**RV DV: RISC-V Arch Test**

**RISC-V Arch Test – Task: 2**

**Following is the link to the github repository for this task:**

<https://github.com/daniyalahmed-10xe/RISC-V_Arch_Test-Task_2.git>

**Test Description:**

This task was a continuation of the previous one. In this program physical memory protection (PMP) has been added to the mode switch program in the last task. This program defined 2 PMP regions, a 4KB TOR region at address ‘0x80001000’ boundry and a 4KB NAPOT region at address ‘0x80004000’. The TOR region has execute permissions only so it throws a load and store exception for the ‘lw’ and ‘sw’ instructions and the NAPOT region has read and execute permissions only (execute permissions are needed to execute any instructions including read instructions) and throws a store exception on ‘sw’. For the second part ot this task, these same protections were to be applied to machine kode which was done by setting the ‘LOCK’ bit which does NOT ignore machine-mode and prevents the changing of CSR values until a hardware reset. Note that since the provided linker file does not allocate any space for the stack, the ‘PROLOGUE’ and ‘EPILOGUE’ can not be used here and are commented out.

**Whats’s the Actual Output? (Screenshots Attatched at the End of File):**

According to the logfile (screenshot provided at the end) The TOR region sunccesfully executes instructions (jalr) with its execute permissions only and throws a load and store execption on ‘lw’ and ‘sw’ while NAPOT region succesfully executes load instructions while throwing a store exception on ‘sw’. Additionally the ‘trapVector’ has been modified and for any of the load (mstatus = 5), store (mstatus = 7), instruction access (mstatus = 1) will simply load the next instruction’s address without changing any modes.

**Following are the answers to the questions asked in this task:**

Instruction access faults happen when the the current mode does not have permission to execute instructions.

The above protections also need to be applied to machine mode because not using the ‘LOCK’ bit in other modes esentially igonores the machine-mode by default so no protections are available for machine-mode until the ‘LOCK’ bit is set in machine-mode but this will prevent any CSR values from being modified until a hardware reset.

**Following is the code written for this task:**

* **task2.S**

#define RVTEST\_DATA\_BEGIN \

.pushsection .tohost,"aw",@progbits; \

.align 6; .global tohost; tohost: .dword 0; .size tohost, 8; \

.align 6; .global fromhost; fromhost: .dword 0; .size fromhost, 8; \

.popsection; \

.align 4; .global begin\_signature; begin\_signature:

#define RVTEST\_CODE\_BEGIN \

.section .text.init; \

.align 6; \

.global \_start; \

\_start: \

j main; \

j writeToHost

RVTEST\_CODE\_BEGIN

main:

# PROLOGUE

# addi sp, sp, -16

# sw ra, 0(sp)

# sw gp, 4(sp)

# sw tp, 8(sp)

# sw fp, 12(sp)

# END PROLOGUE

# CODE

la t0, trapVector

csrw mtvec, t0

li t0, 0

li t0, 0b00011101

slli t0, t0, 8

ori t0, t0, 0b00001100

csrw pmpcfg0, t0

li t0, 0x80001000

srli t0, t0, 2

csrw pmpaddr0, t0

li t0, 0x80004000

srli t0, t0, 2

addi t0, t0, 0x1FF

csrw pmpaddr1, t0

csrr a0, mstatus

li a0, 0

call switchMode

li t0, 0x80000000

jalr t0

li t0, 0x80000000

lw t0, 0(t0)

li t0, 0x80000000

li t1, 0xDEADBEEF

sw t0, 0(t1)

li t0, 0x80004000

jalr t0

li t0, 0x80004000

lw t0, 0(t0)

li t0, 0x80004000

li t1, 0xDEADBEEF

sw t0, 0(t1)

csrr a0, sstatus

ecall

csrr a0, mstatus

li a0, 1

call switchMode

ecall

ecall

csrr a0, mstatus

li t0, 0

li t0, 0b10011101

slli t0, t0, 8

ori t0, t0, 0b10001100

csrw pmpcfg0, t0

li t0, 0x80000000

jalr t0

li t0, 0x80000000

lw t0, 0(t0)

li t0, 0x80000000

li t1, 0xDEADBEEF

sw t0, 0(t1)

li t0, 0x80004000

jalr t0

li t0, 0x80004000

lw t0, 0(t0)

li t0, 0x80004000

li t1, 0xDEADBEEF

sw t0, 0(t1)

# END CODE

# EPILOGUE

# lw fp, 12(sp)

# lw tp, 8(sp)

# lw gp, 4(sp)

# lw ra, 0(sp)

# addi sp, sp, 16

# END EPILOGUE

end\_main: call writeToHost

writeToHost:

# PROLOGUE

# addi sp, sp, -16

# sw ra, 0(sp)

# sw gp, 4(sp)

# sw tp, 8(sp)

# sw fp, 12(sp)

# END PROLOGUE

# CODE

li gp, 1

sw gp, tohost, t5

# END CODE

# EPILOGUE

# lw fp, 12(sp)

# lw tp, 8(sp)

# lw gp, 4(sp)

# lw ra, 0(sp)

# addi sp, sp, 16

# END EPILOGUE

end\_writeToHost: call writeToHost

switchMode:

# PROLOGUE

# addi sp, sp, -16

# sw ra, 0(sp)

# sw gp, 4(sp)

# sw tp, 8(sp)

# sw fp, 12(sp)

# END PROLOGUE

# CODE

mv t0, a0

csrr t1, mstatus

li t6, 0x1800

not t6, t6

and t1, t1, t6

if1: bnez t0, else1

li t6, 0x0800

j end\_if1

else1:

li t6, 0x0000

j end\_if1

end\_if1:

or t1, t1, t6

csrw mstatus, t1

# END CODE

# EPILOGUE

# lw fp, 12(sp)

# lw tp, 8(sp)

# lw gp, 4(sp)

# lw ra, 0(sp)

# addi sp, sp, 16

csrw mepc, ra

# END EPILOGUE

end\_switchMode: mret

trapVector:

# PROLOGUE

# addi sp, sp, -64

# sw gp, 4(sp)

# sw tp, 8(sp)

# sw fp, 12(sp)

# sw s0, 16(sp)

# sw s1, 20(sp)

# sw s2, 24(sp)

# sw s3, 28(sp)

# sw s4, 32(sp)

# sw s5, 36(sp)

# END PROLOGUE

# CODE

csrr s0, mcause

li s1, 9

li s2, 8

li s3, 5

li s4, 7

li s5, 1

if2: bne s0, s1, else2if3

li s1, 0x1800

j end\_if23456

else2if3: bne s0, s2, else3if4

li s1, 0x0800

j end\_if23456

else3if4: bne s0, s3, else4if5

li s1, 0

j end\_if23456

else4if5: bne s0, s4, else5if6

li s1, 0

j end\_if23456

else5if6: bne s0, s5, end\_if23456

li s1, 0

j end\_if23456

end\_if23456:

csrr s0, mstatus

or s0, s0, s1

csrw mstatus, s0

# END CODE

# EPILOGUE

# lw s5, 36(sp)

# lw s4, 32(sp)

# lw s3, 28(sp)

# lw s2, 24(sp)

# lw s1, 20(sp)

# lw s0, 16(sp)

# lw fp, 12(sp)

# lw tp, 8(sp)

# lw gp, 4(sp)

# lw ra, 0(sp)

# addi sp, sp, 64

addi ra, ra, 4

csrw mepc, ra

# END EPILOGUE

end\_trapVector: mret

.data

base:

.word 0xcafebeef

RVTEST\_DATA\_BEGIN

* **link.ld**

OUTPUT\_ARCH( "riscv" )

ENTRY(\_start)

SECTIONS

{

. = 0x80000000;

.text.init : { \*(.text.init) }

. = ALIGN(0x1000);

.tohost : { \*(.tohost) }

. = ALIGN(0x1000);

.text : { \*(.text) }

. = ALIGN(0x1000);

.data : { \*(.data) }

.bss : { \*(.bss) }

\_end = .;

}

**Following are the screenshots of the output for this task:**







