 # Author : Muhammad Imran
 # Date: 16-02-2026
 # Module : RISCV Arch Test
 # Section: RISCV Arch Test
 # Task Name: Task 5

Test Description: [github]

This test verifies the correct implementation of RISC-V Virtual Memory paging, specifically the address translation mechanism using page tables. The objective of the test is to confirm that when virtual memory is enabled (via the satp register), instruction fetches, loads, and stores go through proper virtual-to-physical address translation according to the configured paging mode Sv32.

The test defines the root and leaf page tables sections and then sets up these PTEs with correct flags, permissions and physical page numbers using the pte_setup function. Then it enables virtualization by configuring the satp register. Then test perform the instruction execute and load/store from virtual address in S-mode . If translation is correct and permissions are valid, execution proceeds normally.

pte_setup Function:

This function creates a page table entry to map the virtual address to physical address along with the permissions following the sv32 paging scheme. It first extract the root page table base address from the satp.PPN field

```
core   0: 0x80000040 (0x180024f3) csrr    s1, satp
core   0: 3 0x80000040 (0x180024f3) x9  0x00080002
core   0: 0x80000044 (0x00400937) lui     s2, 0x400
core   0: 3 0x80000044 (0x00400937) x18 0x00400000
core   0: 0x80000048 (0xfff90913) addi    s2, s2, -1
core   0: 3 0x80000048 (0xfff90913) x18 0x003fffff
core   0: 0x8000004c (0x0124f4b3) and     s1, s1, s2
core   0: 3 0x8000004c (0x0124f4b3) x9  0x00080002
core   0: 0x80000050 (0x00c49493) slli    s1, s1, 12
core   0: 3 0x80000050 (0x00c49493) x9  0x80002000
```

Then it extract the VPN[1] and VPN[0] fields from the virtual address

```
55   core   0: 0x80000054 (0x00c55293) srli    t0, a0, 12
56   core   0: 3 0x80000054 (0x00c55293) x5  0x00080004
57   core   0: 0x80000058 (0x3ff2f313) andi    t1, t0, 1023
58   core   0: 3 0x80000058 (0x3ff2f313) x6  0x00000004
59   core   0: 0x8000005c (0x00a2d393) srli    t2, t0, 10
60   core   0: 3 0x8000005c (0x00a2d393) x7  0x00000200
```

Then it configures the PTE from the physical address. It extracts the PPN number, shift left to make space for the permissions bit then it add permissions and at the end it sets the valid bit

```
61   core   0: 0x80000060 (0x00c5d293) srli    t0, a1, 12
62   core   0: 3 0x80000060 (0x00c5d293) x5  0x00080003
63   core   0: 0x80000064 (0x00a29293) slli    t0, t0, 10
64   core   0: 3 0x80000064 (0x00a29293) x5  0x20000c00
65   core   0: 0x80000068 (0x00c2e2b3) or      t0, t0, a2
66   core   0: 3 0x80000068 (0x00c2e2b3) x5  0x20000c00
67   core   0: 0x8000006c (0x0012e293) ori     t0, t0, 1
68   core   0: 3 0x8000006c (0x0012e293) x5  0x20000c01
```

It stores the above translation as a root page table entry if the level argument a3=1 by shifting the VPN[1] by 2 bits because each PTE is 4-byte wide then add this VPN[1] to satp.PPN to get the address of the location where PTE will be stored.

```
69   core   0: 0x80000070 (0x00068a63) beqz    a3, pc + 20
70   core   0: 3 0x80000070 (0x00068a63)
71   core   0: 0x80000074 (0x00239393) slli    t2, t2, 2
72   core   0: 3 0x80000074 (0x00239393) x7  0x00000800
73   core   0: 0x80000078 (0x00748e33) add     t3, s1, t2
74   core   0: 3 0x80000078 (0x00748e33) x28 0x80002800
75   core   0: 0x8000007c (0x005e2023) sw      t0, 0(t3)
76   core   0: 3 0x8000007c (0x005e2023) mem 0x80002800 0x20000c01
```

In case when a3=0 and to store the PTE in leaf page table it first loads the base address of the leaf page table from the root and then adds the VPN[0] to get the address of the leaf page table location where the actual translation will get stored.

```
119  core   0: 0x80000084 (0x00239393) slli    t2, t2, 2
120  core   0: 3 0x80000084 (0x00239393) x7  0x00000800
121  core   0: 0x80000088 (0x00748e33) add     t3, s1, t2
122  core   0: 3 0x80000088 (0x00748e33) x28 0x80002800
123  core   0: 0x8000008c (0x000e2e83) lw      t4, 0(t3)
124  core   0: 3 0x8000008c (0x000e2e83) x29 0x20000c01 mem 0x80002800
125  core   0: 0x80000090 (0x00aede93) srli    t4, t4, 10
126  core   0: 3 0x80000090 (0x00aede93) x29 0x00080003
127  core   0: 0x80000094 (0x00ce9e93) slli    t4, t4, 12
128  core   0: 3 0x80000094 (0x00ce9e93) x29 0x80003000
129  core   0: 0x80000098 (0x00231313) slli    t1, t1, 2
130  core   0: 3 0x80000098 (0x00231313) x6  0x00000010
131  core   0: 0x8000009c (0x006e8eb3) add     t4, t4, t1
132  core   0: 3 0x8000009c (0x006e8eb3) x29 0x80003010
133  core   0: 0x800000a0 (0x005ea023) sw      t0, 0(t4)
134  core   0: 3 0x800000a0 (0x005ea023) mem 0x80003010 0x200010cf
```

Actual Output Explanation:

Test first defines the separation sections for root page table, leaf page table and it also define a section of test page on which it test the execute load/store permissions

At first the program starts in machine mode and sets up the satp.PPN bits which will be used by pte_setup function for page table entry.

```
19  core   0: 0x800000a8 (0x00002297) auipc   t0, 0x2
20  core   0: 3 0x800000a8 (0x00002297) x5  0x800020a8
21  core   0: 0x800000ac (0xf5828293) addi    t0, t0, -168
22  core   0: 3 0x800000ac (0xf5828293) x5  0x80002000
23  core   0: 0x800000b0 (0x00c2d293) srli    t0, t0, 12
24  core   0: 3 0x800000b0 (0x00c2d293) x5  0x00080002
25  core   0: 0x800000b4 (0x18029073) csrw    satp, t0
26  core   0: 3 0x800000b4 (0x18029073) c384_satp 0x00080002
```

Then it calls the pte_setup function which creates the root level page table entry which points to the leaf page table, for this it take the virtual address of the test page, get the root page base address from the satp.PPN and adds it to test page VPN[1] and sets all persimmons to 0. This will create this page table entry in the root

root[VPN1(test)] = pointer to leaf PT

```
71  core   0: 0x80000074 (0x00239393) slli    t2, t2, 2
72  core   0: 3 0x80000074 (0x00239393) x7  0x00000800
73  core   0: 0x80000078 (0x00748e33) add     t3, s1, t2
74  core   0: 3 0x80000078 (0x00748e33) x28 0x80002800
75  core   0: 0x8000007c (0x005e2023) sw      t0, 0(t3)
76  core   0: 3 0x8000007c (0x005e2023) mem 0x80002800 0x20000c01
77  core   0: 0x80000080 (0x00008067) ret
```

This again using the pte_setup function it stores the test page translation along with RWX permissions in the leaf page table. Permissions for test page is as follows

0xCE = 1 1 0 0 1 1 1 1

D A X W R V

```
128  core   0: 3 0x80000094 (0x00ce9e93) x29 0x80003000
129  core   0: 0x80000098 (0x00231313) slli    t1, t1, 2
130  core   0: 3 0x80000098 (0x00231313) x6  0x00000010
131  core   0: 0x8000009c (0x006e8eb3) add     t4, t4, t1
132  core   0: 3 0x8000009c (0x006e8eb3) x29 0x80003010
133  core   0: 0x800000a0 (0x005ea023) sw      t0, 0(t4)
134  core   0: 3 0x800000a0 (0x005ea023) mem 0x80003010 0x200010cf
```

Similarly this test make a translation of .text.init section in the leaf page table with all the permissions, because out test program exist in the this section so that out test can execute properly

```
187   core   0: 0x80000098 (0x00231313) slli    t1, t1, 2
188   core   0: 3 0x80000098 (0x00231313) x6  0x00000000
189   core   0: 0x8000009c (0x006e8eb3) add     t4, t4, t1
190   core   0: 3 0x8000009c (0x006e8eb3) x29 0x80003000
191   core   0: 0x800000a0 (0x005ea023) sw      t0, 0(t4)
192   core   0: 3 0x800000a0 (0x005ea023) mem 0x80003000 0x200000cf
193   core   0: 0x800000a4 (0x00008067) ret
```

It also makes root page and leaf page entry in the leaf page table so that later when we switch privilege mode then lower privilege modes can get the root and leaf page translations and modify the permissions.

After setting translations of all the sections in the leaf page table the test then enables the virtualization by setting the msb of the satp register high.

```
311   core   0: 0x8000014c (0x180022f3) csrr    t0, satp
312   core   0: 3 0x8000014c (0x180022f3) x5  0x00080002
313   core   0: 0x80000150 (0x80000337) lui     t1, 0x80000
314   core   0: 3 0x80000150 (0x80000337) x6  0x80000000
315   core   0: 0x80000154 (0x0062e2b3) or      t0, t0, t1
316   core   0: 3 0x80000154 (0x0062e2b3) x5  0x80080002
317   core   0: 0x80000158 (0x18029073) csrw    satp, t0
318   core   0: 3 0x80000158 (0x18029073) c384_satp 0x80080002
319   core   0: 0x8000015c (0x12000073) sfence.vma zero, zero
```

Now the test program switches to the supervisor mode using switch_mode function and jumps to the test section since we set the translation with RWX for this section in the leaf page table the load, store and the instructions in this will execute successfully there is not page fault error

```
355   core   0: 0x80004000 (0x00500293) li      t0, 5
356   core   0: 1 0x80004000 (0x00500293) x5  0x00000005
357   core   0: 0x80004004 (0x00000317) auipc   t1, 0x0
358   core   0: 1 0x80004004 (0x00000317) x6  0x80004004
359   core   0: 0x80004008 (0xffc30313) addi    t1, t1, -4
360   core   0: 1 0x80004008 (0xffc30313) x6  0x80004000
361   core   0: 0x8000400c (0xdddde3b7) lui     t2, 0xdddde
362   core   0: 1 0x8000400c (0xdddde3b7) x7  0xdddde000
363   core   0: 0x80004010 (0xddd38393) addi    t2, t2, -547
364   core   0: 1 0x80004010 (0xddd38393) x7  0xddddddd
365   core   0: 0x80004014 (0x00732023) sw      t2, 0(t1)
366   core   0: 1 0x80004014 (0x00732023) mem 0x80004000 0xddddddd
```

Now again using the pte_setup function clear the permissions of the test section in the translation available in the leaf page table, permissions are as follows

0xC0 = 1100 0000

Only A and D bits set.

Now when try to execute again the test section it give inst_page_fault

```
426   core   0: 1 0x800000a4 (0x00008067)
427   core   0: 0x80000188 (0x679030ef) jal      pc + 0x3e78
428   core   0: 1 0x80000188 (0x679030ef) x1  0x8000018c
429   core   0: exception trap_instruction_page_fault, epc 0x80004000
430   core   0:           tval 0x80004000
431   core   0: >>>>  trap_vector
432   core   0: 0x80000010 (0x342022f3) csrr    t0, mcause
433   core   0: 3 0x80000010 (0x342022f3) x5  0x0000000c
```

Trying to read any data from this section also give load_page fault

```
462   core   0: 1 0x80000190 (0xe7428293) x5  0x80004000
463   core   0: 0x80000194 (0x0002a303) lw      t1, 0(t0)
464   core   0: exception trap_load_page_fault, epc 0x80000194
465   core   0:           tval 0x80004000
466   core   0: >>>>  trap_vector
467   core   0: 0x80000010 (0x342022f3) csrr    t0, mcause
468   core   0: 3 0x80000010 (0x342022f3) x5  0x0000000d
```

Trying to store data in this section give store page fault

```
500   core   0: 1 0x800001a0 (0x0072f2b3) x5  0x0000000c
501   core   0: 0x800001a4 (0x0262a023) sw      t1, 32(t0)
502   core   0: exception trap_store_page_fault, epc 0x800001a4
503   core   0:           tval 0x0000002c
504   core   0: >>>>  trap_vector
505   core   0: 0x80000010 (0x342022f3) csrr    t0, mcause
506   core   0: 3 0x80000010 (0x342022f3) x5  0x0000000f
```

Finally the program returns in machine mode using ecall and successfully exit the code.