Architectural Test SIG Call –Minutes

Thur, 27Jan2022 8am Pacific → Standard ← Time

See slide 7 for agenda

RISC-V attendance

Only RISC-V Members May Attend

- Non-members are asked to please leave.
- Members share IP protection by virtue of their common membership agreement. Non-members being present jeopardizes that protection
- It is easy to become a member. Check out riscv.org/membership
- If you need work done between non-members or or other orgs and RISC-V, please use a joint working group (JWG).
  - used to allow non-members in SIGs but the SIGs purpose has changed.
- Please put your name and company (in parens after your name) as your zoom name. If you are an individual member just use the word “individual” instead of company name.
- Non-member guests may present to the group but should only stay for the presentation. Guests should leave for any follow on discussions.

RISC-V®

Antitrust Policy Notice

RISC-V International meetings involve participation by industry competitors, and it is the intention of RISC-V International to conduct all its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of, and not participate in, any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.

Examples of types of actions that are prohibited at RISC-V International meetings and in connection with RISC-V International activities are described in the RISC-V International Regulations Article 7 available here: https://riscv.org/regulations/

If you have questions about these matters, please contact your company counsel.

RISC-V®

Collaborative & Welcoming Community

RISC-V is a free and open ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.

We are a transparent, collaborative community where all are welcomed, and all members are encouraged to participate. We are a continuous improvement organization. If you see something that can be improved, please tell us. help@riscv.org

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone.

https://riscv.org/community/community-code-of-conduct/

RISC-V®

SIG Charter

The Architectural Compatibility Test SIG is an umbrella group that will provide guidance, strategy and oversight for the development of tests used to help find incompatibilities with the RISC-V Architecture as a step in the Architectural Compatibility self-certification process

The group will:

- Guide Development of:
  - Architectural tests for RISC-V implementations covering ratified and in-flight specifications for
    - Architectural versions, standard extensions, and implementation options.
  - Tools and infrastructure to help identify architectural incompatibilities in implementations
- Work with LSM and Chairs for resources to get the above work done.
- Mentor or arrange for mentoring for the resources to get the above work done

Adminstrative Pointers

- Chair – Allen Baum allen.baum@esperantotech.com Co-chair – Bill McSpadden bill.mcspadden@seagate.com
- SIG Email sig-arch-test@lists.riscv.org. Notetakers: please send emails to allen.baum@esperantotech.com
- Meetings -Bi-monthly at 8am Pacific time on 2nd/4th Thursdays.
  - See https://docs.google.com/spreadsheets/d/1L15_gHl5b2ApkcHVtpZyl4s_A7sgSrNN zoom link
- Documents, calendar, roster, etc. in
  - https://sites.google.com/a/riscv.org/risc-v-staff/home/tech-groups-cal
  - https://drive.google.com/drive/folders/1DemKMAD3D0Ka1MeESRoVCJipSrwiUlEs

    lifecycle in "policies/supporting docs" folder, gaps in "planning" folder, arch-test specific in "information->content->arch-test")
- Git repositories ←docs riscv → tools

https://github.com/ riscv-non-isa /riscv-arch-test/tree/master/doc	tests	https://github.com/riscv-non-isa/riscv-arch-test
https://github.com/riscv-software-src/riscof/tree/master/docs	riscof	https://github.com/riscv-software-src /riscof
https://github.com/riscv-software-src/riscv-ctg/tree/master/docs	Test Gen.	https://github.com/riscv-software-src /riscv-ctg
https://github.com/riscv-software-src/riscv-isac/tree/master/docs	YAML, WARL config	https://github.com/riscv-software-src /riscv-config/
https://github.com/riscv/sail-riscv/tree/master/doc	Sail formal model	https://github.com/riscv/sail-riscv/
https://github.com/riscv-admin/architecture-test	minutes, charter	

- JIRA: https://jira.riscv.org/projects/CSC/issues/CSC-1?filter=allopenissues
- Sail annotated ISA spec: in https://github.com/rems-project/riscv-isa-manual/blob/sail/

README.SAIL ←how to annotate	annotated unpriv spec→ release/riscv-spec-sail-draft.pdf
release/riscv-spec-sail-draft.pdf ← annotated source	annotated priv spec→ release/riscv-privileged-sail-draft.pdf
https://us02web.zoom.us/rec/share/-XIYazzhIBbQoiZdarCfebdjxjDWiVhf-LxnuVrliN4Bc30yf17ztKkKDU4Og54b.fArPPqnuR-NiXpQU Tutorial Passcode: tHAR#5$V	

Meeting Agenda

0. **Looking for more admins, maintainers for riscv-arch-test git repo !!**

I. Updates, Status, Progress:

I. Updated trap handler passed first simple test, handles nested traps/interrupts, multiple priv modes, -- but won't pass complex tests with MMU enabled

II. Sail PRs merged: configurable test signature, Zfh, Zmmul

III. (former) ACT Vice-Chair is now moved to Risc-V foundation to work on Sail – we need a new Vice-Chair. Please Apply!

II. Next steps and Ongoing maintenance

1. Asynch Event Generator TG – next steps (charter)
2. Discussion: should - should machine generated SAIL to be allowed for CSR read/write legalization?
3. Updates to Current Spec – split into Test Guidelines and Vendor Interface spec
   Vendor Interface is primarily 3 required RVMODEL_ macros (DATA_BEGIN/END, and HALT) and various interrupt/debugmsg/boot code macros
   - Will be updated to add asynch testing (dummy interrupt and event generation device), and external debug ger)
   - this will require specific model interfaces, e.g. wired interrupts, debug messages, timer support)
4. Discussion: other steps for Migration to Framework v.3.0 (riscof). (blocking items):
   a) Reference signature docker image, local podman/docker plugins, remote podman YAML2refsig-AAS implementation
   b) (Sail/Spike model updates, pipecleaning, N people have run it, testing all the "fixed in riscof" issues
   c) Review Pipecleaner tests: What do we need to do to exercise capabilities for Priv Mode tests
5. Dynamic Test Generation
   1. Related: how should we deal with 1GB test directories (FP
6. Discussion: starting a TG to precisely define the ABI for asynch event generation and a reference C-model to be used for Sail
7. Config YAML GUI interface demo

III. Future Agenda items

1. Maintenance updates to V2 to enable future tests
   a) Convert assertions to be out-of-line
   b) add assertion macros for FP, DP, Vreg to arch_test.h, and test_format.spec

Discussion

Status: see previous slide

Issue: Make sure that file format changes are advertised, re:
Configurable Signature: https://github.com/riscv/sail-riscv/issues/111

Background: this was a request from an implementation that had trouble with dumping non-XLEN hex values. This support already existed in OCAML Sail model and in Spike.
This only affects models that explicitly opt-in to this.

Agenda items: setting up a TG for interrupt testing
- Step 1 is to define a charter. Sample charter goals:
  gather SW test framework requirements form AIA, fastInt, Aclint, Clint, groups
  gather test interfaces for different test benches
  spec an async generator device that meets the above requirements

Imperas: this sounds very similar to the openHW verification task group goals: Requirement:
- able to interface to csim, Verilator, System Verilog, Instruction Set simulators & test generator to a virtual peripheral interface (RVVI) (see github virt periph interfaces
- there are multiple versions, but a single spec; select version as a plug-in
- advanced ones are multi-hart
- see OpenHW verification plan for interrupts for cv32e40x core
https://github.com/openhwgroup/core-v-verif/blob/master/cv32e40x/docs/VerifPlans/Simulation/interrupts/CV32E40X_interrupts.xlsx
This is a DV plan, not a compatibility plan, so more comprehensive than required)

Must start with coverage goals to know what the needs will be: what coverage do we want to : e.g
- can an interrupt interrupt every(classes of) instruction?
-- (e.g. arith, branch, load/store, multi-ld/st (vectors, unaligned, push/pop), WFI, exceptions!
-- In different modes, with different priority
-- ensure higher privileged interrupts have higher priority?
-- ensure every interrupt is taken, and serviced?

Also need to know non-goals: speed
-- We may need assertions, but don't care about timing.
-- Horizontal interrupts (which don't change priv mode) are not interesting,
-- Vertical interrupts (which do change priv mode) are interesting,

Does fast interrupt spec testing need any additional features?
(nothing was raised, except how a test can ensure that timings are consistent between Sail and DUT; you may not be able to have consistent timing between interrupt wire assertion and when interrupt is seen by the core)

How do we implement a plugin – need a hook so real silicon
This would similar to how framework handles this now; define a model specific interface macro with parameters write address (model defined), write data (interrupt mask) and control (timing, event type). NumInstRet is an implicit operand.

Decisions & Action Items

Decisions ()

Outstanding Action Items

- find a different place to put coverage reports, e.g. google drive folder < Jenkins file preferred- see next issue>
- Look for and setup ref-signature-as a service site using docker image of Sail and tests < Chair > PLCT has offered resources
- Update all READMEs to point to branch <Incore?>
- Update standard trap handler for added priv levels, custom exception handler registration, < Chair, needs update for VirtMem>
- Contact SW HC & DOC SIG to determine an inline comment->doc tool flow, and determine if docs (as opposed to ISA specs) must be .adoc, or could be .pdf or .hmtl < Chair, Jeff-in progress>
- Develop plugins for podman as well as remote container < HC? >
- Fix tools (ctg, riscv-config, isac, riscof,) to require PRs for changes < done?>
- Set up a TG to define Async Event Generator specs (test interface, Model interface, generator SW that can interface to RTL and simulators, sample shims for Spike and Sail <chair>
- Fix FP D->F convert tests <IIT>?

Draft Internal Test Guidelines

Required Pre-Defined Macros – Macros that every test must include

RVTEST_CODE_BEGIN This saves state and conditionally initializes the trap handler and initializes gprs

RVTEST_CODE_END This conditionally saves the post-test GPR values, transitions to Mmode, conditionally restores pre-test state and causes branches to test halt, then conditionally installs the trap handler

RVTEST_DATA_BEGIN This initializes the a pointer to that trap signature area of the test signature, and reserves space for a pointer to the save area used to save and restore state modified by the trap handler **.**

RVTEST_DATA_END Contains the current trap signature pointer (if traps are enabled) ***FIXME-1/mode
This macro marks the end of the test input data section with label *rvtest_data_end*

RVTEST_CASE(CaseName, CondStr)
execute this case only if condition in cond_str are met
CaseName is arbitrary string
CondStr is evaluated to determine if the test-case is enabled and sets name variable
CondStr can also define compile time macros required for the test-case to be enabled.
The test-case must be delimited with an **#ifdef CaseName/#endif** pair
The format of CondStr can be found in https://riscof.readthedocs.io/en/latest/cond_spec.html#cond-spec

RVTEST_GOTO_MMODE This is used whenever a test halts to put it in Mode so it can restore all state

Helper Macros These are instantiated by the required macros

RVTEST_INIT_GPRS
RVTEST_SAVE_GPRs saves GPRs at end of test if variable *gpr_save* is defined
RVTEST_TRAP_PROLOG sets up trap environment, depends on ***rvtest_strap_routine*** & ***rvtest_vtrap_routine***
RVTEST_TRAP_HANDLER saves trap status in sig & rtn; depends on ***rvtest_strap_routine*** & ***rvtest_vtrap_routine***
RVTEST_TRAP_EPILOG restores trap environment; depends on ***rvtest_strap_routine*** & ***rvtest_vtrap_routine***
RVTEST_TRAP_SAVEAREA set up save area for trap; depends on ***rvtest_strap_routine*** & ***rvtest_vtrap_routine***
RVTEST_SAVE_GPRS(tmpreg, saveaddr) saves all regs except tmpreg to saveaddr at test end if **gpr_save** defined

Required Pre-Defined Variables architecturally defined

RVTEST_DATA_REL_TVAL_MSK (bit-reversed mask of which exceptions store data addrs in xtval.
Defaults to left aligned 0x0F05 (causes 4..7, 13, 15) **update for H-ext
RVTEST_DATA_REL_TVAL_MSK (bit-reversed mask of which exceptions store code addrs in xtval.
Defaults to left aligned 0xD008 (causes 0,1,3,12) **update for H-ext
NUM_SPECD_INTCAUSES (defaults to 16) ***fix for H-ext
NUM_SPECD_EXCPTCAUSES (defaults to 16**)** ***fix for H-ext

Required predefined labels:

rvtest_entry_point The test must define this label to indicate the location to be used by the linker as the entry point in the test. Generally, this would be before the RVMODEL_BOOT macro and should belong to the text.init section.
mtrap_sigptr The test must define this between ***rvmodel_sig_begin*** and ***rvmodel_sig_end*** to mark where normal signatures end and trap signatures begin.
***FIXME: need copies per mode, e.g. ***strap_sigptr*, vtrap_sigptr**
r**vtrap_sigptr.** The test must define this to delimit where the trap signature start
gpr_save The test must define ***gpr_save*** after ***rvmodel_sig_end*** to mark where registers get saved if ***rvtest_gpr_save*** is defined

rvtest_trapsig defines where in the signature area trap signature is stored
rvtest_init
rvtest_code_begin used to relocate code-relative trap status
rvtest_data_[begin/end] used to relocate data-relative trap status
rvtest_[m,s,v]trap_routine used to conditionally instantiate helper macros, depending modes a test will trap into

Optional, Pre-defined Macros

These are helper macros that make test generation easier. The include a set that gives a standard way of storing signatures from the various registers, keeping track of the signature offset, offset overflow, and offset alignment
.
RVTEST_SIGBASE(BaseReg,Val) defines the base register used to update signature values
Register BaseReg is loaded with value Val, hidden_offset is initialized to zero
RVTEST_BASEUPD(BaseReg[oldBase[,newOff]]) [moves &] updates BaseReg past stored signature.
Hidden_offset is re-initialized to 0 afterwards
RVTEST_VALBASEUPD(BaseReg [, Offset])

RVTEST_SIGUPD(BaseReg, SigReg [, Offset])
RVTEST_SIGUPD_F(BaseReg, SigReg, FlagReg [, Offset])
RVTEST_SIGUPD_FID(BaseReg, SigReg, FlagReg [, Offset])
Updates the base reg by hidden or explicit offset. Flagreg is the gpr where fstatus CSR is loaded

Draft: External Arch-Test Spec

Required, Model-defined Macros

These macros are be defined by the owner of the test target in the file **model_test.h**. These macros are required to define the signature regions and also the logic required to halt/exit the test.

RVMODEL_DATA_BEGIN

This macro marks the start of signature regions. The test-target should use this macro to create a label to indicate the beginning of the signature region: .global rvmodel_sig_begin; rvmodel_sig_begin:. This macro must also begin at a 16-byte boundary and must not include anything else.

RVMODEL_DATA_END

This macro marks the end of the signature-region. The test-target must create rvmodel_sig_end global labela indicating the end of the signature region: .globl rvmodel_sig_end; rvmodel_sig_end:. The entire signature region must be included within the RVMODEL_DATA_BEGIN macro and the start of the RVMODEL_DATA_END macro. The RVMODEL_DATA_END macro can also contain other target specific data regions and initializations but only after the end of the signature.

RVMODEL_HALT

This macro must define the test-target halt mechanism. This macro is called when the test is to be terminated either due to completion or due to unsupported behavior. This macro could also include routines to dump the signature region to a file on the host system which can be used for comparison.

Optional labels

Optional Model Defined variables (**FIXME – need to complete)

RVMODEL_ADDR_SZ_	(default to the largest possible size if not defined)
RVMODEL_PHYS_ADDR_SZ	(default to 57 for RV64, 34 for RV32S, 32 for RV32)
RVMODEL_CACHE_BLK_SZ	(default to 64)
NUM_SPECED_INTCAUSES	(default to 16) **FIXME for HEXT
NUM_SPECED_EXCPTCAUSES	(default to 16)) **FIXME for HEXT
FENCEI	(default to fence.i)
RVTEST_DATA_REL_TVAL_MSK	(bit-reversed mask of which exceptions store data addrs in xtval. Defaults to left aligned 0x0F05 (causes 4..7, 13, 15) **update for H-ext
RVTEST_DATA_REL_TVAL_MSK	(bit-reversed mask of which exceptions store code addrs in xtval. Defaults to left aligned 0xD008 (causes 0,1,3,12) **update for H-ext

Model requirements:

Each xTVEC is either arbitrarily writable or initialized to a memory address that has RWX permissions and at least 580 bytes in size (specifically: (XLEN + 3* NUM_SPECD_INTCAUSES + 17) * 4)
The hart exports a 4bit output signal which is the # of retired instructions during each cycle
The hart imports XLEN input interrupt signals
The hart can be configured to have as much memory as a test requires
The risv-config YAML for the core has all model defined variables and optional features implemented (**FIXME:list)
e.g. unaligned access, unaligned_partial_store, Zextensions implemented, opt_except_priorities, ….

Optional, Model-defined Macros

These are macros whose implementation must be defined by the DUT because they are platform specific. They include boot code, debug messaging routines, assertion checking, and eventually interfaces to asynch events like interrupts, concurrent memory accesses, and external debug.

RVMODEL_BOOT contains boot code for the test-target; may include emulation code or trap stub.
If the test-target enforces alignment or value restrictions on the mtvec csr, it is required that this macro sets the value of mtvec to a region which is readable and writable by the M- mode.
****FIXME: must also specify stvec and vstevec and sizes (580 bytes*)
The boot code may include code to copy the data sections from boot device to ram, or any other code that needs to be run prior to running the tests.

RVMODEL_IO_INIT This initializes IO for debug output
This must be invoked if any of the other ***RV_MODEL_IO_**** macros are used

RVMODEL_IO_ASSERT_GPR_EQ(ScrReg, Reg, Value) This outputs a debug message if Reg!=Value
ScrReg is a scratch register used by the output routine; its final value cannot be guaranteed
Can be used to help debug what tests have passed/failed
Note: this macro is currently implemented as an inlined routine. It will eventually be replaced with an out-of-line routine with parameter values in specific registers that is called by an RVTEST_ASSERT macro that calls trampoline table code to handle register save and inline parameter extraction before calling the RVMODEL code .

RVMODEL_IO_WRITE_STR(ScrReg, String) Output debug string, using a scratch register
ScrReg is a scratch register used by the output routine; its final value cannot be guaranteed

RVMODEL_SET_[M/S/V]SW_INT Routines to set the SW interrupt for each mode.
Currently the test forces an empty macro if undefined . Future tests may change this.

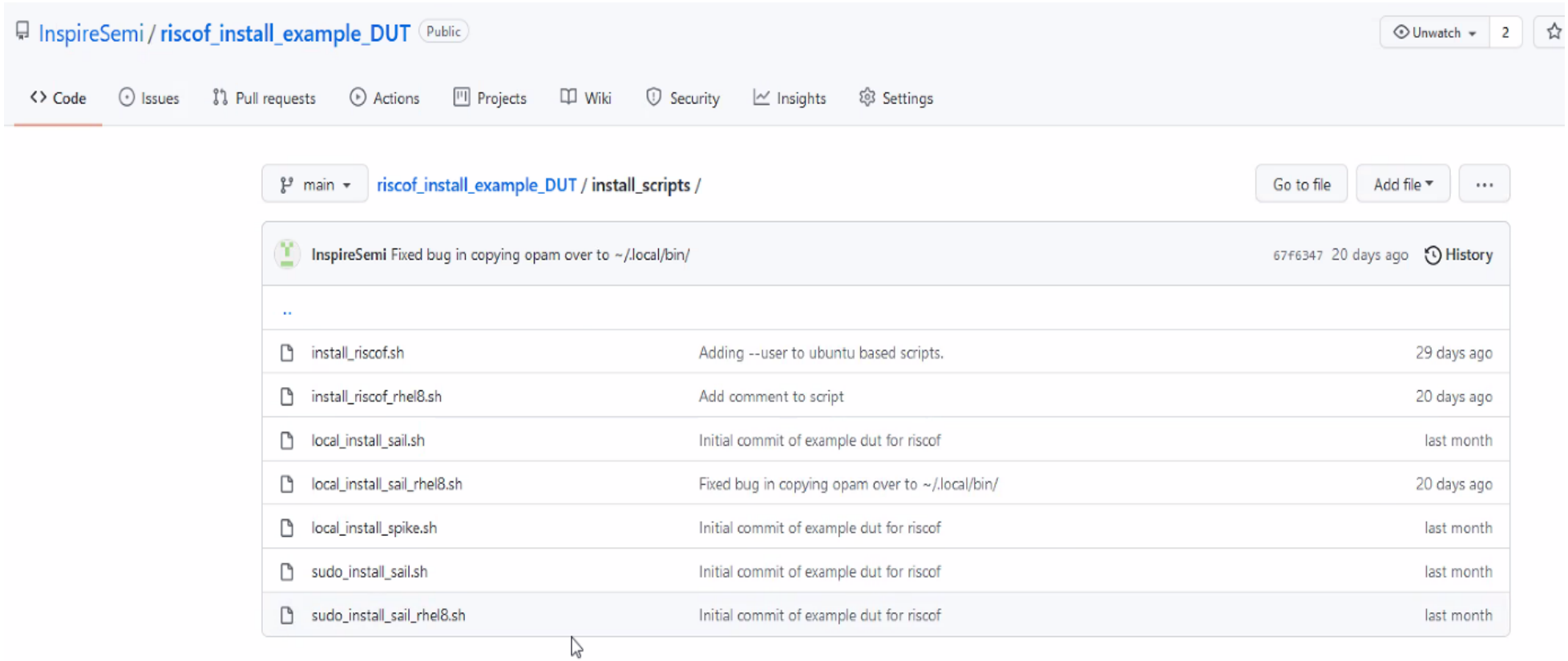
RVMODEL_CLEAR_[M/S/V][SW/TIMER/EXT]_INT. Routines to clear (SW/TMR/EXT) interrupts for each mode.
Currently the test forces an empty macro if undefined . Future tests may change this.

RVMODEL FENCEI Used in the trap handler and setup code to enforce synchronization when code is overwritten
Needed if fencei is not implemented; defaults to fencei

These will be augmented with more general interrupt test macros e.g.:
RVMODEL_ASYNCH_EVENT_ADDR(BaseReg, AddrReg)
RVMODEL_ASYNCH_EVENT_DATA(BaseReg, DataReg)
RVMODEL_ASYNCH_EVENT_CMD(BaseReg, CmdReg, Delta, Cmd, [ResultReg])

BACKUP

Example riscof repo



Pull/Issue Status

Issue#	Date	submitter	title	status	comments
#4	03-Jul-2018	Kasanovic	Section 2.3 Target Environment	Fixed in riscof	Will be closed in V3
#22	24-Nov-18	brouhaha	I-MISALIGN_LDST-01 assumes misaligned data access will trap	^	HW misalign support not configurable
#40	4-Feb-19	debs-sifive	Usage of tohost/fromhost should be removed	|	now
#146-9	01-Dec-20	Imperas	Test I EBREAK,ECALL, MISALIGN_JMP/LDST, OpenHW	|	HW misalign support not configurable
#189	26-Apr-21	neelgala	Proposal to enhance the RVTEST_ISA macro	v	
#115	06-jun-20	adchd	How to support on-board execution?	under discussion	
pull#129	31-jul-20	nmeum	sail-riscv-ocaml: Disable RVC extension on all devices not using it	In process	Who can review this?
pull#184	15-apr-21	dansmathers	Updating http reference for constr	In process	Approved, needs merge
pull#225	08-dec-21	Phthinh	update the K extension for the V.1.0.0 ratified spec	Needs review	Looks good to go
#119	17-jun-20	allenjbaum	Missing RV32i/RV64i test: Fence	Test has been written	Close when RFQ test is merged
#190	26-Apr-21	neelgala	The 16-byte signature boundary issue		
#203	24-Aug-21	Allenjbaum	Fence test has poor coverage		Specifically: test fm bits are ignored
#211	19-sep-21	Neelgala	default rvtest_data should be 16-bytes		
#214	05-oct-21	Allenjbaum	Test Format Spec doesn't specify the order of line in the signature file		Spec clarification
#220	20-oct-21	Davidharrismc	F tests		Add new F tests to makefile so it works OOB
pull#226	17-dec-21	liweiwei90	add support for cbo.zero in cmo extension	Needs changes	

JIRA Status

Issue#	Date	submitter	title	status	comments
CSC-1	20/Aug/20	Ken Dockser	Come up with names for the tests suites that we are creating		1st step done
CSC-2	20/Aug/20	Ken Dockser	Produce concise text to explain the Architecture Tests intent and Limits	done	Will become ACT policy
CSC-3	20/Aug/20	Ken Dockser	Come up with an internal goal for what we wish to accomplish with the Architectural Tests		This is the /test coverpoint YAML
CSC-4	20/Aug/20	Ken Dockser	Develop a roadmap for all the different categories of test suites that will need to be created		Not written
CSC-5	20/Aug/20	Ken Dockser	Develop a roadmap for releases of single-instruction Architecture Tests		Not written
CSC-6	20/Aug/20	Ken Dockser	Develop a reference RTL test fixture that can stimulate and check the CPU under test		Needs more discussion

Non-determinism in Architectural Tests

The RV architecture defines optional and model/µarch defined behavior. This implication: there are tests that have multiple correct answers. E.g.:

- Misaligned accesses: can be handled in HW, by "invisible" traps w/ either misaligned or illegal access causes, and do it differently for the same op accessing the same address at different times (e.g. if the 2nd half was in the TLB or not)
- Unordered Vector Reduce ops: (different results depending on ordering & cancellation)
- Tests involving concurrency will have different results depending on microarchitectural state, speculation, or timing between concurrent threads (e.g. modifying page table entry without fencing)

From the point of view of ACTs, there are 2 (& sometimes more) legal answers. The golden model only generates one. Possible mechanisms to test include:

- Modify (if necessary) & configure reference model to generate each legal result, run it with each config, & accept either result from the DUT (e.g. misalign or un-fenced PTE modification)
- Provide specific handlers for optional traps? (can't test the trap is correct then)
- Use self-testing tests(compare with list or range of allowed outcomes from litmus tests)
- Avoid tests that can generate non-deterministic results
- Ultimately: develop new frameworks that can handle concurrency along with reference models that can generate all legal outcomes
- It is the responsibility of the TG that develops an extension to develop the strategy for testing features and extensions that can have nondeterministic results

Framework Requirements

The framework must:

- Use the TestFormat spec and macros described therein
  - (which must work - including assertions)
- Choose test cases according to equations that reference the YAML configuration
- Define macro variables to be used inside tests based on the YAML configuration
- Include the compliance trap handler(s), & handle its (separate) signature area(s)
- Load, initialize, and run selected tests between two selected models, extract the signatures, compare results, and write out a report file
- Exist in a riscv github repo, with a more than one maintainer.
- Be easy to get running, e.g.:
  - run under a variety of OSes with the minimum number of distro specific tools.
  - Not require sudo privileges
- Have the ability to measure and report coverage for test generation
  - Coverage specification is a separate file
  - Could be a separate app

Test Acceptance Criteria

Tests merged into the ACT test_suite repo must :

- conform to the current format spec (macros, labels, directory structure)
  - including framework-readable configurations - i.e. which ISA extension it will be tested with (using Test_Case macro parameter equations) for each test case
- use only files that are part of the defined support files in the repository, including standard trap handlers
  - TBD: how to install test specific (not model specific) handlers
- Be able to be loaded, initialized, run, signal completion, and have signature results extracted from memory by a/the framework
- run using the SAIL model and not fail any tests
- generate signature values either
  - directly from an instruction result (that can be saved & compared with DUT/sim)
  - by comparing an instruction result with a configuration-independent value range embedded in the test code (e.g. saving above, below, within)
  - by comparing an instruction result with a configuration-independent list of values (e.g saving matches or mismatched)
    - (it can be useful to also return a histogram of value indices that matched)
- Store each signature value into a unique memory location in a signature region that is
  - delimited by standard macros embedded in the test which can be communicated to the test framework
  - pre-initialized to values that are guaranteed not to be produced by a test
- have defined coverage goals in a machine readable form that can be mechanically verified
- improve coverage (compared to existing tests) as measured and reported by a coverage tool (e.g. ISAC)
- use only standard instructions (and fixed size per architecture macros, e.g. LI, LA are allowed)
- be commented in test_case header (ideally listing coverpoint covered)

Tests that are otherwise accepted, but depend on tools or simulators that have not be upstreamed must be put into a <Ext-Name_unratified>/ directory instead of <Ext-Name>/