RV DV: RISC-V Arch Test

RISC-V Arch Test – Task: 1

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

https://github.com/daniyalahmed-10xe/RISC-V_Arch_Test-Task_1.git

Test Description:

This test switched between different privilege modes including Machine Mode, Supervisor Mode, and User Mode. It calls a function ‘switchMode’ with an argument of either ‘0’ or ‘1’ to idnetify which mode to switch into from Supervisor or User respectively. It also defines a trap handler function ‘trapVector’ that identifies the current mode and switeches the user up one privilege level i.e. if current mode is ‘User’, after an ecall, the mode will be switched to ‘Supervisor’. The ‘mstatus’ and ‘sstatus’ bits of the ‘CSR’ register are used to verify the modes. Finally the program ends in machine mode by calling the ‘writeToHost’ function in an infinite loop. 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 program starts in machine mode as can be seen by the log, it then switches to supervisor mode by calling the ‘switchMode’ function with an argument of ‘0’. Then an ecall returns it back to machine mode. The ‘switchMode’ function is called again, this time with an argument of ‘1’, to change the privilege level to user mode and finally an ecall is called twice to return to machine mode before terminating the program with a ‘SUCCESS’ message. The log shows the privilege level chaning from to different modes i.e. ‘M to S’, ‘S to M’, ‘M to U’, ‘U to S’, etc.

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

All programs start in machine mode if the privilege level is not changed, hence there is no need to explicitly start the program in machine mode as it already starts in machine mode.

Following is the code written for this task:

- **task1.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;                                                    \
RVTEST_CODE_BEGIN

main:

        # PROLOGUE
                #addi sp, sp, -16
                #sw ra, 0(sp)
                #sw fp, 4(sp)
        # END PROLOGUE

        la t0, trapVector               # Setup Trap Vector
        csrw mtvec, t0

        csrr a0, mstatus                # Check Machine Mode Status

        li a0, 0                        # Switch to Supervisor Mode
        call switchMode

        csrr a0, sstatus          # Check Supervisor Mode Status

        ecall                                   # Call Trap Vector to Increment Mode to
Machine (Supervisor Mode --Trap--> Machine Mode)

        csrr a0, mstatus                # Check Machine Mode Status

        li a0, 1                        # Switch to User Mode
        call switchMode

        ecall                                   # Call Trap Vector twice to Increment Mode
twice to Machine (User Mode --Trap--> Supervisor Mode --Trap--> Machine Mode)
        ecall

        csrr a0, mstatus                # Check Machine Mode Status

   # EPILOGUE
                #lw fp, 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 fp, 4(sp)
        # END PROLOGUE

        li gp, 1
```

```
        sw gp, tohost, t5

        # EPILOGUE
                #lw fp, 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 fp, 4(sp)
        # END PROLOGUE

        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

        # EPILOGUE
                #lw fp, 4(sp)
                #lw ra, 0(sp)
                #addi sp, sp, 16
                csrw mepc, ra
        # END EPILOGUE

end_switchMode: mret

trapVector:

        # PROLOGUE
                #addi sp, sp, -32
                #sw ra, 0(sp)
                #sw fp, 4(sp)
                #sw s0, 8(sp)
                #sw s1, 12(sp)
```

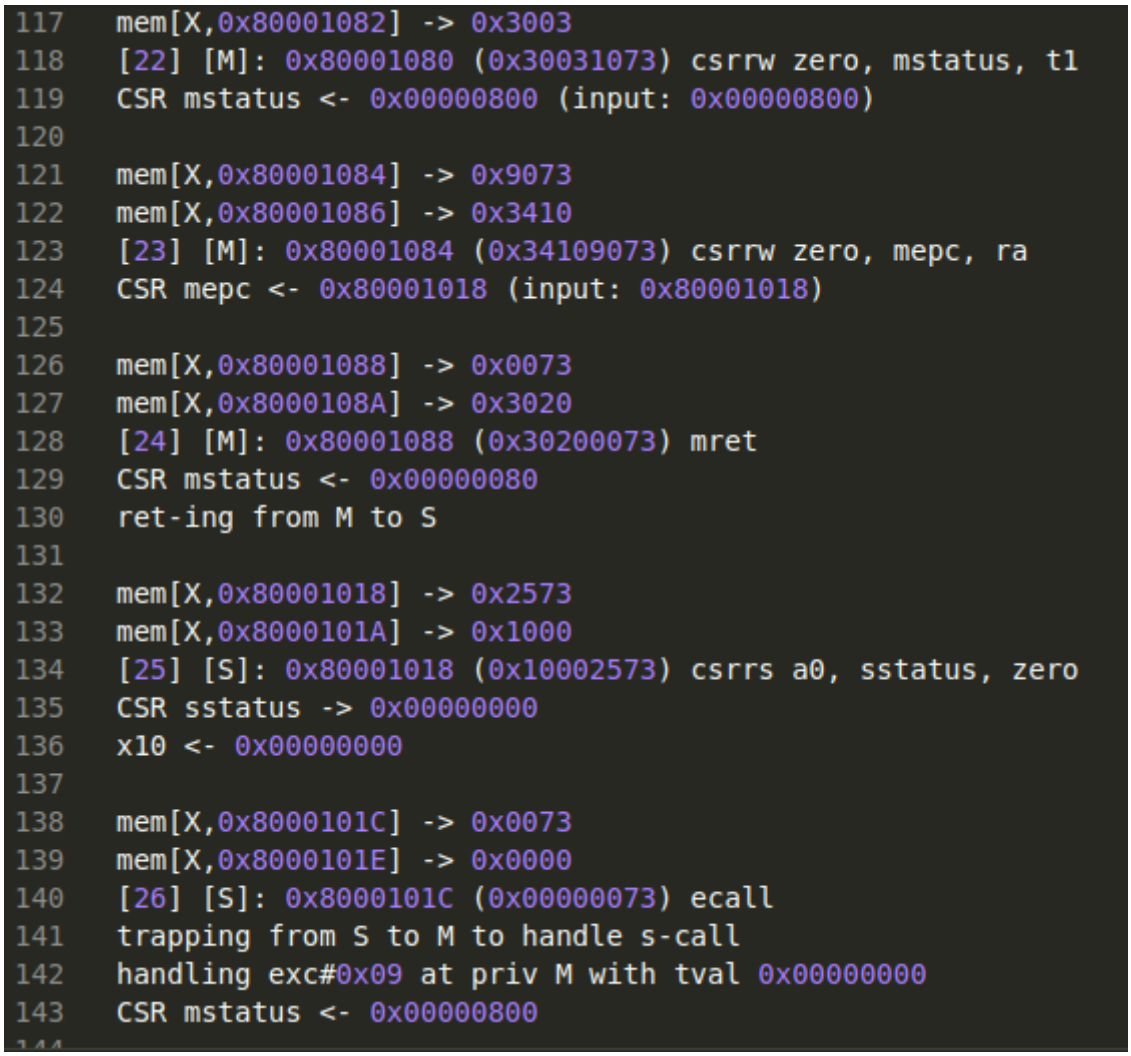
```
        #sw s2, 16(sp)
    # END PROLOGUE

    csrr s0, mcause
    li s1, 9
    li s2, 8

    if2: bne s0, s1, elseif2
        li s1, 0x1800
        j end_if2
    elseif2: bne s0, s2, end_if2
        li s1, 0x0800
        j end_if2
    end_if2:

    csrr s0, mstatus
    or s0, s0, s1
    csrw mstatus, s0

    # EPILOGUE
        #lw s2, 16(sp)
        #lw s1, 12(sp)
        #lw s0, 8(sp)
        #lw fp, 4(sp)
        #lw ra, 0(sp)
        #addi sp, sp, 32
        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:



```
117   mem[X,0x80001082] -> 0x3003
118   [22] [M]: 0x80001080 (0x30031073) csrrw zero, mstatus, t1
119   CSR mstatus <- 0x00000800 (input: 0x00000800)
120
121   mem[X,0x80001084] -> 0x9073
122   mem[X,0x80001086] -> 0x3410
123   [23] [M]: 0x80001084 (0x34109073) csrrw zero, mepc, ra
124   CSR mepc <- 0x80001018 (input: 0x80001018)
125
126   mem[X,0x80001088] -> 0x0073
127   mem[X,0x8000108A] -> 0x3020
128   [24] [M]: 0x80001088 (0x30200073) mret
129   CSR mstatus <- 0x00000080
130   ret-ing from M to S
131
132   mem[X,0x80001018] -> 0x2573
133   mem[X,0x8000101A] -> 0x1000
134   [25] [S]: 0x80001018 (0x10002573) csrrs a0, sstatus, zero
135   CSR sstatus -> 0x00000000
136   x10 <- 0x00000000
137
138   mem[X,0x8000101C] -> 0x0073
139   mem[X,0x8000101E] -> 0x0000
140   [26] [S]: 0x8000101C (0x00000073) ecall
141   trapping from S to M to handle s-call
142   handling exc#0x09 at priv M with tval 0x00000000
143   CSR mstatus <- 0x00000800
```

```
191   mem[X,0x800010C2] -> 0x3004
192   [36] [M]: 0x800010C0 (0x30041073) csrrw zero, mstatus, fp
193   CSR mstatus <- 0x00001800 (input: 0x00001800)
194
195   mem[X,0x800010C4] -> 0x8093
196   mem[X,0x800010C6] -> 0x0040
197   [37] [M]: 0x800010C4 (0x00408093) addi ra, ra, 0x4
198   x1 <- 0x8000101C
199
200   mem[X,0x800010C8] -> 0x9073
201   mem[X,0x800010CA] -> 0x3410
202   [38] [M]: 0x800010C8 (0x34109073) csrrw zero, mepc, ra
203   CSR mepc <- 0x8000101C (input: 0x8000101C)
204
205   mem[X,0x800010CC] -> 0x0073
206   mem[X,0x800010CE] -> 0x3020
207   [39] [M]: 0x800010CC (0x30200073) mret
208   CSR mstatus <- 0x00000080
209   ret-ing from M to M
210
211   mem[X,0x8000101C] -> 0x0073
212   mem[X,0x8000101E] -> 0x0000
213   [40] [M]: 0x8000101C (0x00000073) ecall
214   trapping from M to M to handle m-call
215   handling exc#0x0B at priv M with tval 0x00000000
216   CSR mstatus <- 0x00001800
217
```

```
246   x8 <- 0x00001800
247
248   mem[X,0x800010BC] -> 0x6433
249   mem[X,0x800010BE] -> 0x0094
250   [47] [M]: 0x800010BC (0x00946433) or fp, fp, s1
251   x8 <- 0x00001809
252
253   mem[X,0x800010C0] -> 0x1073
254   mem[X,0x800010C2] -> 0x3004
255   [48] [M]: 0x800010C0 (0x30041073) csrrw zero, mstatus, fp
256   CSR mstatus <- 0x00001808 (input: 0x00001809)
257
258   mem[X,0x800010C4] -> 0x8093
259   mem[X,0x800010C6] -> 0x0040
260   [49] [M]: 0x800010C4 (0x00408093) addi ra, ra, 0x4
261   x1 <- 0x80001020
262
263   mem[X,0x800010C8] -> 0x9073
264   mem[X,0x800010CA] -> 0x3410
265   [50] [M]: 0x800010C8 (0x34109073) csrrw zero, mepc, ra
266   CSR mepc <- 0x80001020 (input: 0x80001020)
267
268   mem[X,0x800010CC] -> 0x0073
269   mem[X,0x800010CE] -> 0x3020
270   [51] [M]: 0x800010CC (0x30200073) mret
271   CSR mstatus <- 0x00000080
272   ret-ing from M to M
```

```
333
334   mem[X,0x8000107C] -> 0x6333
335   mem[X,0x8000107E] -> 0x01F3
336   [64] [M]: 0x8000107C (0x01F36333) or t1, t1, t6
337   x6 <- 0x00000080
338
339   mem[X,0x80001080] -> 0x1073
340   mem[X,0x80001082] -> 0x3003
341   [65] [M]: 0x80001080 (0x30031073) csrrw zero, mstatus, t1
342   CSR mstatus <- 0x00000080 (input: 0x00000080)
343
344   mem[X,0x80001084] -> 0x9073
345   mem[X,0x80001086] -> 0x3410
346   [66] [M]: 0x80001084 (0x34109073) csrrw zero, mepc, ra
347   CSR mepc <- 0x8000102C (input: 0x8000102C)
348
349   mem[X,0x80001088] -> 0x0073
350   mem[X,0x8000108A] -> 0x3020
351   [67] [M]: 0x80001088 (0x30200073) mret
352   CSR mstatus <- 0x00000088
353   ret-ing from M to U
354
355   mem[X,0x8000102C] -> 0x0073
356   mem[X,0x8000102E] -> 0x0000
357   [68] [U]: 0x8000102C (0x00000073) ecall
358   trapping from U to M to handle u-call
359   handling exc#0x08 at priv M with tval 0x00000000
```

```
412   mem[X,0x800010C2] -> 0x3004
413   [79] [M]: 0x800010C0 (0x30041073) csrrw zero, mstatus, fp
414   CSR mstatus <- 0x00000880 (input: 0x00000880)
415
416   mem[X,0x800010C4] -> 0x8093
417   mem[X,0x800010C6] -> 0x0040
418   [80] [M]: 0x800010C4 (0x00408093) addi ra, ra, 0x4
419   x1 <- 0x80001030
420
421   mem[X,0x800010C8] -> 0x9073
422   mem[X,0x800010CA] -> 0x3410
423   [81] [M]: 0x800010C8 (0x34109073) csrrw zero, mepc, ra
424   CSR mepc <- 0x80001030 (input: 0x80001030)
425
426   mem[X,0x800010CC] -> 0x0073
427   mem[X,0x800010CE] -> 0x3020
428   [82] [M]: 0x800010CC (0x30200073) mret
429   CSR mstatus <- 0x00000088
430   ret-ing from M to S
431
432   mem[X,0x80001030] -> 0x0073
433   mem[X,0x80001032] -> 0x0000
434   [83] [S]: 0x80001030 (0x00000073) ecall
435   trapping from S to M to handle s-call
436   handling exc#0x09 at priv M with tval 0x00000000
437   CSR mstatus <- 0x00000880
438
```

```
478
479   mem[X,0x800010BC] -> 0x6433
480   mem[X,0x800010BE] -> 0x0094
481   [92] [M]: 0x800010BC (0x00946433) or fp, fp, s1
482   x8 <- 0x00001880
483
484   mem[X,0x800010C0] -> 0x1073
485   mem[X,0x800010C2] -> 0x3004
486   [93] [M]: 0x800010C0 (0x30041073) csrrw zero, mstatus, fp
487   CSR mstatus <- 0x00001880 (input: 0x00001880)
488
489   mem[X,0x800010C4] -> 0x8093
490   mem[X,0x800010C6] -> 0x0040
491   [94] [M]: 0x800010C4 (0x00408093) addi ra, ra, 0x4
492   x1 <- 0x80001034
493
494   mem[X,0x800010C8] -> 0x9073
495   mem[X,0x800010CA] -> 0x3410
496   [95] [M]: 0x800010C8 (0x34109073) csrrw zero, mepc, ra
497   CSR mepc <- 0x80001034 (input: 0x80001034)
498
499   mem[X,0x800010CC] -> 0x0073
500   mem[X,0x800010CE] -> 0x3020
501   [96] [M]: 0x800010CC (0x30200073) mret
502   CSR mstatus <- 0x00000088
503   ret-ing from M to M
504
```

```
549   mem[X,0x80001044] -> 0x2023
550   mem[X,0x80001046] -> 0xFC3F
551   [105] [M]: 0x80001044 (0xFC3F2023) sw gp, -0x40(t5)
552   htif[0x80000000] <- 0x00000001
553
554   mem[X,0x80001048] -> 0xF0EF
555   mem[X,0x8000104A] -> 0xFF5F
556   [106] [M]: 0x80001048 (0xFF5FF0EF) jal ra, -0xc
557   x1 <- 0x8000104C
558
559   mem[X,0x8000103C] -> 0x0193
560   mem[X,0x8000103E] -> 0x0010
561   [107] [M]: 0x8000103C (0x00100193) addi gp, zero, 0x1
562   x3 <- 0x00000001
563
564   mem[X,0x80001040] -> 0xFF17
565   mem[X,0x80001042] -> 0xFFFF
566   [108] [M]: 0x80001040 (0xFFFFFF17) auipc t5, -0x1
567   x30 <- 0x80000040
568
569   mem[X,0x80001044] -> 0x2023
570   mem[X,0x80001046] -> 0xFC3F
571   [109] [M]: 0x80001044 (0xFC3F2023) sw gp, -0x40(t5)
572   htif[0x80000000] <- 0x00000001
573   htif-syscall-proxy cmd: 0x000000000001
574
575   SUCCESS
```