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

**Darshan H Sonecha**

**Introduction**

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

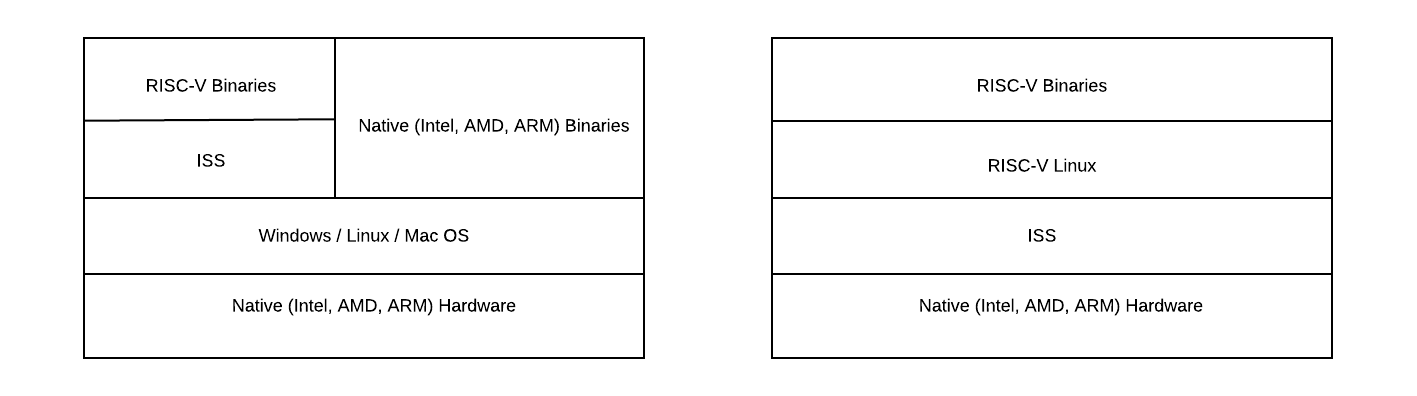
We are to implement the ISS – behavioral simulation model. The implementation of simulator should have the following characteristics - **accuracy**, **speed[[2]](#footnote-2), reproducibility** and **options** yielding flexibility – for example, one of the option family could be for the performance of 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 to allow for the **execution of the program**, **operation** **systems**, and **simulator test code**

The first version of the implementation would be for RV32G ISA, 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 architecture2. 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 behavioral modeling, we may explore design space exploration possibilities - more from the perspective of supporting heterogeneous and homogeneous multicore architectures2, 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**



1. **(b)**

A complete ISA implements the management of processor resource, memory resource, I/O resource and interrupts4.

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

**DS**

**CPUState, Memory, Instruction**

**SM**

**SP**

**Program Flow/Execution Model**

**Data Flow Model**

**Tests**

**Conclusion**

There are various tools envisioned to be used - Microsoft Word, Emacs, IntelliJ IDE, GCC – RISC-V Cross Compiler/GNU Tools Chain, Bluespec System Verilog Simulation Model – Bluespec Inc. and RISC-V Torture, CSMIT, AAPG.

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

};

RV32M\_Opcode\_Map {

u32 R\_mul; // 0000001 | rs2 | rs1 | 000 | rd | 0110011

u32 R\_mulh; // 0000001 | rs2 | rs1 | 001 | rd | 0110011

u32 R\_mulhsu; // 0000001 | rs2 | rs1 | 010 | rd | 0110011

u32 R\_mulhu; // 0000001 | rs2 | rs1 | 011 | rd | 0110011

u32 R\_div; // 0000001 | rs2 | rs1 | 100 | rd | 0110011

u32 R\_divu; // 0000001 | rs2 | rs1 | 101 | rd | 0110011

u32 R\_rem; // 0000001 | rs2 | rs1 | 110 | rd | 0110011

u32 R\_remu; // 0000001 | rs2 | rs1 | 111 | rd | 0110011

}

RV32F\_Opcode\_Map {

u32 I\_flw; //

u32 S\_fsw; //

u32 R4\_fmadd.s; //

u32 R4\_fmsub.s; //

u32 R4\_fnmsub.s; //

u32 R4\_fnmadd.s; //

u32 R\_fadd.s; //

u32 R\_fsub.s; //

u32 R\_fmul.s; //

u32 R\_fdiv.s; //

u32 R\_fsqrt.s; //

u32 R\_fsgnj.s; //

u32 R\_fsgnjn.s; //

u32 R\_fsgnjx.s; //

u32 R\_fmin.s; //

u32 R\_fmax.s; //

u32 R\_fcvt.w.s; //

u32 R\_fcvt.wu.s; //

u32 R\_fmv.x.w; //

u32 R\_feq.s; //

u32 R\_flt.s; //

u32 R\_fle.s; //

u32 R\_fclass.s; //

u32 R\_fvct.s.w; //

u32 R\_fvct.s.wu; //

u32 R\_fmv.w.x; //

}

RV32D\_Opcode\_Map {

u32 I\_fld; //

u32 S\_fsd; //

u32 R4\_fmadd.d; //

u32 R4\_fmsub.d; //

u32 R4\_fnmsub.d; //

u32 R4\_fnmadd.d; //

u32 R\_fadd.d; //

u32 R\_fsub.d; //

u32 R\_fmul.d; //

u32 R\_fdiv.d; //

u32 R\_fsqrt.d; //

u32 R\_fsgnj.d; //

u32 R\_fsgnjn.d; //

u32 R\_fsgnjx.d; //

u32 R\_fmin.d; //

u32 R\_fmax.d; //

u32 R\_fcvt.s.d; //

u32 R\_fcvt.d.s; //

u32 R\_feq.d; //

u32 R\_flt.d; //

u32 R\_fle.d; //

u32 R\_fclass.d; //

u32 R\_fcvt.w.d; //

u32 R\_fvct.s.wu.d; //

u32 R\_fvct.d.w; //

u32 R\_fvct.d.wu; //

}

|  |  |  |
| --- | --- | --- |
|  | f0 / ft0 | FP Temporary |
|  | f1 / ft0 | FP Temporary |
|  | f2 / ft0 | FP Temporary |
|  | f3 / ft0 | FP Temporary |
|  | f4 / ft0 | FP Temporary |
|  | f5 / ft0 | FP Temporary |
|  | f6 / ft0 | FP Temporary |
|  | f7 / ft0 | FP Temporary |
|  | f8 / ft0 | FP Saved register |
|  | f9 / ft0 | FP Saved register |
|  | f10 / ft0 | FP Function argument, return value |
|  | f11 / ft0 | FP Function argument, return value |
|  | f12 / ft0 | FP Function argument |
|  | f13 / ft0 | FP Function argument |
|  | f14 / ft0 | FP Function argument |
|  | f15 / ft0 | FP Function argument |
|  | f16 / ft0 | FP Function argument |
|  | f17 / ft0 | FP Function argument |
|  | f18 / ft0 | FP Saved register |
|  | f19 / ft0 | FP Saved register |
|  | f20 / ft0 | FP Saved register |
|  | f22 / ft0 | FP Saved register |
|  | f23 / ft0 | FP Saved register |
|  | f24 / ft0 | FP Saved register |
|  | f25 / ft0 | FP Saved register |
|  | f26 / ft0 | FP Saved register |
|  | f27 / ft0 | FP Saved register |
|  | f28 / ft0 | FP Temporary |
|  | f29 / ft0 | FP Temporary |
|  | f30 / ft0 | FP Temporary |
|  | f31 / ft0 | FP Temporary |

The floating-point registers of RV32F and RV32D.

RV32A\_Opcode\_Map {

u32 R\_1r.w; //

u32 R\_sc.w; //

u32 R\_amoswap.w; //

u32 R\_amoadd.w; //

u32 R\_amoxor.w; //

u32 R\_amoand.w; //

u32 R\_amoxor.w; //

u32 R\_amoand.w; //

u32 R\_amoor.w; //

u32 R\_amomin.w; //

u32 R\_amomax.w; //

u32 R\_amominu.w; //

u32 R\_amomax.w; //

}

|  |  |  |
| --- | --- | --- |
| 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, Northeastern 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. Full-System Simulation of Java Workloads with RISC-V and the Jikes Research Virtual Machine, Martin Maas, Krste Asanovic, John Kubiatowicz, University of California, Berkeley
10. The RISC-V Reader – An Open Architecture Atlas, First Edition, 1.0.0, David Patterson, Andrew Waterman, November 7, 2017
11. Implementation of Direct Segments of a RISC-V Processors, Nikhita Kunati, Michael M. Switch
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. <https://riscv.org/software-tools/risc-v-gnu-compiler-toolchain/> (Tools)
15. Spike – <https://github.com/riscv/riscv-isa-sim>
16. 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 (First Pass** – Done**)**
4. **ARMISS: An Instruction Set Simulator for the ARM Architecture (First Pass** - Done**)**
5. **ISA Semantics for ARMv8-A, RISC-V, and CHRI-MIPS (First Pass** – Done**)**
6. **Extensible and Configurable RISC-V based Virtual Prototype (First Pass** - Done**)**
7. **RISC5: Implementing the RISC-V ISA in gem5 (First Pass** - Done**)**
8. **Implementation of Direct Segments on a RISC-V Processor (First Pass** – Done**)**
9. **Full-System Simulation of Java Workloads with RISC-V and the Jikes Research Virtual Machine (First Pass** - Done**)**

**TBD/Open Issues** –

1. **Implementation Notes** - Diagrams for simulator execution and data flows.
2. **Implementation Notes** - Diagrams-block level to microarchitecture to software modules and processes (in Rust).
3. **Implementation Idea** - OPCODE interpretation optimizations techniques.
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 for the simulator?
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 the simulator module?
3. **Architecture Specific Question** – Are there co-processors - Functional Units standard across all cores and class of processors (RISC-V).
4. **Implementation Question** - What defines the state of the system? What all the processor subcomponents be inclusive in defining the state of the system?
5. **Implementation Question** - How will memory hierarchy be modeled? How will the delay statics be modeled? How will the instruction memory be stored?
6. **Implementation Question** – What all data transfer sizes will be supported?
7. **Implementation Question** - What is the cycle of instruction in RISC-V?
8. **Implementation Question** - Is it necessary in the ISS – behavioral model - to have cache warming mode for ROI (Region of Interest)? Do we need to look at ROI versus non-ROI regions?
9. **Implementation Question** - Where all would the trace files be created and their formats?
10. **Implementation Question** - Is design space exploration a requirement for this simulator?
11. **Implementation Question** – What are the realization specific micro architectural implications on this ISA?
12. **Implementation Question** - Processor State and Statistics Gathering, what and how?
13. **Design consideration** – How would a multi-threaded model of the program look like? Number of Machine involved?
14. **Goal Statement**: Primary applications for simulators consist of computer architecture studies and performance tuning of compiled software and the compilation process itself.
15. **Implementation Ideas** – Are we to implement dynamic binary translation (to x86\_64) techniques?
16. **Implementation Ideas** - How much detail is enough to model a processor?
17. **Implementation Questions** – What are the interfaces of this ISA that to be plugged into standard buses and memory interfaces?
18. **Implementation Question** –Interrupt handling and system calls (environment interaction), how?
19. **Implementation Question** – How do we address the issue of generating data for power and timing?
20. **Implementation Question** – How would the details of interrupt handling and system calls – interaction sequence as described by the code look like? (Software Side Code and the Model side itself).
21. **Implementation Question** – How would the debugger such as GDB work on the simulator?
22. **Implementation Question** – How do we model memory interactions? Even if these were simply to account for timing and integrity of sequence of execution.
23. **Implementation Question** –How will the issue of compact switching be impacted/addressed, this for the simulator?
24. **Implementation Question** – How does our implementation address the issues of bottlenecks in simulating the CPU core? Does this issue of multi-threading concern Behavioral Modeling?
25. **Implementation Note** – RISC-V follows the *release consistency* memory model.
26. **Implementation Note** – RISC-V follows IEEE-754 2008 floating-point standard. Simulator implementation will be targeted to confirm to the specification.
27. **Implementation Note** – How do we address different simulation paradigms high-level architectural and memory models, and advanced simulation features? Balancing achieving of high accuracy with reduced simulation time.
28. **Implementation Question** – Will implement system call emulation (SE) mode that will replace program’s own system calls with ones on/to the host?
29. **Implementation Question –** How are the checkpoint saved and used to resume the simulation?
30. **Implementation Question –** Will implementation support features and techniques such as phase analysis? Implications?
31. **Implementation Question** – What all are the data sets that will need to be generated to feed as input to external tools so that we can get estimations such as power, temperature, voltage noise etc.? Is some of these tabulated so the we can embed in Behavioral Model.
32. **Implementation Question** – What all are the test programs we should be executing to test the Behavioral Model and development tools setup/flow?
33. **Implementation Question** – How do we separate microarchitecture implementation form the general-purpose functions allowing adaption of simulator for different classes of processors?
34. **Implementation Question** – How does a simulator such as this help support the need for hardware-software co-design? RISC-V is an open source ISA and can be implemented on an FPGAs for full-system simulation that can yield better results.
35. **Implementation Question** – Particularly for managed-language research how do you achieve realism, fidelity and simulation speed required to simulate managed-language workloads. Can we port something like JikesRVM to this simulator? Are FPGA based simulators better for these sorts of research?
36. **Implementation Question** – Should we be targeting to use test-suites by RISC-V Compliance Working Group?
37. **Implementation Question** – Should we have comparable output as with Spike or other simulators? Should we look to compare trace files?
38. **Implementation Question** – What sort of test tools should we look forward to integrating with the simulator?
39. **Implementation Comment** - Along with behavioral aspects entailing instruction set, hardware-software interfacing, simulator could also look at the concurrency mode and interrupt behavior.
40. **Implementation Question** – What are the CPU performance benchmarking programs we should support to bring out any deficiency in implementation?
41. **Implementation Question** – How would we test the entire software stack particularly entailing network card, disk drive etc.?
42. **Implementation Question** – How would we simulate multicore processors?
43. **Implementation Question** – How do we handle disruptive miss events?
44. **Implementation Question** – Are there any implications of interval simulation/analysis technique or aspects of the same on the simulator we are developing?
45. **Implementation Question** – What kind of disruptive miss events circumventing solutions modeling we intend to support with this simulator?
46. **Implementation Question** – GUI screen shots and terminal - showing the possible set of control commands for the simulator.

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)