# Author : Muhammad Imran
# Date: 26-02-2026
# Module : RISCV Arch Test
# Section: RISCV Arch Test
# Task Name: Grand Assessment

Task Description:

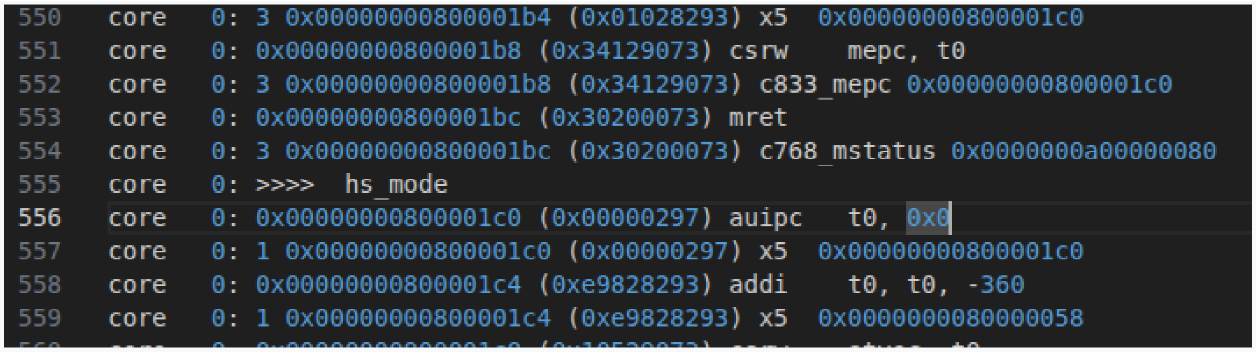
Implementation of hypervisor managing a guest OS in VS-mode with VU-mode application.

1: Confirm M→HS→VS→VU mode transitions worked

The test starts in M-mode configured the machine mode trap vector, configured the guest mode exceptions to HS mode trap vector and initializes the hgapt for G-stage translation, setup the PTEs for host stage translation (stage-2 Translation) and switches privilege mode to HS mode using the function switch_hs.

M→HS Transition

In Spike logs, the privilege level is shown as the digit after 'core 0:' on register lines: 3=M-mode, 1=HS-mode/VS-mode, 0=VU-mode.

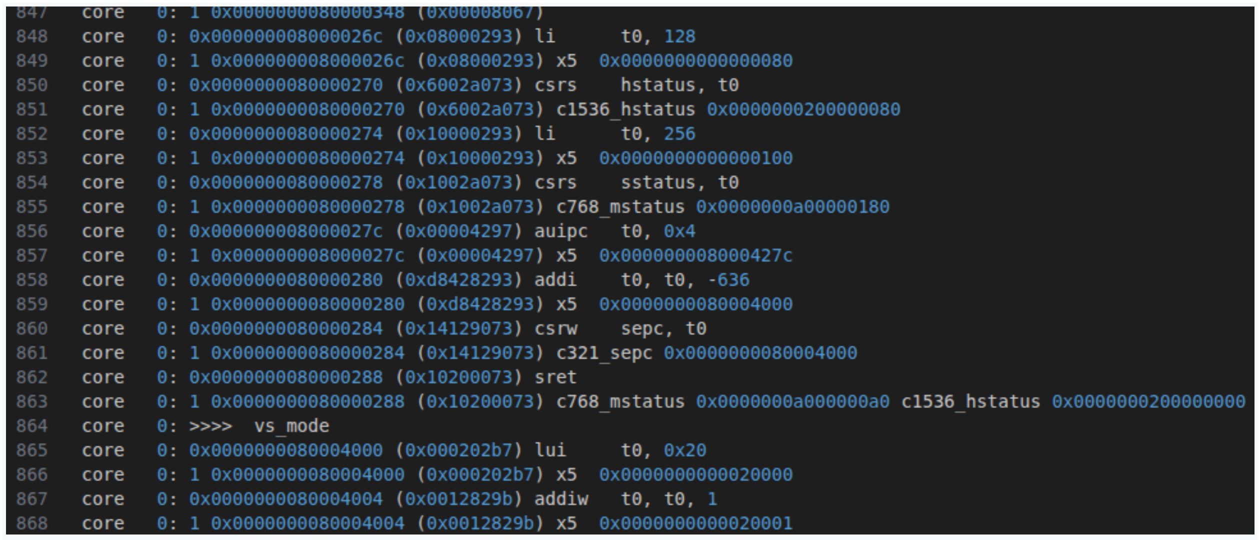


```
core   0: 3 0x00000000800001b4 (0x01028293) x5  0x00000000800001c0
core   0: 0x00000000800001b8 (0x34129073) csrw    mepc, t0
core   0: 3 0x00000000800001b8 (0x34129073) c833_mepc 0x00000000800001c0
core   0: 0x00000000800001bc (0x30200073) mret
core   0: 3 0x00000000800001bc (0x30200073) c768_mstatus 0x0000000a00000080
core   0: >>>>  hs_mode
core   0: 0x00000000800001c0 (0x00000297) auipc   t0, 0x0
core   0: 1 0x00000000800001c0 (0x00000297) x5  0x00000000800001c0
core   0: 0x00000000800001c4 (0xe9828293) addi    t0, t0, -360
core   0: 1 0x00000000800001c4 (0xe9828293) x5  0x0000000080000058
```

HS→VS Transition

HS mode setups the supervisor trap vector to handle the traps of HS mode and the traps of guest delegated to HS mode, after enabling the stage-1 translation by setting the vsatp then it setup the PTEs for stage-1 translation, finally it returns to VS-mode by setting the hstatus.SVP and sstatus.SPP bits using sret

entering VS-mode at vs_mode (0x80004000).

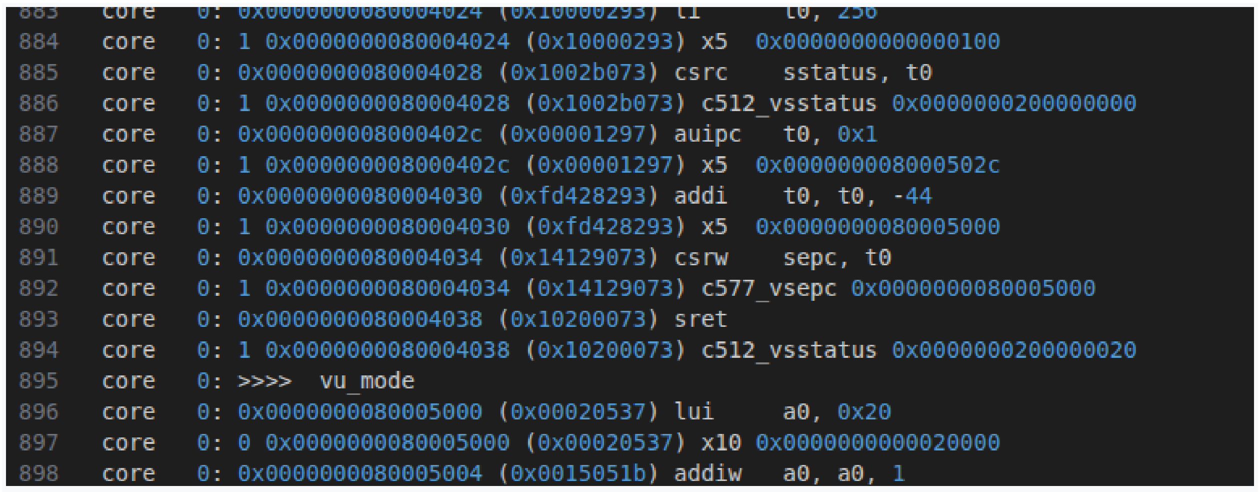


```
core   0: 1 0x0000000080000348 (0x00008067)
core   0: 0x000000008000026c (0x08000293) li      t0, 128
core   0: 1 0x000000008000026c (0x08000293) x5  0x0000000000000080
core   0: 0x0000000080000270 (0x6002a073) csrs    hstatus, t0
core   0: 1 0x0000000080000270 (0x6002a073) c1536_hstatus 0x0000000200000080
core   0: 0x0000000080000274 (0x10000293) li      t0, 256
core   0: 1 0x0000000080000274 (0x10000293) x5  0x0000000000000100
core   0: 0x0000000080000278 (0x1002a073) csrs    sstatus, t0
core   0: 1 0x0000000080000278 (0x1002a073) c768_mstatus 0x0000000a00000180
core   0: 0x000000008000027c (0x00004297) auipc   t0, 0x4
core   0: 1 0x000000008000027c (0x00004297) x5  0x000000008000427c
core   0: 0x0000000080000280 (0xd8428293) addi    t0, t0, -636
core   0: 1 0x0000000080000280 (0xd8428293) x5  0x0000000080004000
core   0: 0x0000000080000284 (0x14129073) csrw    sepc, t0
core   0: 1 0x0000000080000284 (0x14129073) c321_sepc 0x0000000080004000
core   0: 0x0000000080000288 (0x10200073) sret
core   0: 1 0x0000000080000288 (0x10200073) c768_mstatus 0x0000000a000000a0 c1536_hstatus 0x0000000200000000
core   0: >>>>  vs_mode
core   0: 0x0000000080004000 (0x000202b7) lui     t0, 0x20
core   0: 1 0x0000000080004000 (0x000202b7) x5  0x0000000000020000
core   0: 0x0000000080004004 (0x0012829b) addiw   t0, t0, 1
core   0: 1 0x0000000080004004 (0x0012829b) x5  0x0000000000020001
```

VS→VU Transition

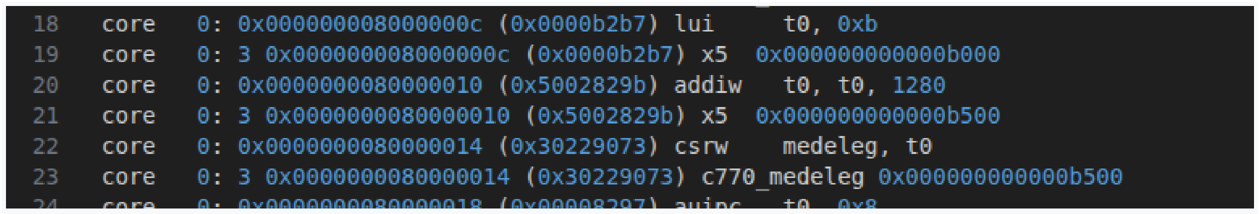
In VS-mode it sets the sstatus.SPP to zero and loads the address of VS-mode to sepc it jumps to VU mode using sret

VU-mode entry at vu_mode (0x80005000).



```
core   0: 0x0000000080004024 (0x10000293) li      t0, 256
core   0: 1 0x0000000080004024 (0x10000293) x5  0x0000000000000100
core   0: 0x0000000080004028 (0x1002b073) csrc    sstatus, t0
core   0: 1 0x0000000080004028 (0x1002b073) c512_vsstatus 0x0000000200000000
core   0: 0x000000008000402c (0x00001297) auipc   t0, 0x1
core   0: 1 0x000000008000402c (0x00001297) x5  0x000000008000502c
core   0: 0x0000000080004030 (0xfd428293) addi    t0, t0, -44
core   0: 1 0x0000000080004030 (0xfd428293) x5  0x0000000080005000
core   0: 0x0000000080004034 (0x14129073) csrw    sepc, t0
core   0: 1 0x0000000080004034 (0x14129073) c577_vsepc 0x0000000080005000
core   0: 0x0000000080004038 (0x10200073) sret
core   0: 1 0x0000000080004038 (0x10200073) c512_vsstatus 0x0000000200000020
core   0: >>>>  vu_mode
core   0: 0x0000000080005000 (0x00020537) lui     a0, 0x20
core   0: 0 0x0000000080005000 (0x00020537) x10 0x0000000000020000
core   0: 0x0000000080005004 (0x0015051b) addiw   a0, a0, 1
```

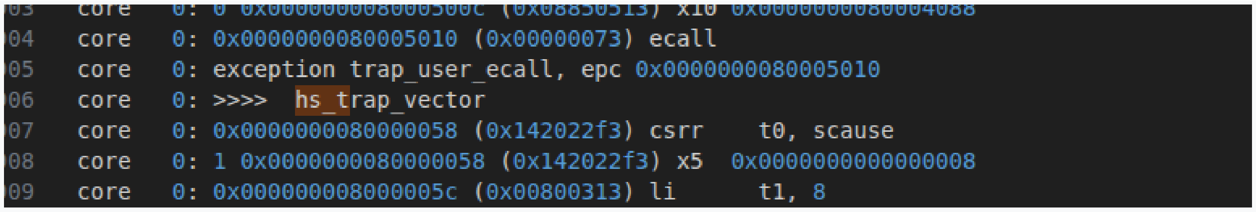
2: Verify medeleg includes both bit 8 (VU ecall) and bit 10 (VS ecall)



```
core   0: 0x000000008000000c (0x0000b2b7) lui     t0, 0xb
core   0: 3 0x000000008000000c (0x0000b2b7) x5  0x000000000000b000
core   0: 0x0000000080000010 (0x5002829b) addiw   t0, t0, 1280
core   0: 3 0x0000000080000010 (0x5002829b) x5  0x000000000000b500
core   0: 0x0000000080000014 (0x30229073) csrw    medeleg, t0
core   0: 3 0x0000000080000014 (0x30229073) c770_medeleg 0x000000000000b500
core   0: 0x0000000080000018 (0x00008297) auipc   t0, 0x8
```

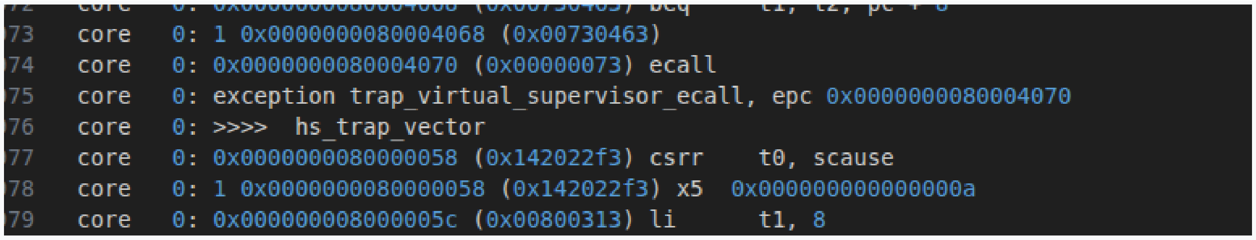
3: Confirm HS trap handler checks scause=10 for VS ecalls (NOT 9)

VU ecall handling (first trap): scause=8



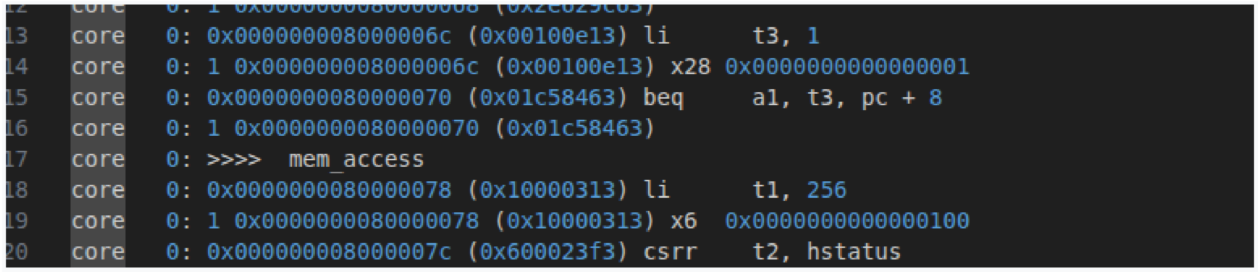
```
core   0: 0 0x000000008000500c (0x08850513) x10 0x0000000080004088
core   0: 0x0000000080005010 (0x00000073) ecall
core   0: exception trap_user_ecall, epc 0x0000000080005010
core   0: >>>>  hs_trap_vector
core   0: 0x0000000080000058 (0x142022f3) csrr    t0, scause
core   0: 1 0x0000000080000058 (0x142022f3) x5  0x0000000000000008
core   0: 0x000000008000005c (0x00800313) li      t1, 8
```

VS ecall handling (second trap): scause=10



```
core   0: 0x0000000080004068 (0x00730463) beq     t1, t2, pc + 8
core   0: 1 0x0000000080004068 (0x00730463)
core   0: 0x0000000080004070 (0x00000073) ecall
core   0: exception trap_virtual_supervisor_ecall, epc 0x0000000080004070
core   0: >>>>  hs_trap_vector
core   0: 0x0000000080000058 (0x142022f3) csrr    t0, scause
core   0: 1 0x0000000080000058 (0x142022f3) x5  0x000000000000000a
core   0: 0x000000008000005c (0x00800313) li      t1, 8
```

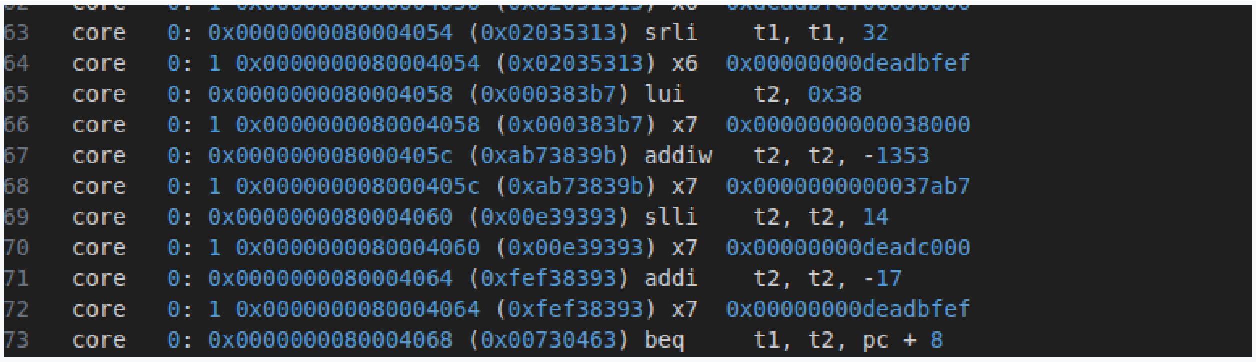
4: Verify VU-mode uses command codes in a1 to distinguish different ecall purposes



```
core   0: 1 0x0000000080000068 (0x2e029c63)
core   0: 0x000000008000006c (0x00100e13) li      t3, 1
core   0: 1 0x000000008000006c (0x00100e13) x28 0x0000000000000001
core   0: 0x0000000080000070 (0x01c58463) beq     a1, t3, pc + 8
core   0: 1 0x0000000080000070 (0x01c58463)
core   0: >>>>  mem_access
core   0: 0x0000000080000078 (0x10000313) li      t1, 256
core   0: 1 0x0000000080000078 (0x10000313) x6  0x0000000000000100
core   0: 0x000000008000007c (0x600023f3) csrr    t2, hstatus
```

5: Verify VS-mode stored data correctly (0xDEADBEEF)

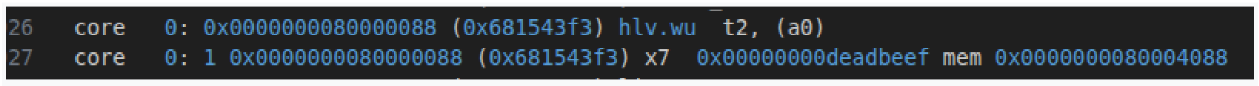
In VS-mode, the guest OS stores the test value 0xDEADBEEF at vs virtual address 0x80004088. This verifies VS-mode execution and correct address translation.



```
core   0: 1 0x0000000080004050 (0x02031313) x6  0xdeadbfef00000000
core   0: 0x0000000080004054 (0x02035313) srli    t1, t1, 32
core   0: 1 0x0000000080004054 (0x02035313) x6  0x00000000deadbfef
core   0: 0x0000000080004058 (0x000383b7) lui     t2, 0x38
core   0: 1 0x0000000080004058 (0x000383b7) x7  0x0000000000038000
core   0: 0x000000008000405c (0xab73839b) addiw   t2, t2, -1353
core   0: 1 0x000000008000405c (0xab73839b) x7  0x0000000000037ab7
core   0: 0x0000000080004060 (0x00e39393) slli    t2, t2, 14
core   0: 1 0x0000000080004060 (0x00e39393) x7  0x00000000deadc000
core   0: 0x0000000080004064 (0xfef38393) addi    t2, t2, -17
core   0: 1 0x0000000080004064 (0xfef38393) x7  0x00000000deadbfef
core   0: 0x0000000080004068 (0x00730463) beq     t1, t2, pc + 8
```

6: Confirm HLV.WU (not HLV.W) read correct value from VS memory without sign extension

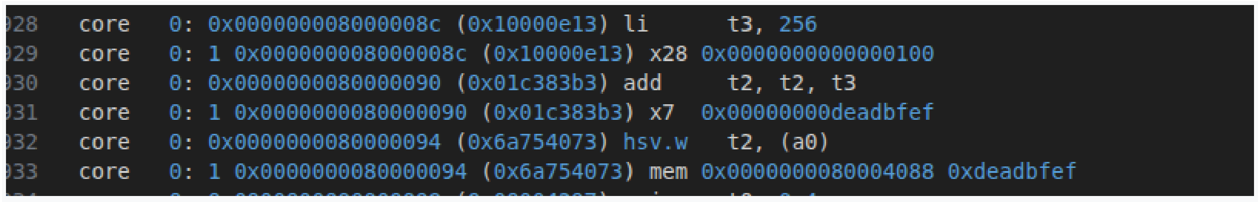
HLV.WU reads a 32-bit word from guest memory using two-stage translation (VS-stage via vsatp, then G-stage via hgatp). The 'U' suffix ensures the result is zero-extended, preserving 0xDEADBEEF correctly without sign extension.



```
core   0: 0x0000000080000088 (0x681543f3) hlv.wu  t2, (a0)
core   0: 1 0x0000000080000088 (0x681543f3) x7  0x00000000deadbeef mem 0x0000000080004088
```

7: Verify HSV.W wrote modified value (0xDEADBFEF) back to VS memory

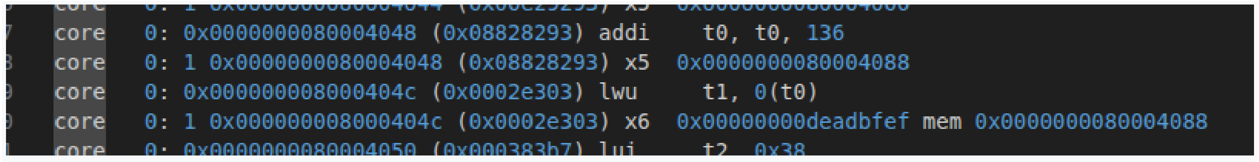
After adding 0x100 to the loaded value, HSV.W writes the result back to the same guest virtual address through two-stage translation.



```
core   0: 0x000000008000008c (0x10000e13) li      t3, 256
core   0: 1 0x000000008000008c (0x10000e13) x28 0x0000000000000100
core   0: 0x0000000080000090 (0x01c383b3) add     t2, t2, t3
core   0: 1 0x0000000080000090 (0x01c383b3) x7  0x00000000deadbfef
core   0: 0x0000000080000094 (0x6a754073) hsv.w   t2, (a0)
core   0: 1 0x0000000080000094 (0x6a754073) mem 0x0000000080004088 0xdeadbfef
```

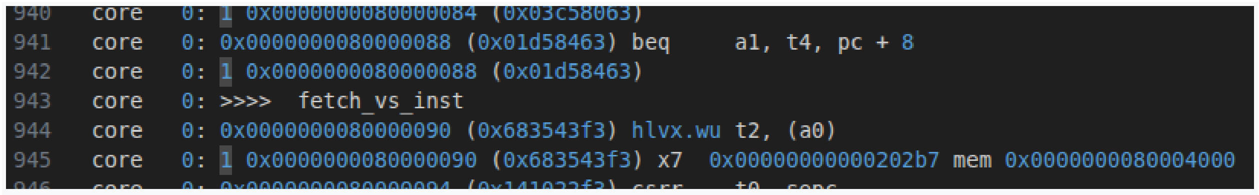
8: Confirm VS-mode uses LWU (not LW) to load and verify modified value

VS-mode (vs_verify) reloads the modified value and confirms it equals 0xDEADBFEF. The code uses LW + shift to zero-extend (equivalent to LWU semantics), avoiding sign-extension issues on 64-bit RISC-V.



```
core   0: 0x0000000080004048 (0x08828293) addi    t0, t0, 136
core   0: 1 0x0000000080004048 (0x08828293) x5  0x0000000080004088
core   0: 0x000000008000404c (0x0002e303) lwu     t1, 0(t0)
core   0: 1 0x000000008000404c (0x0002e303) x6  0x00000000deadbfef mem 0x0000000080004088
core   0: 0x0000000080004050 (0x000383b7) lui     t2, 0x38
```

9: Verify HLVX.WU successfully fetched instruction from VS code region

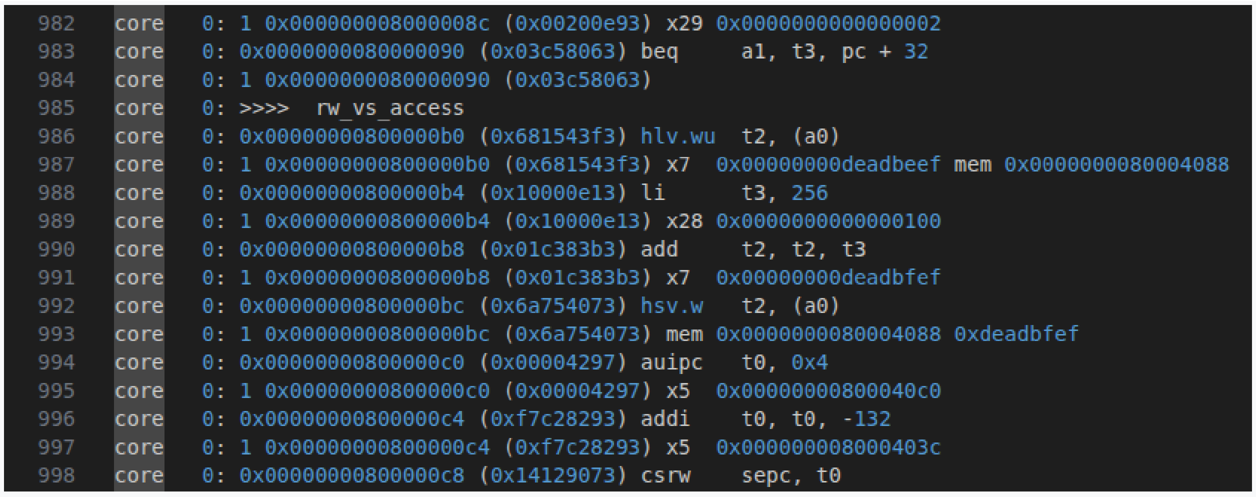


```
core   0: 1 0x0000000080000084 (0x03c58063)
core   0: 0x0000000080000088 (0x01d58463) beq     a1, t4, pc + 8
core   0: 1 0x0000000080000088 (0x01d58463)
core   0: >>>>  fetch_vs_inst
core   0: 0x0000000080000090 (0x683543f3) hlvx.wu t2, (a0)
core   0: 1 0x0000000080000090 (0x683543f3) x7  0x00000000000202b7 mem 0x0000000080004000
```

10: Confirm two-stage translation occurred

With both stages active, a guest virtual address (GVA) is first translated by vsatp (GVA → GPA) and then by hgatp (GPA →

HPA). The HLV.WU and HSV.W instructions in the HS trap handler transparently perform this full two-stage walk.

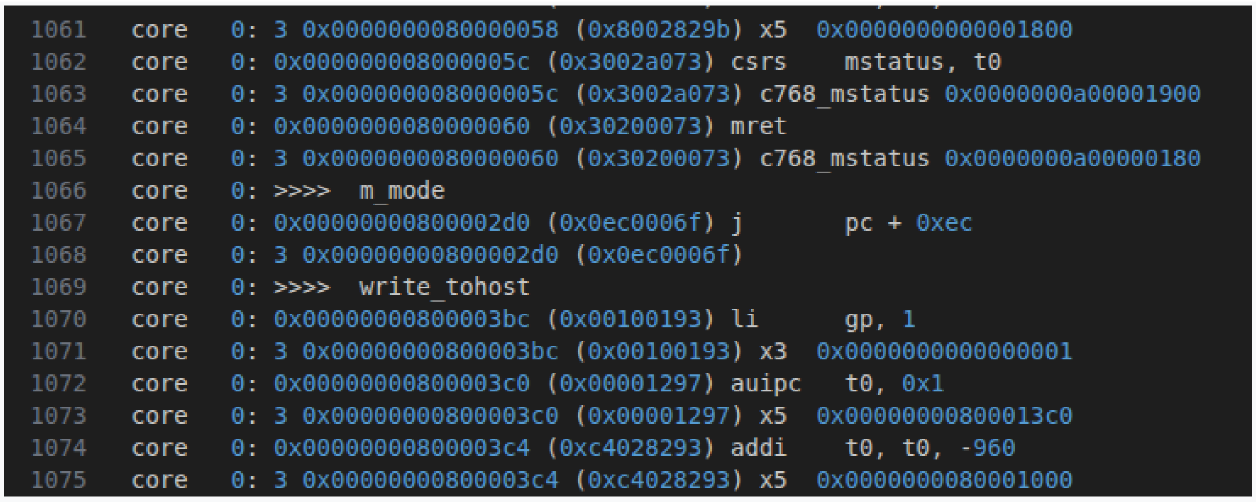

```
core   0: 1 0x000000008000008c (0x00200e93) x29 0x0000000000000002
core   0: 0x0000000080000090 (0x03c58063) beq     a1, t3, pc + 32
core   0: 1 0x0000000080000090 (0x03c58063)
core   0: >>>>  rw_vs_access
core   0: 0x00000000800000b0 (0x681543f3) hlv.wu  t2, (a0)
core   0: 1 0x00000000800000b0 (0x681543f3) x7  0x00000000deadbeef mem 0x0000000080004088
core   0: 0x00000000800000b4 (0x10000e13) li      t3, 256
core   0: 1 0x00000000800000b4 (0x10000e13) x28 0x0000000000000100
core   0: 0x00000000800000b8 (0x01c383b3) add     t2, t2, t3
core   0: 1 0x00000000800000b8 (0x01c383b3) x7  0x00000000deadbfef
core   0: 0x00000000800000bc (0x6a754073) hsv.w   t2, (a0)
core   0: 1 0x00000000800000bc (0x6a754073) mem 0x0000000080004088 0xdeadbfef
core   0: 0x00000000800000c0 (0x00004297) auipc   t0, 0x4
core   0: 1 0x00000000800000c0 (0x00004297) x5  0x00000000800040c0
core   0: 0x00000000800000c4 (0xf7c28293) addi    t0, t0, -132
core   0: 1 0x00000000800000c4 (0xf7c28293) x5  0x000000008000403c
core   0: 0x00000000800000c8 (0x14129073) csrw    sepc, t0
```

11: Verify proper mode transitions: M→HS→VS→VU→HS→VS→HS→M

Mode transitions verified manually from the spike log

12: Confirm test exited successfully in M-mode with tohost=1

The HS-mode ecall (mcause=9) is handled by m_trap_vector. It advances mepc by 4 to skip the ecall and returns to HS-mode. HS-mode then falls through to write_tohost, writing 1 to the tohost address — the PASS signal recognized by Spike.


```
core   0: 3 0x0000000080000058 (0x8002829b) x5  0x0000000000001800
core   0: 0x000000008000005c (0x3002a073) csrs    mstatus, t0
core   0: 3 0x000000008000005c (0x3002a073) c768_mstatus 0x0000000a00001900
core   0: 0x0000000080000060 (0x30200073) mret
core   0: 3 0x0000000080000060 (0x30200073) c768_mstatus 0x0000000a00000180
core   0: >>>>  m_mode
core   0: 0x00000000800002d0 (0x0ec0006f) j       pc + 0xec
core   0: 3 0x00000000800002d0 (0x0ec0006f)
core   0: >>>>  write_tohost
core   0: 0x00000000800003bc (0x00100193) li      gp, 1
core   0: 3 0x00000000800003bc (0x00100193) x3  0x0000000000000001
core   0: 0x00000000800003c0 (0x00001297) auipc   t0, 0x1
core   0: 3 0x00000000800003c0 (0x00001297) x5  0x00000000800013c0
core   0: 0x00000000800003c4 (0xc4028293) addi    t0, t0, -960
core   0: 3 0x00000000800003c4 (0xc4028293) x5  0x0000000080001000
```