**Rust RISC-V ISA Simulator with Qt GUI**

**Darshan H Sonecha**

**Introduction**

This document represents design and implementation of RISC-V ISA simulator, particularly geared to suite towards SHAKTI. The simulator is to allow for development and test of simple programs to compilers to operating systems. There are various kinds of simulation in the real-world - architectural level **simulation**, **direct execution**, **threaded code**, and **instruction set simulators 1,[[1]](#footnote-1)**. ISS are also referred to as “complete system instruction set simulator,” as well as " **1**.

We are to implement ISS – behavioral simulation model. The implementation of simulator should have following characteristics - **accuracy**, **speed[[2]](#footnote-2), reproducibility** and **options** yielding flexibility – for example, one of the option family could be for performance of the program being executed. **Extensibility** and **Statistics** are two additional features/characteristics to be built upon as needed. Few of these characteristics would be traded off amongst or against each other.

The implementation is to be in Rust. The goal of this simulator is all to allow for **execution of program**, **operation** **systems** and **simulator test code**

The first version of the implementation would be for RV32I ISA only, additional extensions and expansions to follow later. The ISA is the interface between hardware and software and is a major portion of what makes up an architecture3. In following sections, we will discuss **1)** Block Level Architecture, **2)** Data Structures, Software Modules, Software Processes **3)** Program Flow/Execution Model, **4)** Data Flow Model, **5)** Tests and **6)** Conclusion. We will discuss various implementation details including optimization details that improve in the performance of the simulator.

As in with other simulation tools but with the constraint that this ISS is more a behavioral modeling, we may explore design space exploration possibilities - more from the perspective of supporting heterogeneous and homogeneous multicore architectures3, without timing constraints. As well there will be traces that will be generated to study the simulation flow and aid in debugging of the logic. There are existing simulator architectures for exploration of architectural and microarchitectural features

such as Sniper. As well there are reference RISC-V ISA functional simulator such as Spike.

**Block Level Architect**

**Data Structures, Software Modules, Software Processes**

**Program Flow/Execution Model**

**Data Flow Model**

**Tests**

**Conclusion**

There various tools used – such as Microsoft Word, Emacs, IntelliJ IDE, GCC – RISC-V Cross Compiler/GNU Tools Chain, Bluespec System Verilog Simulation Model – Bluespec Inc. (accuracy of implementation) and RISC-V Torture, CSMIT, AAPG (used by Processor team as necessary).

Data Structures:

RV32I\_Opcode\_Map {

u32 U\_lui; // imm[31:12] | rd | 0110111

u32 U\_auipc; // imm[31:12] | rd | 0010111

u32 J\_jal; // imm[20|10:1|11|19:12] | rd | 1101111

u32 I\_jalr; // imm[11:0] | rs1 | 000 | rd | 1100111

u32 B\_beq; // imm[12|10:5] | rs2 | rs1 | 000 | imm[4:1|11] | 1100011

u32 B\_bne; // imm[12|10:5] | rs2 | rs1 | 001 | imm[4:1|11] | 1100011

u32 B\_blt; // imm[12|10:5] | rs2 | rs1 | 100 | imm[4:1|11] | 1100011

u32 B\_bge; // imm[12|10:5] | rs2 | rs1 | 101 | imm[4:1|11] | 1100011

u32 B\_bltu; // imm[12|10:5] | rs2 | rs1 | 110 | imm[4:1|11] | 1100011

u32 B\_bgeu; // imm[12|10:5] | rs2 | rs1 | 111 | imm[4:1|11] | 1100011

u32 I\_1b; // imm[11:0] | rs1 | 000 | rd | 0000011

u32 I\_1h; // imm[11:0]. | rs1 | 001 | rd | 0000011

u32 I\_1w; // imm[11:0] | rs1 | 010 | rd | 0000011

u32 I\_1lu; // imm[11:0] | rs1 | 100 | rd | 0000011

u32 I\_1hu; // imm[11:0] | rs1 | 101 | rd | 0000011

u32 S\_sb; // imm[11:5] | rs2 | rs1 | 000 | imm[4:0] | 0100011

u32 S\_sh; // imm[11:5] | rs2 | rs1 | 001 | imm[4:0] | 0100011

u32 S\_sw; // imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011

u32 I\_addi; // imm[11:0] | rs1 | 000 | rd | 0010011

u32 I\_slti; // imm[11:0] | rs1 | 010 | rd | 0010011

u32 I\_sltiu; // imm[11:0] | rs1 | 011 | rd | 0010011

u32 I\_xori; // imm[11:0] | rs1 | 100 | rd | 0010011

u32 I\_ori; // imm[11:0] | rs1 | 110 | rd | 0010011

u32 I\_andi; // imm[11:0] | rs1 | 111 | rd | 0010011

u32 I\_slli; // 0000000 | shamt | rs1 | 001 | rd | 0010011

u32 I\_srli; // 0000000 | shamt | rs1 | 101 | rd | 0010011

u32 I\_srai; // 0100000 | shamt | rs1 | 101 | rd | 0010011

u32 R\_add; // 0000000 | rs2 | rs1 | 000 | rd | 0110011

u32 R\_sub; // 0100000 | rs2 | rs1 | 000 | rd | 0110011

u32 R\_sll // 0000000 | rs2 | rs1 | 001 | rd | 0110011

u32 R\_slt // 0000000 | rs2 | rs1 | 010 | rd | 0110011

u32 R\_sltu; // 0000000 | rs2 | rs1 | 011 | rd | 0110011

u32 R\_xor; // 0000000 | rs2 | rs1 | 100 | rd | 0110011

u32 R\_srl; // 0000000 | rs2 | rs1 | 101 | rd | 0110011

u32 R\_sra; // 0100000 | rs2 | rs1 | 101 | rd | 0110011

u32 R\_or; // 0000000 | rs2 | rs1 | 110 | rd | 0110011

u32 R\_and; // 0000000 | rs2 | rs1 | 111 | rd | 0110011

u32 I\_fence; // 0000 | pred | succ |00000| 000 | 00000 | 0001111

u32 I\_fence.i; // 0000 | 0000 | 0000 |00000| 001 | 00000 | 0001111

u32 I\_ecall; // 000000000000 | 00000| 000 | 00000 | 1110011

u32 I\_ebreak; // 000000000001 | 00000 | 000 | 00000 | 1110011

u32 I\_csrrw; // csr | rs1 | 001 | rd | 1110011

u32 I\_csrrs; // csr | rs1 | 010 | rd | 1110011

u32 I\_csrrc; // csr | rs1 | 011 | rd | 1110011

u32 I\_csrrwi; // csr | zimm | 101 | rd | 1110011

u32 I\_csrrsi; // csr | zimm | 110 | rd | 1110011

u32 I\_csrrci; // csr | zimm | 111 | rd | 1110011

};

|  |  |  |
| --- | --- | --- |
| U-type for long immediate | U\_lui | U-type long upper immediate |
| Does U imply Unsigned? | U\_auipc | U-type add upper immediate to pc |
|  | U\_jal | U-type jump and link |
|  | U\_jalr | U-type jump and link register |
| B-type for conditional branches | B\_beq | B-type branch equal |
| Does B imply Branch? | B\_bne | B-type branch not equal |
|  | B\_blt | B-type branch less than |
|  | B\_bge | B-type greater than or equal |
|  | B\_bltu | B-type branch less than unsigned |
|  | B\_bgeu | B-type greater than or equal unsigned |
| I-type for short immediate | I\_lb | I-type load byte |
| Does I imply Immediate? | I\_lh | I-type load halfword |
| \_l - load | I\_lw | I-type load word |
|  | I\_lbu | I-type load byte unsigned |
|  | I\_lhu | I-type load halfword unsigned |
| S-type instruction is for Store | S\_sb | S-type store byte |
| Does S imply Store? | S\_sh | S-type store halfword |
|  | S\_sw | S-type word |
| I-type for short immediate | I\_addi | I-type add immediate |
| Does I imply Immediate? | I\_slti | I-type set less than immediate |
|  | I\_sltiu | I-type set less than immediate unsigned |
|  | I\_xori | I-type exclusive or immediate |
|  | I\_ori | I-type or immediate |
|  | I\_andi | I-type and immediate |
|  | I\_slli | I-type shift left logical immediate |
|  | I\_srli | I-type shift right logical immediate |
|  | I\_srai | I-type shift right arithmetic immediate |
| R-type for register-register operation | R\_add | R-type add |
| Does R imply register? | R\_sub | R-type subtract |
|  | R\_sll | R-type shift left logical |
|  | R\_slt | R-type set less than |
|  | R\_sltu | R-type set less then unsigned |
|  | R\_xor | R-type exclusive or |
|  | R\_srl | R-type shift right logical |
|  | R\_sra | R-type shift right arithmetic |
|  | R\_or | R-types or |
|  | R\_and | R-type and |
| I-type for short immediate | I\_fence | I-type fence loads & stores |
| Does I imply immediate? | I\_fence.i | I-type instruction & data |
|  | I\_ecall | I-type environment call |
|  | I\_ebreak | I-type environment break |
|  | I\_csrrw | I-type control status register read & write |
|  | I\_csrrs | I-type control status register read & set bit |
|  | I\_csrrc | I-type control status register read & clear bit |
|  | I\_csrrwi | I-type control status register read & write immediate |
|  | I\_csrrsi | I-type control status register read & set bit immediate |
|  | I\_csrrci | I-type control status register read & clear bit immediate |

0

31

|  |  |
| --- | --- |
| x0 / zero | Hardwired zero |
| x1 / ra | Return address |
| x2 / sp | Stack pointer |
| x3 / gp | Global pointer |
| x4 / tp | Thread pointer |
| x5 / t0 | Temporary |
| x6 / t1 | Temporary |
| x7 / t2 | Temporary |
| x8 / s0 / fp | Saved register, frame pointer |
| x9 / s1 | Saved register |
| x10 / a0 | Function argument, return value |
| x11 / a1 | Function argument, return value |
| x12 / a2 | Function argument |
| x13 / a3 | Function argument |
| x14 / a4 | Function argument |
| x15 / a5 | Function argument |
| x16 / a6 | Function argument |
| x17 / a7 | Function argument |
| x18 / s2 | Saved register |
| x19 / s3 | Saved register |
| x20 / s4 | Saved register |
| x21 / s5 | Saved register |
| x22 / s6 | Saved register |
| x23 / s7 | Saved register |
| x24 / s8 | Saved register |
| x25 / s9 | Saved register |
| x26/ s10 | Saved register |
| x27 /s11 | Saved register |
| x28 / t3 | Temporary |
| x29 / t4 | Temporary |
| x30 / t5 | Temporary |
| x31 / t6 | Temporary |

32

31

0

|  |
| --- |
| PC |

32

References:

1. ARMSim: An Instruction-Set Simulator for the ARM processor, Alpa Shah, Columbia University
2. Flexible Timing Simulation of RISC-V Processors with Sniper, Neethu Bal Mallya, Cecilia Gonzalez-Alvarez, Trevor E. Carlson, CARRV 2018, June 2018
3. Fast, Accurate, and Validated Full-System Software Simulation of x86 Hardware, Frederick Ryckbosch, Stijn Polfliet, Lieven Eeckhout, Ghent University, IEEE Computer Society, 2010.
4. ARMISS: An Instruction Set Simulator for the ARM Architecture, Mingsong Lv, Qingxu Deng, Nan Guan, Yaming Xie, Ge Yu, Institute of Computer Software and Theory, Northestern University
5. ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS, Alasdair Armstrong, University of Cambridge, UK, et. al., January 2019.
6. SHAKTI: An Open-Source Processor Ecosystem, Neel Gala, G.S.Madhusudan, InCore Semiconductors Pvt. Ltd., Paul George, Anmore Sahoo, Arjun Menon, V. Kamakoti, Indian Institute of Technology, Madras, Advanced Computing & Communications, Processor Ecosystem, Volume 02 Issue 03 September 2018.
7. Extensible and Configurable RISC-V based Virtual Prototype, Vladimir Herdt, Daniel GroBe, Hoang M. Le, Rolf Drechsler, Institute of Computer Science, University of Bremen; Cyber-Physical Systems, DFKI GmbH, Bremen, Germany.
8. RISC5: Implementing the RISC-V ISA in gem5, Alec Roelke, Mircea R. Stan; University of Virginia.
9. The RISC-V Reader – An Open Architecture Atlas, First Edition, 1.0.0, David Patterson, Andrew Waterman, November 7, 2017
10. Implementation of Direct Segments of a RISC-V Processors, Nikhita Kunati, Michael M. Switch
11. The Rust Programming Language, Steve Klabnik and Carlo Nichols with contributions from the Rust Community, no scratch press, San Francisco, CA.
12. Mastering Qt 5, Packet Publishing, December 2016.
13. <https://riscv.org/software-tools/risc-v-gnu-compiler-toolchain/> (Tools)
14. Spike – <https://github.com/riscv/riscv-isa-sim>
15. Computer Architecture, A Quantitative Approach, Sixth Edition, John L. Hennessy & David Paterson

Paper Notes:

1. **ARMSim: An Instruction-Set Simulator for the ARM processor (First Pass -** Done**)**
2. **Flexible Timing Simulation of RISC-V Processors with Sniper (First Pass -** Done**)**
3. **FAST, ACCURATE, and Validated Full-System Software Simulation of x86 Hardware**
4. **ARMISS: An Instruction Set Simulator for the ARM Architecture**
5. **ISA Semantics for ARMv8-A, RISC-V, and CHRI-MIPS**
6. **Extensible and Configurable RISC-V based Virtual Prototype**
7. **RISC5: Implementing the RISC-V ISA in gem5**
8. **Implementation of Direct Segments on a RISC-V Processor**

**TBD/Open Issues** –

1. **Implementation Notes** - Diagrams for simulator execution and data flows. (Diagrams could be Flow Charts, UML or Blocks)
2. **Implementation Notes** - Diagrams-block level to microarchitecture to software modules and processes (in Rust). (These Diagrams shows mostly mapping between blocks to microarchitecture to software processes.)
3. **Implementation Idea** - Tree for OPCODE walk through – Could be part of Block Leve data flow – initial block, probably a fastest way to categorize the “next instruction”.
4. Features: {

**What does it mean to be accurate?**

**What does it mean to support speed?**

**What does it mean to have options?** Option Family 1: Performance family options could mean varying cache and register file sizes and instantiating subcomponents (co-processors) and memory hierarchy.

}

1. **Implementation Idea** - How about SaaS Model - essentially implementing Simulator as a Cloud Service?
2. **Implementation Idea** - How would you run time control this simulator, purely from GUI or would there be CLI? Would it be better to have a Python based control embedded with simulator module to control it’s parameters?
3. **Architecture Specific Question** – Are there co-processors - Functional Units standard across all cores (RISC-V Shakti).
4. **Implementation Question** - What defines the state of the system? What all the processor sub components be inclusive in defining state of the system?
5. **Implementation Question** - How will memory hierarchy be modeled in terms of delay statistics – delay(x), delay(x+1), … How is the memory modeled? How will the instruction memory be stored? Data transfer sizes?
6. **Implementation Question** - What is the cycle of instruction in RISC-V? This in order to instructions are put thru to yield the end result.
7. **Implementation Question** - Is it necessary in ISS – behavioral model - to have cache warming mode for ROI (Region of Interest-Front End)? Do we need to look at ROI versus non-ROI regions? We will possibly not have BE – Back End?
8. **Implementation Question** - Where would the trace files be created? What would be the trace file format of our own?
9. **Implementation Question** - Is design space exploration a requirement for this simulator or just a BDD? Should we be concerned with microarchitectural implementation of an ISA – this from the perspective of design space exploration?
10. **Implementation Question** - Processor State Modeling.
11. **Implementation Question** - Statistics Gathering.
12. **Design consideration** – Multithreaded Implementation (for Simulator)? Multi or Single Machine?
13. **Goal Statement**: Primary applications for simulators consist of computer architecture studies and performance tuning of compiled software and the compilation process itself.
14. **Design Ideas** - Instead of decoding the operation fields each time an instruction is executed; the instruction is translated once into a form that is faster to execute. This idea has been used in a variety of simulators for a number of applications. (ARMSim).
15. **Implementation Ideas** - What are the other open issues we need to consider – System Binaries, Binary Data Representation, Low Startup and models at various stages?

1. Instruction set simulator [ISS] execute target machine programs by simulating the effects of each instruction on a target machine, one instruction at a time. [↑](#footnote-ref-1)
2. Reasonable speed of execution as supported by the underlying hardware. [↑](#footnote-ref-2)