Author : Irtaza Hyder

Date : 2026-02-09

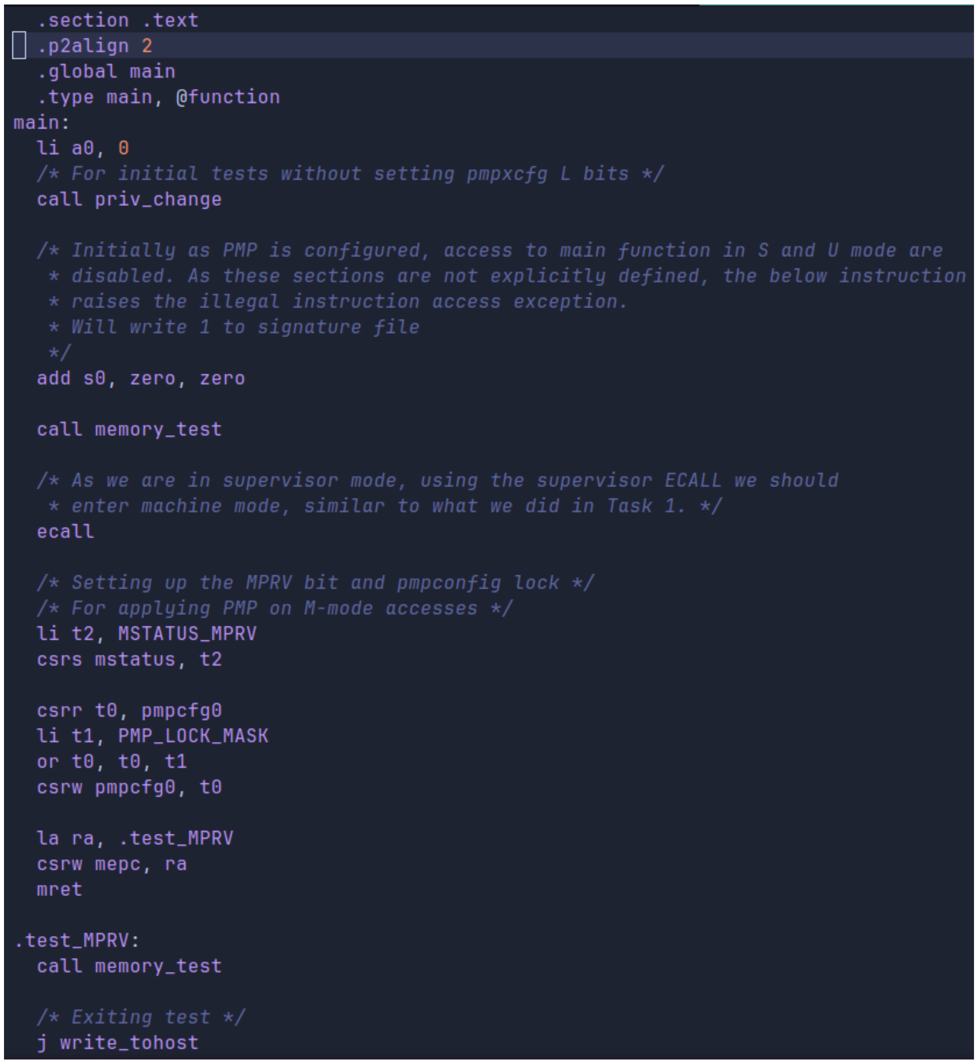
Module : RISC-V Arch Test

Section : RISC-V Arch Test

Task Name : Task 2

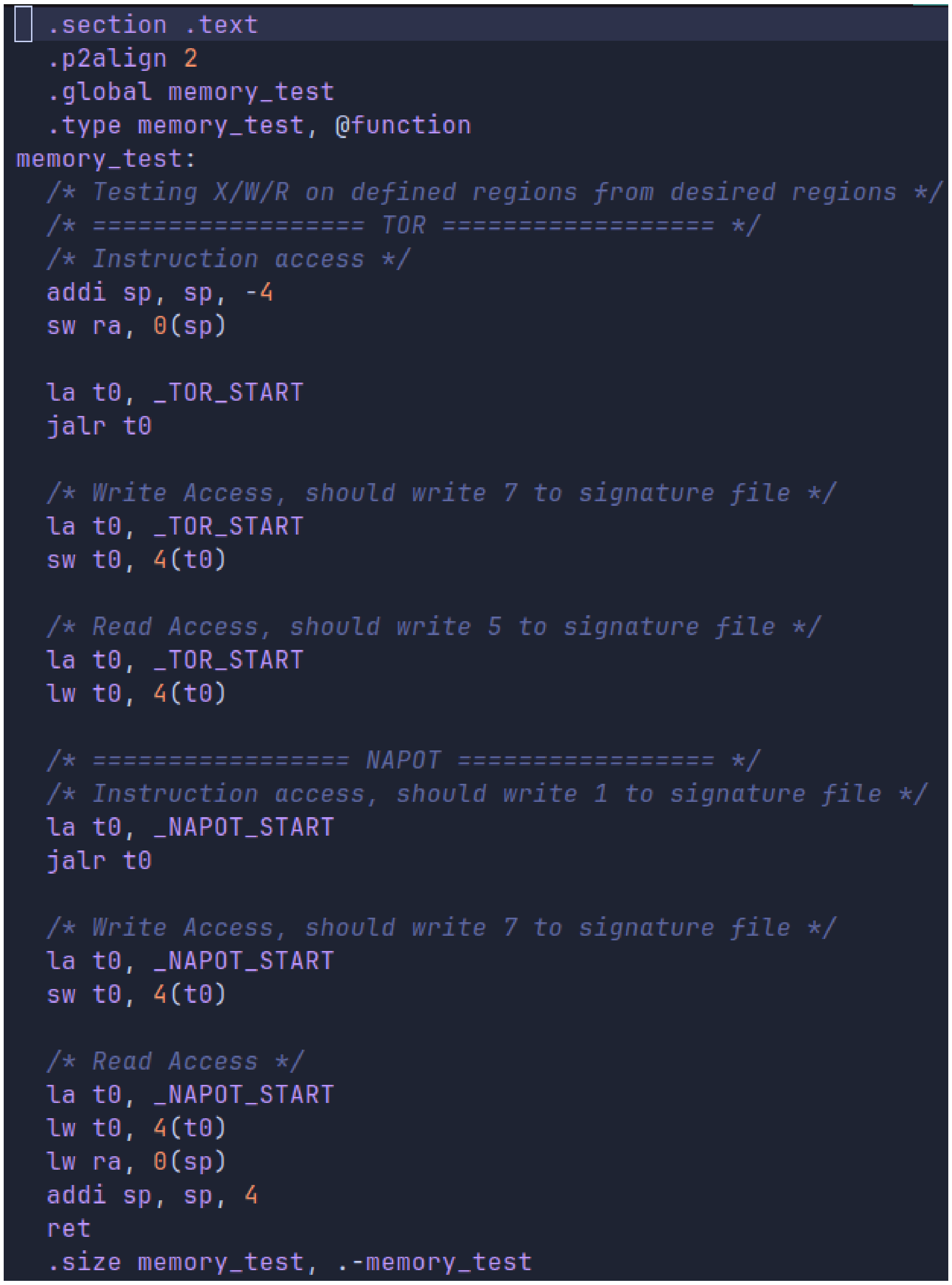
Description: Write an assembly test to handle exceptions in S mode

Test Description



```
  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  li a0, 0
  /* For initial tests without setting pmpxcfg L bits */
  call priv_change

  /* Initially as PMP is configured, access to main function in S and U mode are
   * disabled. As these sections are not explicitly defined, the below instruction
   * raises the illegal instruction access exception.
   * Will write 1 to signature file
   */
  add s0, zero, zero

  call memory_test

  /* As we are in supervisor mode, using the supervisor ECALL we should
   * enter machine mode, similar to what we did in Task 1. */
  ecall

  /* Setting up the MPRV bit and pmpconfig lock */
  /* For applying PMP on M-mode accesses */
  li t2, MSTATUS_MPRV
  csrs mstatus, t2

  csrr t0, pmpcfg0
  li t1, PMP_LOCK_MASK
  or t0, t0, t1
  csrw pmpcfg0, t0

  la ra, .test_MPRV
  csrw mepc, ra
  mret

.test_MPRV:
  call memory_test

  /* Exiting test */
  j write_tohost
```

test.S: lines 3-43. The test first changes from Machine mode to Supervisor mode and then runs the memory tests (explained below). Then we use ecall to upgrade from supervisor to machine mode (like in Task 1). MPRV bit is set (Modify PriVilege), and the lock bit is set for all the memory regions in pmpcfg0 for implementing PMP for machine mode access as well. Finally the memory tests are executed again to observe if PMP affects Machine mode accesses.

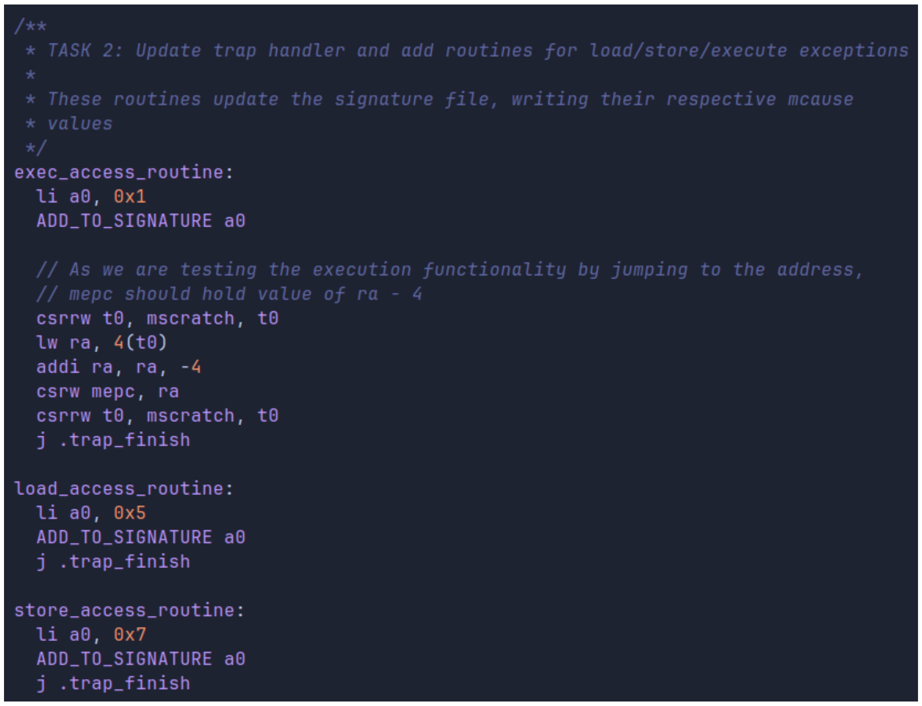


```
  .section .text
  .p2align 2
  .global memory_test
  .type memory_test, @function
memory_test:
  /* Testing X/W/R on defined regions from desired regions */
  /* ================== TOR ================== */
  /* Instruction access */
  addi sp, sp, -4
  sw ra, 0(sp)

  la t0, _TOR_START
  jalr t0

  /* Write Access, should write 7 to signature file */
  la t0, _TOR_START
  sw t0, 4(t0)

  /* Read Access, should write 5 to signature file */
  la t0, _TOR_START
  lw t0, 4(t0)

  /* ================= NAPOT ================= */
  /* Instruction access, should write 1 to signature file */
  la t0, _NAPOT_START
  jalr t0

  /* Write Access, should write 7 to signature file */
  la t0, _NAPOT_START
  sw t0, 4(t0)

  /* Read Access */
  la t0, _NAPOT_START
  lw t0, 4(t0)
  lw ra, 0(sp)
  addi sp, sp, 4
  ret
  .size memory_test, .-memory_test
```

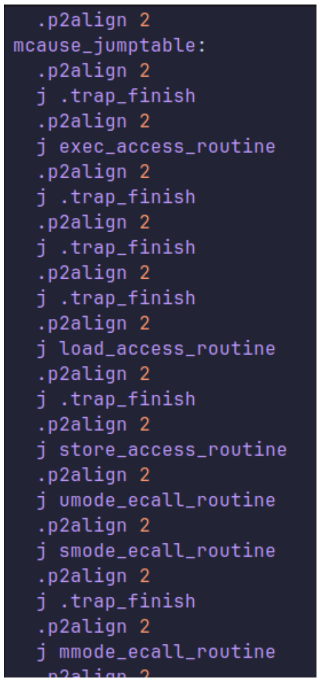
test.S: line 45-85. Execute, store and read are tested firstly in the TOR and then in the NOTAP defined region. If any instruction/load/store access fault is encountered, the mcause is written to the signature.txt file.

Output and Q&A

1. Enhance your trap handler from task 1 to handler load/store/execute exceptions



```
/**
 * TASK 2: Update trap handler and add routines for load/store/execute exceptions
 *
 * These routines update the signature file, writing their respective mcause
 * values
 */
exec_access_routine:
  li a0, 0x1
  ADD_TO_SIGNATURE a0

  // As we are testing the execution functionality by jumping to the address,
  // mepc should hold value of ra - 4
  csrrw t0, mscratch, t0
  lw ra, 4(t0)
  addi ra, ra, -4
  csrw mepc, ra
  csrrw t0, mscratch, t0
  j .trap_finish

load_access_routine:
  li a0, 0x5
  ADD_TO_SIGNATURE a0
  j .trap_finish

store_access_routine:
  li a0, 0x7
  ADD_TO_SIGNATURE a0
  j .trap_finish
```

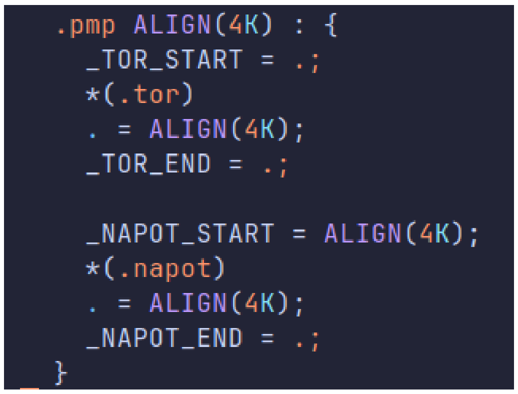


```
  .p2align 2
mcause_jumptable:
  .p2align 2
  j .trap_finish
  .p2align 2
  j exec_access_routine
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j load_access_routine
  .p2align 2
  j .trap_finish
  .p2align 2
  j store_access_routine
  .p2align 2
  j umode_ecall_routine
  .p2align 2
  j smode_ecall_routine
  .p2align 2
  j .trap_finish
  .p2align 2
  j mmode_ecall_routine
```

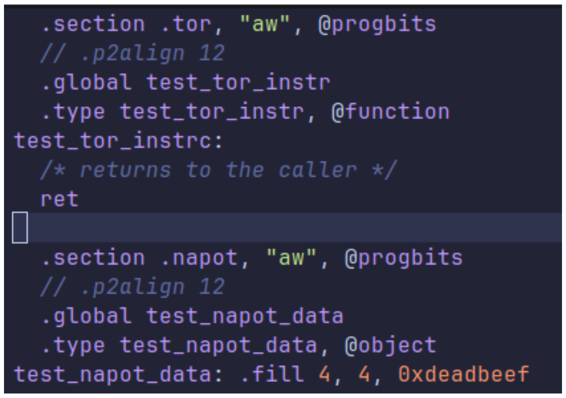
left) init.S: lines 169-196. The trap handler for access faults write their mcause values (in the form of a single byte) into the signature file. This is later compared to find if the test has passed or failed.

right) init.S: lines 126-151. The updated jumptable entries.

2. Configure two 4KB PMP regions one using TOR method and other using NAPOT method.



```
  .pmp ALIGN(4K) : {
    _TOR_START = .;
    *(.tor)
    . = ALIGN(4K);
    _TOR_END = .;

    _NAPOT_START = ALIGN(4K);
    *(.napot)
    . = ALIGN(4K);
    _NAPOT_END = .;
  }
```



```
  .section .tor, "aw", @progbits
  // .p2align 12
  .global test_tor_instr
  .type test_tor_instr, @function
test_tor_instrc:
  /* returns to the caller */
  ret

  .section .napot, "aw", @progbits
  // .p2align 12
  .global test_napot_data
  .type test_napot_data, @object
test_napot_data: .fill 4, 4, 0xdeadbeef
```

left) link.ld lines: 31-41. Declaring the TOR and NAPOT regions in the linker file.

right) pmp_section.S. Defining the .tor and .napot regions in assembly. The .tor region has only the `ret` instruction (hence when we do the function call, the ret instruction would execute and we would return to our original code). The .napot region has defined 16 bytes (repeated word 0xdeadbeef).

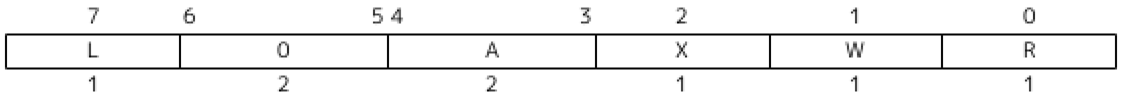


Figure 34. PMP configuration register format.

RISC-V ISA Manual: Volume II: Privilege Architecture (page 68). pmpXcfg fields. L = lock, A = address, X = execute allowed, W = write allowed, R = read allowed.

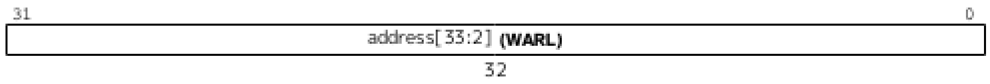
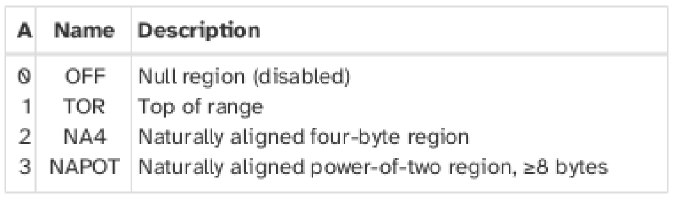


Figure 32. PMP address register format, RV32.

RISC-V ISA Manual: Volume II: Privilege Architecture (page 68). As seen in the diagram, pmpaddrX requires the given address to be right shifted the by 2 bits.

Table 18. Encoding of A field in PMP configuration registers.

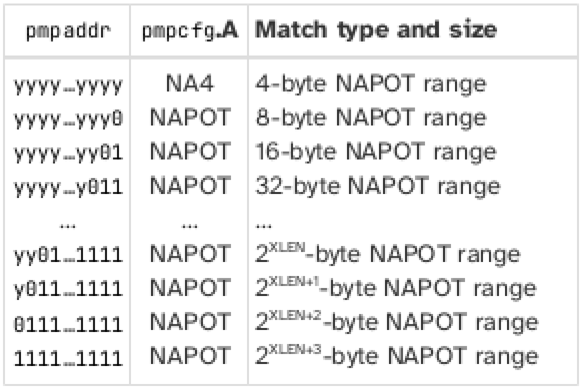


A	Name	Description
0	OFF	Null region (disabled)
1	TOR	Top of range
2	NA4	Naturally aligned four-byte region
3	NAPOT	Naturally aligned power-of-two region, ≥8 bytes

RISC-V ISA Manual: Volume II: Privilege Architecture (page 69). Address field encoding for pmpXcfg.

TOR: Top of the range region is configured by defining the region in the linker file, then loading the _TOR_START address in pmpaddr2 and _TOR_END in pmpaddr3 (after shifting them right by 2 bits). The pmp3cfg[4:3] are set to 01.

Table 19. NAPOT range encoding in PMP address and configuration registers.



pmpaddr	pmpcfg.A	Match type and size
yyyy…yyyy	NA4	4-byte NAPOT range
yyyy…yyy0	NAPOT	8-byte NAPOT range
yyyy…yy01	NAPOT	16-byte NAPOT range
yyyy…y011	NAPOT	32-byte NAPOT range
…	…	…
yy01…1111	NAPOT	2^{XLEN}-byte NAPOT range
y011…1111	NAPOT	2^{XLEN+1}-byte NAPOT range
0111…1111	NAPOT	2^{XLEN+2}-byte NAPOT range
1111…1111	NAPOT	2^{XLEN+3}-byte NAPOT range

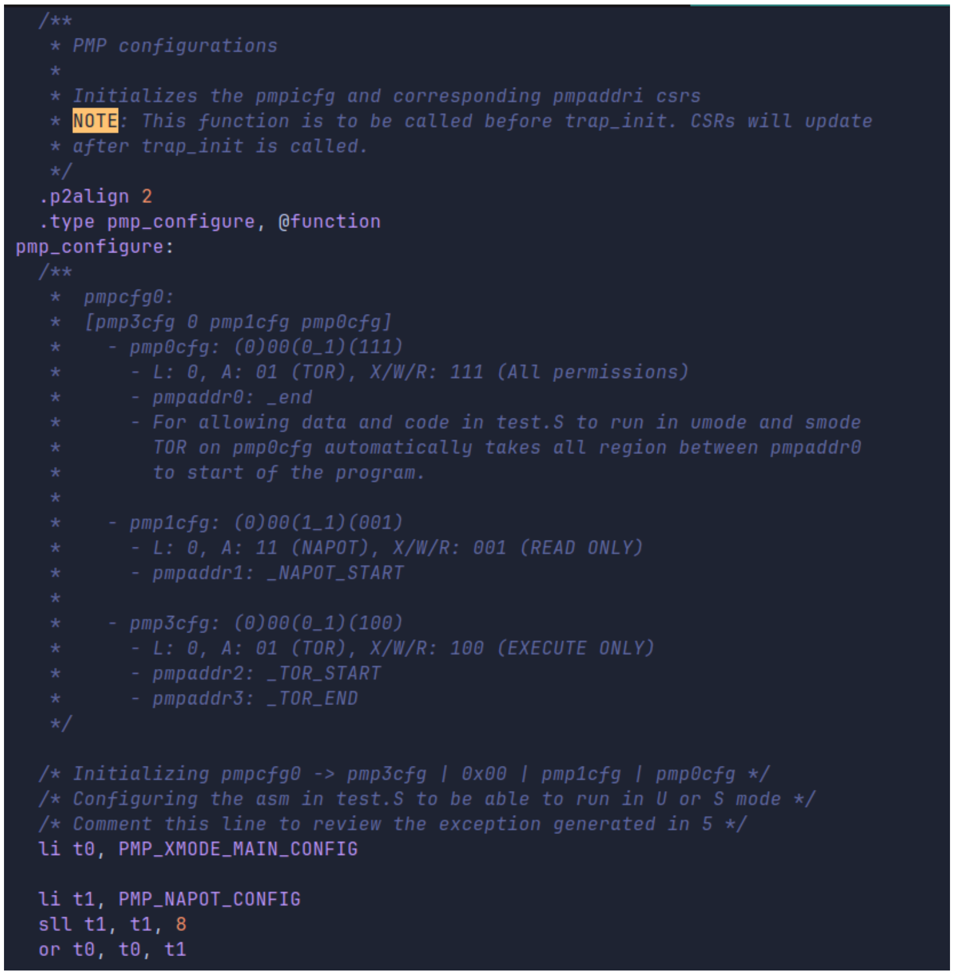
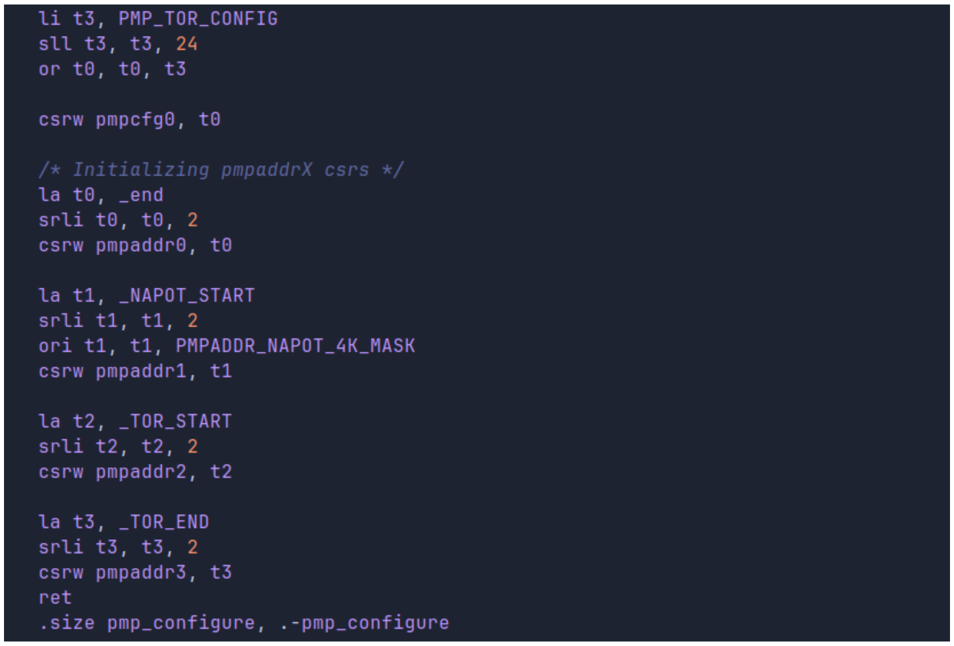
RISC-V ISA Manual: Volume II: Privilege Architecture (page 69). Mask calculation for NAPOT. As the page size if 4KB (2^12), the (12 - 3) bit is set to 0, and further bits are set to 1 (mask 01_1111_1111)

NAPOT: Naturally aligned power of two region is configured by defining the region in the linker file, then loading the _NAPOT_START address in pmpaddr1 (after shifting the address right by 2 bits and masking it with 01_1111_1111). The pmp1cfg[4:3] are set to 11.

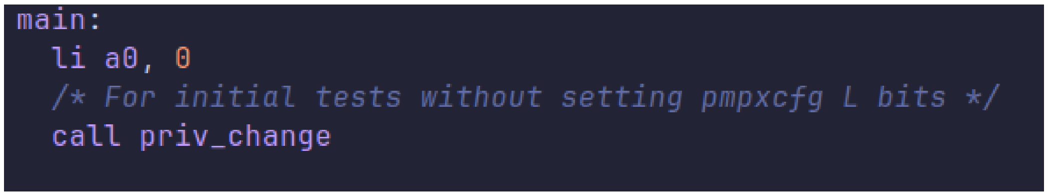
3. Grant read permission to NAPOT and execute permission to TOR region

For the NAPOT section, we assign the value 0001_1001 in pmpcfg1. The last 3 bits (001) configures the NAPOT region to have ONLY read access.

Similarly for the TOR section, we assign the value 0000_1100 in pmpcfg3. The last 3 bits (100) configure the TOR region to have ONLY execute access.

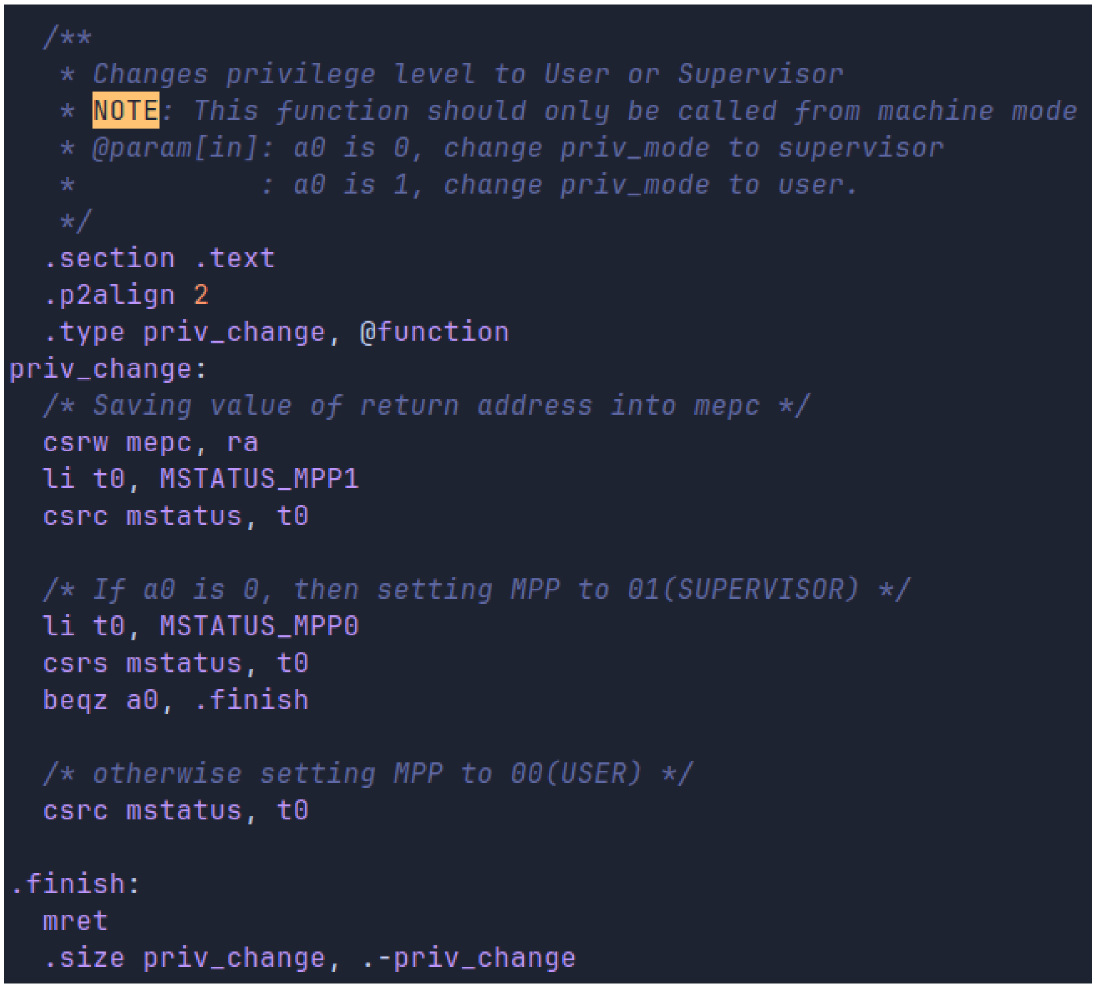

```
  /**
   * PMP configurations
   *
   * Initializes the pmpicfg and corresponding pmpaddri csrs
   * NOTE: This function is to be called before trap_init. CSRs will update
   * after trap_init is called.
   */
  .p2align 2
  .type pmp_configure, @function
pmp_configure:
  /**
   *  pmpcfg0:
   *  [pmp3cfg 0 pmp1cfg pmp0cfg]
   *    - pmp0cfg: (0)00(0_1)(111)
   *      - L: 0, A: 01 (TOR), X/W/R: 111 (All permissions)
   *      - pmpaddr0: _end
   *      - For allowing data and code in test.S to run in umode and smode
   *        TOR on pmp0cfg automatically takes all region between pmpaddr0
   *        to start of the program.
   *
   *    - pmp1cfg: (0)00(1_1)(001)
   *      - L: 0, A: 11 (NAPOT), X/W/R: 001 (READ ONLY)
   *      - pmpaddr1: _NAPOT_START
   *
   *    - pmp3cfg: (0)00(0_1)(100)
   *      - L: 0, A: 01 (TOR), X/W/R: 100 (EXECUTE ONLY)
   *      - pmpaddr2: _TOR_START
   *      - pmpaddr3: _TOR_END
   */

  /* Initializing pmpcfg0 -> pmp3cfg | 0x00 | pmp1cfg | pmp0cfg */
  /* Configuring the asm in test.S to be able to run in U or S mode */
  /* Comment this line to review the exception generated in 5 */
  li t0, PMP_XMODE_MAIN_CONFIG

  li t1, PMP_NAPOT_CONFIG
  sll t1, t1, 8
  or t0, t0, t1
```

```
  li t3, PMP_TOR_CONFIG
  sll t3, t3, 24
  or t0, t0, t3

  csrw pmpcfg0, t0

  /* Initializing pmpaddrX csrs */
  la t0, _end
  srli t0, t0, 2
  csrw pmpaddr0, t0

  la t1, _NAPOT_START
  srli t1, t1, 2
  ori t1, t1, PMPADDR_NAPOT_4K_MASK
  csrw pmpaddr1, t1

  la t2, _TOR_START
  srli t2, t2, 2
  csrw pmpaddr2, t2

  la t3, _TOR_END
  srli t3, t3, 2
  csrw pmpaddr3, t3
  ret
  .size pmp_configure, .-pmp_configure
```
init.S: lines 12-75. Configuring the TOR and NAPOT regions and their respective CSRs.

4. **Jump to S mode.**



```
main:
  li a0, 0
  /* For initial tests without setting pmpxcfg L bits */
  call priv_change
```

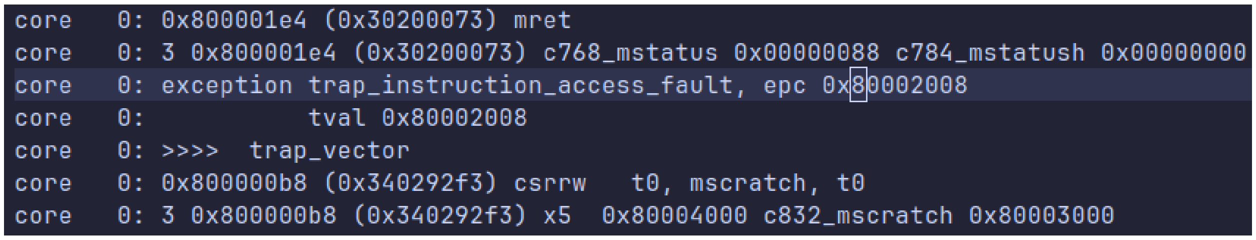
test.S: lines 7-10.



```
  /**
   * Changes privilege level to User or Supervisor
   * NOTE: This function should only be called from machine mode
   * @param[in]: a0 is 0, change priv_mode to supervisor
   *           : a0 is 1, change priv_mode to user.
   */
  .section .text
  .p2align 2
  .type priv_change, @function
priv_change:
  /* Saving value of return address into mepc */
  csrw mepc, ra
  li t0, MSTATUS_MPP1
  csrc mstatus, t0

  /* If a0 is 0, then setting MPP to 01(SUPERVISOR) */
  li t0, MSTATUS_MPP0
  csrs mstatus, t0
  beqz a0, .finish

  /* otherwise setting MPP to 00(USER) */
  csrc mstatus, t0

.finish:
  mret
  .size priv_change, .-priv_change
```

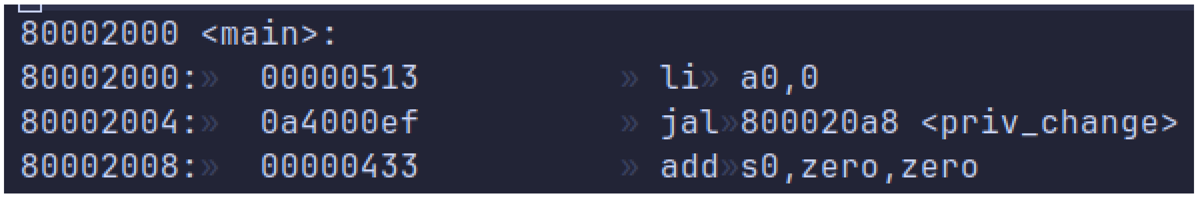
test.S: lines 87-112. Priv_change function used to change from Machine to Supervisor mode.

5. **You will get an Instruction access fault exception after entering into lower modes. Why is that? (Try to resolve it yourself)**



```
core   0: 0x800001e4 (0x30200073) mret
core   0: 3 0x800001e4 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
core   0: exception trap_instruction_access_fault, epc 0x80002008
core   0:           tval 0x80002008
core   0: >>>>  trap_vector
core   0: 0x800000b8 (0x340292f3) csrrw   t0, mscratch, t0
core   0: 3 0x800000b8 (0x340292f3) x5  0x80004000 c832_mscratch 0x80003000
```

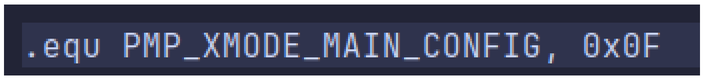
spike.log: A instruction access fault is raised after mret is executed (after we have downgraded from Machine to Supervisor). The exception is due to the assembly in test.S not being in any valid defined PMP region.



```
80002000 <main>:
80002000:»  00000513          » li» a0,0
80002004:»  0a4000ef          » jal»800020a8 <priv_change>
80002008:»  00000433          » add»s0,zero,zero
```

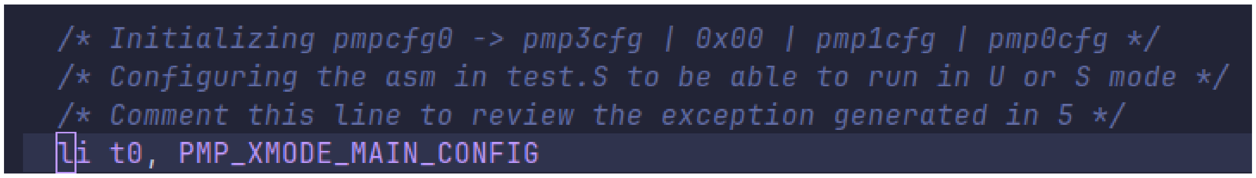
test.disasm: lines 325-328. Instruction at address 80002008 gives an instruction access fault. Because we have switched to a lower privilege mode and the region as disabled (00) by default, S & U mode cannot execute the instructions.

FIX:



```
.equ PMP_XMODE_MAIN_CONFIG, 0x0F
```

macro.S: line 6

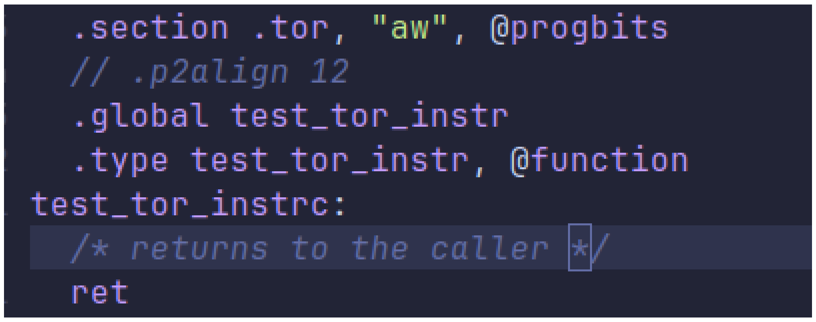


```
  /* Initializing pmpcfg0 -> pmp3cfg | 0x00 | pmp1cfg | pmp0cfg */
  /* Configuring the asm in test.S to be able to run in U or S mode */
  /* Comment this line to review the exception generated in 5 */
  li t0, PMP_XMODE_MAIN_CONFIG
```

init.S: lines 42-45. We configure another PMP region (using pmp0cfg). This region is defined as TOR and has read, write and access privileges. The pmpaddr0 is set to _end (which means all sections other than NATOP and TOR have R/W/X permissions). As TOR is used in pmpaddr0, user and supervisor mode can access sections having address less than _end.

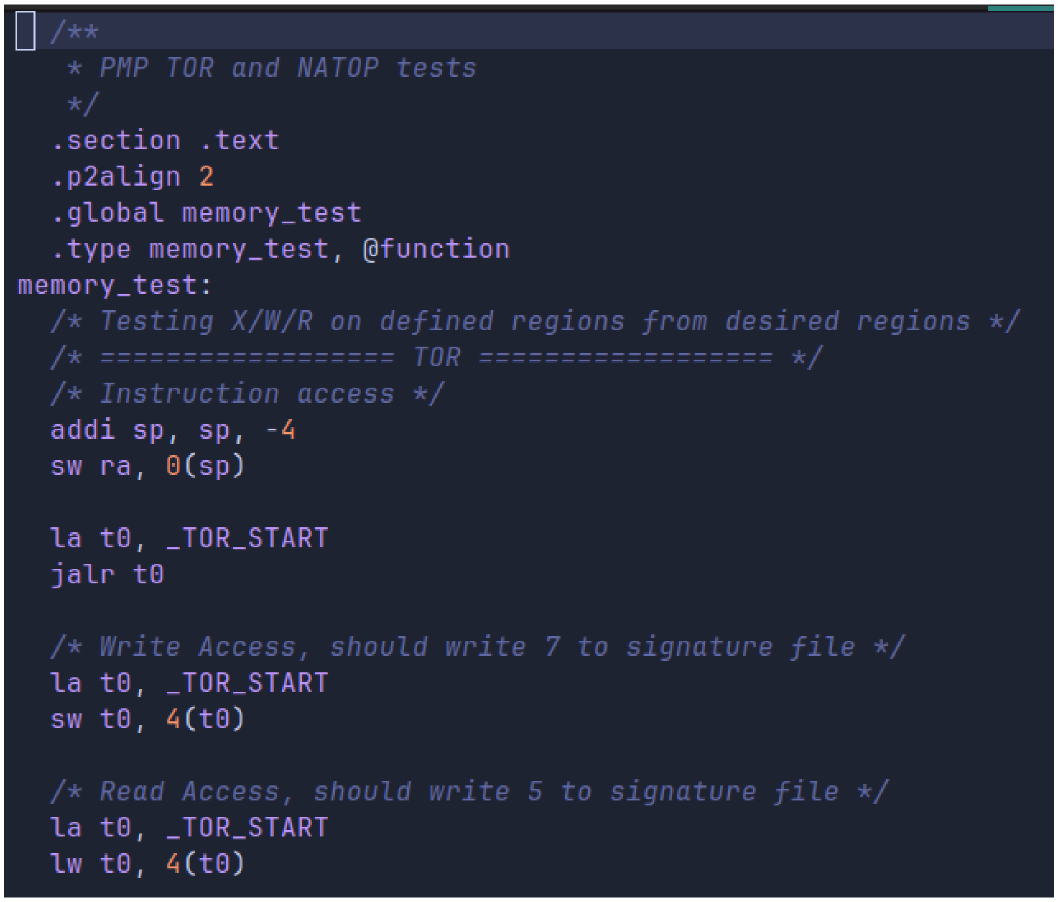
6. **Now (after resolving v), do read, write and execute from both the pmp region. You should get appropriate Access faults.**

TOR Section:



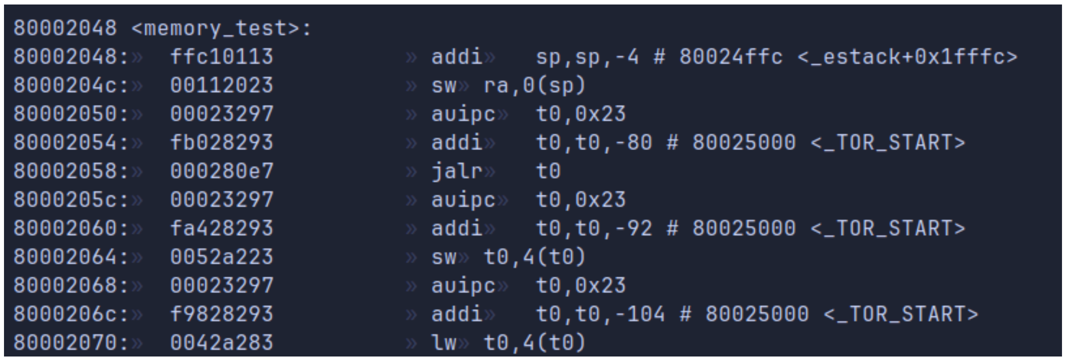
```
   .section .tor, "aw", @progbits
   // .p2align 12
   .global test_tor_instr
   .type test_tor_instr, @function
 test_tor_instrc:
   /* returns to the caller */
   ret
```

pmp_section.S: lines 1-6



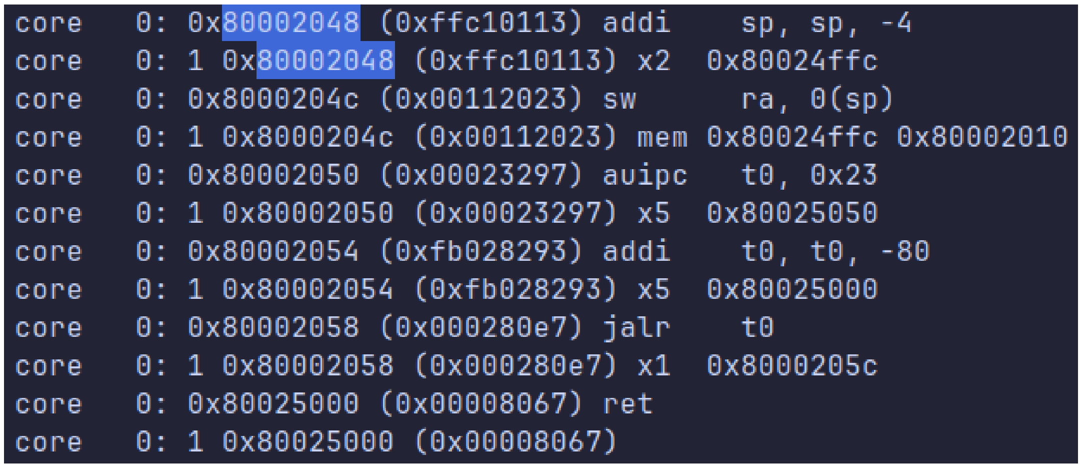
```
  /**
   * PMP TOR and NATOP tests
   */
  .section .text
  .p2align 2
  .global memory_test
  .type memory_test, @function
memory_test:
  /* Testing X/W/R on defined regions from desired regions */
  /* ================== TOR ================== */
  /* Instruction access */
  addi sp, sp, -4
  sw ra, 0(sp)

  la t0, _TOR_START
  jalr t0

  /* Write Access, should write 7 to signature file */
  la t0, _TOR_START
  sw t0, 4(t0)

  /* Read Access, should write 5 to signature file */
  la t0, _TOR_START
  lw t0, 4(t0)
```

test.S: lines 45-68. Testing the TOR section.



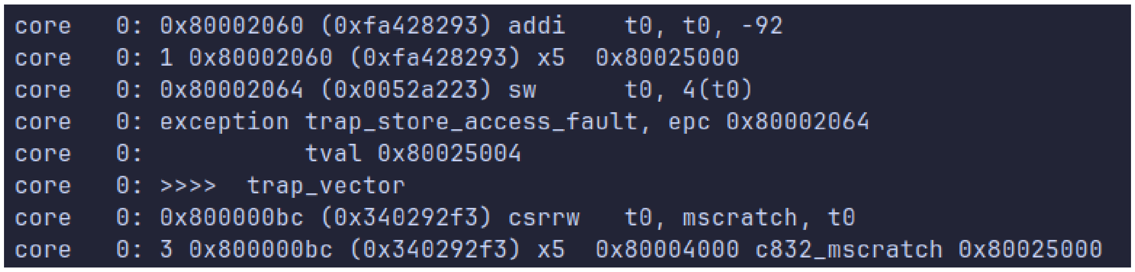
```
80002048 <memory_test>:
80002048:»  ffc10113          » addi»   sp,sp,-4 # 80024ffc <_estack+0x1fffc>
8000204c:»  00112023          » sw» ra,0(sp)
80002050:»  00023297          » auipc»  t0,0x23
80002054:»  fb028293          » addi»   t0,t0,-80 # 80025000 <_TOR_START>
80002058:»  000280e7          » jalr»   t0
8000205c:»  00023297          » auipc»  t0,0x23
80002060:»  fa428293          » addi»   t0,t0,-92 # 80025000 <_TOR_START>
80002064:»  0052a223          » sw» t0,4(t0)
80002068:»  00023297          » auipc»  t0,0x23
8000206c:»  f9828293          » addi»   t0,t0,-104 # 80025000 <_TOR_START>
80002070:»  0042a283          » lw» t0,4(t0)
```

test.disass: lines 363-374. Disassembly of memory test for TOR section.



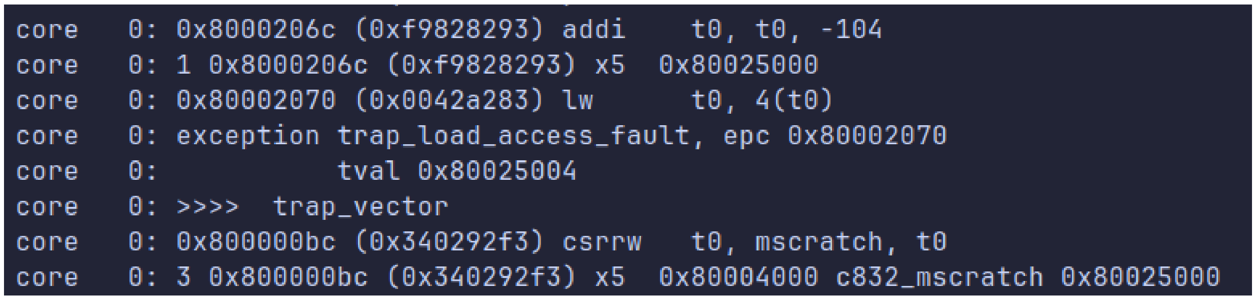
```
core   0: 0x80002048 (0xffc10113) addi    sp, sp, -4
core   0: 1 0x80002048 (0xffc10113) x2  0x80024ffc
core   0: 0x8000204c (0x00112023) sw      ra, 0(sp)
core   0: 1 0x8000204c (0x00112023) mem 0x80024ffc 0x80002010
core   0: 0x80002050 (0x00023297) auipc   t0, 0x23
core   0: 1 0x80002050 (0x00023297) x5  0x80025050
core   0: 0x80002054 (0xfb028293) addi    t0, t0, -80
core   0: 1 0x80002054 (0xfb028293) x5  0x80025000
core   0: 0x80002058 (0x000280e7) jalr    t0
core   0: 1 0x80002058 (0x000280e7) x1  0x8000205c
core   0: 0x80025000 (0x00008067) ret
core   0: 1 0x80025000 (0x00008067)
```

spike.log: lines 130-141. Successfully able to execute the `ret` instruction in the TOR section.



```
core   0: 0x80002060 (0xfa428293) addi    t0, t0, -92
core   0: 1 0x80002060 (0xfa428293) x5  0x80025000
core   0: 0x80002064 (0x0052a223) sw      t0, 4(t0)
core   0: exception trap_store_access_fault, epc 0x80002064
core   0:           tval 0x80025004
core   0: >>>>  trap_vector
core   0: 0x800000bc (0x340292f3) csrrw   t0, mscratch, t0
core   0: 3 0x800000bc (0x340292f3) x5  0x80004000 c832_mscratch 0x80025000
```

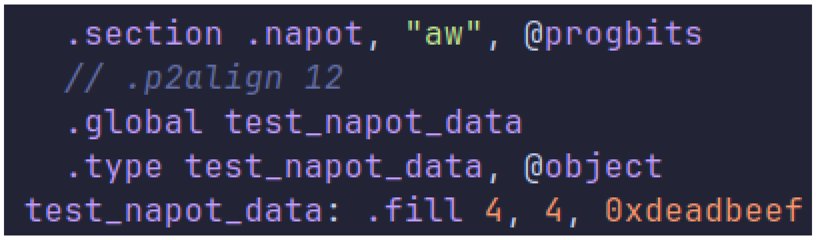
spike.log: lines 144-151. When trying to save words in the TOR region, we get store_access_fault.



```
core   0: 0x8000206c (0xf9828293) addi    t0, t0, -104
core   0: 1 0x8000206c (0xf9828293) x5  0x80025000
core   0: 0x80002070 (0x0042a283) lw      t0, 4(t0)
core   0: exception trap_load_access_fault, epc 0x80002070
core   0:           tval 0x80025004
core   0: >>>>  trap_vector
core   0: 0x800000bc (0x340292f3) csrrw   t0, mscratch, t0
core   0: 3 0x800000bc (0x340292f3) x5  0x80004000 c832_mscratch 0x80025000
```

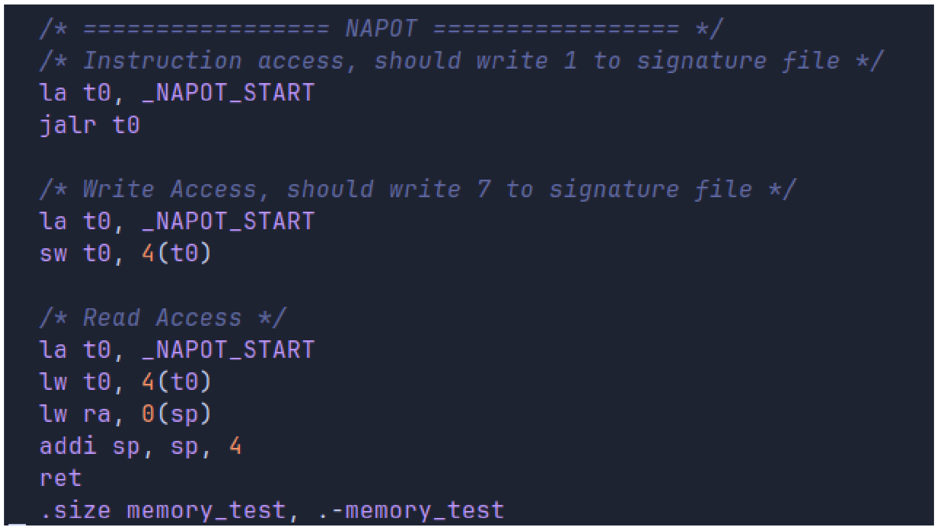
spike.log: lines 330-337. When trying to load a word in the TOR region, we get load_access_fault.

NAPOT Section:



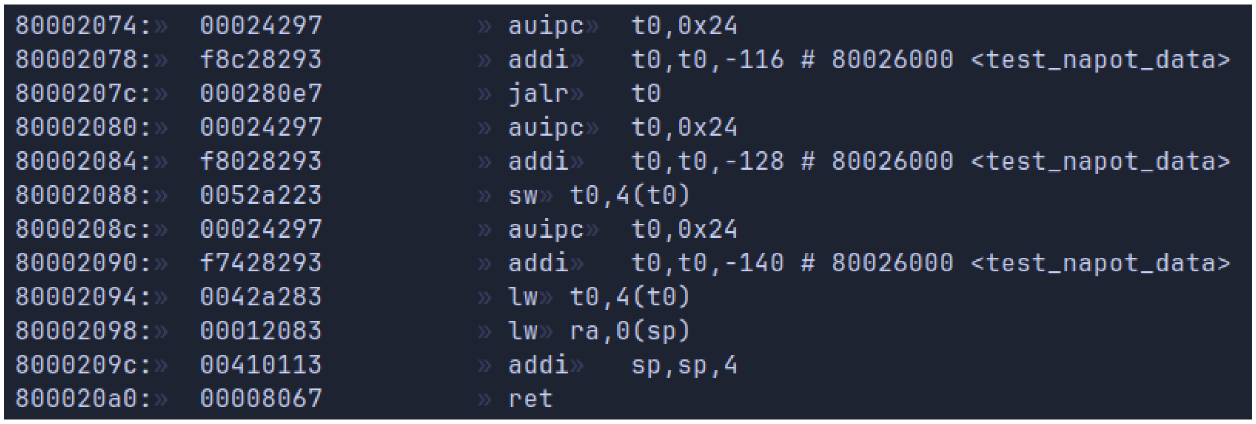
```
   .section .napot, "aw", @progbits
   // .p2align 12
   .global test_napot_data
   .type test_napot_data, @object
test_napot_data: .fill 4, 4, 0xdeadbeef
```

pmp_section.S: lines 9-13



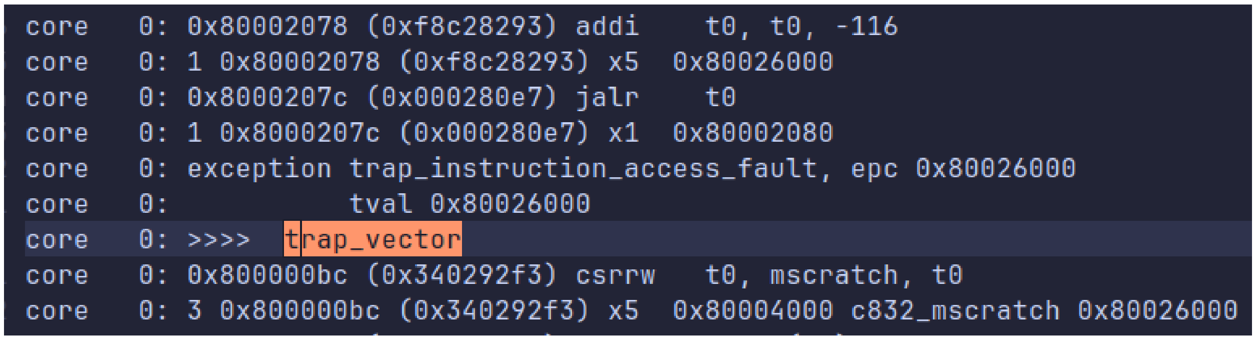
```
/* ================ NAPOT ================ */
/* Instruction access, should write 1 to signature file */
la t0, _NAPOT_START
jalr t0

/* Write Access, should write 7 to signature file */
la t0, _NAPOT_START
sw t0, 4(t0)

/* Read Access */
la t0, _NAPOT_START
lw t0, 4(t0)
lw ra, 0(sp)
addi sp, sp, 4
ret
.size memory_test, .-memory_test
```

test.S: lines 70-85. Testing the TOR section.



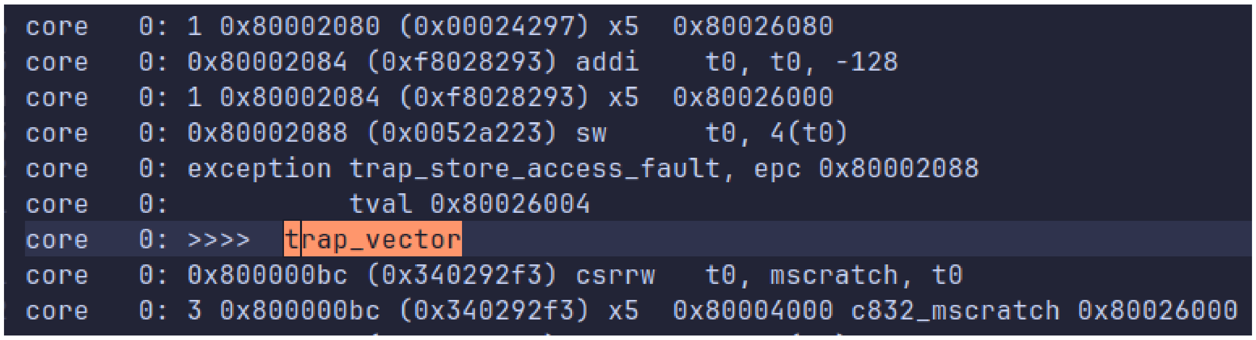
```
80002074:»  00024297          » auipc»  t0,0x24
80002078:»  f8c28293          » addi»   t0,t0,-116 # 80026000 <test_napot_data>
8000207c:»  000280e7          » jalr»   t0
80002080:»  00024297          » auipc»  t0,0x24
80002084:»  f8028293          » addi»   t0,t0,-128 # 80026000 <test_napot_data>
80002088:»  0052a223          » sw» t0,4(t0)
8000208c:»  00024297          » auipc»  t0,0x24
80002090:»  f7428293          » addi»   t0,t0,-140 # 80026000 <test_napot_data>
80002094:»  0042a283          » lw» t0,4(t0)
80002098:»  00012083          » lw» ra,0(sp)
8000209c:»  00410113          » addi»   sp,sp,4
800020a0:»  00008067          » ret
```

test.diass: lines 375-386. Disassembly of memory test for NAPOT section.



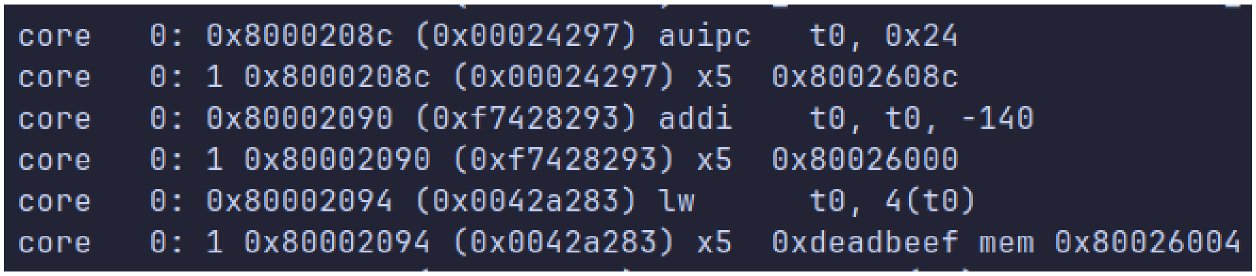
```
core   0: 0x80002078 (0xf8c28293) addi    t0, t0, -116
core   0: 1 0x80002078 (0xf8c28293) x5  0x80026000
core   0: 0x8000207c (0x000280e7) jalr    t0
core   0: 1 0x8000207c (0x000280e7) x1  0x80002080
core   0: exception trap_instruction_access_fault, epc 0x80026000
core   0:           tval 0x80026000
core   0: >>>>  trap_vector
core   0: 0x800000bc (0x340292f3) csrrw   t0, mscratch, t0
core   0: 3 0x800000bc (0x340292f3) x5  0x80004000 c832_mscratch 0x80026000
```

spike.log: lines 516-524. Instruction Access fault when trying to execute instruction in NATOP region.



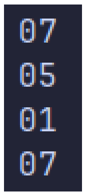
```
core   0: 1 0x80002080 (0x00024297) x5  0x80026080
core   0: 0x80002084 (0xf8028293) addi    t0, t0, -128
core   0: 1 0x80002084 (0xf8028293) x5  0x80026000
core   0: 0x80002088 (0x0052a223) sw      t0, 4(t0)
core   0: exception trap_store_access_fault, epc 0x80002088
core   0:           tval 0x80026004
core   0: >>>>  trap_vector
core   0: 0x800000bc (0x340292f3) csrrw   t0, mscratch, t0
core   0: 3 0x800000bc (0x340292f3) x5  0x80004000 c832_mscratch 0x80026000
```

spike.log: lines 713-720. Store Access fault when trying to store variables in the NATOP region.



```
core   0: 0x8000208c (0x00024297) auipc   t0, 0x24
core   0: 1 0x8000208c (0x00024297) x5  0x8002608c
core   0: 0x80002090 (0xf7428293) addi    t0, t0, -140
core   0: 1 0x80002090 (0xf7428293) x5  0x80026000
core   0: 0x80002094 (0x0042a283) lw      t0, 4(t0)
core   0: 1 0x80002094 (0x0042a283) x5  0xdeadbeef mem 0x80026004
```

spike.log: lines 897-902. Successfully able to load data from the NATOP region.

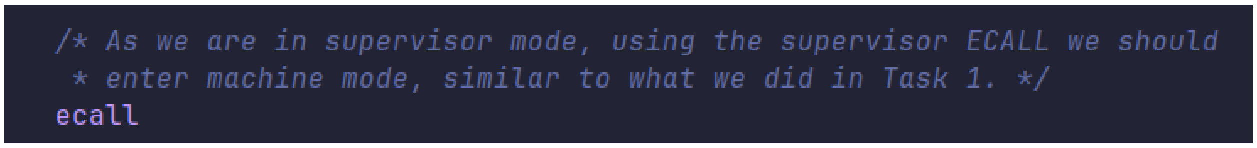


```
07
05
01
07
```

signature.txt: lines 1-4. The first two faults are for the TOR section. Mcause 07 and 05 (corresponding to store and load access respectively) are written in the signature file by the trap handler. This means that from X/W/R, only X was successful.

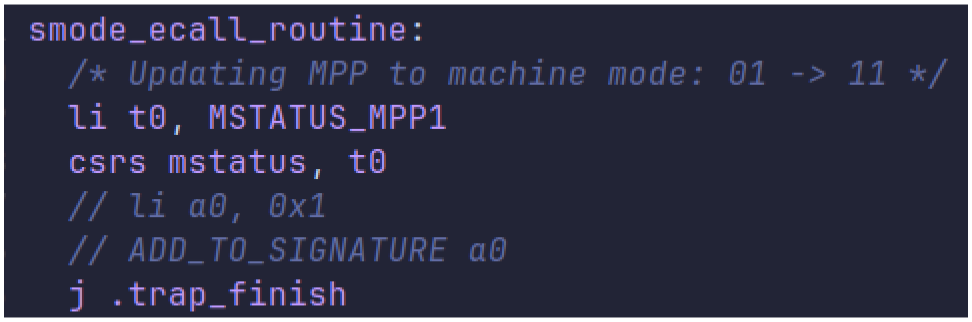
The later two faults are for NATOP section. Mcause 01 and 07 (corresponding to instruction and load access fault respectively) are written in the signature file by the trap handler. This means that from the X/W/R, only R was successful.

7. **Jump to machine mode and now apply above (point iii) mentioned pmp restrictions for machine mode too. What do you need to do for this ?**



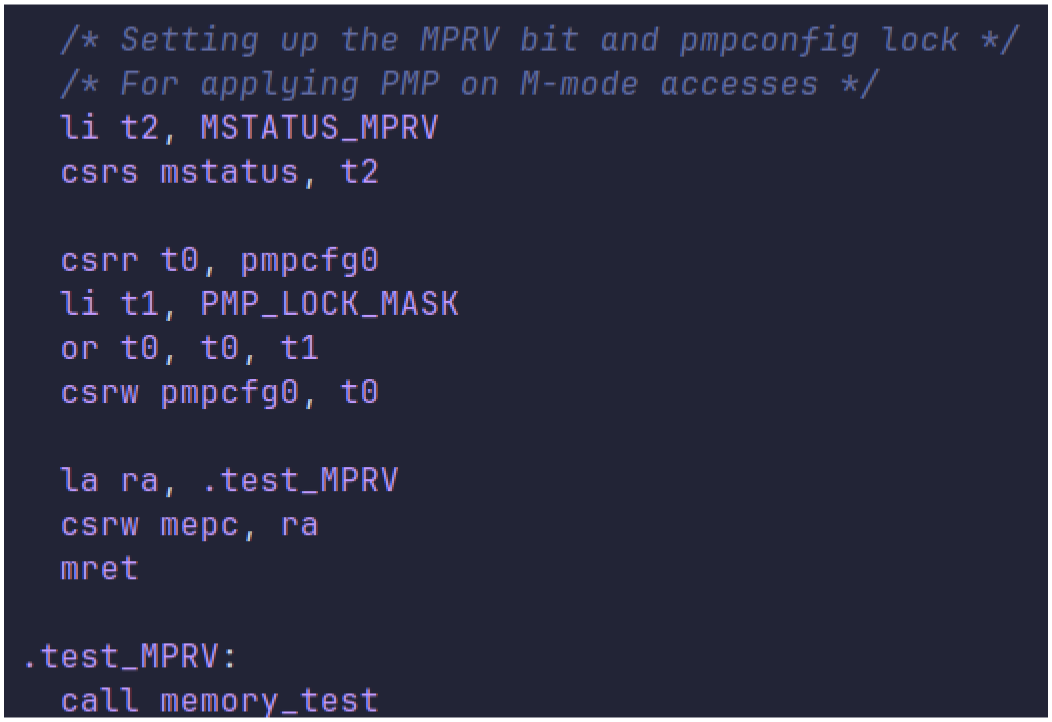
```
/* As we are in supervisor mode, using the supervisor ECALL we should
 * enter machine mode, similar to what we did in Task 1. */
ecall
```

test.S: line 27



```
smode_ecall_routine:
  /* Updating MPP to machine mode: 01 -> 11 */
  li t0, MSTATUS_MPP1
  csrs mstatus, t0
  // li a0, 0x1
  // ADD_TO_SIGNATURE a0
  j .trap_finish
```

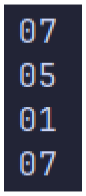
init.S: line 213. Going from supervisor mode to machine mode.



```
  /* Setting up the MPRV bit and pmpconfig lock */
  /* For applying PMP on M-mode accesses */
  li t2, MSTATUS_MPRV
  csrs mstatus, t2

  csrr t0, pmpcfg0
  li t1, PMP_LOCK_MASK
  or t0, t0, t1
  csrw pmpcfg0, t0

  la ra, .test_MPRV
  csrw mepc, ra
  mret

.test_MPRV:
  call memory_test
```

test.S: line 25-40. Setting the MPRV bit in mstatus, and then setting the lock bit in pmp0cfg, pmp1cfg, pmp2cfg & pmp3cfg. This allows us to enable pmp restrictions on machine mode accesses.

8. **Now again read, write and execute from both the pmp region. You should get appropriate Access faults.**



```
07
05
01
07
```

signature.txt: lines 5-8. The same access faults that were observed in S mode are observed in M mode.