RISC-V Arch Test

Task 2

[GITHUB REPO]

Test Description:

This assembly test verifies RISC-V Physical Memory Protection (PMP) functionality across Supervisor and Machine modes. The test includes:

1. PMP Configuration:

Two 4KB memory regions:

- TOR (Top-of-Range) region: execute only (X) from 0x80002000 to 0x80003000.
- NAPOT (Naturally Aligned Power-of-Two) region: read only (R) at 0x80003000.

2. Privilege Mode Switching:

- The test switches from Machine mode (M-mode) to Supervisor mode (S-mode).
- In S-mode, accesses to memory regions are tested to verify proper PMP enforcement.

3. Trap Handling:

- Custom trap handler captures load, store, and instruction access faults.
- Faulting instructions are skipped or redirected to valid regions, enabling the test to continue.

4. Access Tests:

- Supervisor Mode:
- Execute TOR region - allowed.
- Read NAPOT region - allowed.

- Write NAPOT region - store access fault.
- Execute NAPOT region - instruction access fault.
- Read/Write TOR region - load/store access faults.

5. Machine Mode:

- PMP entries locked.
- MPRV enabled to apply S-mode PMP permissions from M-mode.
- Same access tests repeated to confirm enforcement.

6. Pass/Fail Indication:

- Signature 0x55555555, test passed.
- Signature 0xdeadbeef, test failed.

PMP Configuration:

```
/* Protect for entry1 region start */
li t0, 0x80002000
srli t0, t0, 2
csrw pmpaddr1, t0

li t0, (0x0F << 8)
csrr t1, pmpcfg0
or t1, t1, t0
csrw pmpcfg0, t1

/*  TOR REGION */
/* 4KB at execute only */
li t0, 0x80003000
srli t0, t0, 2
csrw pmpaddr2, t0

/* entry1 = TOR + X (0x0C) */
li t0, (0x0C << 16)
csrr t1, pmpcfg0
or t1, t1, t0
csrw pmpcfg0, t1
```

```
/* NAPOT REGION */
/* 4KB at 0x80003000 read only */
li t0, 0x80003000
srli t0, t0, 2
li t1, 0x1FF
or t0, t0, t1
csrw pmpaddr3, t0

/* entry2 = NAPOT + R (0x19) */
li t0, (0x19 << 24)
csrr t1, pmpcfg0
or t1, t1, t0
csrw pmpcfg0, t1
```

For TOR region, it is configured for Execute permissions only using the pmpaddr2 register.

For NAPOT register, it is configured for Read only permission using the pmpaddr3 register.

M-mode lock configuration:

```
/* Lock PMP entries
 * entry0 for TOR RX + L
 * entry1 for TOR X + L
 * entry2 for NAPOT R + L
 */
li t0, 0x88
li t1, (0x8F << 8)
li t2, (0x8C << 16)
li t3, (0x99 << 24)
or t0, t0, t1
or t0, t0, t2
or t0, t0, t3
csrw pmpcfg0, t0
```

M-mode is configured for lock bit so that all the permission for pmp are enforced to M-mode.

Supervisor Tests:

```
supervisor_tests:

    /* execute TOR (allowed) */
    la t0, tor_execution
    jalr t0

    /* Read NAPOT (allowed) */
    li t0, 0x80003000
    lw t1, 0(t0)

    /* Write NAPOT (Store access fault) */
    sw t1, 0(t0)

    /* Execute NAPOT (Instruction access fault) */
    la t0, napot_execution
    jalr t0

    /* Read TOR (Load access fault) */
    li t0, 0x80002000
    lw t1, 0(t0)

    /* Write TOR (Store access fault) */
    sw t1, 0(t0)

    /* Switch to M-mode using handler */
    ecall
```

Same set of tests are applied to m-mode after locked bit configuration to see the result.

After checking all the permission successfully in S-mode we do ecall to switch back to machine mode. Ecall handler is responsible for this conversation.

Store access fault NAPOT region:

```
800000c4:   0062a023            sw t1,0(t0)
```

```
core   0: 0x800000c4 (0x0062a023) sw      t1, 0(t0)
core   0: exception trap_store_access_fault, epc 0x800000c4
core   0:           tval 0x80003000
core   0: >>>>  trap_vector
```

Instruction access fault NAPOT region:

```
800000cc:   f3828293            addi  t0,t0,-200 # 80003000 <napot_execution>
800000d0:   000280e7            jalr  ra,0(t0)
```

```
core   0: 0x800000d0 (0x000280e7) jalr    t0
core   0: 1 0x800000d0 (0x000280e7) x1  0x800000d4
core   0: exception trap_instruction_access_fault, epc 0x80003000
core   0:           tval 0x80003000
core   0: >>>>  trap_vector
```

Load access fault TOR region:

```
800000d8:   0002a303            lw t1,0(t0) # 80002000 <tor_execution>
```

```
core   0: 1 0x800000d4 (0x800022b7) x5  0x80002000
core   0: 0x800000d8 (0x0002a303) lw      t1, 0(t0)
core   0: exception trap_load_access_fault, epc 0x800000d8
core   0:           tval 0x80002000
core   0: >>>>  trap_vector
```

Store access fault TOR region:

```
800000dc:   0062a023            sw t1,0(t0)
```

```
core   0: 0x800000dc (0x0062a023) sw      t1, 0(t0)
core   0: exception trap_store_access_fault, epc 0x800000dc
core   0:           tval 0x80002000
core   0: >>>>  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.

```
80000374 <test_pass>:
80000374:   555551b7            lui   gp,0x55555
80000378:   55518193            addi  gp,gp,1365 # 55555555 <_start-0x2aaaaaab>
```

```
core   0: 0x80000378 (0x55518193) addi    gp, gp, 1365
core   0: 3 0x80000378 (0x55518193) x3  0x55555555
```

Why do you get an Instruction Access Fault after entering S-mode?

When we transition to a lower privilege mode (S or U), all memory accesses are checked against the PMP. If the memory location where our code resides is not explicitly covered by a PMP entry with Execute (X) permissions, the processor will trigger an Instruction Access Fault.

How do you apply PMP restrictions to Machine Mode?

The "L" (Lock) Bit: By default, PMP rules do not apply to M-mode. To enforce them, you must set the Lock bit (bit 7) in the pmpcfg register for the specific entry.

Once the L-bit is set, the permissions apply to all modes, including M-mode.