Author : Irtaza Hyder

Date : 2026-02-14

Module : RISC-V Arch Test

Section : RISC-V Arch Test

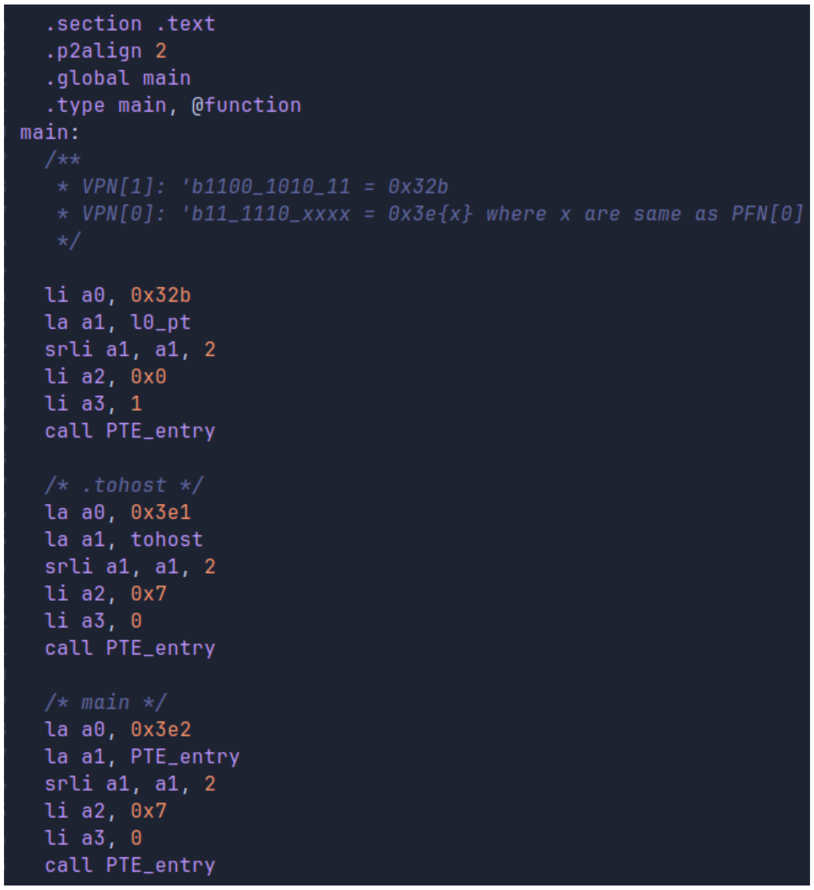
Task Name : Task 5

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

Test Description

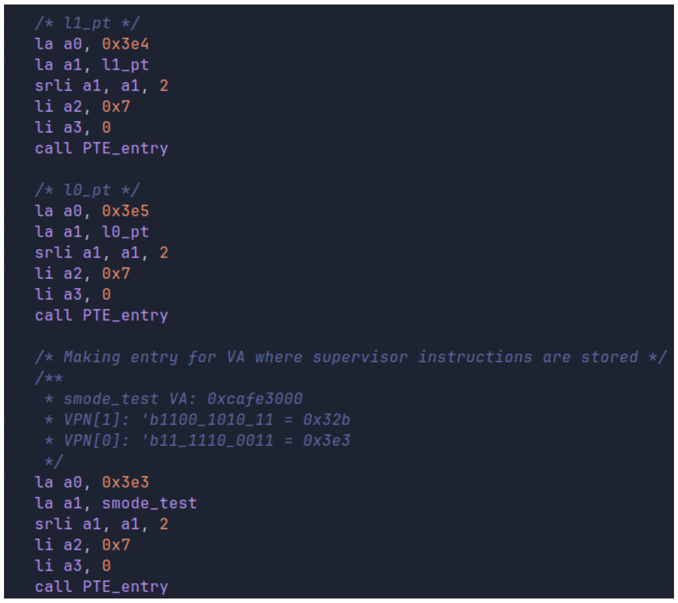
Setup

Before we run the test, we first set privilege to machine mode. Then we create PTE's in L1 and L0 pages. All of the L1 PTE's point to the same L0 page table. The physical address of different sections are stored in L0 pages as shown below:

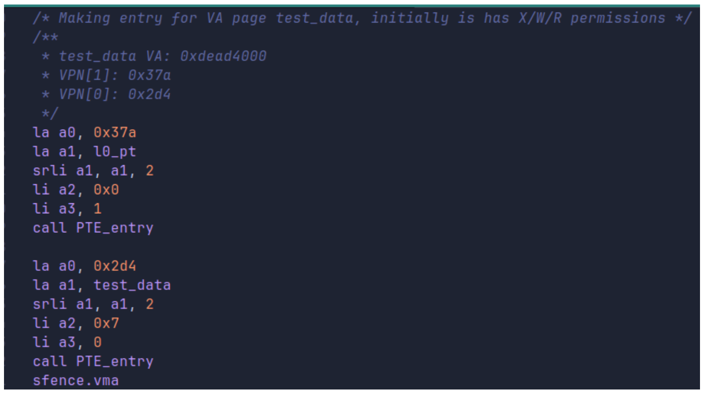


```
  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  /**
   * VPN[1]: 'b1100_1010_11 = 0x32b
   * VPN[0]: 'b11_1110_xxxx = 0x3e{x} where x are same as PFN[0]
   */

  li a0, 0x32b
  la a1, l0_pt
  srli a1, a1, 2
  li a2, 0x0
  li a3, 1
  call PTE_entry

  /* .tohost */
  la a0, 0x3e1
  la a1, tohost
  srli a1, a1, 2
  li a2, 0x7
  li a3, 0
  call PTE_entry

  /* main */
  la a0, 0x3e2
  la a1, PTE_entry
  srli a1, a1, 2
  li a2, 0x7
  li a3, 0
  call PTE_entry
```

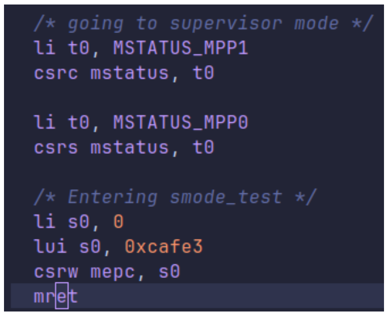
Setting PTE for .tohost section and main. The tohost section contains the signature file, which we read while in supervisor mode, so we need to assign it a virtual address. Similarly, as we update the PTE enteries in supervisor mode, we require the PTE_entry functions VA as well.


```
/* l1_pt */
la a0, 0x3e4
la a1, l1_pt
srli a1, a1, 2
li a2, 0x7
li a3, 0
call PTE_entry

/* l0_pt */
la a0, 0x3e5
la a1, l0_pt
srli a1, a1, 2
li a2, 0x7
li a3, 0
call PTE_entry

/* Making entry for VA where supervisor instructions are stored */
/**
 * smode_test VA: 0xcafe3000
 * VPN[1]: 'b1100_1010_11 = 0x32b
 * VPN[0]: 'b11_1110_0011 = 0x3e3
 */
la a0, 0x3e3
la a1, smode_test
srli a1, a1, 2
li a2, 0x7
li a3, 0
call PTE_entry
```

Setting PTE for l1_pt and l0_pt. These are altered by the PTE_entry function, and the l0_pt is updated in supervisor mode (requiring us to provide it with a virtual address). Setting smode_test function's VA. This is the location for all our supervisor mode tests.


```
/* Making entry for VA page test_data, initially is has X/W/R permissions */
/**
 * test_data VA: 0xdead4000
 * VPN[1]: 0x37a
 * VPN[0]: 0x2d4
 */
la a0, 0x37a
la a1, l0_pt
srli a1, a1, 2
li a2, 0x0
li a3, 1
call PTE_entry

la a0, 0x2d4
la a1, test_data
srli a1, a1, 2
li a2, 0x7
li a3, 0
call PTE_entry
sfence.vma
```

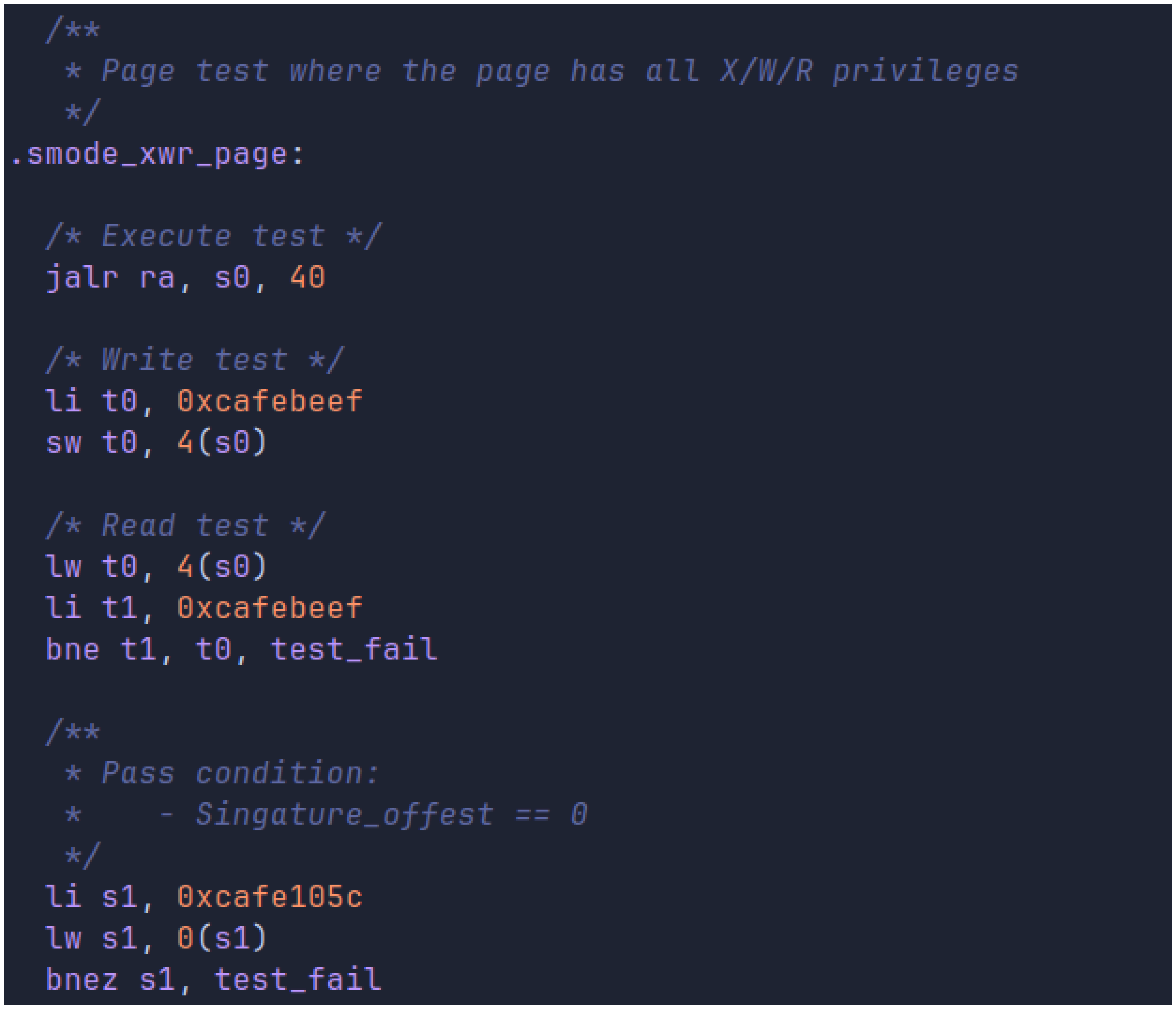
Setting the test_data section's VA. This is the section which we alter and test the X/W/R instructions in supervisor mode.


```
/* going to supervisor mode */
li t0, MSTATUS_MPP1
csrc mstatus, t0

li t0, MSTATUS_MPP0
csrs mstatus, t0

/* Entering smode_test */
li s0, 0
lui s0, 0xcafe3
csrw mepc, s0
mret
```

Entering supervisor mode.

test.S: lines: 3-97

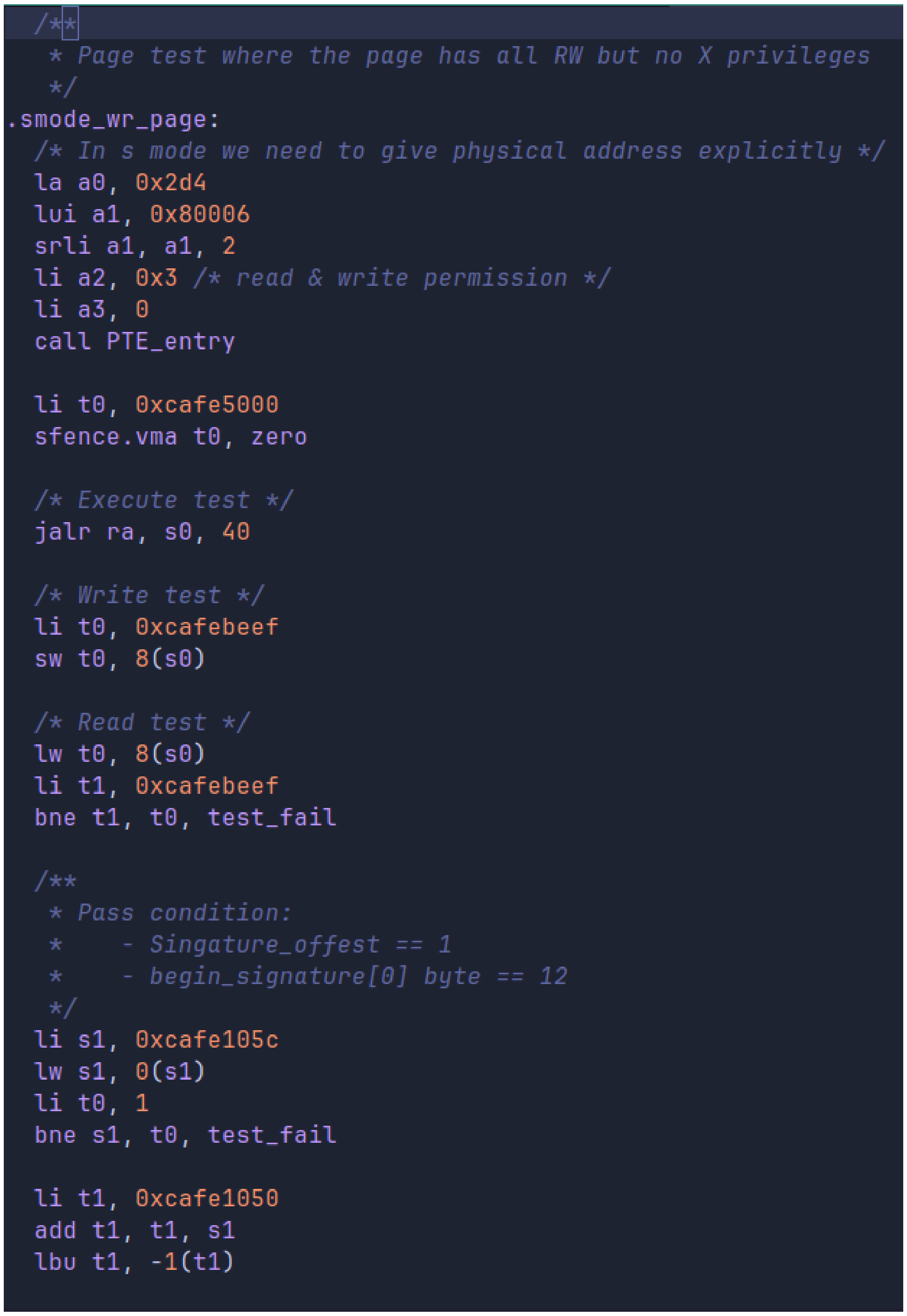
Test 1:



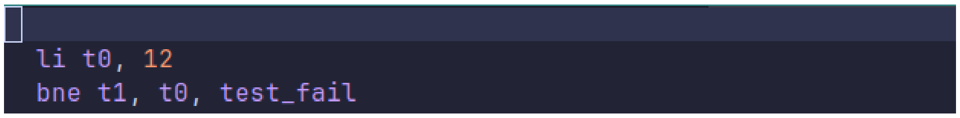
```
  /**
   * Page test where the page has all X/W/R privileges
   */
.smode_xwr_page:

  /* Execute test */
  jalr ra, s0, 40

  /* Write test */
  li t0, 0xcafebeef
  sw t0, 4(s0)

  /* Read test */
  lw t0, 4(s0)
  li t1, 0xcafebeef
  bne t1, t0, test_fail

  /**
   * Pass condition:
   *    - Singature_offest == 0
   */
  li s1, 0xcafe105c
  lw s1, 0(s1)
  bnez s1, test_fail
```

Testing page with X/W/R access. The trap handler should not be invoked. If the trap handler is invoked the singature_offset variable is incremented by 1, hence the pass condition for this tests is signature_offset == 0.

Test 2:



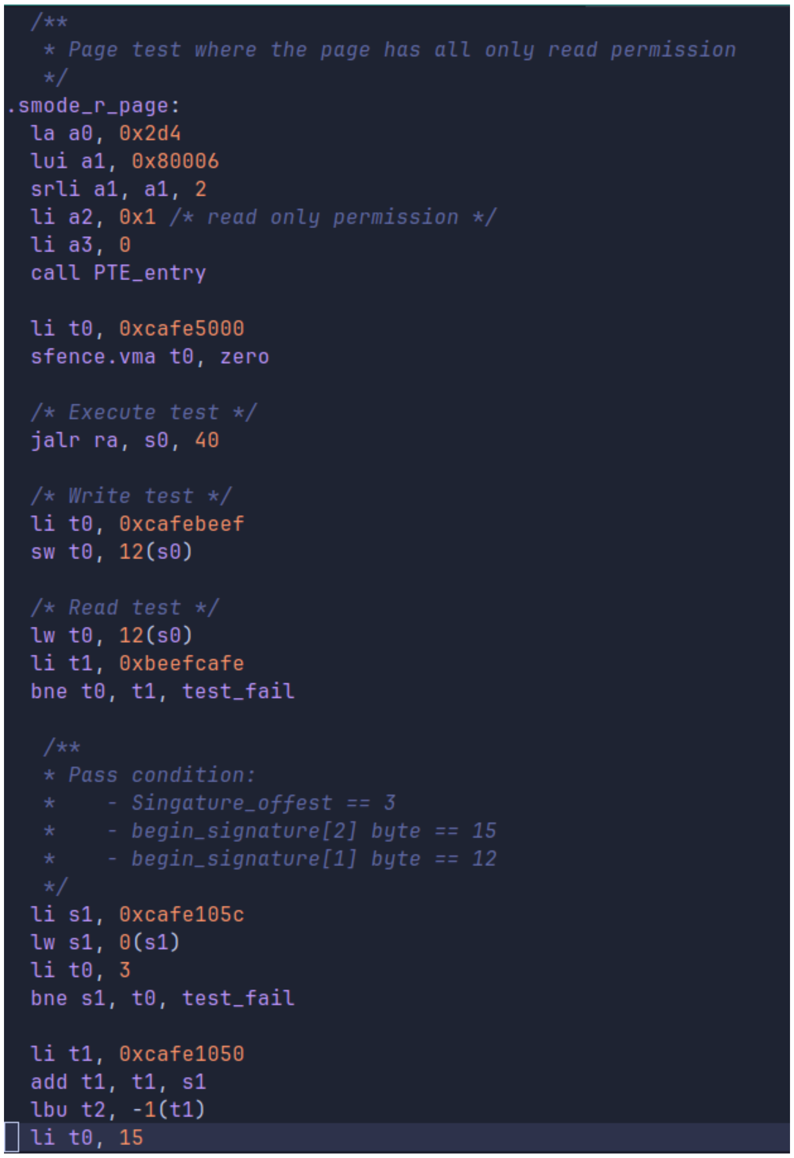
```
  /**
   * Page test where the page has all RW but no X privileges
   */
.smode_wr_page:
  /* In s mode we need to give physical address explicitly */
  la a0, 0x2d4
  lui a1, 0x80006
  srli a1, a1, 2
  li a2, 0x3 /* read & write permission */
  li a3, 0
  call PTE_entry

  li t0, 0xcafe5000
  sfence.vma t0, zero

  /* Execute test */
  jalr ra, s0, 40

  /* Write test */
  li t0, 0xcafebeef
  sw t0, 8(s0)

  /* Read test */
  lw t0, 8(s0)
  li t1, 0xcafebeef
  bne t1, t0, test_fail

  /**
   * Pass condition:
   *    - Singature_offest == 1
   *    - begin_signature[0] byte == 12
   */
  li s1, 0xcafe105c
  lw s1, 0(s1)
  li t0, 1
  bne s1, t0, test_fail

  li t1, 0xcafe1050
  add t1, t1, s1
  lbu t1, -1(t1)
```



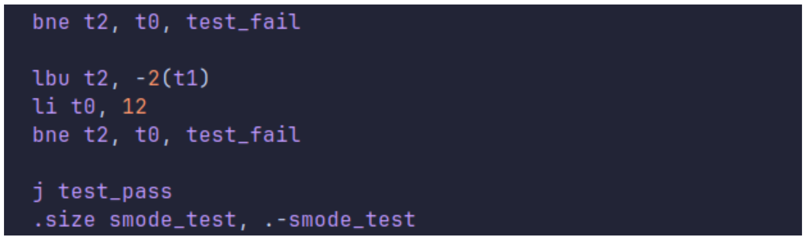
```
  li t0, 12
  bne t1, t0, test_fail
```

Updating the page (containing test_data section's PA). It now has R/W access. The trap handler would be invoked once, and would write 0x0c in signature_file (1st byte) showing a instruction_page_fault. The pass condition is: signature_offset == 1 && signature_offset[0] byte = 0x0c

Test 3:



```
  /**
   * Page test where the page has all only read permission
   */
.smode_r_page:
  la a0, 0x2d4
  lui a1, 0x80006
  srli a1, a1, 2
  li a2, 0x1 /* read only permission */
  li a3, 0
  call PTE_entry

  li t0, 0xcafe5000
  sfence.vma t0, zero

  /* Execute test */
  jalr ra, s0, 40

  /* Write test */
  li t0, 0xcafebeef
  sw t0, 12(s0)

  /* Read test */
  lw t0, 12(s0)
  li t1, 0xbeefcafe
  bne t0, t1, test_fail

   /**
   * Pass condition:
   *    - Singature_offest == 3
   *    - begin_signature[2] byte == 15
   *    - begin_signature[1] byte == 12
   */
  li s1, 0xcafe105c
  lw s1, 0(s1)
  li t0, 3
  bne s1, t0, test_fail

  li t1, 0xcafe1050
  add t1, t1, s1
  lbu t2, -1(t1)
  li t0, 15
```



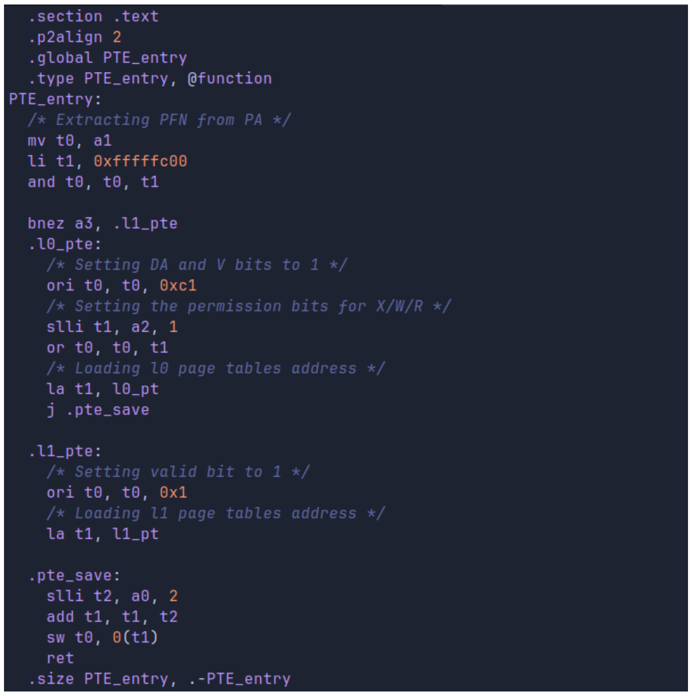
```
  bne t2, t0, test_fail

  lbu t2, -2(t1)
  li t0, 12
  bne t2, t0, test_fail

  j test_pass
  .size smode_test, .-smode_test
```

Updating the page (containing test_data section's PA). It now has ONLY R access. The trap handler would be invoked twice, and would write 0x0c (2nd bytes) and 0xf (3rd byte) respectively in signature_file. The pass condition is: signature_offset == 3 && signature_offset[1] byte = 0x0c && signature_offset[1] byte = 0x0f.

test.S: lines 99-224.

Output and Q&A

A. **Write an assembly function to setup the page table entry, Function must have following arguments:**
   - **Virtual address → a0 = Virtual_address[31:0]**
   - **Physical address → a1 = Physical address[33:0]**
   - **Permissions → a2[2:0] = X/W/R permission**
   - **level: Either 0 or 1. This sets the PTE specified by the level parameter →a3[0] = Level**



```
  .section .text
  .p2align 2
  .global PTE_entry
  .type PTE_entry, @function
PTE_entry:
  /* Extracting PFN from PA */
  mv t0, a1
  li t1, 0xfffffc00
  and t0, t0, t1

  bnez a3, .l1_pte
  .l0_pte:
    /* Setting DA and V bits to 1 */
    ori t0, t0, 0xc1
    /* Setting the permission bits for X/W/R */
    slli t1, a2, 1
    or t0, t0, t1
    /* Loading l0 page tables address */
    la t1, l0_pt
    j .pte_save

  .l1_pte:
    /* Setting valid bit to 1 */
    ori t0, t0, 0x1
    /* Loading l1 page tables address */
    la t1, l1_pt

  .pte_save:
    slli t2, a0, 2
    add t1, t1, t2
    sw t0, 0(t1)
    ret
  .size PTE_entry, .-PTE_entry
```

test.S: lines 248-293. The PTE_entry function. It adds entries to the L1 and L0 page tables. It can be invoked in both machine and supervisor mode. Note: L1 PTEs have only their valid bits set. Whereas L0 PTEs have their valid bits, permission bits given by user, and D & A bits set.

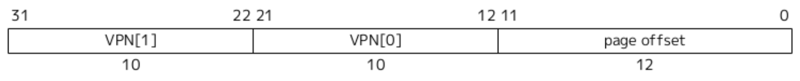


Figure 65. Sv32 virtual address.

Sv32 page tables consist of 2^{10} page-table entries (PTEs), each of four bytes. A page table is exactly the size of a page and must always be aligned to a page boundary. The physical page number of the root page table is stored in the **satp** register.

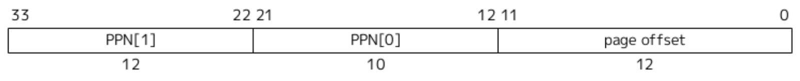


Figure 66. SV32 physical address.

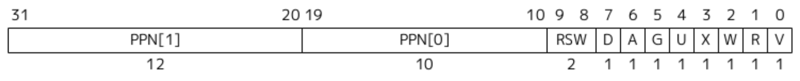


Figure 67. Sv32 page table entry.

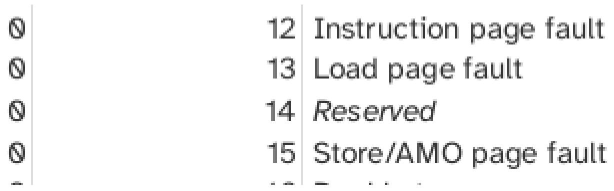
RISC-V ISA, Volume II: Privilege Architecture, page 131. The figure shows the VA, PA, and PTE formats.

B. **Write assembly test:**

   a. **Start in M mode. Setup the PTEs for Instruction memory address (or DMem: load/store addresses).**

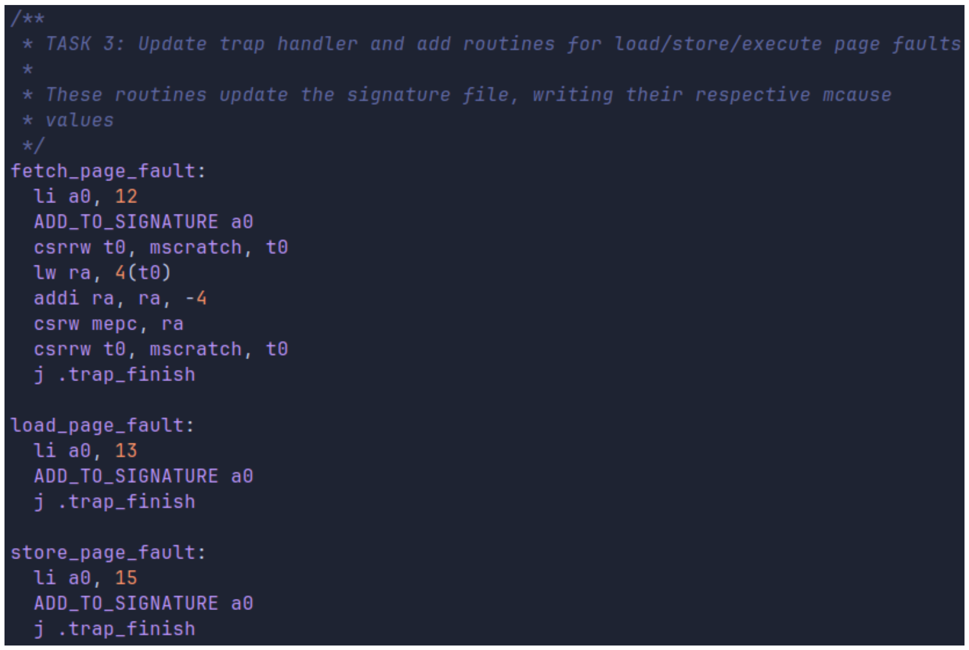
Kindly review the **Test Description** section for screenshots.

   b. **Setup your trap-handler to handle the exception appropriately.**

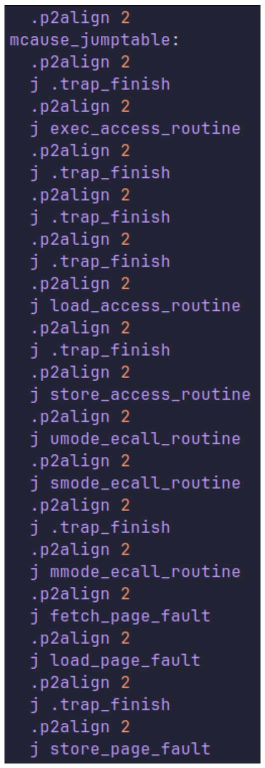


0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault

RISC-V ISA, Volume II: Privilege Architecture, Table 16 (page 60). The figure shows mcause values for page fault related exceptions.



```
/**
 * TASK 3: Update trap handler and add routines for load/store/execute page faults
 *
 * These routines update the signature file, writing their respective mcause
 * values
 */
fetch_page_fault:
  li a0, 12
  ADD_TO_SIGNATURE a0
  csrrw t0, mscratch, t0
  lw ra, 4(t0)
  addi ra, ra, -4
  csrw mepc, ra
  csrrw t0, mscratch, t0
  j .trap_finish

load_page_fault:
  li a0, 13
  ADD_TO_SIGNATURE a0
  j .trap_finish

store_page_fault:
  li a0, 15
  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
  .p2align 2
  j fetch_page_fault
  .p2align 2
  j load_page_fault
  .p2align 2
  j .trap_finish
  .p2align 2
  j store_page_fault
```

left) fetch/load/store page_fault routines right) Updates to jumptable to handle page fault exceptions (mcause: instruction/fetch = 12, load = 13 and store = 15).

c. Set satp appropriately to enable virtualization with SV32 translation scheme.

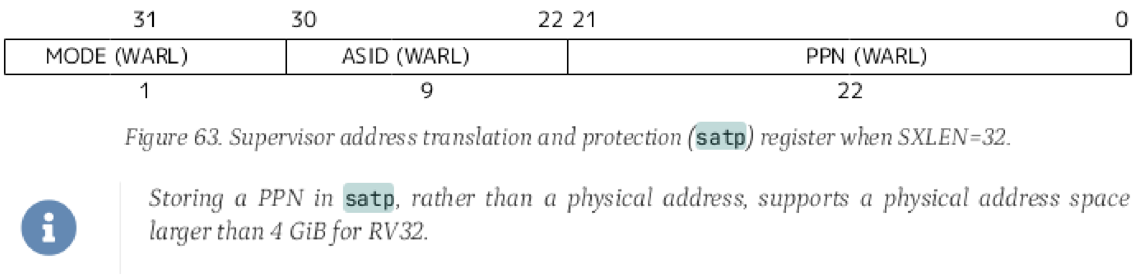


Figure 63. Supervisor address translation and protection (satp) register when SXLEN=32.

Storing a PPN in satp, rather than a physical address, supports a physical address space larger than 4 GiB for RV32.

RISC-V ISA, Volume II: Privilege Architecture, Page 124

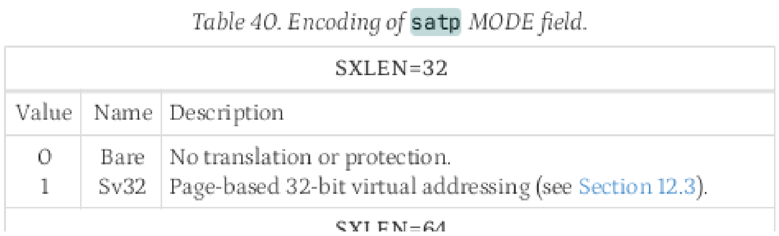
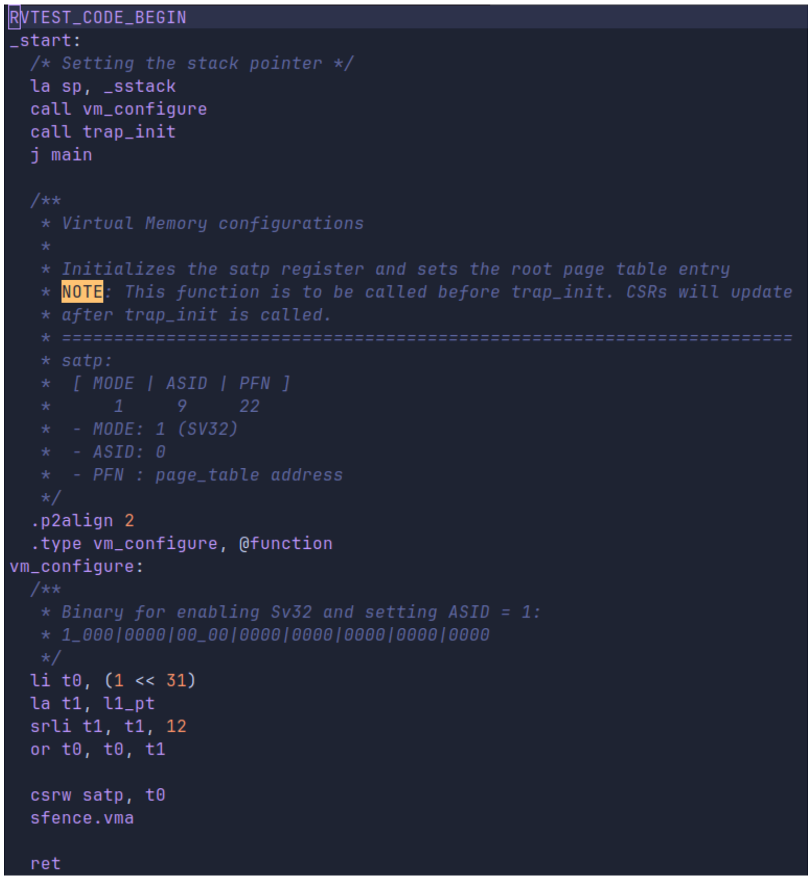


Table 40. Encoding of satp MODE field.

SXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing (see Section 12.3).
SXLEN=64		

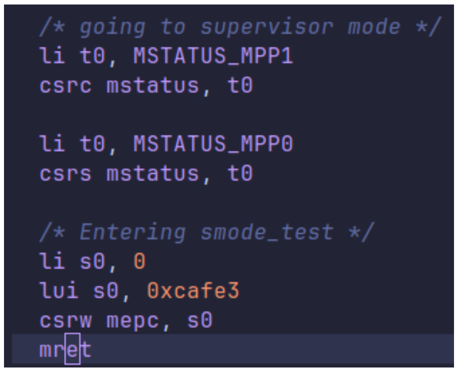
RISC-V ISA, Volume II: Privilege Architecture, Page 126



```
RVTEST_CODE_BEGIN
_start:
  /* Setting the stack pointer */
  la sp, _sstack
  call vm_configure
  call trap_init
  j main

  /**
   * Virtual Memory configurations
   *
   * Initializes the satp register and sets the root page table entry
   * NOTE: This function is to be called before trap_init. CSRs will update
   * after trap_init is called.
   * ===================================================================
   * satp:
   *  [ MODE | ASID | PFN ]
   *      1     9     22
   *  - MODE: 1 (SV32)
   *  - ASID: 0
   *  - PFN : page_table address
   */
  .p2align 2
  .type vm_configure, @function
vm_configure:
  /**
   * Binary for enabling Sv32 and setting ASID = 1:
   * 1_000|0000|00_00|0000|0000|0000|0000|0000
   */
  li t0, (1 << 31)
  la t1, l1_pt
  srli t1, t1, 12
  or t0, t0, t1

  csrw satp, t0
  sfence.vma

  ret
```

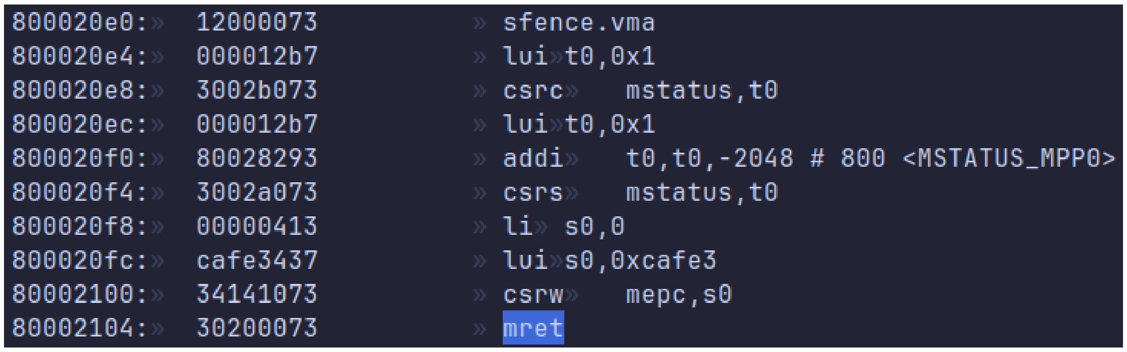
init.S: lines 5-39. Setting up satp register to use sv32 tranlsation scheme.

d. **Goto S/U mode by mret such that mepc contain the Virtual address. (Translation must be enabled).**



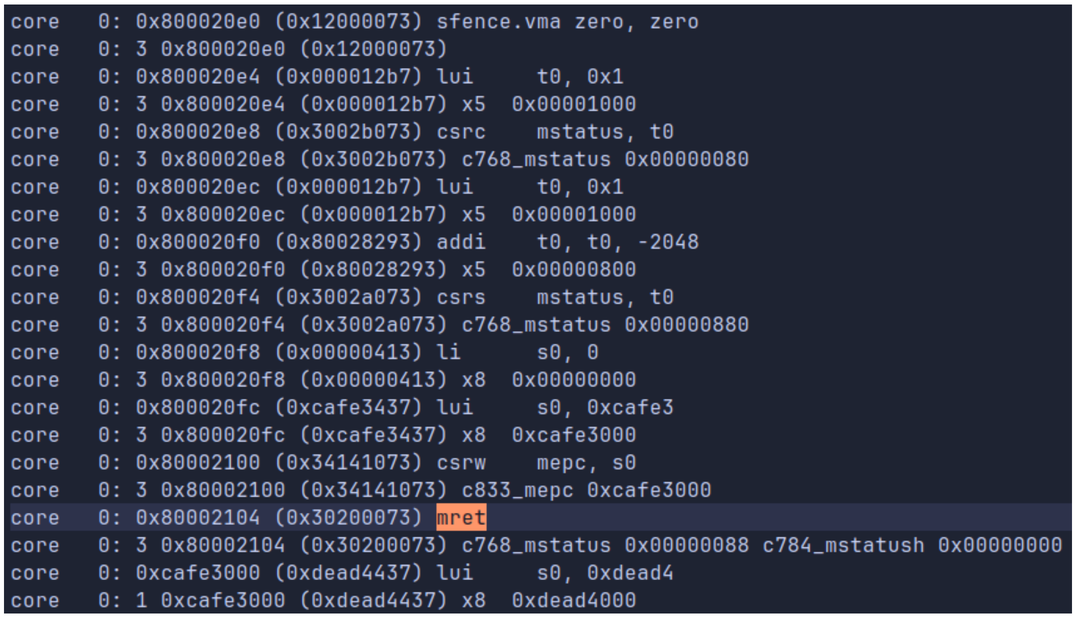
```
/* going to supervisor mode */
li t0, MSTATUS_MPP1
csrc mstatus, t0

li t0, MSTATUS_MPP0
csrs mstatus, t0

/* Entering smode_test */
li s0, 0
lui s0, 0xcafe3
csrw mepc, s0
mret
```

test.S: lines 93-102



```
800020e0:»  12000073        » sfence.vma
800020e4:»  000012b7        » lui»t0,0x1
800020e8:»  3002b073        » csrc»   mstatus,t0
800020ec:»  000012b7        » lui»t0,0x1
800020f0:»  80028293        » addi»   t0,t0,-2048 # 800 <MSTATUS_MPP0>
800020f4:»  3002a073        » csrs»   mstatus,t0
800020f8:»  00000413        » li» s0,0
800020fc:»  cafe3437        » lui»s0,0xcafe3
80002100:»  34141073        » csrw»   mepc,s0
80002104:»  30200073        » mret
```

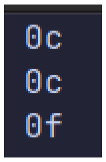
test.disass: lines 412-421



```
core   0: 0x800020e0 (0x12000073) sfence.vma zero, zero
core   0: 3 0x800020e0 (0x12000073)
core   0: 0x800020e4 (0x000012b7) lui     t0, 0x1
core   0: 3 0x800020e4 (0x000012b7) x5  0x00001000
core   0: 0x800020e8 (0x3002b073) csrc    mstatus, t0
core   0: 3 0x800020e8 (0x3002b073) c768_mstatus 0x00000080
core   0: 0x800020ec (0x000012b7) lui     t0, 0x1
core   0: 3 0x800020ec (0x000012b7) x5  0x00001000
core   0: 0x800020f0 (0x80028293) addi    t0, t0, -2048
core   0: 3 0x800020f0 (0x80028293) x5  0x00000800
core   0: 0x800020f4 (0x3002a073) csrs    mstatus, t0
core   0: 3 0x800020f4 (0x3002a073) c768_mstatus 0x00000880
core   0: 0x800020f8 (0x00000413) li      s0, 0
core   0: 3 0x800020f8 (0x00000413) x8  0x00000000
core   0: 0x800020fc (0xcafe3437) lui     s0, 0xcafe3
core   0: 3 0x800020fc (0xcafe3437) x8  0xcafe3000
core   0: 0x80002100 (0x34141073) csrw    mepc, s0
core   0: 3 0x80002100 (0x34141073) c833_mepc 0xcafe3000
core   0: 0x80002104 (0x30200073) mret
core   0: 3 0x80002104 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
core   0: 0xcafe3000 (0xdead4437) lui     s0, 0xdead4
core   0: 1 0xcafe3000 (0xdead4437) x8  0xdead4000
```

spike.log: lines: 394-415. Output shows our privlege has changed from machine to supervisor, and we are using the virtual address (0xcafe3000) showing that address translation is working.

e. **Try accessing the addresses with and without PTE (RWX) permissions: check the CSRs (mstatus, mcause, mepc) for the corresponding page-fault.**



```
0c
0c
0f
```

signature.txt: We observe that our 1st test (X/W/R on X/W/R page) raise no exceptions and hence write nothing in the signature file. The 2nd test (X/W/R on W/R page) raises fetch page fault exception (mcause 12 or 0xc), stored in the 1st byte in the signature.txt. Our final test (X/W/R on R ONLY page) raises the fetch and store page faults (mcause 12/0x0c and 15/0x0f respectively), stored in the 2nd and 3rd byte in our signature file.