**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 but 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.

**Block Level Architect**

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

**Program Flow/Execution Model**

**Data Flow Model**

**Tests**

**Conclusion**

Paper Notes:

1. **Flexible Timing Simulation of RISC-V Processors with Sniper**

Frontend

* ROI – Region of Interest in the application is to be simulated in detail, the code sections outside of the ROI could be simulated in functional cache warming mode (where the memory subsystem is warmed before ROI execution) or could be fast-forwarded without cache warming.
* Instruction instrument callbacks: Module that intercepts each executed instruction.
* (System Call instrumentation and thread instrumentation – OS Calls)
* Like SIFT do we need to have a trace file format of our own?

Scheduler and Backend

* This is the main component of the timing simulator.

(**Question**: Do we need to worry about scheduler?).

* Each application thread in the original program will have a matching thread in the Sniper.

1. **ARMSim: An Instruction-Set Simulator for the ARM processor**

* ARMSim is a lightweight ISA (Instruction Setup Architecture) level simulator and a trace generator too.
* The deterministic behavior of simulators makes programs execution reproducible, and thus helps in locating problems.

Simulation Strategies:

**Instruction Set Simulators**: ISS execute target machine pragmas by simulating the effects of each instruction on a target machine, one instruction at a time.

This is useful for experiments and debugging purposes.

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.

**Structure of ARMSim (Behavioral Model)**

* System Binaries
* Binary Data Representation
* Determinism
* Low Startup
* Extensible
* Statistics
* Various stages of model

1. **FAST, ACCURATE, and Validated Full-System Software Simulation of x86 Hardware**

* Validate the timing model against real hardware using a set of microbenchmarks.

1. **ISA Semantics for ARMv8-A, RISC-V, and CHRI-MIPS**
2. **ARMISS: An Instruction Set Simulator for the ARM Architecture**
3. **RISC5: Implementing the RISC-V ISA in gem5**
4. **Implementation of Direct Segments on a RISC-V Processor**
5. **Extensible and Configurable RISC-V based Virtual Prototype**

Tools:

1. Microsoft Word
2. Emacs, IntelliJ IDE Community 2018.3
3. GCC – RISC-V Cross Compiler, GNU
4. Bluespec System Verilog Simulation Model for accuracy of implementation, Bluespec Inc.
5. Additional tools used by Class-C Processor team – RISC-V Torture, CSMIT, AAPG etc. as applicable.

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 | 0000011

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

u32 I\_sltiu; // imm[11:0] | rs1 | 011 | rd | 0000011

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

u32 I\_ori; // imm[11:0] | rs1 | 110 | rd | 0000011

u32 I\_andi; // imm[11:0] | rs1 | 111 | rd | 0000011

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\_1b | I-type |
| Does I imply Immediate? | I\_1h |  |
|  | I\_1w |  |
|  | I\_1bu |  |
|  | I\_1hu |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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. The RISC-V Reader – An Open Architecture Atlas, First Edition, 1.0.0, David Patterson, Andrew Waterman, November 7, 2017
3. Flexible Timing Simulation of RISC-V Processors with Sniper, Neethu Bal Mallya, Cecilia Gonzalez-Alvarez, Trevor E. Carlson, CARRV 2018, June 2018
4. Fast, Accurate, and Validated Full-System Software Simulation of x86 Hardware, Frederick Ryckbosch, Stijn Polfliet, Lieven Eeckhout, Ghent University, IEEE Computer Society, 2010.
5. 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
6. ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS, Alasdair Armstrong, University of Cambridge, UK, et. al., January 2019.
7. 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.
8. 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.
9. RISC5: Implementing the RISC-V ISA in gem5, Alec Roelke, Mircea R. Stan; University of Virginia.
10. Implementation of Direct Segments of a RISC-V Processors, Nikhita Kunati, Michael M. Switch
11. <https://riscv.org/software-tools/risc-v-gnu-compiler-toolchain/> (Tools)
12. The Rust Programming Language, Steve Klabnik and Carlo Nichols with contributions from the Rust Community, no scratch press, San Francisco, CA.
13. Mastering Qt 5, Packet Publishing, December 2016.
14. Spike –

**TBD/Open Issues** –

1. Block diagram for simulator execution and data flow
2. Transition diagrams-block level to microarchitecture to software modules and processes (in Rust).
3. Tree for OPCODE walk through – Could be part of Block Leve data flow – initial block, probably a fastest way to categorize the “next instruction”.

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. How about SaaS Model, essentially allowing for execution over cloud for this Simulator Model?
2. How would you run time control this simulator, purely GUI based control or would there be CLI based control supposed? Would having a Python based control embedded with simulator module to control it’s parameter be better?
3. Co-processors are not Functional Units across all standard cores (RISC-V Shakti).
4. What defines the state of the system:
   1. What all the processor sub components be inclusive in defining state of the system?
   2. How will memory hierarchy be modeled in terms of delay statistics – delay(x), delay(x+1), …
5. What is the cycle of instruction in RISC-V? This in order to instructions are put thru to yield the end result.
6. Is it necessary in ISS – behavioral model - to have cache warming mode for ROI (Region of Interest)? (FE)
7. Where would the trace files be created?
8. Is design space exploration a requirement for this simulator or just a BDD?
9. How is the memory modeled?
   1. How will the instruction memory be stored?
   2. Data transfer sizes?
10. Processor State Modeling.
11. Statistics Gathering.
12. Should we be concerned with microarchitectural implementation of an ISA – this from the perspective of design space exploration?
13. What are the components of this simulator – back-end(not concerned about timing, in my opinion not necessary to map application thread to simulator thread), front-end and trace files?
14. Design consideration – multithreaded (Given, no need to specify).
15. **Goal Statement**: Primary applications for simulators consist of computer architecture studies and performance tuning of compiled software and the compilation process itself.

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