Verifying a RISC-V Processor

Nirav Dave **Prashanth Mundkur**

SRI, International

Verified Software

Recent work on verifying key components of software toolchain:

- compilers (*e.g.*,CompCert, CakeML, *etc.*)
- OS kernels (*e.g.*,seL4)
- hypervisors (*e.g.*,minVisor (x86), XMHF (x86))

Verified Software

Recent work on verifying key components of software toolchain:

- compilers (*e.g.*,CompCert, CakeML, *etc.*)
- OS kernels (*e.g.*,seL4)
- hypervisors (*e.g.*,minVisor (x86), XMHF (x86))

Catch: all assume correct hardware.

Real-world hardware

Intel Specification Update (328899-024), March 2015

- HSD1: LBR, BTS, BTM May Report a Wrong Address when an Exception/Interrupt Occurs in 64-bit Mode
- HSD3: MCi-Status Overflow Bit May Be Incorrectly Set on a Single Instance of a DTLB Error
- HSD5: MONITOR or CLFLUSH on the Local XAPIC's Address Space Results in Hang
- HSD20: Accessing Physical Memory Space 0-640K through the Graphics Aperture May Cause Unpredictable System Behavior
- HSD27: Processor May Enter Shutdown Unexpectedly on a Second Uncorrectable Error

Real-world hardware

Intel Specification Update (328899-024), March 2015

- HSD1: LBR, BTS, BTM May Report a Wrong Address when an Exception/Interrupt Occurs in 64-bit Mode
- HSD3: MCi-Status Overflow Bit May Be Incorrectly Set on a Single Instance of a DTLB Error
- HSD5: MONITOR or CLFLUSH on the Local XAPIC's Address Space Results in Hang
- HSD20: Accessing Physical Memory Space 0-640K through the Graphics Aperture May Cause Unpredictable System Behavior
- HSD27: Processor May Enter Shutdown Unexpectedly on a Second Uncorrectable Error

140 items, all marked *No Fix*.

Real-world hardware

Cortex-A53 errata (ARM-EPM-048406 v17.0, 2015)

- 812869: Instruction stream might be corrupted
- 835769: AArch64 multiply-accumulate instruction might produce incorrect result
- 843419: A load or store might access an incorrect address
- 814270: Misaligned PC and out-of-range address aborts might be taken to incorrect exception level
- 845719: A load might read incorrect data

Real-world hardware

Cortex-A53 errata (ARM-EPM-048406 v17.0, 2015)

- 812869: Instruction stream might be corrupted
- 835769: AArch64 multiply-accumulate instruction might produce incorrect result
- 843419: A load or store might access an incorrect address
- 814270: Misaligned PC and out-of-range address aborts might be taken to incorrect exception level
- 845719: A load might read incorrect data
- . . . (37 pages)

Real-world hardware

RISC-V ?

Minimum prerequisites for RISC-Verified

- unambiguous formal ISA specification

Minimum prerequisites for RISC-Verified

- unambiguous formal ISA specification
- *a* processor implementation amenable to verification

Minimum prerequisites for RISC-Verified

- unambiguous formal ISA specification
- *a* processor implementation amenable to verification
- formal link between the two

Formal ISA specification

- Specifying RISC-V in the L3 DSL
  - generates executable ISA interpreter (Standard ML)
  - exports definitions for theorem prover (HOL4)
- Use interpreter as reference oracle for processor implementations

L3 DSL

- constructs for registers, instructions and state
- ML-like data-types
- strong bit-level type-checking
- library for bit-vector ops, floating-point, etc.

Current status

- implements interrupts and virtual memory
- unoptimized (no icache or dcache)
- passes most riscv-tests (-p-, -pt-, -v-)

Available at `github.com/pmundkur/l3riscv`

Clarifications needed

Which has priority: a synchronous exception or an interrupt?

Clarifications needed

Misaligned fetch is trapped before side-effects in JAL/JALR.

Next Steps

Completing the executable specification:

- RVC
- floating-point
- SBI (when available)
- boot single-core Linux/FreeBSD
- use for tandem-verification (*e.g.*,with Flute from Bluespec, Inc)

Next Steps

- usable HOL4 formal definitions
- prove non-interference and information flow in low-level privileged code
- tie to architectural predicates for hardware implementations

Processor verification via Architectural Extraction

Motivation

- Verify architectural and μ-architectural variants of RISC-V
- Robust to design and implementation changes
- Quick formal verification of design
  - using architect-friendly approach

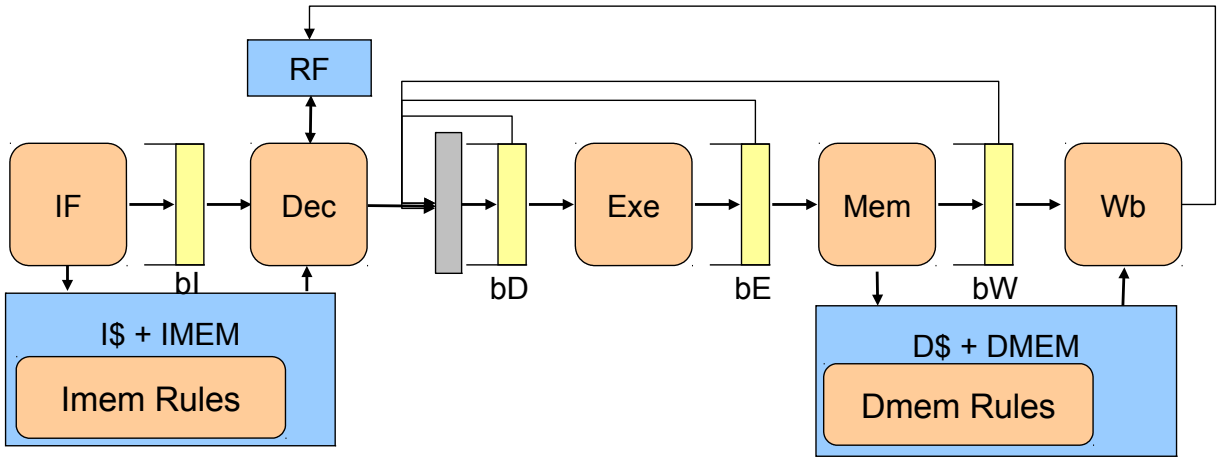
Motivation

- Verify architectural and μ-architectural variants of RISC-V
  - using parametrized BluespecVerilog designs
- Robust to design and implementation changes

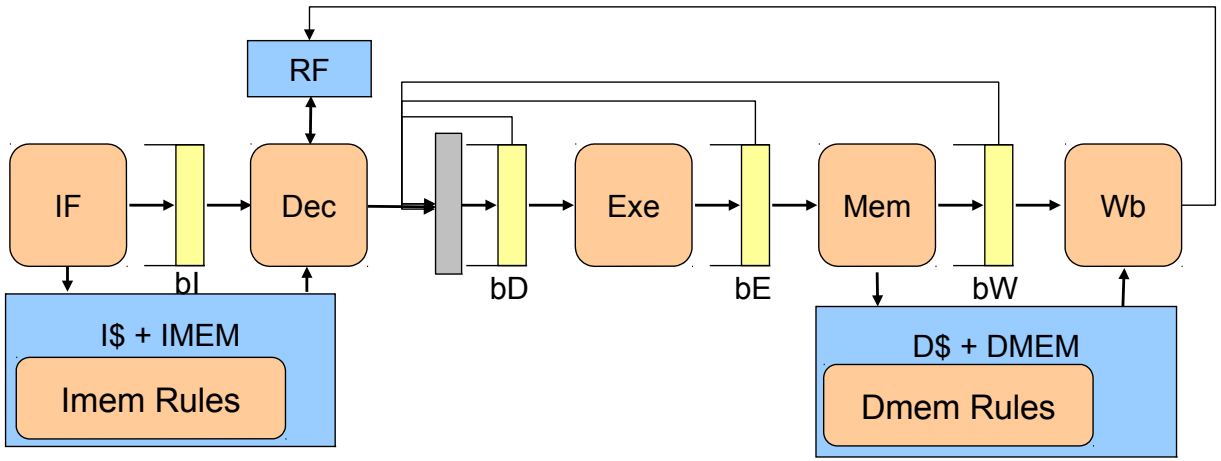
- Quick formal verification of design
  - using architect-friendly approach

BSV-Based Design

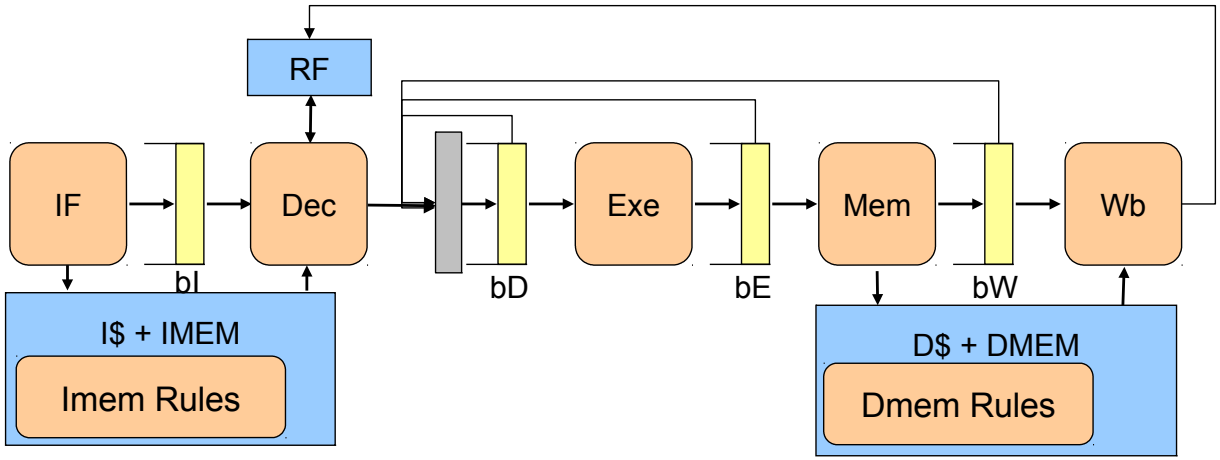
- Break ISA-level transactions into smaller transactions with better cycle-level parallelization
  - add/modify system state (*e.g.*,pipeline buffers)
  - re-order operations / perform speculation
- Transactions in design $\equiv$ rules in BSV
  - design correctness $\equiv$ re-arrangement of μ-transactions into ISA-level operations



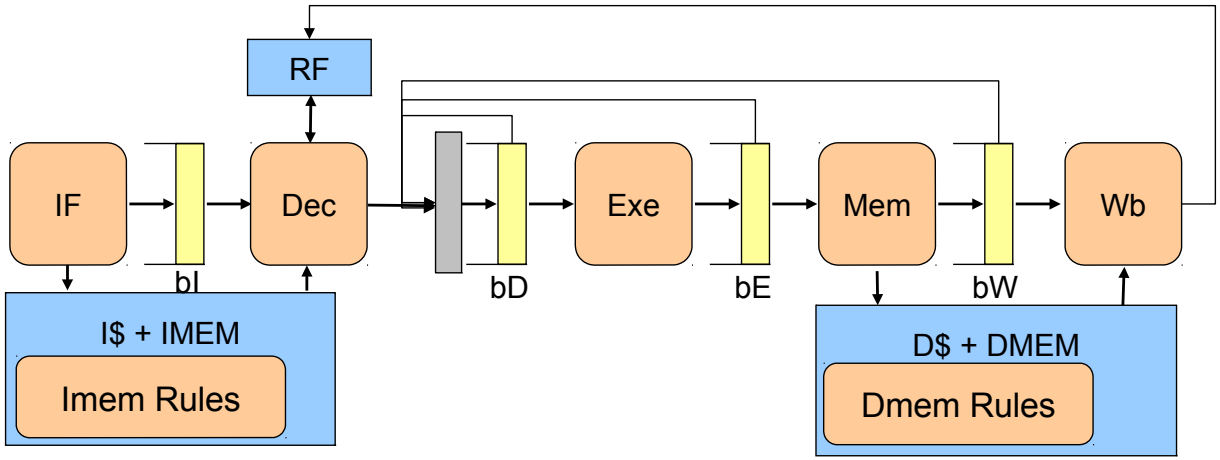
- **Pipeline view:** stages execute (mostly) in reverse order
  - concurrent execution of instructions
- **ISA view:** stages execute in pipeline order
  - one instruction at-a-time



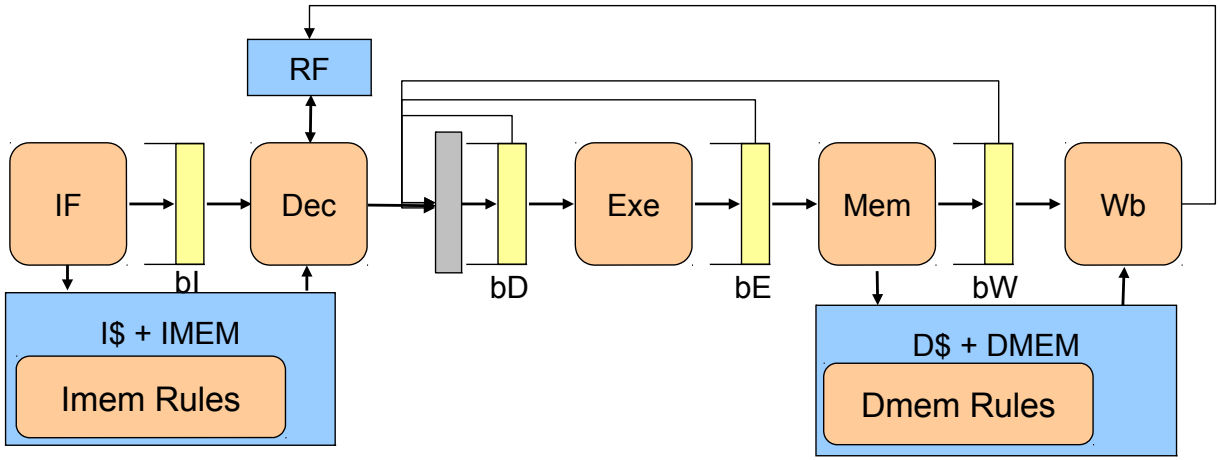
- find a mapping of μ-architectural rule exections to coarser (ISA-level) rule executions
- pipeline correctness derived from commutativity of rule execution order



- Two rule sequences are equivalents iff they start and end in equivalent states



- Two rule sequences are equivalents iff they start and end in equivalent states
  - what if instructions are partially executed?



- Two rule sequences are equivalents iff they start and end in equivalent states
  - what if instructions are partially executed?
  - easier for ISA-level states (*i.e.*,flushed pipelines)

Formalizing this observation

- Given predicate on μ-state that picks out ISA-level states, search for covers of all possible rule traces

Formalizing this observation

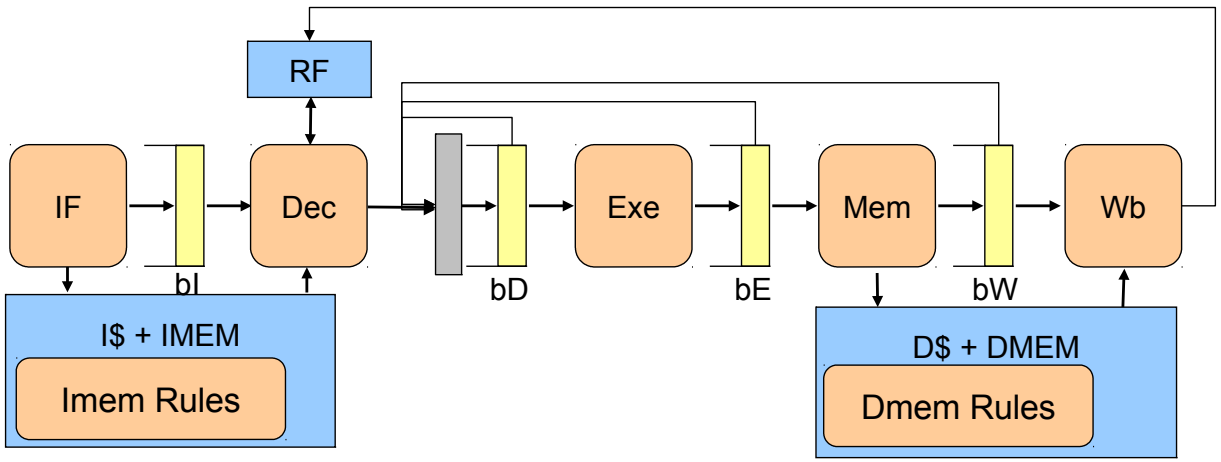
- Given predicate on μ-state that picks out ISA-level states, search for covers of all possible rule traces
- *i.e.*,perform a stuttering bisimulation search

Formalizing this observation

- Given predicate on μ-state that picks out ISA-level states, search for covers of all possible rule traces
- *i.e.*,perform a stuttering bisimulation search
  - robust to modular changes and pipeline variations

Formalizing this observation

- Given predicate on μ-state that picks out ISA-level states, search for covers of all possible rule traces
- *i.e.*,perform a stuttering bisimulation search
  - robust to modular changes and pipeline variations
- Does not guarantee correctness of implemented ISA
  - only provides an abstracted "ISA"-level model



- Expected traces (mapping to ISA-level executions):
  - Single instruction:
    [IF, IMem, Dec, Exe, Mem, Dmem, Wb]
  - Branch misprediction:
    [IF, IMem, Dec, Exe] [IF, IMem, Dec]
  - Pipeline bug:
    [(IF, IMem)×2, Dec, Exe, Dec, Exe, (Mem, DMem, Wb)×2]

Flute μ-architecture

- analysis-friendly pipeline design from Bluespec, Inc
- modularized pipeline stages allow changes in pipeline timing

Current status

First steps: basic correctness

- verify FIFO channels maintain atomicity expectations
  - show potentially unsafe use of BSV RWires are correct
- show processor pipeline $\equiv$ to unpipelined processor
  - no order-dependant pipeline bugs

Eventual Goal

Full abstraction of Flute Microprocessor to ISA-level design

- Block-level microarchitectural modular predicates to allow easy refinement
- Compilation of ISA-level traces to L3 RISC-V model for functional verification
  - Verify implemented Flute matches L3 design at ISA-level

Questions?

Thank you!

Improving the Spec

Ambiguities in the spec

- sstatus/mstatus projections for read/write
- pseudo-code for instructions (e.g. CSRR[SC])
- interrupt / exception priorities