 # 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.

Each mode switch is verified by jumping to the corresponding verify label, which try to execute instructions not supported by the current mode and will raise an exception, during the handling of that exception previous privilege mode bits are extracted, written to the signature file and verified.

Switching to U-Mode from S-Mode is done by a function **u_mode_switch** which modifies the **status.SPP** bits and returns back to caller using **sret** and by setting **sepc** to the next instruction in main.

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
```

Verification of M-Mode at start:

M-mode is verified by calling **verify_m_mode** function which try to read the CSR **mstatus** which can only be read in machine mode, any other mode will trying to read this csr will trap write **0xff** in the signature file and exit the code. If it it successfully read this CSR then it will record the machine mode code in the signature file and return to the main.

```
27  core   0: 0x800000c0 (0x00300513) li      a0, 3
28  core   0: 3 0x800000c0 (0x00300513) x10 0x00000003
29  core   0: 0x800000c4 (0x058000ef) jal     pc + 0x58
30  core   0: 3 0x800000c4 (0x058000ef) x1  0x800000c8
31  core   0: 0x8000011c (0x300022f3) csrr    t0, mstatus
32  core   0: 3 0x8000011c (0x300022f3) x5  0x00000000
33  core   0: 0x80000120 (0x00300313) li      t1, 3
34  core   0: 3 0x80000120 (0x00300313) x6  0x00000003
35  core   0: 0x80000124 (0x006f0023) sb      t1, 0(t5)
36  core   0: 3 0x80000124 (0x006f0023) mem 0x80000470 0x03
37  core   0: 0x80000128 (0x00008067) ret
```

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  0xffffe7ff
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
```

Verification of Current Mode(S-mode):

Current S_mode is verified by calling the function **verify_s_mode.** This function takes an argument (0, 1, 3) which indicates trap occured during the verification of current mode and returns to main, it will not exit the test. S-mode is verified by executing an instruction which is not supported by the s-mode and will raise an exception. **Verify_s_mode** tries to access **mstatus** register which will raise an illegal instruction

exception

```
1   core   0: 0x800000d0 (0x00100513) li      a0, 1
2   core   0: 1 0x800000d0 (0x00100513) x10 0x00000001
3   core   0: 0x800000d4 (0x040000ef) jal     pc + 0x40
4   core   0: 1 0x800000d4 (0x040000ef) x1  0x800000d8
5   core   0: 0x80000114 (0x300022f3) csrr    t0, mstatus
6   core   0: exception trap_illegal_instruction, epc 0x80000114
7   core   0:           tval 0x300022f3
8   core   0: >>>>  trap_vector
9   core   0: 0x80000020 (0xf8010113) addi    sp, sp, -128
```

and the program will jump to **trap_vector** in machine mode. From where the previous privilege mode bits are extracted and comapred with the argument if MPP bits and the argument matches then it will record it in the signature file otherwise it will jump to fail and exit the test.

In short, on successful verification the current mode will be recorded in the signature file and the test will return to main.

Reading mstatus from the trap caused by verify_s_mode function

```
171   core   0: 0x80000318 (0x300022f3) csrr    t0, mstatus
172   core   0: 3 0x80000318 (0x300022f3) x5  0x00000800
173   core   0: 0x8000031c (0x00b2d293) srli    t0, t0, 11
174   core   0: 3 0x8000031c (0x00b2d293) x5  0x00000001
```

Writing it to signature file at the 2nd location

```
183   core   0: 0x80000330 (0x09c30663) beq     t1, t3, pc + 140
184   core   0: 3 0x80000330 (0x09c30663)
185   core   0: 0x800003bc (0x005f00a3) sb      t0, 1(t5)
186   core   0: 3 0x800003bc (0x005f00a3) mem 0x80000471 0x01
```

Returning to main after verification in the same privilege mode

```
250   core   0: 3 0x8000043c (0x08010113) x2  0x80007000
251   core   0: 0x80000440 (0x30200073) mret
252   core   0: 3 0x80000440 (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
253   core   0: 0x80000118 (0x00008067) ret
254   core   0: 1 0x80000118 (0x00008067)
```

ECALL in S-Mode:

After successfully verifying and switching to the **S_mode** an **ECALL[3]** triggers the trap and program jumps to the trap_vector using the address stored in the **mtvec** which at first stores the context using **SAVE_REG** macro on the stack

```
253   core   0: 0x80000118 (0x00008067) ret
254   core   0: 1 0x80000118 (0x00008067)
255   core   0: 0x800000d8 (0x00000073) ecall
256   core   0: exception trap_supervisor_ecall, epc 0x800000d8
257   core   0: >>>>  trap_vector
```

Then the trap vector reads the **mcause** csr and discards the MSB bit because of exception. And jumps to the **s_trap_handler.**

```
334   core   0: 0x800000b8 (0x18728a63) beq     t0, t2, pc + 404
335   core   0: 3 0x800000b8 (0x18728a63)
336   core   0: 0x8000024c (0x341023f3) csrr    t2, mepc
337   core   0: 3 0x8000024c (0x341023f3) x7  0x800000d8
338   core   0: 0x80000250 (0x00438393) addi    t2, t2, 4
```

[2] s_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 machie mode. It can be verified by the below logs

```
156   core   0: 0x80001158 (0x341023f3) csrr    t2, mepc
157   core   0: 3 0x80001158 (0x341023f3) x7  0x800010f4
158   core   0: 0x8000115c (0x00438393) addi    t2, t2, 4
159   core   0: 3 0x8000115c (0x00438393) x7  0x800010f8
160   core   0: 0x80001160 (0x34139073) csrw    mepc, t2
161   core   0: 3 0x80001160 (0x34139073) c833_mepc 0x800010f8
162   core   0: 0x80001164 (0x300022f3) csrr    t0, mstatus
163   core   0: 3 0x80001164 (0x300022f3) x5  0x00000000
164   core   0: 0x80001168 (0x00300313) li      t1, 3
165   core   0: 3 0x80001168 (0x00300313) x6  0x00000003
166   core   0: 0x8000116c (0x00b31313) slli    t1, t1, 11
```

Finally it restores the context and returns to the address stored in the mepc using mret which successfully switches the mode to **M-mode** as it can be verified by calling the **verify_m_mode** function.

```
248   core   0: 0x800012cc (0x08010113) addi    sp, sp, 128
249   core   0: 3 0x800012cc (0x08010113) x2  0x80007000
250   core   0: 0x800012d0 (0x30200073) mret
251   core   0: 3 0x800012d0 (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
```

Now the program is again in M-mode, so we can call the **switch_mode** function and change the privilege mode but this time it is called with argument **a0 = 1** which switches the mode to user

```
258   core   0: 0x80001104 (0x00100513) li      a0, 1
259   core   0: 3 0x80001104 (0x00100513) x10 0x00000001
260   core   0: 0x80001108 (0x018000ef) jal     pc + 0x18
261   core   0: 3 0x80001108 (0x018000ef) x1  0x8000110c
```

switch_mode stores the address of the next instruction in **mepc** and changes the **MPP** filed of **mstatus** csr to '00' and return using **mret.** This time it will return in **U-mode**

```
279   core   0: 3 0x80001140 (0x03e50063)
280   core   0: 0x80001160 (0x341f9073) csrw    mepc, t6
281   core   0: 3 0x80001160 (0x341f9073) c833_mepc 0x8000110c
282   core   0: 0x80001164 (0x30029073) csrw    mstatus, t0
283   core   0: 3 0x80001164 (0x30029073) c768_mstatus 0x00000080
284   core   0: 0x80001168 (0x30200073) mret
285   core   0: 3 0x80001168 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
```

Verification of Current Mode(U-mode):

Verification of U-mode is done following the same steps used in verification of S-mode. A function **verify_u_mode** with and argument a0 = 0 is called. Which tries to execute illegal instruction

```
457   core   0: 0 0x800000e8 (0x024000ef) x1  0x800000ec
458   core   0: 0x8000010c (0x300022f3) csrr    t0, mstatus
459   core   0: exception trap_illegal_instruction, epc 0x8000010c
460   core   0:           tval 0x300022f3
461   core   0: >>>>  trap_vector
```

On trapping the illegal instruction the previous privilege mode bits are extracted and compared with the input argument if match it will write the mode to the signature file at 3rd location and successfully return otherwise it will fail and exit the test.

MPP bits extraction and comparison with input argument and writing to the signature file

```
554   core   0: 0x80000318 (0x300022f3) csrr    t0, mstatus
555   core   0: 3 0x80000318 (0x300022f3) x5  0x00000080
556   core   0: 0x8000031c (0x00b2d293) srli    t0, t0, 11
557   core   0: 3 0x8000031c (0x00b2d293) x5  0x00000000
558   core   0: 0x80000320 (0x00300313) li      t1, 3
559   core   0: 3 0x80000320 (0x00300313) x6  0x00000003
560   core   0: 0x80000324 (0x0062f333) and     t1, t0, t1
561   core   0: 3 0x80000324 (0x0062f333) x6  0x00000000
562   core   0: 0x80000328 (0x00030663) beqz    t1, pc + 12
563   core   0: 3 0x80000328 (0x00030663)
564   core   0: 0x80000334 (0x005f0123) sb      t0, 2(t5)
565   core   0: 3 0x80000334 (0x005f0123) mem 0x80000472 0x00
```

Returning to caller after trap handling and then returing to main

```
core   0: 0x800003b8 (0x30200073) mret
core   0: 3 0x800003b8 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
core   0: 0x80000110 (0x00008067) ret
core   0: 0 0x80000110 (0x00008067)
core   0: 0x800000ec (0x00000073) ecall
```

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**

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

Trap_vector stores the context using **SAVE_REG** macro and read the **mcause.** It reads **macuse = 8** and jumps to the **u_trap_handler**

```
358   core   0: 3 0x80001094 (0x07f12e23) mem 0x80006ffc 0x8000110c
359   core   0: 0x80001098 (0x342022f3) csrr    t0, mcause
360   core   0: 3 0x80001098 (0x342022f3) x5  0x00000008
361   core   0: 0x8000109c (0x00129293) slli    t0, t0, 1
362   core   0: 3 0x8000109c (0x00129293) x5  0x00000010
363   core   0: 0x800010a0 (0x0012d293) srli    t0, t0, 1
364   core   0: 3 0x800010a0 (0x0012d293) x5  0x00000008
365   core   0: 0x800010a4 (0x00800313) li      t1, 8
366   core   0: 3 0x800010a4 (0x00800313) x6  0x00000008
367   core   0: 0x800010a8 (0x00900393) li      t2, 9
368   core   0: 3 0x800010a8 (0x00900393) x7  0x00000009
369   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
```

Switching from S-mode to U-mode (S -> U)

In this test at first the privilege mode is switched from **M -> S** by calling the **switch mode** function

Returning in S-mode and calling **u_mode_switch** function

```
004   core   0: 0x80000188 (0x30200073) mret
005   core   0: 3 0x80000188 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
006   core   0: 0x800000fc (0x030000ef) jal     pc + 0x30
007   core   0: 1 0x800000fc (0x030000ef) x1  0x80000100
```

U_mode_switch function reads the **sstatus** csr and modifies the **SPP** (previous previlege mode bit) to **'0'** which indicates that after **sret** the program will return to mode indicated by the this bit which is set to **U** mode. Reading **status,** modifying and returning in **U-mode**

```
1012   core   0: 0x80000134 (0x100022f3) csrr    t0, sstatus
1013   core   0: 1 0x80000134 (0x100022f3) x5  0x00000000
1014   core   0: 0x80000138 (0x00000393) li      t2, 0
1015   core   0: 1 0x80000138 (0x00000393) x7  0x00000000
1016   core   0: 0x8000013c (0x00839393) slli    t2, t2, 8
1017   core   0: 1 0x8000013c (0x00839393) x7  0x00000000
1018   core   0: 0x80000140 (0x0072f2b3) and     t0, t0, t2
1019   core   0: 1 0x80000140 (0x0072f2b3) x5  0x00000000
1020   core   0: 0x80000144 (0x10029073) csrw    sstatus, t0
1021   core   0: 1 0x80000144 (0x10029073) c768_mstatus 0x00000088
1022   core   0: 0x80000148 (0x10200073) sret
```

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 Output:

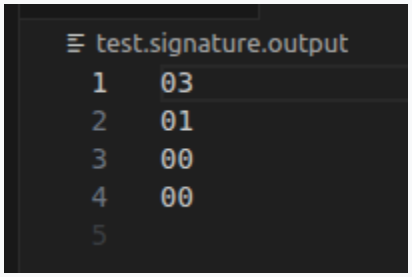
03 ====> M-Mode verification record

02 =====> S-Mode verification record

00 =====> U-Mode verification record

00 ======> Error record (this will be 0xff in case of any error)

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. It is verifies using **verify_m_mode** function.

How is mode switching verified?

It is verified using following function

verify_m_mode, verify_s_mode, verify_u_mode.

verify_m_mode tries to execute the highest privilege instruction if it successfully executes which indicates the current mode is M-mode, it will write to the signature file and ret.

verify_s_mode this tries to read CSR which is not supported by S-mode, if it raises a trap then our current mode is S-mode. In the trap handler it will extract the MPP bits, write them in the signature files and return to the same privilege mode.

Verify_u_mode has the same verification steps as the verify_s_mode.

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)