RISC-V Assembly Test Infrastructure

Stephen Twigg
UC Berkeley

sdtwigg@eecs.berkeley.edu
Basic overview

- riscv-tests/isa/make
  - rv64ui-v-add

- Test Virtual Machines
  - Defines what capabilities test program needs from the processor, memory.
  - Separate folder in riscv-tests/isa for each

- 4 Target Environments
  - Defines what the surrounding test system looks like (multicore, virtual memory, interrupts)
  - Separate folder in riscv-tests-env for each
Test Virtual Machines

- Set of allowed instructions and registers
  - E.g. RVTEST_RV64UF

- Execution start and end
  - RVTEST_CODE_BEGIN, RVTEST_CODE_END

- Test data input
  - Invoked with the .data assembler directive

- Test output data
  - RVTEST_DATA_BEGIN, RV_TEST_DATA_END
  - Dump signature via +signature=[filename] (FESVR feature)
Set of Allowed Instructions

- Top-of-file Macro:
  - RVTEST_RV[32/64]U(F/V)
  - RVTEST_RV[32/64]S

- Adjusts execution scripts to turn on associated parts of processor
  - 32 -> decode as 32 bit instructions
  - F -> Floating point unit
  - V -> FPU and Accelerator (Hwacha assumed)

- Static checkers could check to ensure enclosed instructions subset of stipulated set

- e.g. RVTEST_RV32UV
Test Pass/Failure

- RVTEST_PASS, RVTEST_FAIL macros
  - Signal program is completed to fesvr and write error code to the tohost register

- RVTEST_CODE_END is RVTEST_PASS

- Unlikely Aside: Understanding tohost register
  - LSB indicates whether syscall (0) or termination (1)
  - All other bits are metadata
    - Notice how RVTEST_FAIL shifts error code left
#include "riscv_test.h"  # Hook into Target Environment

RVTEST_RV64U  # Define TVM used by program.

# Test code region.  # Start of test code.
RVTEST_CODE_BEGIN
    lw     x2, testdata
    addi   x2, 1    # Should be 42 into $2.
    sw     x2, result
    # Store result into memory overwriting 1s.
    li      x3, 42  # Desired result.
    bne    x2, x3, fail
    # Fail out if doesn't match.
fail:
    RVTEST_PASS  # Signal success.

RVTEST_FAIL
RVTEST_CODE_END  # End of test code.
Sample Simple Assembly Test (Data 2/2)

# Input data section.
# This section is optional, and this data is NOT saved in the output.
.data
    .align 3
testdata:
    .dword 41

# Output data section.
RVTEST_DATA_BEGIN    # Start of test output data region.
    .align 3
result:
    .dword -1
RVTEST_DATA_END      # End of test output data region.
Four Target Environments

- **Physical (p)**
  - Physical memory
  - Only core 0 allowed to boot

- **Physical-multicore (pm)**
  - Physical memory
  - All cores allowed to boot
  - Special environment

- **Physical-interrupt (pt)**
  - Physical Memory
  - Only core 0 allowed to boot
  - Every 100 cycles a timer interrupt fires

- **Virtual (v)**
  - Virtual memory (small kernel for handling included)
  - Only core 0 allowed to boot
Advised mechanism for adding own tests

- Fork (or clone) riscv-tests
- Determine which test virtual machine you want
- Go into directory for associated TVM in riscv-tests/isa
- Add assembly file for your test
- Modify Makefrag for your test
  - Add your test name to either *_sc_tests or *_mc_tests
# Makefrag for rv64ui tests

rv64ui_sc_tests = \n  add \n  xor \n
rv64ui_mc_tests = \n  lrsc

rv64ui_p_tests = $(addprefix rv64ui-p-, $(rv64ui_sc_tests))
rv64ui_pm_tests = $(addprefix rv64ui-pm-, $(rv64ui_mc_tests))
rv64ui_pt_tests = $(addprefix rv64ui-pt-, $(rv64ui_sc_tests))
rv64ui_v_tests = $(addprefix rv64ui-v-, $(rv64ui_sc_tests))

spike_tests += $(rv64ui_p_tests) $(rv64ui_pm_tests) $(rv64ui_pt_tests) $(rv64ui_v_tests)
- Target environment linker scripts
  - Execution begins at 0x2000

- Before core boot FESVR writes to 0x0
  - 0x0 = Size of memory in MB
  - 0x4 = Number of cores in the system

- `riscv-tests/isa/macros` contains many recipes for individual assembly tests
Sample Full Test (isa/rv32ui/add.S)

# See LICENSE for license details.
#******************************************************************************
# add.S
#------------------------------------------------------
#
# Test add instruction.
#
#include "riscv_test.h"
#include "test_macros.h"
RVTEST_RV32U
RVTEST_CODE_BEGIN
# Arithmetic tests

```c
TEST_RR_OP( 2, add, 0x00000000, 0x00000000, 0x00000000);
TEST_RR_OP( 3, add, 0x00000002, 0x00000001, 0x00000001);
TEST_RR_OP( 4, add, 0x0000000a, 0x00000003, 0x00000007);
TEST_RR_OP( 5, add, 0xffff8000, 0x00000000, 0xffff8000);
TEST_RR_OP( 6, add, 0x80000000, 0x80000000, 0x00000000);
TEST_RR_OP( 7, add, 0x7fff8000, 0x80000000, 0xffff8000);
TEST_RR_OP( 8, add, 0x00007fff, 0x00000000, 0x00007fff);
TEST_RR_OP( 9, add, 0x7fffffff, 0x7fffffff, 0x00000000);
TEST_RR_OP(10, add, 0x80007ffe, 0x7fffffff, 0x00007fff);
TEST_RR_OP(11, add, 0x80007ffe, 0x80000000, 0x00007fff);
TEST_RR_OP(12, add, 0x7fff7ffe, 0x80000000, 0xffff8000);
TEST_RR_OP(13, add, 0xffffffff, 0x00000000, 0xffffffff);
TEST_RR_OP(14, add, 0x00000000, 0xffffffff, 0x00000001);
TEST_RR_OP(15, add, 0xfffffffff, 0xfffffffff, 0xfffffffff);
TEST_RR_OP(16, add, 0x80000000, 0x00000001, 0x7fffffff);
```
# Source/Destination tests

TEST_RR_SRC1_EQ_DEST( 17, add, 24, 13, 11 );
TEST_RR_SRC2_EQ_DEST( 18, add, 25, 14, 11 );
TEST_RR_SRC12_EQ_DEST( 19, add, 26, 13 );

# Bypassing tests

TEST_RR_DEST_BYPASS( 20, 0, add, 24, 13, 11 );
TEST_RR_DEST_BYPASS( 21, 1, add, 25, 14, 11 );
TEST_RR_DEST_BYPASS( 22, 2, add, 26, 15, 11 );
TEST_RR_SRC12_BYPASS( 23, 0, 0, add, 24, 13, 11 );
TEST_RR_SRC12_BYPASS( 24, 0, 1, add, 25, 14, 11 );
TEST_RR_SRC12_BYPASS( 25, 0, 2, add, 26, 15, 11 );
TEST_RR_SRC12_BYPASS( 26, 1, 0, add, 24, 13, 11 );
TEST_RR_SRC12_BYPASS( 27, 1, 1, add, 25, 14, 11 );
TEST_RR_SRC12_BYPASS( 28, 2, 0, add, 26, 15, 11 );
TEST_RR_SRC21_BYPASS( 29, 0, 0, add, 24, 13, 11 );
TEST_RR_SRC21_BYPASS( 30, 0, 1, add, 25, 14, 11 );
TEST_RR_SRC21_BYPASS( 31, 0, 2, add, 26, 15, 11 );
TEST_RR_SRC21_BYPASS( 32, 1, 0, add, 24, 13, 11 );
TEST_RR_SRC21_BYPASS( 33, 1, 1, add, 25, 14, 11 );
TEST_RR_SRC21_BYPASS( 34, 2, 0, add, 26, 15, 11 );
Final tests and end

TEST_RR_ZEROSRC1(35, add, 15, 15 );
TEST_RR_ZEROSRC2(36, add, 32, 32 );
TEST_RR_ZEROSRC12(37, add, 0 );
TEST_RR_ZERODEST(38, add, 16, 30 );
TEST_PASSFAIL

RVTEST_CODE_END

.data

RVTEST_DATA_BEGIN
TEST_DATA
RVTEST_DATA_END
riscv-torture

- Currently private repository, elements subject to change before public release

- Generates random assembly tests
  - Interleaving small code-snippets into one large program

- Runs spike as golden model compares against other supplied chip emulators to compare
  - Copy state (registers, etc.) to test signature memory portion and compare these signatures