asm_test introduction document

V1.0

FRASER INNOVATION INC

Version Control

version	Date	Description
1.0	09/29/2020	Initial Release

Copyright Notice:

Thank you for purchasing the FPGA development board. Please read the manual carefully before using the product and make sure that you know how to use the product correctly. Improper operation may damage the development board. This manual is constantly updated, and it is recommended that you download the latest version when using.

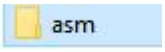
Official Shopping Website:

https://fpgamarketing.com

Contents

1. Introduction

asm_test hosts unit tests for different features of FII_RISC-V V3.01 processor. This CPU is fully pipelined and include irq feature. FII_RISC-V V3.01 include mul and div instructions. Its ROM write and read depth is set to be 16384 bits.

asm_test project contains test cases assembly files ,main file and header files for macros. It is build through FreedomStudio platform.asm_test will test all of the FII_RISC-V V3.01 processor supported instructions . The corresponding assembly files were stored in folder asm: 

and listed below in Figure1.

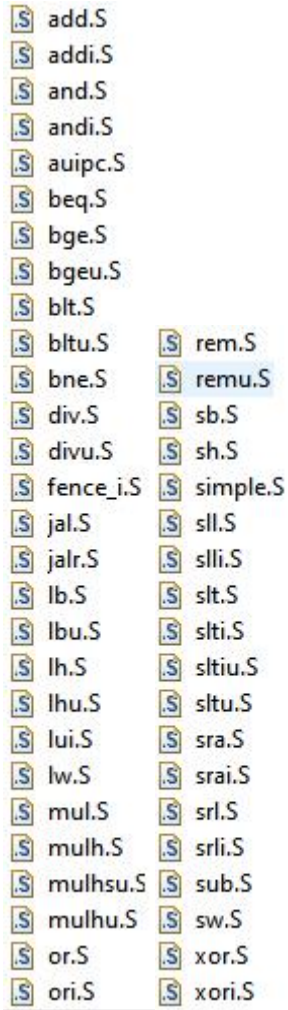


Figure1 instruction test cases list

2. Basic Features

2.1)asm_test

Each test case program should first include the riscv_test.h and test_macros.h header file, which defines the macros used by instruction assembly test file.

A test program for a RISC-V instruction is written within a single assembly language file, which is passed through the C preprocessor, and all regular assembly directives can be used.

The test program will begin execution at the first instruction after defining the function name, and continue until execution reaches TEST_PASSFAIL macro or jalr t6, 0(t6) and then jump out from the current function. The testing result will be stored into register x26 and x27. if the result is implicitly a success, then 0x1 will be stored into x26 and x27. A test can explicitly fail by invoking the RVTEST_FAIL macro, and x27 will be set to 0x0. The related code is shown below in Figure2.

```
#define RVTEST_PASS
        li x26, 0x01;
        li x27, 0x01;


#define TESTNUM gp

#define RVTEST_FAIL
        li x26, 0x01;
        li x27, 0x00;
```

Figure2 code for RVTEST_PASS and RVTEST_FAIL

All user registers (pc, x0-x31, fsr) can be accessed inside test program, so the test case functions will store sp at the beginning of funtion and restore sp at the end of the function. The value of register ra is return address, so it will also be stored for returning to main program. The corresponding code is shown below in Figure3.

```
24      addi x25, sp, 0
25      addi t6, ra, 0

85     addi sp, x25, 0
86     jalr t6, 0(t6)
```

Figure3 store and restore code for register sp and ra

asm_test will print out the testing result of all instructions through serial port. The test result shows that all instructions except fence has passed the test, This is because FII_RISC-V V3.01 does not support fence instruction. Figure4 shows the error report for fence instruction.

```
test cast _test_fence_i
Assertion : [((temp & 0x03) == 0x03)]  File: ../main.c, Line 77 [Date: Sep 29 2020 Time: 09:34:03] error_n
um = 2
OK, temp = 0x00000001
```

Figure4 error report for fence instruction

2.2)jal_test

jal_test is a project which only runs an example test function _test_jal. The assembly code is shown in Figure 5.

```
#*****************************************************************************
# jal.S
#-----------------------------------------------------------------------------
#
# Test jal instruction.
#

#include "riscv_test.h"
#include "test_macros.h"

RVTEST_RV64U
.section .text.init
.align 6
.globl _test_jal
.type _test_jal,@function
_test_jal:

  #-------------------------------------------------------------
  # Test 2: Basic test
  #-------------------------------------------------------------

    addi x25, sp, 0
    addi t6, ra, 0

test_2:
  li  TESTNUM, 2
  li  ra, 0

  jal x4, target_2
linkaddr_2:
  nop
  nop

  j fail

target_2:
  la  x2, linkaddr_2
  bne x2, x4, fail

  #-------------------------------------------------------------
  # Test delay slot instructions not executed nor bypassed
  #-------------------------------------------------------------

  TEST_CASE( 3, ra, 3, \
    li  ra, 1; \
    jal x0, 1f; \
    addi ra, ra, 1; \
    addi ra, ra, 1; \
    addi ra, ra, 1; \
    addi ra, ra, 1; \
1:  addi ra, ra, 1; \
    addi ra, ra, 1; \
  )

  TEST_PASSFAIL

    addi sp, x25, 0
    jalr t6, 0(t6)

```

Figure5 Test case for jal instruction

The example program contains self-checking code to test the result of jal instruction. A test can explicitly fail by invoking the RVTEST_FAIL macro, and it will continue run inside infinite loop called loop fail. The corresponding code is shown below in Figure6.

```
126 #define RVTEST_PASS                                     \
127         li x26, 0x01;                                   \
128         li x27, 0x01;
129
130 #define TESTNUM gp
131
132 #define RVTEST_FAIL                                     \
133         li x26, 0x01;                                   \
134         li x27, 0x00;                                   \
135 loop_fail:                                              \
136         j loop_fail
137
```

Figure6 code for RVTEST_FAIL

3. References

1. https://github.com/riscv/riscv-tests