RISC V ARCH TEST
Task

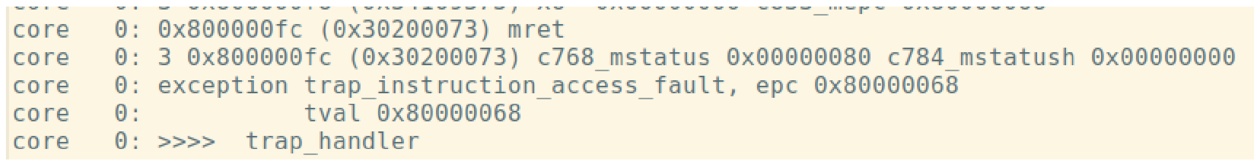
Task Description: To understand PMP (Physical Memory Protection) from privileged spec Then create two regions 4kb one by NAPTO method and other by TOR and give read permission to NAPOT region and execute permission to TOR region then solve the exception caused (access Fault) by giving the appropriate permissions to the regions in the last also apply PMP permissions to M mode.

Note: We update the trap handler so it now updates the permission for regions according to fault eg, instruction access fault occurs, then the trap handler gives ex permission TO both regions

Note: I created another section for the trap handler in memory so we can handle the case when We lock in 0x80000000 -0x80001000 in m modes ,so code smoothly test the lock feature Of PMP

Note: i created 0x80003000 -0x80004000 region for Tranphandler and gave Ex permission to

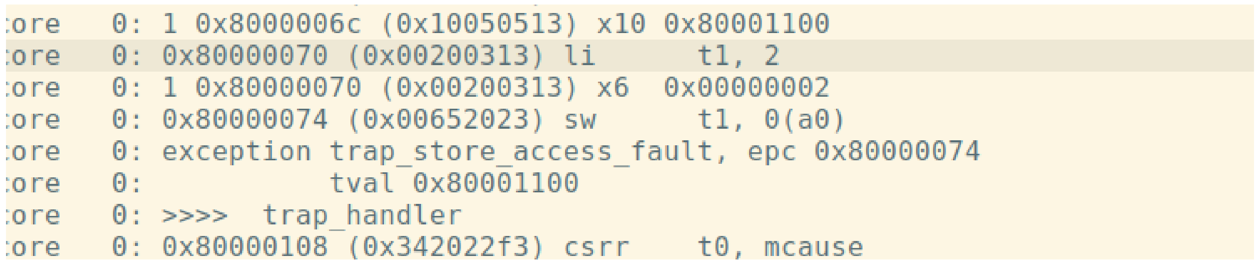
Step1: first setup pmpcfg0 for pmpadd0,addr1,addr2 ,pmpaddr0 for NAPOT and reset of two for for TOR Region and switch to supervisor mode
NAPOT Region starts from 0x80000000 -0x80001000 with r permissions
TOR Region 0x80002000-0x80003000 with ex permissions
And then switch to Supervisor mode to check the effect of PMP
Expected output would be inst_access fault: because test init_test region start From 0x80000000, it is the NAPOT region with only read permission


```
core   0: 0x800000fc (0x30200073) mret
core   0: 3 0x800000fc (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
core   0: exception trap_instruction_access_fault, epc 0x80000068
core   0:           tval 0x80000068
core   0: >>>>  trap_handler
```

Trap Handler will update the permissions for the NAPOT region

Step 2: After giving the ex permissions to the region, write to any of the regions. The expected exception is a store access fault. The trap handler handle this exception and gave read and write permission to both regions.

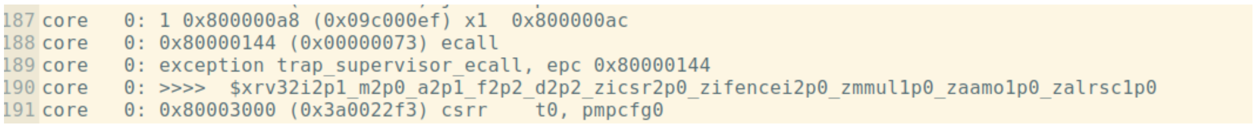
Sw at address 0x80001100 expected fault : store access_fault


```
ore   0: 1 0x8000006c (0x10050513) x10 0x80001100
ore   0: 0x80000070 (0x00200313) li      t1, 2
ore   0: 1 0x80000070 (0x00200313) x6  0x00000002
ore   0: 0x80000074 (0x00652023) sw      t1, 0(a0)
ore   0: exception trap_store_access_fault, epc 0x80000074
ore   0:           tval 0x80001100
ore   0: >>>>  trap_handler
ore   0: 0x80000108 (0x342022f3) csrr    t0, mcause
```

RISC V ARCH TEST
Task

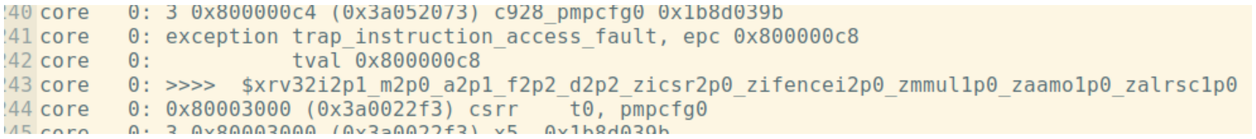
Step 3: Switch to M mode by calling the supervisor ecall then Apply the same permission again as we started, and this time set L bit (lock bit) in the Pmpcfg0, after set up the lock bit, The permissions are effective in machine mode access as well

Switching to M-mode



```
core   0: 1 0x800000a8 (0x09c000ef) x1  0x800000ac
core   0: 0x80000144 (0x00000073) ecall
core   0: exception trap_supervisor_ecall, epc 0x80000144
core   0: >>>>  $xrv32i2p1_m2p0_a2p1_f2p2_d2p2_zicsr2p0_zifencei2p0_zmmul1p0_zaamo1p0_zalrsc1p0
core   0: 0x80003000 (0x3a0022f3) csrr    t0, pmpcfg0
```

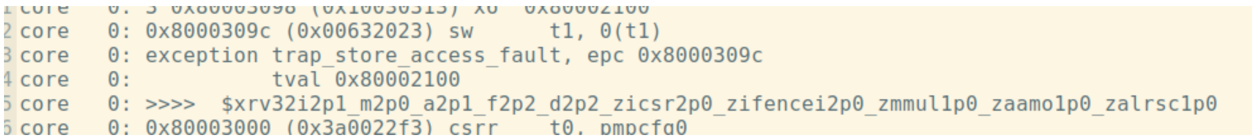
Now set the lock bit in pmpcfg, then the expected exception is an inst_access fault



```
core   0: 3 0x800000c4 (0x3a052073) c928_pmpcfg0 0x1b8d039b
core   0: exception trap_instruction_access_fault, epc 0x800000c8
core   0:           tval 0x800000c8
core   0: >>>>  $xrv32i2p1_m2p0_a2p1_f2p2_d2p2_zicsr2p0_zifencei2p0_zmmul1p0_zaamo1p0_zalrsc1p0
core   0: 0x80003000 (0x3a0022f3) csrr    t0, pmpcfg0
core   0: 3 0x80003000 (0x3a0022f3) x5  0x1b8d039b
```

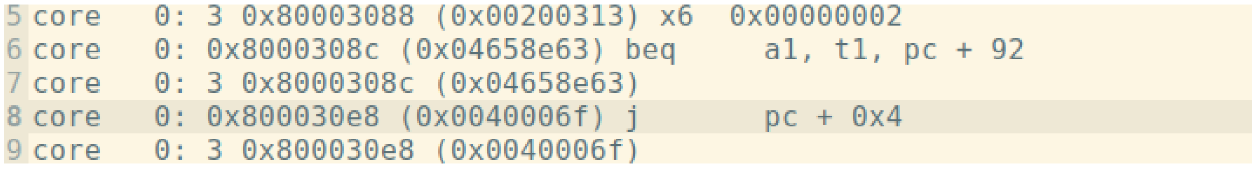
As Trap handler is not in the same region as program code so we also test the store word test by storing the word in 0x80002000 region

Expected expectation is a store access fault



```
core   0: 3 0x80003098 (0x10030313) x6  0x80002100
core   0: 0x8000309c (0x00632023) sw      t1, 0(t1)
core   0: exception trap_store_access_fault, epc 0x8000309c
core   0:           tval 0x80002100
core   0: >>>>  $xrv32i2p1_m2p0_a2p1_f2p2_d2p2_zicsr2p0_zifencei2p0_zmmul1p0_zaamo1p0_zalrsc1p0
core   0: 0x80003000 (0x3a0022f3) csrr    t0, pmpcfg0
```

Note: we are in M mode, so we can exit the code by j test pass



```
core   0: 3 0x80003088 (0x00200313) x6  0x00000002
core   0: 0x8000308c (0x04658e63) beq     a1, t1, pc + 92
core   0: 3 0x8000308c (0x04658e63)
core   0: 0x800030e8 (0x0040006f) j       pc + 0x4
core   0: 3 0x800030e8 (0x0040006f)
```