# RISC-V Arch Test

# Task 3

**Test Description:**

This test verifies correct handling of an illegal instruction exception generated in User mode and delegated to Supervisor mode.

The test starts execution in Machine mode, configures exception delegation using the medeleg CSR, installs a Supervisor-mode trap handler, and transitions execution through Supervisor mode into User mode.

An illegal instruction is deliberately executed in User mode by accessing an Supervisor mode only CSR (sstatus). The resulting exception is delegated to Supervisor mode, where it is identified using scause, handled by advancing the exception program counter (sepc), and execution is returned back to User mode using sret.

Successful handling of the exception results in a test pass indication written to the tohost register.

**Program Flow:**

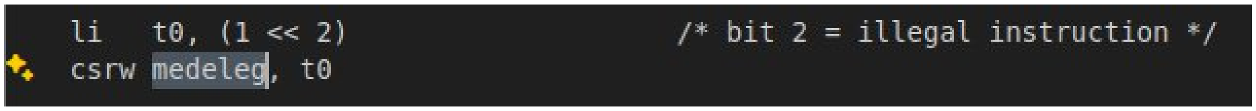
- The program starts in Machine mode (M-mode) at _start and transfers control to main.
- In main, illegal instruction exceptions (cause = 2) are delegated to Supervisor mode using the medeleg CSR, and the Supervisor trap vector (stvec) is set to s_trap_vector.
- The function switch_to_supervisor configures mstatus.MPP for Supervisor mode, sets mepc to s_entry, and executes mret to transition from M-mode to S-mode.
- At s_entry, switch_to_user clears sstatus.SPP, sets sepc to u_entry, and executes sret to transition from S-mode to U-mode.

- In User mode, a privileged instruction (csrr t0, sstatus) is executed intentionally, causing an illegal instruction exception (scause = 2).
- Due to exception delegation, control transfers to Supervisor mode, and execution jumps to s_trap_vector.
- The trap handler reads scause and sepc, detects the illegal instruction, and branches to handle_illegal.
- In handle_illegal, sepc is incremented to skip the faulting instruction, and sret returns execution back to User mode.
- Execution resumes normally after the illegal instruction, and the test signals pass by writing to tohost. Any unexpected exception results in test failure.

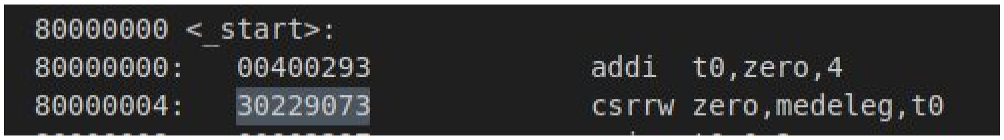
**Exception Delegation:**

By default, all exceptions are handled in Machine mode.

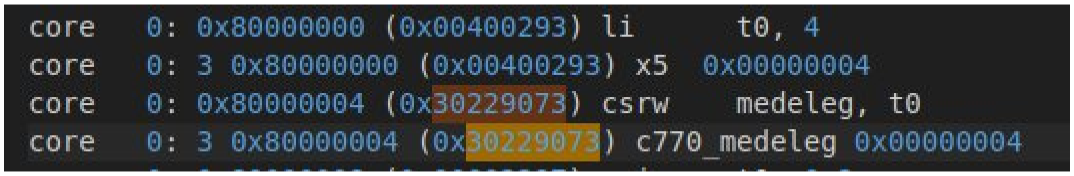
The medeleg CSR is configured to delegate illegal instruction exceptions (scause = 2) to Supervisor mode.



```
    li   t0, (1 << 2)                   /* bit 2 = illegal instruction */
    csrw medeleg, t0
```



```
80000000 <_start>:
80000000:    00400293             addi  t0,zero,4
80000004:    30229073             csrrw zero,medeleg,t0
```

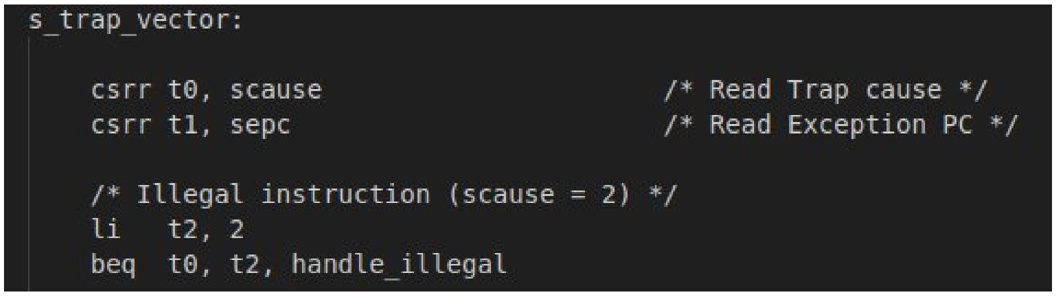


```
core   0: 0x80000000 (0x00400293) li      t0, 4
core   0: 3 0x80000000 (0x00400293) x5  0x00000004
core   0: 0x80000004 (0x30229073) csrw    medeleg, t0
core   0: 3 0x80000004 (0x30229073) c770_medeleg 0x00000004
```

**Supervisor Trap Vector:**

The Supervisor-mode trap handler reads scause to determine the exception type.

If the exception corresponds to an illegal instruction (scause = 2), control is transferred to the illegal instruction handler.

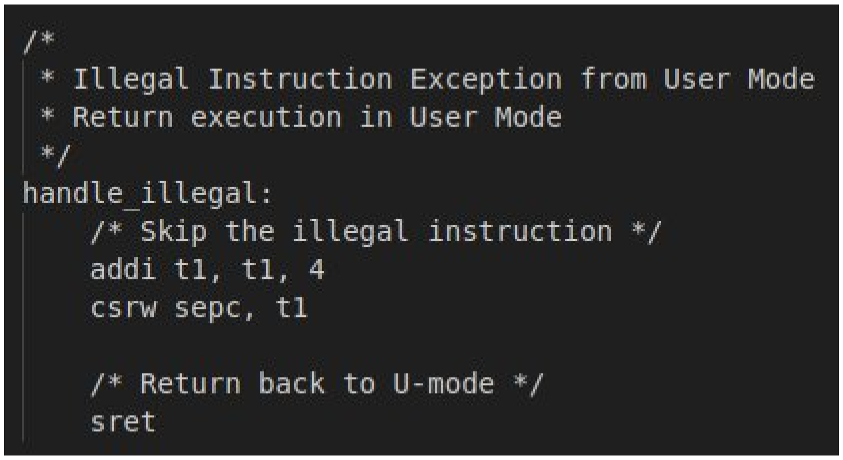


```
s_trap_vector:

    csrr t0, scause                      /* Read Trap cause */
    csrr t1, sepc                        /* Read Exception PC */

    /* Illegal instruction (scause = 2) */
    li   t2, 2
    beq  t0, t2, handle_illegal
```

**Illegal Instruction Handle:**

The exception program counter (sepc) is advanced to skip the faulting instruction.

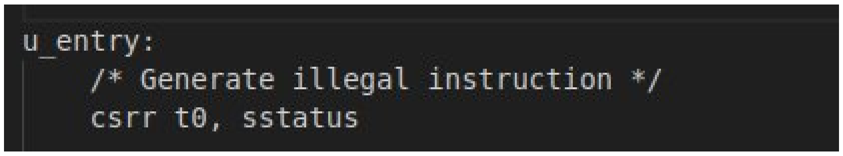
The sret instruction returns execution back to User mode.



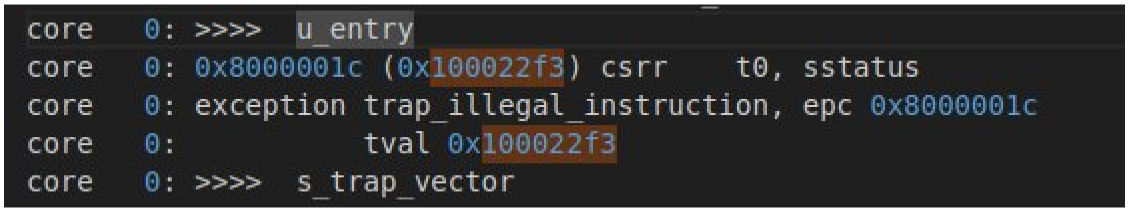
```
/*
 * Illegal Instruction Exception from User Mode
 * Return execution in User Mode
 */
handle_illegal:
    /* Skip the illegal instruction */
    addi t1, t1, 4
    csrw sepc, t1

    /* Return back to U-mode */
    sret
```

**Exception: (User Mode)**

We try to access  the sstatus CSR in User mode which is illegal and generates an illegal instruction exception.

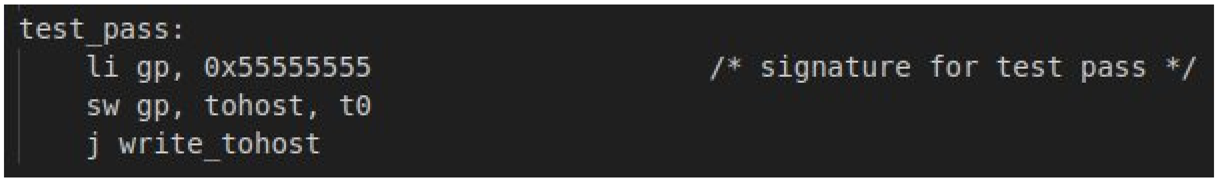


```
u_entry:
    /* Generate illegal instruction */
    csrr t0, sstatus
```



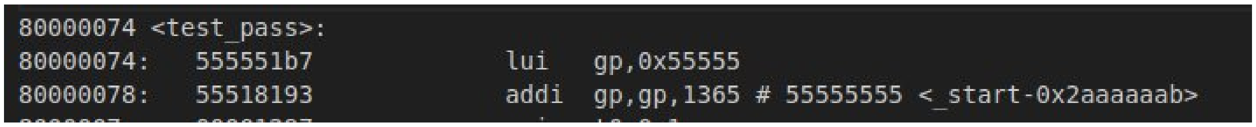
```
core   0: >>>>  u_entry
core   0: 0x8000001c (0x100022f3) csrr    t0, sstatus
core   0: exception trap_illegal_instruction, epc 0x8000001c
core   0:           tval 0x100022f3
core   0: >>>>  s_trap_vector
```

**Test_pass:**

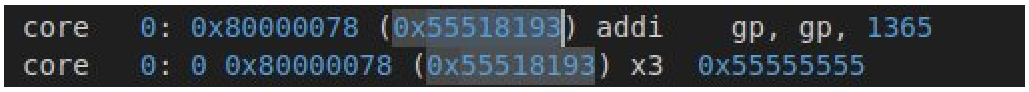


```
test_pass:
    li gp, 0x55555555                    /* signature for test pass */
    sw gp, tohost, t0
    j write_tohost
```

On successful program execution, it will jump to test_pass label which is loading a value 0x5555555 as a pass signature to the register gp. This value is only a marker which symbolizes that the test is passed.



```
80000074 <test_pass>:
80000074:   555551b7            lui   gp,0x55555
80000078:   55518193            addi  gp,gp,1365 # 55555555 <_start-0x2aaaaaab>
```



```
core   0: 0x80000078 (0x55518193) addi    gp, gp, 1365
core   0: 0 0x80000078 (0x55518193) x3  0x55555555
```