruction `sw tg7,0(tg6)` at address `0x0290` is at the top, and `sw tg4,4(tg8)` at address `0x0298` is highlighted.
- Load Buffer Panel:** A table showing the state of the load buffer. The instruction `lw tg4,0(tg8)` at address `0x0294` is at the top, and `lw tg2,4(tg8)` at address `0x0298` is highlighted.
- IPC Panel:** Shows the Instruction Per Cycle (IPC) value as 0.57.
- Branch accuracy Panel:** Shows the branch accuracy as 57.14%.
- Register File Panel:** A table showing the state of the register file. The register `x1` is highlighted.
- Store Buffer Detail Panel:** A detailed view of the store buffer showing the instruction `sw tg7,0(tg6)` at address `0x0290` and the data `0x0290`.
- Load Buffer Detail Panel:** A detailed view of the load buffer showing the instruction `lw tg4,0(tg8)` at address `0x0294` and the data `0x0294`.

Red arrows indicate the flow of data from the Program panel to the Fetch and Decode blocks, and from the Store Buffer to the Load Buffer.

The screenshot displays the Memory Subsystem interface of the Jiri Jaros Web-Based Simulator. The interface includes a sidebar with navigation icons, a main content area with several panels, and a right-hand sidebar with performance metrics.

Left Sidebar: Contains navigation icons for Home, Back, Forward, Search, and Settings.

Main Content Area:

- L_S Panel:** Shows Cycle 1/1 and instruction `sw tg2,0(tg8)`.
- Memory Panel:** Shows Cycle 1/1, Cache, and instruction `sw tg7,0(tg6)`.
- Main Memory Panel:** Displays a list of memory addresses from `0x0000` to `0x0048` with corresponding hex values. Some values are highlighted in blue.
- Cache Panel:** Shows Cache configuration: Lane Size 16B, Address 3 index bits 4 offset bits, Eviction Policy LRU. It contains a table with Index, Tag, and Line columns.

Cache Table:

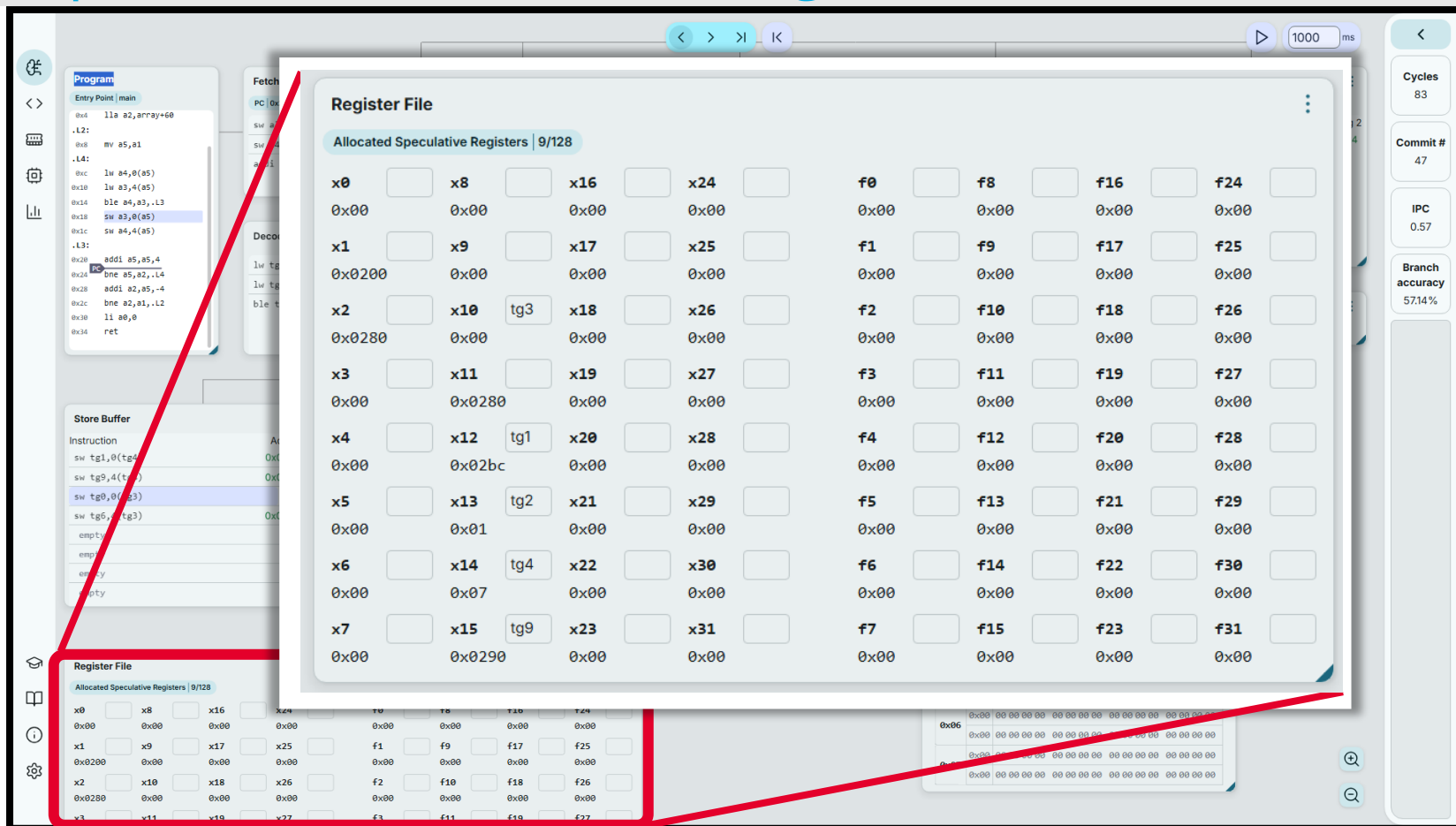
Index	Tag	Line
0x00	0x05	05 00 00 00 06 00 00 00 01 00 00 00 02 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x01	0x05	07 00 00 00 01 00 00 00 08 00 00 00 04 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x02	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x03	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x04	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x05	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x06	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x07	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	0x00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Right Sidebar:

- Branch Issue Window:** Shows Instructions 1, Instruction `bne tg7,a2,.L4`, Arg 1 `tg7`, Arg 2 `700 -24`.
- Branch:** Shows Cycle 1/2, Instruction `sw tg6,tg0,.L3`.
- Performance Metrics:** Cycles 83, Commit # 47, IPC 0.57, Branch accuracy 57.14%.

Bottom Bar: Displays a row of memory addresses: `x3`, `x11`, `x19`, `x27`, `f3`, `f11`, `f19`, `f27`.

Graphical User Interface – Register Fields



The screenshot displays the Jiri simulator's graphical user interface. A central window titled "Register File" is open, showing a table of 32 registers (x0 to x31 and f0 to f31). The registers are organized into two columns of 16 registers each. The first column contains integer registers (x0-x31) and the second column contains floating-point registers (f0-f31). Each register has a small square icon next to its name and a text box showing its current value. The "Allocated Speculative Registers" are indicated as 9/128. A red arrow points from the "Register File" window to the "Program" window on the left, which shows the assembly code being executed. The "Program" window lists instructions such as `li a2,array+60`, `mv a5,a1`, `lw a4,0(a5)`, `lw a3,4(a5)`, `ble a4,a3,-L3`, `sw a3,0(a5)`, `sw a4,4(a5)`, `addi a5,a5,4`, `bne a5,a2,-L4`, `addi a2,a5,-4`, `bne a2,a1,-L2`, `li a0,0`, and `ret`. The "Store Buffer" window below the program window shows the state of the store buffer, including instructions like `sw tg1,0(tg4)`, `sw tg9,4(tg3)`, `sw tg0,0(tg3)`, and `sw tg5,-1(tg3)`. The "Register File" window is also visible in the bottom left corner, showing the same register table. The right side of the GUI features a sidebar with performance metrics: Cycles (83), Commit # (47), IPC (0.57), and Branch accuracy (57.14%).

Allocated Speculative Registers 9/128															
x0		x8		x16		x24		f0		f8		f16		f24	
0x00		0x00		0x00		0x00		0x00		0x00		0x00		0x00	
x1		x9		x17		x25		f1		f9		f17		f25	
0x0200		0x00		0x00		0x00		0x00		0x00		0x00		0x00	
x2		x10	tg3	x18		x26		f2		f10		f18		f26	
0x0280		0x00		0x00		0x00		0x00		0x00		0x00		0x00	
x3		x11		x19		x27		f3		f11		f19		f27	
0x00		0x0280		0x00		0x00		0x00		0x00		0x00		0x00	
x4		x12	tg1	x20		x28		f4		f12		f20		f28	
0x00		0x02bc		0x00		0x00		0x00		0x00		0x00		0x00	
x5		x13	tg2	x21		x29		f5		f13		f21		f29	
0x00		0x01		0x00		0x00		0x00		0x00		0x00		0x00	
x6		x14	tg4	x22		x30		f6		f14		f22		f30	
0x00		0x07		0x00		0x00		0x00		0x00		0x00		0x00	
x7		x15	tg9	x23		x31		f7		f15		f23		f31	
0x00		0x0290		0x00		0x00		0x00		0x00		0x00		0x00	

Processor Configuration

Name

Buffers

Functional Units
Cache
Memory
Branch

Buffers

Re-order buffer size ①

Committed instructions per cycle ①

Flush penalty ①

Fetch instructions per cycle ①

Branch follow limit ①

Name
Buffers
Functional Units
Cache
Memory

Branch

Branch Prediction

Branch target buffer size ①

Pattern history table size ①

Predictor type in PHT

Zero bit
One bit

Two bit

Predictor default state ①

Strongly Not Taken

Weakly Not Taken
Weakly Taken
Strongly Taken

☐ Use global history vector

Name
Buffers

Functional Units

Cache
Memory
Branch

Functional Units

Unit	Latency	Operations	
FX	2	Bitwise 1 Addition 1 Multiplication 2 Division	✗
FP	2	Bitwise 1 Addition 1 Multiplication 2 Division	✗
L_S	1	N/A	✗
Branch	2	N/A	✗
Memory	1	N/A	✗

Add a Unit

FX

FP
L_S
Branch
Memory

Name

Base latency

This unit can perform:

☐ Addition (+)
☐ Bitwise (and, or, xor)
☐ Division (/)
☐ Multiplication (*)
☐ Special (sqrt, ...)

Add Unit

Name	Buffers	Functional Units	Cache	Memory	Branch
------	---------	------------------	-------	--------	--------

Cache

☒ Use cache

Cache lines ⓘ

Cache line size (B) ⓘ

Cache associativity ⓘ

Cache replacement policy

Store behavior ⓘ

Lane replacement delay ⓘ

Cache access delay ⓘ

Name	Buffers	Functional Units	Cache	Memory	Branch
------	---------	------------------	-------	--------	--------

Memory

Load buffer size ⓘ

Store buffer size ⓘ

Store latency ⓘ

Load latency ⓘ

Call stack size ⓘ


Number of speculative registers ⓘ


Code Editor

Optimization

- ☐ Do not optimize
☒ Optimize (O2)
☐ Optimize (O3)
☐ Optimize for size

 Compile

 Load Example

 Show Memory

[Tips for writing code](#)

C Code

 Load


 Save


```


1 // 1) Define your own matrix in memory tab
2 extern int matrix[];
3 // 2) Change the dimensions
4 #define rows_c 30
5 #define cols_c 30
6 int result[rows_c*cols_c] = {0};
7
8 // Function to transpose a matrix
9 void transpose(int *dst, int *src, int rows, int cols) {
10     for (int i = 0; i < rows; i++) {
11         for (int j = 0; j < cols; j++) {
12             dst[j * rows + i] = src[i * cols + j];
13         }
14     }
15 }
16
17 int main() {
18     transpose(result, matrix, rows_c, cols_c);
19 }
20


```

ASM Code

 Load

 Save

 Check

 Entry Point: main

```

1 transpose:
2     ble a2,zero,.L1
3     slli t3,a3,2
4     add a6,a1,t3
5     slli a7,a2,2
6     li t1,0
7 .L3:
8     sub a5,a6,t3
9     mv a4,a0
10    ble a3,zero,.L6
11 .L4:
12    lw a1,0(a5)
13    addi a5,a5,4
14    sw a1,0(a4)
15    add a4,a4,a7
16    bne a5,a6,.L4
17 .L6:
18    addi t1,t1,1
19    addi a0,a0,4
20    add a6,a6,t3
21    bne a2,t1,.L3
22 .L1:
23    ret
24 main:
25    addi sp,sp,-16
26    lla a0,result
27    li a3,30

```

New Object

ImportExport

Memory Objects

matrix

Pointer Name ⓘ
matrix

Data Type ⓘ
Integer

Alignment (log Bytes) ⓘ
4

Values

ListRepeated ConstantRandom Values

Random values will be generated in an inclusive range.
Number of Elements
900
Inclusive Range
0 - 100

Summary

Array called **matrix** of 900 elements (3600 bytes). Alignment is $2^4 = 16$ bytes.

UpdateDelete

Simulation Statistics

IPC

0.52

Clocks

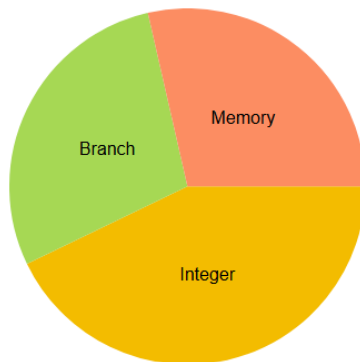
58

Branch Prediction
Accuracy

55.56%

Static Instruction Mix

Instruction mix of the program



Table

Chart

Dynamic Instruction Mix

Instruction mix of committed instructions

Category	Value	Proportion
Integer	7	23.33%
Float	0	0.00%
Branch	9	30.00%
Memory	14	46.67%
Other	0	0.00%

Table

Chart

Functional Units

Amount of time a unit spends computing

Name	Busy Cycles	Utilization
L_S	27	46.55%
FX	19	32.76%
Branch	0	0.00%
Memory	17	29.31%
FP	0	0.00%

Detailed Statistics

Attribute	Value
Prediction Accuracy	0.56
Committed Instructions	30
Clock Cycles	58
Flushed Instructions	31
ROB Flushes	4
Correctly Predicted Branches	5
Conditional Branches	9
Taken Branches	6
Max Allocated Registers	10
Arithmetic Intensity	0.23
FLOPS	0.00
IPC	0.52

Cache Statistics

Attribute	Value
Read Accesses	10
Write Accesses	5
Hits	13
Misses	2
Total Delay	0
Bytes Written	20
Bytes Read	40

Instruction Statistics

Heat Map of execution counts

Cache Hits

Committed

main:

ll a a1,array

ll a a2,array+60

.L2:

mv a5,a1

.L4:

66.67% lw a4,0(a5)

83.33% lw a3,4(a5)

ble a4,a3,.L3

100.00% sw a3,0(a5)

100.00% sw a4,4(a5)

.L3:

addi a5,a5,4

bne a5,a2,.L4

addi a2,a5,-4

bne a2,a1,.L2

li a0,0

ret

Implementation and Deployment

- **Fully Containerized Solution:**

Implemented in Docker for seamless deployment and scalability.

- **Extensive Static Unit Testing**

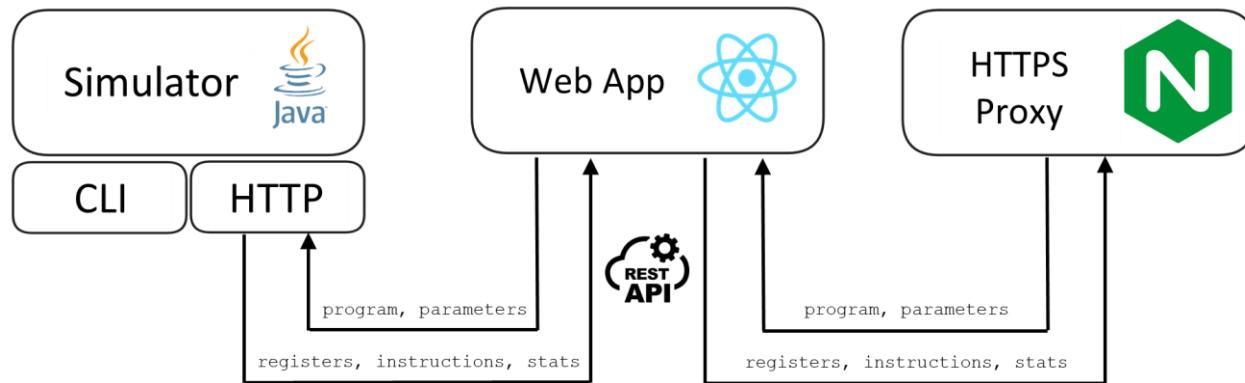
Achieves 83% code coverage.

- **Robust Dynamic Testing:**

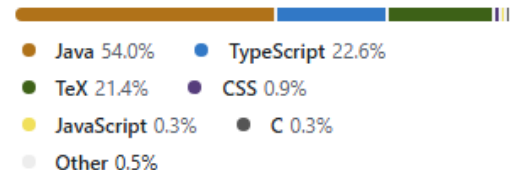
Supports 100 concurrent users with a median latency under 1.2 seconds on an Intel i5-8300 laptop with 16GB RAM.



[Source Codes](#)



Languages



- **Accessibility**

The simulator simplifies understanding of superscalar RISC-V processors for education and research.

- **Versatility**

Supports C/assembly programs, code optimization, and architectural benchmarking.

- **Impact:**

Empowers IT students and HPC developers to optimize code and design custom RISC-V processors.

- **Adoption**

It is currently used by 200+ students in the Computation Systems Architectures course at Brno University of Technology.

- **Processor Enhancements**

Add advanced features like vector units, pipelined functional units, improved branch predictors, and deeper cache hierarchies.

- **Development Tools**

Improve the code editor and debugging environment with features like breakpoints, watches, dynamic memory allocation, and atomic operations.

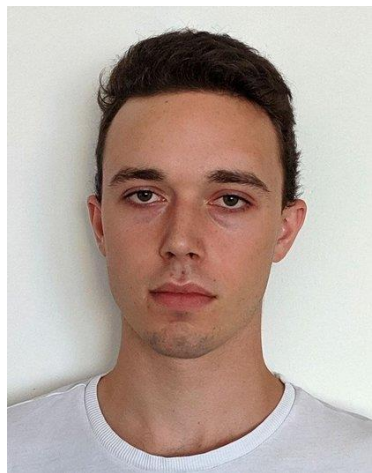
- **Expanded Metrics**

Incorporate runtime statistics for chip area estimation and power consumption analysis using realistic manufacturing technology.

- **Community Engagement**

Continue development and improvement via open-source contributions on GitHub and increase accessibility through the live simulator instance.

This simulator was brought to life by three talented master's students, guided by an enthusiastic supervisor.



Michal Majer



Jakub Horky



Jan Vavra



Jiri Jaros