# Author : Muhammad Imran
# Date: 11-02-2026
# Module : RISCV Arch Test
# Section: RISCV Arch Test
# Task Name: Task 3

Github Link: [RISCV-ARCH TEST Task-3]()

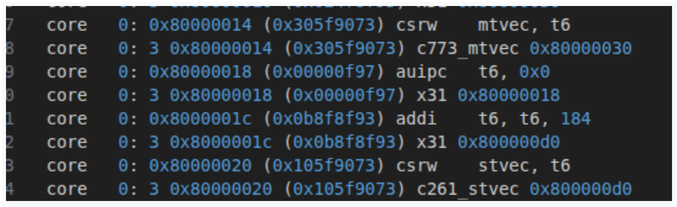
Test Description:

This test verifies the exception delegation to S-mode exception handler in a RISCV system. The program begins execution in M-mode, where it initializes stack, trap vectors (mtvec, stvec) and explicitly sets the medeleg register to delegate the Illegal Instruction exception (cause code 2) to S-mode. After this setup, the processor transitions from M-mode to U-mode using the **switch_mode** function implemented in task1. While running in U-mode, the test executes an invalid instruction **(0xffffffff)** to trigger an Illegal Instruction exception. Because the corresponding bit in medeleg is set, the exception must be routed to the S-mode trap handler instead of the M-mode handler. The S-mode handler verifies correct delegation by reading scause, confirms the cause value, updates sepc to skip the faulting instruction, and returns using sret. Successful completion demonstrates that exception delegation mechanisms operate according to the RISC-V Privileged Specification.

Actual Output:

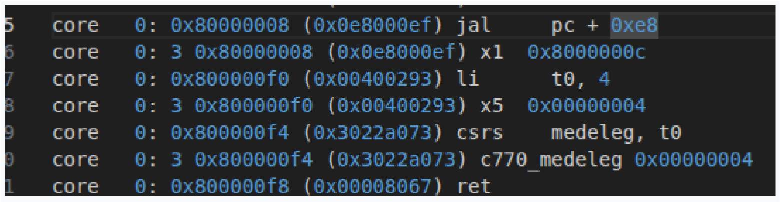
The program starts in **M** mode and sets up the stack, machine mode trap vector address, supervisor mode trap vector address.

mtvec (machine mode trap handler address) stvec (supervisor mode trap handler addres) set up



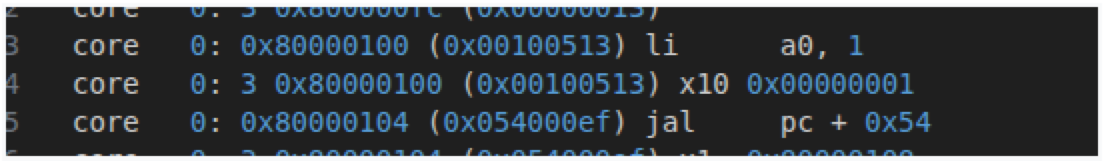
```
core   0: 0x80000014 (0x305f9073) csrw    mtvec, t6
core   0: 3 0x80000014 (0x305f9073) c773_mtvec 0x80000030
core   0: 0x80000018 (0x00000f97) auipc   t6, 0x0
core   0: 3 0x80000018 (0x00000f97) x31 0x80000018
core   0: 0x8000001c (0x0b8f8f93) addi    t6, t6, 184
core   0: 3 0x8000001c (0x0b8f8f93) x31 0x800000d0
core   0: 0x80000020 (0x105f9073) csrw    stvec, t6
core   0: 3 0x80000020 (0x105f9073) c261_stvec 0x800000d0
```

it also set up the medeleg[2] bit to delegate the illegal instruction exception to the supervisor mode.



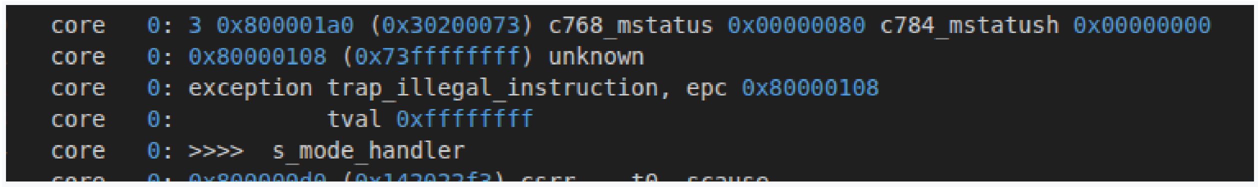
```
core   0: 0x80000008 (0x0e8000ef) jal     pc + 0xe8
core   0: 3 0x80000008 (0x0e8000ef) x1  0x8000000c
core   0: 0x800000f0 (0x00400293) li      t0, 4
core   0: 3 0x800000f0 (0x00400293) x5  0x00000004
core   0: 0x800000f4 (0x3022a073) csrs    medeleg, t0
core   0: 3 0x800000f4 (0x3022a073) c770_medeleg 0x00000004
core   0: 0x800000f8 (0x00008067) ret
```

After initial setup the program then jumps to the **main** label where it switches privilege mode from **M** to **U** using the **switch_mode** function implemented in the **task1.**



```
core   0: 0x80000100 (0x00100513) li      a0, 1
core   0: 3 0x80000100 (0x00100513) x10 0x00000001
core   0: 0x80000104 (0x054000ef) jal     pc + 0x54
```

The **switch_mode** function returns in **U** mode, then the test tries to execute an illegal instruction **.word 0xffffffff** which will raise and illegal instruction exception. The program switches to the **M** mode and reads the **medeleg** csr it finds the corrosponding bit of the **medeleg** register is set instead of jumping to the **mtvec** it will jump to **stvec** and handle the exception in **s_mode_handler**



```
core   0: 3 0x800001a0 (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
core   0: 0x80000108 (0x73ffffffff) unknown
core   0: exception trap_illegal_instruction, epc 0x80000108
core   0:           tval 0xffffffff
core   0: >>>>  s_mode_handler
```

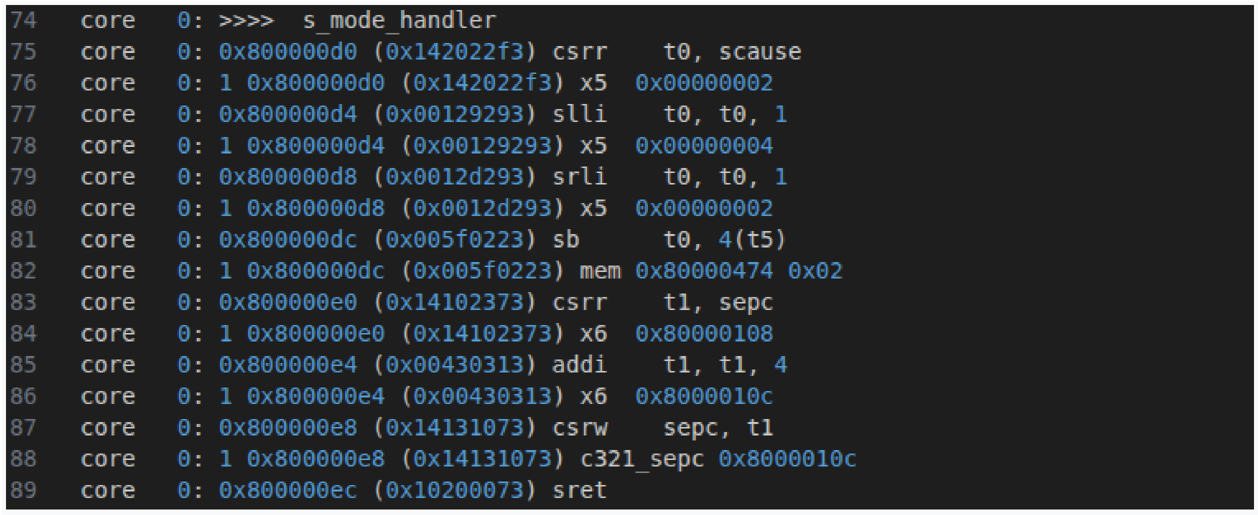
Illegal instruction handling in S Mode handler:

Read scause (expected value = 2). Store scause value into signature region.

Read sepc and increment by 4 to skip illegal instruction.

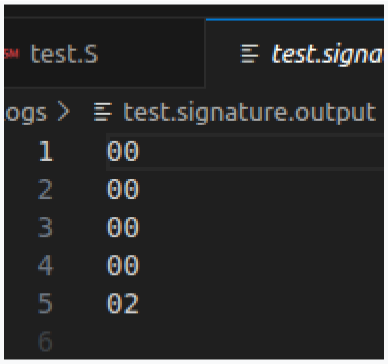
Write updated value back to sepc.

Return using sret.



```
core   0: >>>>  s_mode_handler
core   0: 0x800000d0 (0x142022f3) csrr    t0, scause
core   0: 1 0x800000d0 (0x142022f3) x5  0x00000002
core   0: 0x800000d4 (0x00129293) slli    t0, t0, 1
core   0: 1 0x800000d4 (0x00129293) x5  0x00000004
core   0: 0x800000d8 (0x0012d293) srli    t0, t0, 1
core   0: 1 0x800000d8 (0x0012d293) x5  0x00000002
core   0: 0x800000dc (0x005f0223) sb      t0, 4(t5)
core   0: 1 0x800000dc (0x005f0223) mem 0x80000474 0x02
core   0: 0x800000e0 (0x14102373) csrr    t1, sepc
core   0: 1 0x800000e0 (0x14102373) x6  0x80000108
core   0: 0x800000e4 (0x00430313) addi    t1, t1, 4
core   0: 1 0x800000e4 (0x00430313) x6  0x8000010c
core   0: 0x800000e8 (0x14131073) csrw    sepc, t1
core   0: 1 0x800000e8 (0x14131073) c321_sepc 0x8000010c
core   0: 0x800000ec (0x10200073) sret
```

For this test one extra byte is added in the signature region which records the illegal instruction cause during the trap handling in s-mode. Reset of the bytes are 00 used for error and recording the privilege modes during mode verification. Switch mode verification is not implemented in this task.



```
00
00
00
00
02
```

At the end two **ECALL** instructions are executed to switch back to the **M** mode so that the test can exit in **M** mode. Each ecall raises an exception which is handled by the trap handler implemented in the previous task1.

Reference:

Machine Trap Delegation Registers: Privileged Architecture v1.12, Sec 3.1.8