Task 5

1. Write a test for VM in spike/sail (Virtual Memory task - SV32)

a. Pre-requisites

- Read RISCV privileged architecture spec for Virtrual Memory
- Read about mstatus, mtvec, mcause, mepc CSRs and different operating modes (Machine, Supervisor and User modes)

b. Write an assembly function to setup the page table entry, Function must have following arguments:

- Virtual address.
- Physical address.
- Permissions
- level: Either 0 or 1. This sets the PTE specified by the level parameter

c. Write assembly test:

- Start in M mode. Setup the PTEs for Instruction memory address (or DMem: load/store addresses).
- Setup your trap-handler to handle the exception appropriately.
- Set satp appropriately to enable virtualization with SV32 translation scheme.
- Goto S/U mode by mret such that mepc contain the Virtual address. (Translation must be enabled)
- Try accessing the addresses with and without PTE (RWX) permissions: check the CSRs (mstatus, mcause, mepc) for the corresponding page-fault.

➢ Solution:

Github Link:

https://github.com/BilalAli10x/Riscv_Arch_Test/tree/main/BilalAli_Task5

Test Description

This test validates the **SV32 virtual memory system**:

1. Configures **page table entries (PTEs)** for instruction and data memory addresses.
2. Initializes a **trap handler** to handle Load, Store, and Instruction page faults.
3. Enables virtual memory via the satp CSR.
4. Performs a **mode switch** from M-mode -> S-mode using mret with mepc pointing to virtual addresses.
5. Accesses memory addresses with and without permissions to verify correct **page-fault exceptions**.

Implementation Overview

➢ PTE Setup Function (setup_pte)

Argument	Description
a0	Virtual Address (VA)
a1	Physical Address (PA)
a2	Permissions (R/W/X/A/D bits)
a3	Level (0 = leaf page, 1 = root-page)

- Constructs SV32 PTEs according to Privileged Spec

- Writes entries to root or leaf page table depending on level
- Sets Valid (V) bit and permission flags

➢ Page Table Initialization

VA	PA	Permissions	Notes
0x80001000	0x80001000	R/W/X/A/D	Valid leaf page
0x80100000	0x80000000	None	Should trigger page fault
0x80200000	0x80002000	R/W/A	Non-executable page (X=0)

- Initializes root page table
- Configures leaf pages with proper permission flags

➢ Virtual Memory Enable

- Set satp for Sv32 mode
- Flush TLB using sfence.vma
- Switch to S-mode via mret with mepc = main

➢ Memory Access Tests

Address	Access Type	Expected Result	mcause
0x80001000	Execute	Successs	–
0x80001000	Load	Success	–
0x80100000	Load	Load Page Fault	13
0x80100000	Store	Store Page Fault	15
0x80200000	Execute	Instruction PF	12
0x80200000	Load	Success	–

- Trap handler increments mepc to skip faulting instruction
- Validates mcause and mstatus after each access

➢ Trap Handler

- Handles Instruction/Load/Store page faults
- Updates mepc to skip the faulting instruction
- Provides a mechanism to recover and continue test

Spike Log Snapshot:

```
core   0: 0x80001014 (0x000f2f83) lw      t6, 0(t5)
core   0: exception trap_load_page_fault, epc 0x80001014
core   0:           tval 0x80100000
(spike)
core   0: >>>>  trap_handler
core   0: 0x80000140 (0x342022f3) csrr    t0, mcause
core   0: 3 0x80000140 (0x342022f3) x5  0x0000000d
```

```
core   0: 0x80001020 (0x01ff2023) sw      t6, 0(t5)
core   0: exception trap_store_page_fault, epc 0x80001020
core   0:           tval 0x80100000
(spike)
core   0: >>>>  trap_handler
core   0: 0x80000140 (0x342022f3) csrr    t0, mcause
core   0: 3 0x80000140 (0x342022f3) x5  0x0000000f
```

```
core   0: exception trap_instruction_page_fault, epc 0x80200000
core   0:           tval 0x80200000
(spike)
core   0: >>>>  trap_handler
core   0: 0x80000140 (0x342022f3) csrr    t0, mcause
core   0: 3 0x80000140 (0x342022f3) x5  0x0000000c
```

Reference to Specification

RISC-V Privileged Architecture Specification

- **Section 3.1.15** - mcause
    - Instruction page fault = 12
    - Load page fault = 13
    - Store/AMO page fault = 15
- **Section 4.1.11:** satp register
- **Section 4.2.1:** Supervisor Fence Instruction
- **Section 4.3:** 32-bit Virtual-Memory Systems
- **Section 4.3.2:** Virtual Address Translation Process