 # Author : Muhammad Imran
 # Date: 05-02-2026
 # Module : RISCV Arch Test
 # Section: RISCV Arch Test
 # Task Name: Task 1

Github link: RISCV Arch Test- Task 1

Test Description:

This test implements a function which switches the privilege mode based on the input argument. If the argument is '0' it switches the privilege mode to supervisor mode modifying the **mstatus, mepc, and mret** CSR registers and if the argument is '1' it switches the privilege mode to User mode. It also implements the trap vector which jumps to the appropriate trap handler based on the exception raised. If there is User **ECALL** exception it jumps to the **u_trap_handler** which stores the context using **REG_STORE,** handles the exception restore the context using **REG_RESTORE** and return to 1 higher privilege mode, the trap handler itself runs in the machine mode, the trap delegation is not used. If there is Supervisor **ECALL** exception it jumps to the **s_trap_handler** which handlers the exception and return to 1 higher privilege mode.

Finally the test always exit in M-mode.

Actual Output:

_start is the entry point of the program. Which initializes the stack and sets up the **mtvec** for trap handling. Any trap occurs during the execution of the program the program will jump to the address store in **mtvec.** Writing to **mtvec** verifies that the program starts in **M-mode[1]** because mtvec is machine mode csr and it can only be accessed in the machine mode. After that it jumps to the **main** from where the actual

program execution starts.

```
3   core   0: 0x00001004 (0x02028593) addi    a1, t0, 32
4   core   0: 3 0x00001004 (0x02028593) x11 0x00001020
5   core   0: 0x00001008 (0xf1402573) csrr    a0, mhartid
6   core   0: 3 0x00001008 (0xf1402573) x10 0x00000000
7   core   0: 0x0000100c (0x0182a283) lw      t0, 24(t0)
8   core   0: 3 0x0000100c (0x0182a283) x5  0x80001000 mem 0x00001018
9   core   0: 0x00001010 (0x00028067) jr      t0
10  core   0: 3 0x00001010 (0x00028067)
11  core   0: 0x80001000 (0x00006117) auipc   sp, 0x6
12  core   0: 3 0x80001000 (0x00006117) x2  0x80007000
13  core   0: 0x80001004 (0x00010113) mv      sp, sp
14  core   0: 3 0x80001004 (0x00010113) x2  0x80007000
15  core   0: 0x80001008 (0x00000317) auipc   t1, 0x0
16  core   0: 3 0x80001008 (0x00000317) x6  0x80001008
17  core   0: 0x8000100c (0x01030313) addi    t1, t1, 16
18  core   0: 3 0x8000100c (0x01030313) x6  0x80001018
19  core   0: 0x80001010 (0x30531073) csrw    mtvec, t1
20  core   0: 3 0x80001010 (0x30531073) c773_mtvec 0x80001018
21  core   0: 0x80001014 (0x0a40006f) j       pc + 0xa4
```

Switch to S-mode (M -> S)

main sets up the argument to call the **switch_mode** function. Initially the argument is set to '0'. Finally it calls the **switch_mode** function.

```
26  core   0: 3 0x800010bc (0x00000013)
27  core   0: 0x800010c0 (0x00000513) li      a0, 0
28  core   0: 3 0x800010c0 (0x00000513) x10 0x00000000
29  core   0: 0x800010c4 (0x008000ef) jal     pc + 0x8
30  core   0: 3 0x800010c4 (0x008000ef) x1  0x800010c8
31  core   0: 0x800010cc (0x300022f3) csrr    t0, mstatus
```

switch_mode The switch_mode function is called only in Machine mode. It updates the **mstatus MPP** field to select the previous privilege mode (01 = S-Mode) and writes the return address stored in **ra** into **mepc**. Executing **mret** jumps to the address stored in **mepc** with the privilege mode saved in **MPP** bits. When **a0=0**, execution switches to Supervisor mode.

```
47  core   0: 0x8000110c (0x00008f93) mv      t6, ra
48  core   0: 3 0x8000110c (0x00008f93) x31 0x800010e8
49  core   0: 0x80001110 (0x300022f3) csrr    t0, mstatus
50  core   0: 3 0x80001110 (0x300022f3) x5  0x00000000
51  core   0: 0x80001114 (0x00300313) li      t1, 3
52  core   0: 3 0x80001114 (0x00300313) x6  0x00000003
53  core   0: 0x80001118 (0x00b31313) slli    t1, t1, 11
54  core   0: 3 0x80001118 (0x00b31313) x6  0x00001800
55  core   0: 0x8000111c (0xfff34313) not     t1, t1
56  core   0: 3 0x8000111c (0xfff34313) x6  0xfffffe7ff
57  core   0: 0x80001120 (0x0062f2b3) and     t0, t0, t1
58  core   0: 3 0x80001120 (0x0062f2b3) x5  0x00000000
59  core   0: 0x80001124 (0x00100f13) li      t5, 1
60  core   0: 3 0x80001124 (0x00100f13) x30 0x00000001
61  core   0: 0x80001128 (0x00050663) beqz    a0, pc + 12
62  core   0: 3 0x80001128 (0x00050663)
63  core   0: 0x8000112c (0x03e50063) beq     a0, t5, pc + 32
64  core   0: 3 0x8000112c (0x03e50063)
65  core   0: 0x8000114c (0x341f9073) csrw    mepc, t6
66  core   0: 3 0x8000114c (0x341f9073) c833_mepc 0x800010e8
67  core   0: 0x80001150 (0x30029073) csrw    mstatus, t0
68  core   0: 3 0x80001150 (0x30029073) c768_mstatus 0x00000000
69  core   0: 0x80001154 (0x30200073) mret
70  core   0: 3 0x80001154 (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
```

Switch from S ⇒ U Mode

After successfully switching to **S** mode the program jump to the **switch_to_u_mode.** This label uses supervisor mode registers, it first reads the **sstatus** register and clear the **SPP** bit which indicate that on **sret** the program will execute in user mode.

```
core   0: >>>>  s_mode_code
core   0: 0x80000048 (0x05c0006f) j       pc + 0x5c
core   0: 1 0x80000048 (0x05c0006f)
core   0: 0x800000a4 (0x100022f3) csrr    t0, sstatus
core   0: 1 0x800000a4 (0x100022f3) x5  0x00000000
core   0: 0x800000a8 (0xeff00313) li      t1, -257
core   0: 1 0x800000a8 (0xeff00313) x6  0xfffffeff
core   0: 0x800000ac (0x0062f2b3) and     t0, t0, t1
core   0: 1 0x800000ac (0x0062f2b3) x5  0x00000000
core   0: 0x800000b0 (0x10029073) csrw    sstatus, t0
core   0: 1 0x800000b0 (0x10029073) c768_mstatus 0x00000080
core   0: 0x800000b4 (0x00000397) auipc   t2, 0x0
core   0: 1 0x800000b4 (0x00000397) x7  0x800000b4
core   0: 0x800000b8 (0xf9838393) addi    t2, t2, -104
core   0: 1 0x800000b8 (0xf9838393) x7  0x8000004c
core   0: 0x800000bc (0x14139073) csrw    sepc, t2
core   0: 1 0x800000bc (0x14139073) c321_sepc 0x8000004c
core   0: 0x800000c0 (0x10200073) sret
core   0: 1 0x800000c0 (0x10200073) c768_mstatus 0x000000a0
core   0: >>>>  u_mode_code
```

ECALL in U-Mode:

After returning in **U-mode** and **ECALL** in user mode will trigger the trap and execution jumps to the **trap_vect** using the address stored in **mtvec**

```
core   0: 0 0x80001114 (0x01de2223) mem 0x80001304 0x00000003
core   0: 0x80001118 (0x00000073) ecall
core   0: exception trap_user_ecall, epc 0x80001118
core   0: >>>>  trap_vector
core   0: 0x80001018 (0xf8010113) addi    sp, sp, -128
core   0: 3 0x80001018 (0xf8010113) x2  0x80006f80
```

Trap_vector read the **mcause.** It reads **macuse = 8** and jumps to the **u_trap_handler**

```
core   0: 3 0x80001094 (0x07f12e23) mem 0x80006ffc 0x8000110c
core   0: 0x80001098 (0x342022f3) csrr    t0, mcause
core   0: 3 0x80001098 (0x342022f3) x5  0x00000008
core   0: 0x8000109c (0x00129293) slli    t0, t0, 1
core   0: 3 0x8000109c (0x00129293) x5  0x00000010
core   0: 0x800010a0 (0x0012d293) srli    t0, t0, 1
core   0: 3 0x800010a0 (0x0012d293) x5  0x00000008
core   0: 0x800010a4 (0x00800313) li      t1, 8
core   0: 3 0x800010a4 (0x00800313) x6  0x00000008
core   0: 0x800010a8 (0x00900393) li      t2, 9
core   0: 3 0x800010a8 (0x00900393) x7  0x00000009
core   0: 0x800010ac (0x0c628063) beq     t0, t1, pc + 192
```

u_trap_handler stores the address of the next instruction in the **mepc** that will be executed in machine mode after mret. Modify the **MPP** bits of the **mstatus** register to change the privilege mode to the s_mode. Finally it restores the context and return in s-mode. It can be verified by the below logs

```
371   core   0: 0x8000116c (0x341023f3) csrr    t2, mepc
372   core   0: 3 0x8000116c (0x341023f3) x7  0x80001118
373   core   0: 0x80001170 (0x00438393) addi    t2, t2, 4
374   core   0: 3 0x80001170 (0x00438393) x7  0x8000111c
375   core   0: 0x80001174 (0x34139073) csrw    mepc, t2
376   core   0: 3 0x80001174 (0x34139073) c833_mepc 0x8000111c
377   core   0: 0x80001178 (0x300022f3) csrr    t0, mstatus
378   core   0: 3 0x80001178 (0x300022f3) x5  0x00000080
379   core   0: 0x8000117c (0x00300313) li      t1, 3
380   core   0: 3 0x8000117c (0x00300313) x6  0x00000003
381   core   0: 0x80001180 (0x00b31313) slli    t1, t1, 11
382   core   0: 3 0x80001180 (0x00b31313) x6  0x00001800
383   core   0: 0x80001184 (0xfff34313) not     t1, t1
384   core   0: 3 0x80001184 (0xfff34313) x6  0xfffffe7ff
385   core   0: 0x80001188 (0x0062f2b3) and     t0, t0, t1
386   core   0: 3 0x80001188 (0x0062f2b3) x5  0x00000080
387   core   0: 0x8000118c (0x00100313) li      t1, 1
388   core   0: 3 0x8000118c (0x00100313) x6  0x00000001
389   core   0: 0x80001190 (0x00b31313) slli    t1, t1, 11
390   core   0: 3 0x80001190 (0x00b31313) x6  0x00000800
391   core   0: 0x80001194 (0x0062e2b3) or      t0, t0, t1
392   core   0: 3 0x80001194 (0x0062e2b3) x5  0x00000880
393   core   0: 0x80001198 (0x30029073) csrw    mstatus, t0
394   core   0: 3 0x80001198 (0x30029073) c768_mstatus 0x00000880
395   core   0: 0x8000119c (0x00412083) lw      ra, 4(sp)
```

Now the program is in **S_mode** another ECALL will be trapped and returned in **M_mode,** finally the program can exit in **M-mode.**

Finally program exits by writing to the host

```
366   core   0: 0x80000450 (0x00100193) li      gp, 1
367   core   0: 3 0x80000450 (0x00100193) x3  0x00000001
368   core   0: 0x80000454 (0x00001297) auipc   t0, 0x1
369   core   0: 3 0x80000454 (0x00001297) x5  0x80001454
370   core   0: 0x80000458 (0xbac28293) addi    t0, t0, -1108
371   core   0: 3 0x80000458 (0xbac28293) x5  0x80001000
372   core   0: 0x8000045c (0x0032a023) sw      gp, 0(t0)
373   core   0: 3 0x8000045c (0x0032a023) mem 0x80001000 0x00000001
374   core   0: 0x80000460 (0xff1ff06f) j       pc - 0x10
375   core   0: 3 0x80000460 (0xff1ff06f)
376   core   0: 0x80000450 (0x00100193) li      gp, 1
```

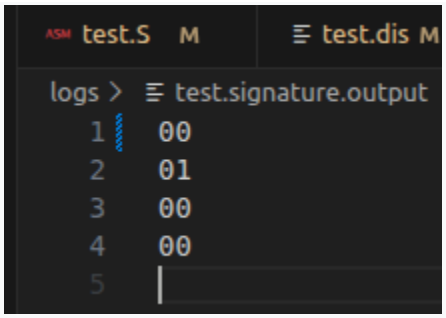
After returning in U-mode two ECALLS are executed

1st ECALL U ⇒ S

2ns ECALL S ⇒ M

Finally the program can exit in M mode.

Signature file:



How does the test start in M-mode?

The hart starts in Machine mode after reset, and the first executed instruction at **_start** runs in M-mode.

How is mode switching verified?

Mode switching in this test is verified by transitioning the processor through all privilege levels (Machine → Supervisor → User → Supervisor → Machine) and validating each transition using traps and CSR checks. Initially, the code runs in M-mode and switches to S-mode by modifying the mstatus.MPP bits and executing mret, which should cause execution to resume in S-mode at the correct return address. From S-mode, it switches to U-mode using sret after clearing the SPP bit in sstatus and setting sepc appropriately. The verification continues by executing ecall instructions in U-mode and S-mode to intentionally generate traps. The trap handler checks the mcause register to confirm whether the trap originated from U-mode (mcause = 8) or S-mode (mcause = 9), and it records the previous privilege level (MPP) into the signature region for validation. The handler then updates mstatus.MPP to the next intended mode and uses mret to return. If all transitions occur correctly and control flow proceeds as expected, the program writes 1 to tohost, indicating success; otherwise, it branches to fail. Thus, correct mode switching is verified through controlled privilege transitions,CSR inspection, and final pass/fail signaling.

How does the test exit?

The test always returns to Machine mode and writes to the **tohost** register, which terminates Spike execution.

Reference to RISC-V Spec

[1] Privilege Modes: Privileged Architecture v1.12, Sec 3.1

[2] Trap Handling: Sec 3.2-3.3 (mcause, mepc, mtvec, mret)

[3]ECALL Causes: Sec 4.3.1 (8 = U-mode, 9 = S-mode)