RISC-V arch test
Module: Privileged spec
Task 1

Test description(what test is trying to do)

Test description(what test is trying to doTest is trying to switch the modes to supervisor or user from machine mode. The test is started in main function which calls the switch mode function in which we are trying to switch to supervisor or user mode by playing with the mstatus bits. After changing the mode, test moves back to the main function where ecall interrupt or exception is being called that transfers control to the trap handler whose address is stored in mtvec. Trap is being handled and then returned back to the main function.

- **What's the actual output?(Try explaining this with snapshots of your log file)**

```
39 core   0: 0x0000000080000078 (0x01c3e3b3) or      t2, t2, t3
40 core   0: 3 0x0000000080000078 (0x01c3e3b3) x7  0x0000000a00000800
41 core   0: 0x000000008000007c (0x30039073) csrw    mstatus, t2
42 core   0: 3 0x000000008000007c (0x30039073) c768_mstatus 0x0000000a00000800
43 core   0: 0x0000000080000080 (0x00008067) ret
```

The value of mstatus is updated to be in the supervisor mode

```
29 core   0: 3 0x0000000080000080 (0x00008067)
30 core   0: 0x000000008000001c (0x00000073) ecall
31 core   0: exception trap_machine_ecall, epc 0x000000008000001c
32 core   0: >>>>  trap_handler
33 core   0: 0x000000008000009c (0xff010113) addi    sp, sp, -16
34 core   0: 3 0x000000008000009c (0xff010113) x2  0x0000000080001ff0
35 core   0: 0x00000000800000a0 (0x00512023) sw      t0, 0(sp)
36 core   0: 3 0x00000000800000a0 (0x00512023) mem 0x0000000080001ff0 0x00000080
37 core   0: 0x00000000800000a4 (0x00612223) sw      t1, 4(sp)
38 core   0: 3 0x00000000800000a4 (0x00612223) mem 0x0000000080001ff4 0x00000000
39 core   0: 0x00000000800000a8 (0x00712423) sw      t2, 8(sp)
40 core   0: 3 0x00000000800000a8 (0x00712423) mem 0x0000000080001ff8 0x00000800
41 core   0: 0x00000000800000ac (0x01c12623) sw      t3, 12(sp)
```

Ecall is called and control transferred to trap handler

```
28 core   0: 3 0x0000000080000104 (0x01010113) x2  0x0000000080002000
29 core   0: 0x0000000080000108 (0x341022f3) csrr    t0, mepc
30 core   0: 3 0x0000000080000108 (0x341022f3) x5  0x000000008000001c
31 core   0: 0x000000008000010c (0x00428293) addi    t0, t0, 4
32 core   0: 3 0x000000008000010c (0x00428293) x5  0x0000000080000020
33 core   0: 0x0000000080000110 (0x34129073) csrw    mepc, t0
34 core   0: 3 0x0000000080000110 (0x34129073) c833_mepc 0x0000000080000020
35 core   0: 0x0000000080000114 (0x30200073) mret
36 core   0: 3 0x0000000080000114 (0x30200073) c768_mstatus 0x0000000a00000080
37 core   0: 0x0000000080000020 (0x00100513) li      a0, 1
38 core   0: 3 0x0000000080000020 (0x00100513) x10 0x0000000000000001
39 core   0: 0x0000000080000024 (0x030000ef) jal     pc + 0x30
40 core   0: 3 0x0000000080000024 (0x030000ef) x1  0x0000000080000028
41 core   0: >>>>  switch_mode
```

Returned from the trap handler to the address stored in the mepc register + 4 and then switch mode function is called that will change to user mode by updating the mstatus register

```
33 core   0: 3 0x0000000080000064 (0x02650063)
34 core   0: >>>>  set_user_mode
35 core   0: 0x0000000080000084 (0x300023f3) csrr    t2, mstatus
36 core   0: 3 0x0000000080000084 (0x300023f3) x7  0x0000000a00000080
37 core   0: 0x0000000080000088 (0x00001e37) lui     t3, 0x1
38 core   0: 3 0x0000000080000088 (0x00001e37) x28 0x0000000000001000
39 core   0: 0x000000008000008c (0x01c3e3b3) or      t2, t2, t3
40 core   0: 3 0x000000008000008c (0x01c3e3b3) x7  0x0000000a00001080
41 core   0: 0x0000000080000090 (0x30039073) csrw    mstatus, t2
42 core   0: 3 0x0000000080000090 (0x30039073) c768_mstatus 0x0000000a00000080
43 core   0: 0x0000000080000094 (0x00008067) ret
44 core   0: 3 0x0000000080000094 (0x00008067)
45 core   0: 0x0000000080000028 (0x00000073) ecall
```

Updated mstatus with the user mode values

```
28 core   0: 0x0000000080000094 (0x00008067) ret
29 core   0: 3 0x0000000080000094 (0x00008067)
30 core   0: 0x0000000080000028 (0x00000073) ecall
31 core   0: exception trap_machine_ecall, epc 0x0000000080000028
32 core   0: >>>>  trap_handler
33 core   0: 0x000000008000009c (0xff010113) addi    sp, sp, -16
34 core   0: 3 0x000000008000009c (0xff010113) x2  0x0000000080001ff0
35 core   0: 0x00000000800000a0 (0x00512023) sw      t0, 0(sp)
36 core   0: 3 0x00000000800000a0 (0x00512023) mem 0x0000000080001ff0 0x00000080
37 core   0: 0x00000000800000a4 (0x00612223) sw      t1, 4(sp)
38 core   0: 3 0x00000000800000a4 (0x00612223) mem 0x0000000080001ff4 0x00000001
39 core   0: 0x00000000800000a8 (0x00712423) sw      t2, 8(sp)
40 core   0: 3 0x00000000800000a8 (0x00712423) mem 0x0000000080001ff8 0x00001080
41 core   0: 0x00000000800000ac (0x01c12623) sw      t3, 12(sp)
```

Trap is called and the control is transferred to trap handler

- **Add snapshot or reference of spec, if possible for expected behaviour**

RV32 and Figure 3.7 for RV64. The `mstatus` register keeps track of and controls the hart's current operating state. A restricted view of `mstatus` appears as the `sstatus` register in the S-level ISA.

31	30 … 23	22	21	20	19	18	17
SD	WPRI	TSR	TW	TVM	MXR	SUM	MPRV
1	8	1	1	1	1	1	1

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 3.6: Machine-mode status register (`mstatus`) for RV32.

63	62 … 38	37	36	35 34	33 32	31 … 23	22	21	20	19	18
SD	WPRI	MBE	SBE	SXL[1:0]	UXL[1:0]	WPRI	TSR	TW	TVM	MXR	SUM
1	25	1	1	2	2	9	1	1	1	1	1

17	16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
MPRV	XS[1:0]	FS[1:0]	MPP[1:0]	VS[1:0]	SPP	MPIE	UBE	SPIE	WPRI	MIE	WPRI	SIE	WPRI
1	2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 3.7: Machine-mode status register (`mstatus`) for RV64.

For RV32 only, `mstatush` is a 32-bit read/write register formatted as shown in Figure 3.8. Bits 30:4 of `mstatush` generally contain the same fields found in bits 62:36 of `mstatus` for RV64. Fields

How can you start your test in M-mode?

By default the program runs in M-mode, so we don't need to set it to M-mode because the program will already be running in M-mode.

Verify that you have switched to the required mode(Figure out How to verify this)

We can verify that we have switched to the required mode by exploring the mstatus register, specifically MPP bits of the mstatus register that tells us the privilege mode in which the program is running.

Try switching to U mode. (figure out how you can do this using standard way)

We can switch to U mode by setting the MPP bits to 01 i.e 12 and 11 bit in mstatus register.