ISA Formal Spec Technical Group Update

Rishiyur S. Nikhil
nikhil at bluespec.com

7th RISC-V Workshop
Wednesday, November 29, 2017

Technical Group Members

Industry/Labs

BAE Systems:
- Arun Thomas

Bluespec:
- Rishiyur Nikhil (Chair)

Dover Microsystems:
- Greg Sullivan

LowRISC:
- Alex Bradbury

Microsemi:
- Stuart Hoad

NVIDIA:
- Daniel Lustig
- Joe Xie

SiFive:
- Jacob Chang

Syntacore
- Pavel Smirnov

Academia

IIT Madras
- Rahul Bodduna
- Neel Gala
- Vinod Ganesan
- G. Madhusudan

MIT:
- Thomas Bourgeat
- Adam Chlipala
- Murali Vijayaraghavan
- Andy Wright

U.Cambridge, England:
- Shaked Flur
- Christopher Pulte
- Peter Sewell

U.St.Andrews, Scotland:
- Susmit Sarkar

Individuals

- Don Bailey
- Michael Clark
- James Cloos
- Dan Hopper
- Po-wei Huang
- Prashant Mundur (SRI)
- Clifford Wolf

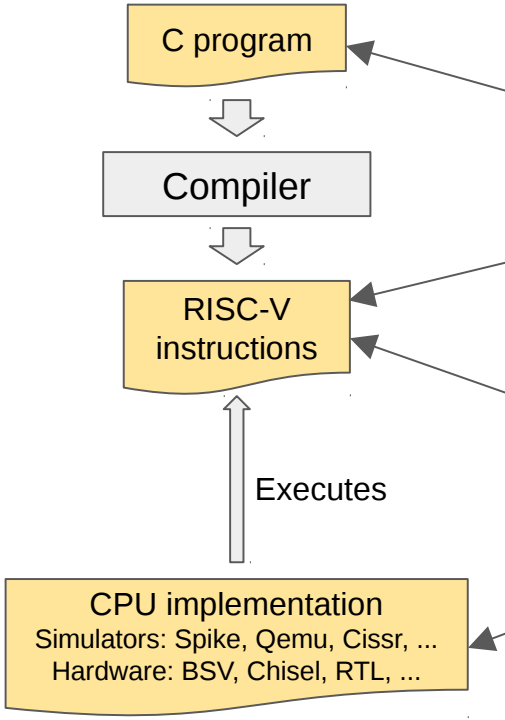
About 10-13 people attend our weekly con-call.
Many participate in multiple TGs.

Plan:

- What is an ISA Formal Spec? Of what use is an ISA formal spec?
- Our approach
- Excerpts of the spec, to provide flavor
- Status and plans

Of What Use is an ISA Formal Spec?

It's a key requirement to be able definitively to answer questions such as the following:



Compiler correctness
- Will executing these two produce the same results?
- For all inputs?
- For all C and corresponding RISC-V programs?

Implementation correctness
- Will executing this RISC-V program on this implementation produce correct results?
- For all RISC-V programs?

There are many other uses as well; the unifying theme is: Provable Correctness
See also the talk that follows this one,
"Strong Formal Verification for RISC-V: From Instruction-Set Manual to RTL" by Adam Chlipala (MIT) for an ambitious project for end-to-end correct RISC-V CPU implementations.

Of What Use is an ISA Formal Spec? (Contd.)

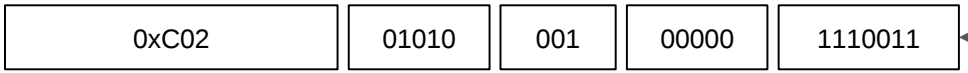
Some examples of actual bugs found using Formal Verification, by Clifford Wolf (member of our TG):

- JALR does not clear LSB after computing effective address
  - Usually not encountered in running compiled code
- Bypassing/forwarding/speculation errors in the CPU pipeline
  - May need a very carefully constructed test to provoke
- Disabling the ‘C’ extension dynamically, when PC is not 32b aligned.
- ...

These bugs were in implementations, a simulator, and even in the English language spec (and even in the formal spec!)

Of course, each of these could be provoked by a suitable test program,
but they are often corner cases that may be unanticipated (so we never wrote a test)
or, even if anticipated, may be very difficult to conduct a test that provokes it.

What is an ISA Formal Spec?



What exactly does this instruction do?

= CSRRW x0, instret, a0

If a0 contains 100, does instret contain 100 or 101 after this?

An ISA Formal Spec is a <u>precise</u> definition of the functionality of

1. *each RISC-V instruction*
2. *Any RISC-V hart (sequences of instructions)*
3. *Any collection of RISC-V harts*

Note: functional spec only; does not aim to characterize performance or resources.

Note: 1 and 2 can mostly be defined and understood independent of the memory model (cf. Dan Lustig's talk on Memory Models, yesterday)

What is an ISA Formal Spec? (Contd.)

We want our Formal Spec to be

- *Clear and understandable to the human reader*
  - *Not cluttered with implementation considerations; no micro-architectural detail*
  - *Accessible to HW implementers, simulator implementers, compiler writers, OS implementers, concurrent shared data structure authors, ...*

- *Precise and complete*
  - *(including allowed non-determinism of some instructions)*

- *Machine readable*
- *Executable (run RISC-V programs, boot an OS)*
- *Usable with various formal tools (theorem provers, model checkers, verifiers, …)*

English text specs, and instruction-set simulators (Spike, Qemu, Cissr) can be regarded as specs, but typically do not meet many of these goals.

Our approach

Given the diversity of tools for formal methods, there is no widely accepted “common language” for this

- *ISA-centric DSLs: … L3, Sail, ARM Specification Language*
- *General-purpose: …, Haskell, Coq, PVS, ...*

We’ve chosen to write the spec in a simple, pure functional language

- Easy to understand, for the human reader
- This is the preferred representation for most tools for formal methods.

So, our approach:

- Use a simple “Functional Programming 101” subset of Haskell
  - (Avoiding the more esoteric features of Haskell)
- Directly executable in Haskell
- Provide parser(s) to connect to other formal tools and formats[1] (Coq, PVS, Isabelle/HoL, L3, Sail, …)

[1] *It is perhaps inevitable that others will want/ write RISC-V ISA Formal Specs in other formal notations, for their use cases, to connect to existing tooling, etc.*
It will be necessary to prove equivalence between different renderings of the Spec, and to create translators between renderings.

Our approach (contd.)

Provide the spec in stages, corresponding to organization of the English-language spec:

- First targets (currently in good shape):
  - RV32I/RV64I, M, Priv Spec M, assume simple memory model

- Later (not necessarily in this order) add support for
  - standard extensions: C, A, F, D
  - Priv Spec S
  - Multi-hart: integrate with Formal Memory Model
  - newer standard extensions: Vector, Bit Manip, …

A group at U.Cambridge (Peter Sewell, Shaked Flur, Christopher Pulte) has independently been pursuing a formalization of the RISC-V ISA and Memory Model, using "Sail", their DSL for ISA specs, based on many years of experience formalizing ARM/MIPS/Power ISAs and Memory Models. They are also active participants in this Technical Group.

Some excerpts, to give you a flavor of the look and feel: *Decode*

https://github.com/mit-plv/riscv-semantics

```
-- Data type declaration for decoded instructions
data Instruction =
  InvalidInstruction |
  Lw    { rd :: Register, rs1 :: Register, oimm12 :: MachineInt } |
  Addi  { rd :: Register, rs1 :: Register, imm12  :: MachineInt } |
  Csrrw { rd :: Register, rs1 :: Register, csr12  :: MachineInt }
  ...

-- Decode function from 32b words to decoded-instruction data type
decode :: Int -> MachineInt -> Instruction
decode xlen inst = decode_sub opcode
  where
    opcode  = bitSlice inst 0 7        -- = Verilog's: inst [6:0]
    funct3  = bitSlice inst 12 15
    rs1     = bitSlice inst 15 20
    ...
    decode_sub opcode
      ...
      | opcode==opcode_LOAD, funct3==funct3_LW            = Lw  {rd=rd, rs1=rs1, oimm12=oimm12}
      | opcode==opcode_LOAD, funct3==funct3_LD,  xlen==64 = Ld  {rd=rd, rs1=rs1, oimm12=oimm12}
      | opcode==opcode_OP_IMM, funct3==funct3_ADDI        = Addi  {rd=rd, rs1=rs1, imm12=imm12}
      | opcode==opcode_SYSTEM, funct3==funct3_CSRRW       = Csrrw {rd=rd, rs1=rs1,   csr12=csr12}
      ...
```

Some excerpts, to give you a flavor of the look and feel: *Instruction execution*

https://github.com/mit-plv/riscv-semantics

```
execute (Addi rd rs1 imm12) = do
  x <- getRegister rs1
  setRegister rd (x + fromIntegral imm12)

execute (Beq rs1 rs2 sbimm12) = do
  x <- getRegister rs1
  y <- getRegister rs2
  pc <- getPC
  when (x == y) (setPC (pc + fromIntegral sbimm12))

execute (Lw rd rs1 oimm12) = do
  a <- getRegister rs1
  x <- loadWord (a + fromIntegral oimm12)
  setRegister rd x

execute (Csrrw rd rs1 csr12) = do
  x <- getRegister rs1
  when (rd /= 0) (do
    y <- getCSR (lookupCSR csr12)
    setRegister rd y)
  setCSR (lookupCSR csr12) x
```

Will get more detailed as we address allowed non-determinism due to real memory models.

Some excerpts, to give you a flavor of the look and feel: *Programs*

https://github.com/mit-plv/riscv-semantics

```
run_loop = do
  ...
  pc <- getPC
  inst <- loadWord pc
  size <- getXLEN
  execute (decode size (fromIntegral inst))
  interrupt <- ... checkInterrupt ...
  if interrupt then do
    setCSRField Field.MEIP 1
  step
  run_loop
```

This is just an illustrative run-loop showing how to compose 'decode' and 'execute', for a simple, single-hart, simple-memory-model.
It'll get more detailed as we go to multi-hart and real memory models.

Status and plans

- Done: RV32I/RV64I, M, Priv Spec M, ignoring mem model issues
- Soon: Priv Spec S
- Next: A, C, F, D, integration with Memory Model
- Later: Other standard extensions (Vector, Bit Manip, …)

Several groups are using or are planning to use ISA Formal Specs for Formal Verification:

- *MIT (see next talk by Adam Chlipala)*
- *Clifford Wolf*
- *Bluespec*
- *SiFive*
- *SRI*
- *U.Cambridge (Peter Sewell et.al.) also have a Sail model for the core RISC-V user-level ISA, integrated with their operational concurrent model of the RISC-V Weak Memory Model.*
- *… and possibly many more ...*

Questions? Discussion?
Thank you!

Please see me or any member of the group

- if you have more questions
- would like to participate
- use the spec in your work
- etc.