Compliance Task Group Call – Agenda

Weds, May 22 , 2019 8am Pacific →Daylight← Time

Charter

The Compliance Task Group will

- Develop a framework for RISC-V tests, taking into account approved specifications for:
  - Architectural versions (e.g. RV32I, RV32E, RV64I, RV128I)
  - Standard Extensions (M,A,F,D,Q,L,C,B,J,T,P,V,N)
  - All spec'ed implementation options
    - (incl. MHSU modes, optional CSRs, optional CSR bits)
- Develop a method for selecting and configuring appropriate tests for a RISC-V implementation, taking into account:
  - Platform profile and Execution Environment (EE)
  - Implemented architecture, extensions, and options
- Develop a method to apply the appropriate tests to an implementation and verify that it meets the standard
  - test result signature stored in memory will be compared to a golden model result signature

Adminstrative Pointers

- Chair – Allen Baum allen.baum@esperantotech.com
- Co-chair – Stuart Hoad stuart.hoad@microchip.com
- TG membership- Sue Leininger sue@riscv.org
  - Send email to her - you must have a workspace.riscv.org login
- TG Email compliance@workspace.riscv.org
  - Notetakers: please send emails to allen.baum@esperantotech.com
- Meetings -Bi-monthly at 8/9am Pacific time on 2nd/4th Wednesdays
  - Location is https://zoom.us/j/6213886723
- Documents, calendar, roster, etc. in https://lists.riscv.org/g/tech-compliance/
  see /documents, /calendars subdirectories
- Git repositories
  - https://github.com/riscv/riscv-compliance/
  - https://github.com/rsnikhil/Experimental_RISCV_Feature_Model
  - https://github.com/rsnikhil/Forvis_RISCV-ISA-Spec
  - https://gitlab.com/incoresemi/riscof (Shakti framework)

Attendees

• Allen Baum	(Esperanto Technologies)
• Jeffery Osier-Mixon	
• Jeremy Benet	(Embocosm)
• Ilja Stepanov	(Syntacore)
• Grant Martin	(Cadence)
• Jacob Chang	(SiFive)
• Marek Masarik	(Codasip)
• Milan Skala	(Codasip)
• Neel Gala	(Shakti)
• Deborah Soung	(SiFive)

Meeting Agenda

1. Continue TestSpec proposed changes pull #57;
   - Review of last meeting discussion, (sec 1.3 Future Work updated to indicate WIP)
   - Directory restructuring/renaming RV<xlen><mode><extension*> (see slide 9)
     - Placement/naming of dependent extensions (e.g.should C.MUL be, in RV32C, RV32M, or new in RV32CM, RV32MC?)
   - Review of new standard macro proposals to simplify test authoring and give more common structure to tests (see slide 10)
     - RVTEST_SIGBASE(basereg,val) sets signature base register to be used in tests; could gen code to move it if redefined
     - RVTEST_CASE(name,<cond_str?>) keeps track of test#, updates sigbase, could also provide conditions for framework
     - RVTEST_SIGUPDATE(srcreg [, assertion_val]) stores signature value at offset from sigbase; if enabled, could generate debug code

     – depends on existence of ARM SETA/Microsoft"=" macro direct equivalents
   - Allowing absolute addresses (see slide 11)
   - Test selection proposal: central database vs. included in std test macro (see RVTEST_CASE) (see slide 10)
   - Other "To Be Discussed" sections should be reviewed
2. Test Coverage discussion (see slide 12)
   - Questions from last meeting: SOC vs architecture coverage
     - e.g. how do have adequate coverage for platform limited features (memory map)
     - ARM includes memory maps in AVS tests
     - How did Microchip deal with this?
   - Testing eBreak without ending test: do we need to? (eBreak expects special noop before/after inst for debug)
   - Instruction categories & coverage types (e.g. all,walking0/1,)

4. WARL proposal – approve, change, reject: (see slide 13, pull 52)
   - Restrict field definition to 3 types (+variants), reduce compliance effort
     - Amend to deal with specific illegal address exception
   - Applies to fields without specific architectural requirements
   - Add Priv spec wording that compliance will only check fields meeting these definitions (if time permits) test spec

Discussion

Pulls

1. #51: reviewed, out of date, will be merged after update – done
2. #54 grift: point Ben Selfridge at my revised code for misaligned test
3. #55 this is the test to point it to Neil will review

   Bugs found and (allegedly) fixed, proper commenting replacement, more comments

1. Test Spec : get lee, simon to comment on 1.3 (and other proposals
   1. future work sec 1.3 – get Simon to review
   2. Directory restructuring- no comments, no volunteers to fix current mess
   3. New standard Macros: Question of whether "hidden variable" was possiblestandard flow of running through CPP enables it
   4. Need to combine multiple riscv.h files into a single file
   5. Need to update test spec to clarify issues:

      4.4.1.5: **test selection**: remove/rewrite to use **Common compliance test pool reference document** & define syntax

      4.4.1.6: Absolute address restriction: remove after review

      4.4.1.TBD1: to Harvard Architectures: remove reference

      4.4.1.TBD.2: Macro restrictions: rewrite to explain a macro must only exist in one file

      4.4.1.TBD.3: New std Macros: rmv, then add documentation for new macros in 4.4 (requires major test work)

      4.4.2.TBD.1, .2: Find answers to YAML and document

      4.4.2.TBD.3: Use of #ifdef & future headers: Remove everything after "[AJB] Note"
2. Directory structure, Test coverage (slide 9)

3. WARL proposal: Neel is updating Forvis to add WARL types to YAML

Conclusions & Action Items

- Decisions

1. Get review of #54
2. no objections heard except removal of absolute address restriction and note that spec is WIP and doesn’t match current tests
3. no objections to restructuring (no volunteers either)
4. Neel is prototyping

- Action Items

1. fix bugs in I_MISALIGN_JMP
2. Update spec (Done), gather feedback about absolute address issues
3. Need someone restructure test directory and collateral that uses it. Allen will document new structure first
4. Neel will present YAML to support WARL & possibly framework that uses it . Need to write tests that take advantage of it

Backup from discussions

Instruction Test Coverage

Classes of things we want to test for

- Decode
  - Immediate – test all bits in either polarity will affect output
  - Register specifiers – test that changing any bit will affect output, ensure all regs are tested
  - Variations – test values of opcodes suffixes that have any string after a "." in its opcode
- Register combinations
  - Destructive (dest = either src) and non-destructive
  - Non-updating (i.e., targeting X0), or non-supplying (X0 as an input)
  - All registers (or immediate bit) should be used per instruction *category*
- Special and exception cases
  - Explicitly defined (e.g. shifts>=XLEN & RD=X0
  - Implicitly defined – corner cases
    - Maximal and minimal inputs, or creating maximal outputs
    - Inputs that special case outputs (mostly FP cases, also. shiftamt>=XLEN)
    - Outputs crossing value boundary (e.g. address cross word/page/superpage/VA boundary, FP crossing exponent boundary)
- Some of this is in "Future Work" document in Item 5 of the "Progress: Test Development" section.
  - See: https://docs.google.com/document/d/1mDOjdz2wy7aI9-vBB6nm-aPVm4O6TPSqqX7-0HwFaj8

proposed
coverage & categories

Arith[I],	*W1/0,crys*
Logical[I],	*W1/0*
Shift[I],	*W1/0/msk,+*
Auipc,Lui,	
Ld,St,	*W1/0, bndXing*
Br,	*W1/0, bndXing*
Jmp ,	*W1/0, bndXing*
Ebreak,	
Ecall	

Directory structure / Test naming

- Last meeting: decided to keep hierarchical directory
- Framework should use a unified database that lists each test and the conditions under which it should be executed
  - E.g. mode, extension supported address widths, CSRs implemented, etc
  - Slightly redundant with directory structure,testnames, but more flexible
- Directory format is RV[**32i | 32e | 64i][MSU_]{extension}**
  - **U** tests should not touch privileged resources, nor trap
  - _ tests can be executed in any mode with identical results
  - **M/S** tests may simply be U tests with an environment that causes a trap
  - Compressed ops should be located in the appropriate extension directory; database will label it to only be executed if C-ext is implemented& enabled

Standard test structure & macros

to make writing tests easier

- RVTEST_SIGBASE(basereg,val) – defines base register used to store signature values, initializes it, and clears sig_offset
  - (e.g .set SIGBASE val; .set SIGOFF 0; la SIGBASE, val)
  - Could this change during test? (&auto generate a move from old to new base)?
- RVTEST_PART(name,cond_str)
  - This increments the test_section variable & lays out requirements for each test section;
  - Framework will removes the section if conditions unmet
  - Do we need Test_case_end? E.g. can Test_case_start end the previous part & start a new one
  - Also need a way to pass values of options into code – here or elsewhere?
- RVTEST_SIGUPDATE(srcreg [, assertion_val])
  - Store srcreg into sigoffset(sigbase) and increments sigoffset
    - ST srcreg, SIGOFF(SIGBASE); .set SIGOFF SIGOFF +XLEN/8
  - Optional assertion value; Framework uses it to build a signature file, and will insert checking/printing code for debug if separately enabled
- RVTEST_SIGBASE() and RVTEST_SIGUPDATE each replace a single instruction, and perform framework housekeeping; RVTEST_PART is used by framework and only performs housekeeping

Absolute Addresses in tests

The current compliance test specification is intended not to be self-checking, but instead to store a signature.
Self-checking compares signature to a "correct" value, and branches to a success or failure label as a result.

The spec has this requirement:
Tests shall not store absolute addresses from the program in the signature.

This is somewhat in conflict with the directive that test not be self-checking:
Any test whose correct result depends on a PC needs to be specially written to avoid an absolute address.
Examples: tests that check for the correct return address of a JAL[R], or the TVAL or EPC of a trap.
In order to check for these are correct, the test must convert these to a relative address by subtracting a known absolute address, either by
- having an absolute address in the code (e.g. the entry point), or
- having some previously saved PC in a register (e.g. from an AUIPC)

The former is a crude work-around for self checking code; the only difference is that it doesn't branch to separate success/fail labels.
The latter is a self referential; it tests that the relative difference is correct, but there is no guarantee that both absolute addresses are correct.

Tests that depend on data addresses have similar issues.

- what is the rationale for the prohibition on absolute addresses?
- what would break if we stored absolute addresses

My opinion: serves no useful purpose, makes tests longer. complicates test development, and hides some classes of errors.
Possible downside: would not be possible to store universal signatures,
e.g a signature that would work regardless of the platform configuration and load addresses,
t would have to be computed dynamically by running the test on both the golden model and the device under test.

Is that a good enough reason give the drawbacks?

WARL field Restriction Proposal

3 Modes with variants:

- Range(Mode{Saturate|Unchgd |Addr} Upper,Lower)
- Bitmask(Mask<==1 is RW>, FixedVal<val if Mask=0>)
- Distinct(Mode{*see table*} {LegalList = Val,Val…,Val}

Range Mode:

```
 Saturate: if       (WriteVal) <Lower, Lower
              elseif (WriteVal) >Upper, Upper
              else    (WriteVal)
Unchged: if       (WriteVal <Lower) | (WriteVal >Upper), CurrVal
Addr:       out of range flips MSB
```

Mask: (WriteVal & Mask) |(FixedVal & ~Mask)

Distinct		Legal		Legal				Legal	
Mode	<1	1	2	3	4,5	6	7,8	9	>9
Largest			9		9	9	9		
Smallest			1		1	1	1		
NextUp	1	1	3		9	9	9		9
NextDn			1	3	3	3	3	9	
NearUp			3		3	9	9		
NearDn			1		3	3	9		
UnChgd	curr Val		curr Val		curr Val	curr Val	curr Val		curr Val

Distinct: if Write is not an element of LegalList, depending on Mode either

```
 Unchd:    Currval
 Largest:   max(LegalList)
 Smallest:  min(LegalList)
 NextUp:    ceiling(WriteVal)  //next   larger element of list or largest
 NextDn:    floor(   WriteVal)  //next smaller element of list or smallest
 NearUp:    ceiling(WriteVal) //pick   closest element of list, ties go up
 NearDn:    floor(   WriteVal) //pick   closest element of list, ties go down
```

Note: These restrictions apply to fields that do not have architectural requirements, but that are completely implementation defined, i.e. their final value does not depend on any other state variable in the implementation.