Author : Irtaza Hyder

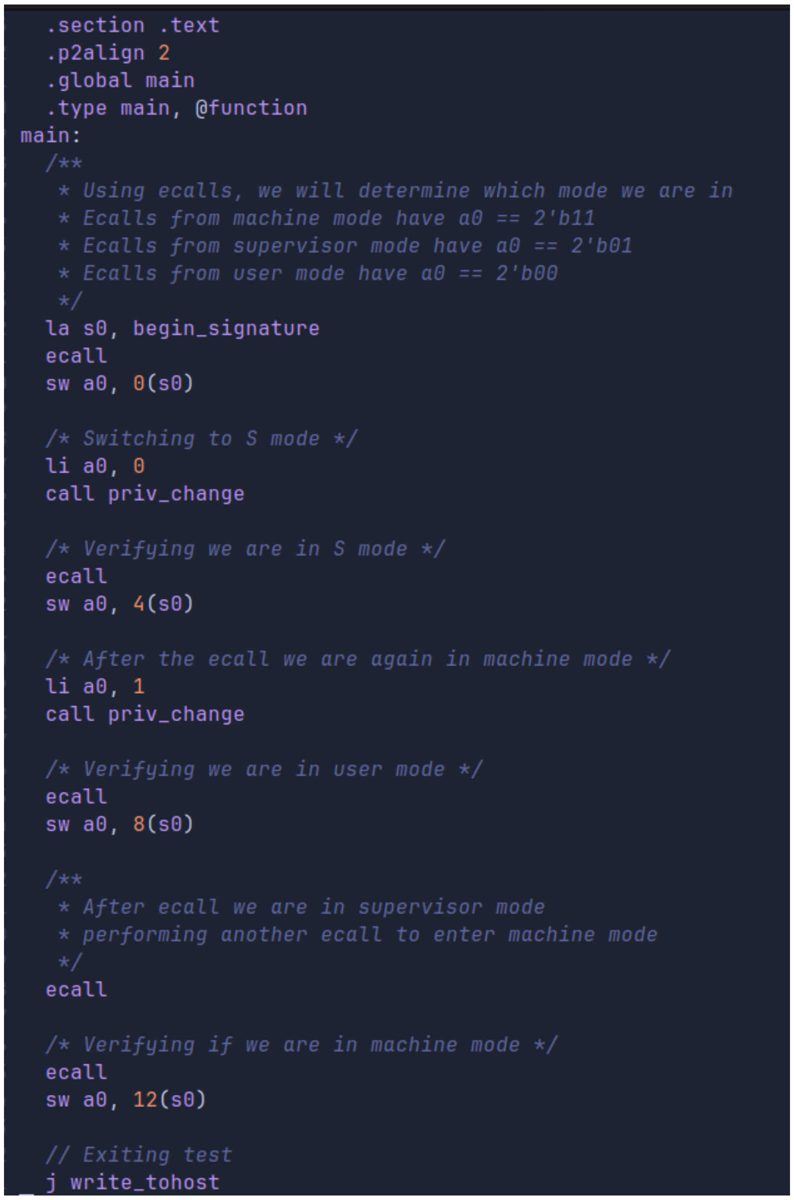
Date : 2026-02-05

Module : RISC-V Arch Test

Section : RISC-V Arch Test

Task Name : Task 1

Test Description



```
  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  /**
   * Using ecalls, we will determine which mode we are in
   * Ecalls from machine mode have a0 == 2'b11
   * Ecalls from supervisor mode have a0 == 2'b01
   * Ecalls from user mode have a0 == 2'b00
   */
  la s0, begin_signature
  ecall
  sw a0, 0(s0)

  /* Switching to S mode */
  li a0, 0
  call priv_change

  /* Verifying we are in S mode */
  ecall
  sw a0, 4(s0)

  /* After the ecall we are again in machine mode */
  li a0, 1
  call priv_change

  /* Verifying we are in user mode */
  ecall
  sw a0, 8(s0)

  /**
   * After ecall we are in supervisor mode
   * performing another ecall to enter machine mode
   */
  ecall

  /* Verifying if we are in machine mode */
  ecall
  sw a0, 12(s0)

  // Exiting test
  j write_tohost
```

test.S: main function

The test first starts in machine mode. To verify we are in machine mode, we execute the ecall function. The ecall routine for machine mode ecall writes 3 to register a0. Hence MPP (11) is written in signature.txt.

Then the downgrades to S mode by invoking the change_priv function and giving it a0 == 0. To verify that we are in S mode a supervisor ecall is executed. The ecall causes the privilege mode to upgrade to machine mode, but the original MPP is written into a0. Hence MPP (01) is written in signature.txt.

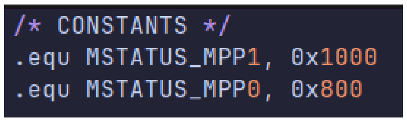
To downgrade to user mode we again invoke the priv_change function with a0 == 1. The user mode is verified by using a user ecall. This causes the privilege mode of the program to upgrade to supervisor, and the original MPP is written into a0. Hence MPP (00) is written in signature.txt.

We finally invoke ecall twice, shifting from U -> S -> M mode. Then the program goes into an infinite loop of write_tohost.

Output and Q&A

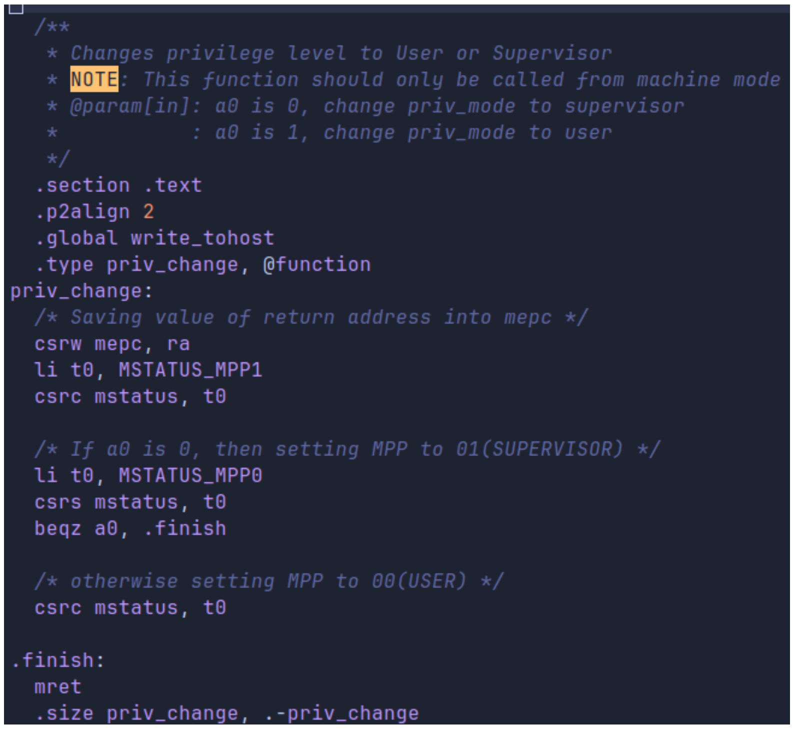
Description: Implement and configure a function and a trap handler to switch modes

a. **Pre-requisites: mstatus specific fields, mepc, mtval, mtvec, mret, mcause, ecall**
b. **Implement the following in your test:**
    1. **Function: Implement a function which will take an argument (0 or 1) and then jump to relevant mode (0 for supervisor and 1 for user). (This function is only called in machine mode).**



```
/* CONSTANTS */
.equ MSTATUS_MPP1, 0x1000
.equ MSTATUS_MPP0, 0x800
```

macro.S: lines 1-3

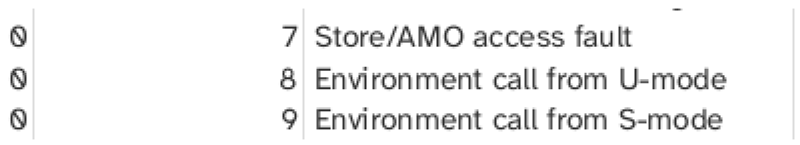


```
  /**
   * Changes privilege level to User or Supervisor
   * NOTE: This function should only be called from machine mode
   * @param[in]: a0 is 0, change priv_mode to supervisor
   *          : a0 is 1, change priv_mode to user
   */
  .section .text
  .p2align 2
  .global write_tohost
  .type priv_change, @function
priv_change:
  /* Saving value of return address into mepc */
  csrw mepc, ra
  li t0, MSTATUS_MPP1
  csrc mstatus, t0

  /* If a0 is 0, then setting MPP to 01(SUPERVISOR) */
  li t0, MSTATUS_MPP0
  csrs mstatus, t0
  beqz a0, .finish

  /* otherwise setting MPP to 00(USER) */
  csrc mstatus, t0

.finish:
  mret
  .size priv_change, .-priv_change
```

test.S: lines 60-86

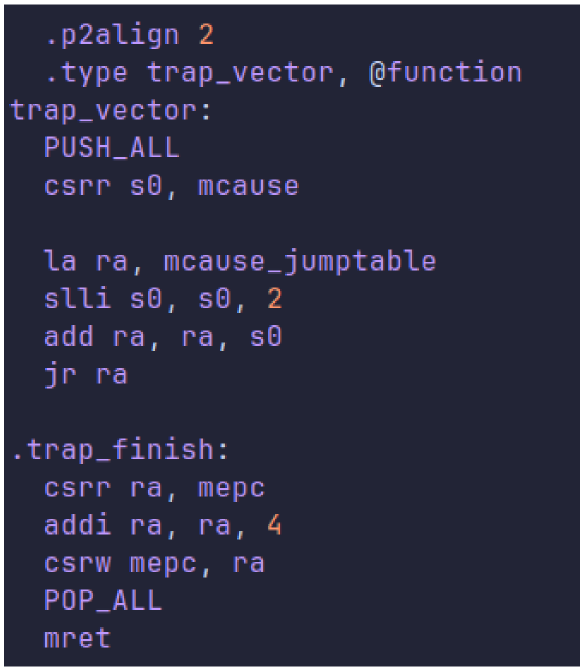
2. **Trap handler: Implement a trap handler function for mcause (SUPERVISOR_ECALL and USER_ECALL) and jump to 1 higher mode e.g if there is USER_ECALL then after handling the trap it should return in Supervisor mode.**

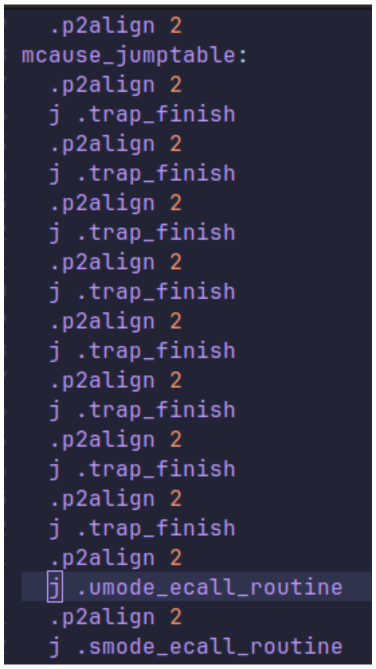
   The trap handler uses a jump table to find the desired routine. As the mcause of USER_ECALL is 8 and SUPERVISOR_ECALL is 9, the mcause value is first shifted by 2 bits and the address is added the the start of the jump table to determine the required routine to be executed.



0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode

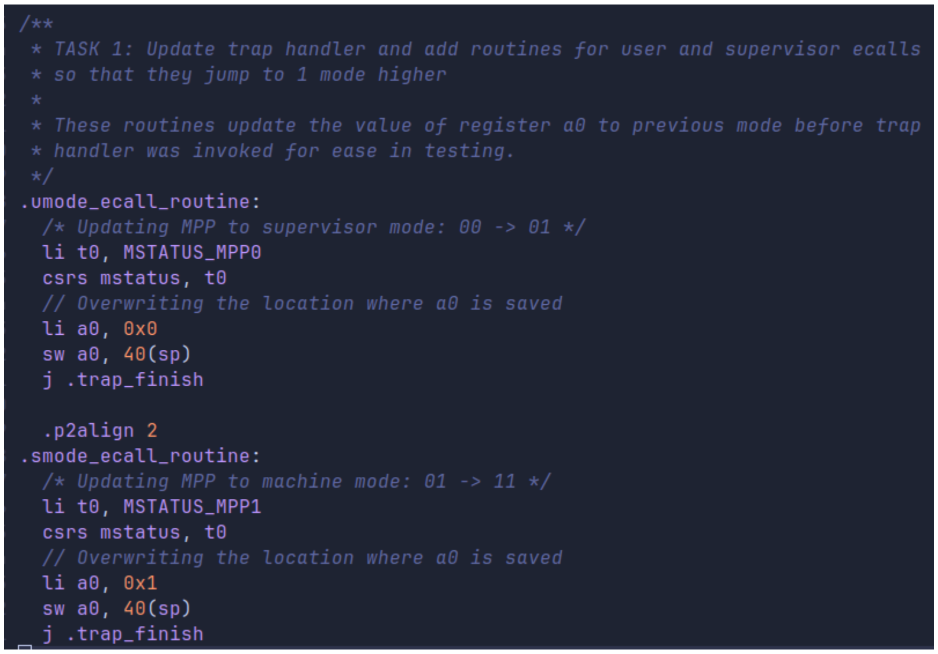
The RISC-V ISA Volume II: Privileged Architecture. Table 14, page 49.


```
  .p2align 2
  .type trap_vector, @function
trap_vector:
  PUSH_ALL
  csrr s0, mcause

  la ra, mcause_jumptable
  slli s0, s0, 2
  add ra, ra, s0
  jr ra

.trap_finish:
  csrr ra, mepc
  addi ra, ra, 4
  csrw mepc, ra
  POP_ALL
  mret
```


```
  .p2align 2
mcause_jumptable:
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .trap_finish
  .p2align 2
  j .umode_ecall_routine
  .p2align 2
  j .smode_ecall_routine
```

Left) trap handler implementation (init.S: lines 34-50)

Right) Jump table: Note that only the required sub-routines are shown. (init.S: lines 52-93)

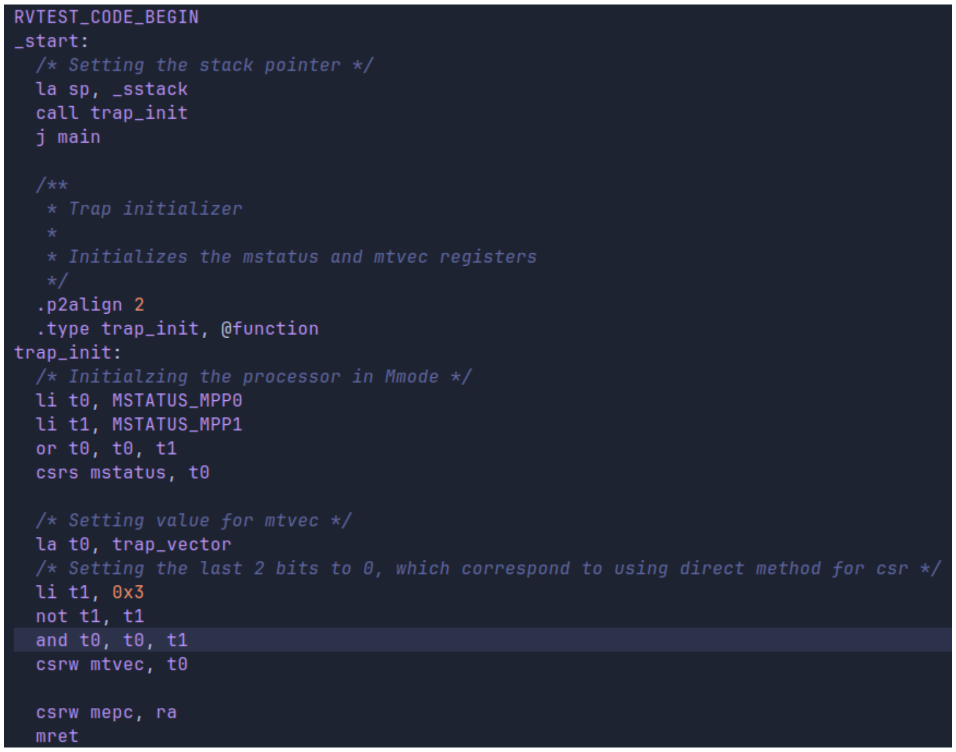

```
/**
 * TASK 1: Update trap handler and add routines for user and supervisor ecalls
 * so that they jump to 1 mode higher
 *
 * These routines update the value of register a0 to previous mode before trap
 * handler was invoked for ease in testing.
 */
.umode_ecall_routine:
  /* Updating MPP to supervisor mode: 00 -> 01 */
  li t0, MSTATUS_MPP0
  csrs mstatus, t0
  // Overwriting the location where a0 is saved
  li a0, 0x0
  sw a0, 40(sp)
  j .trap_finish

  .p2align 2
.smode_ecall_routine:
  /* Updating MPP to machine mode: 01 -> 11 */
  li t0, MSTATUS_MPP1
  csrs mstatus, t0
  // Overwriting the location where a0 is saved
  li a0, 0x1
  sw a0, 40(sp)
  j .trap_finish
```

Routines for user ecall and supervisor ecall. (init.S: lines 102-119)

a. **Write assembly test:**

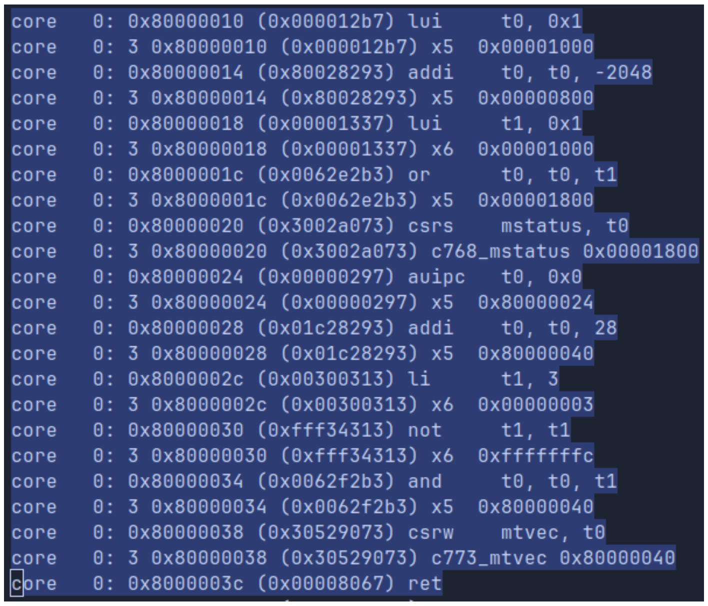
i. **Your test should start in M mode (How can you start your test in M-mode?)**

trap_init sets the MSTATUS MPP bits to 11, which correspond to machine mode. This ensures that when we jump to the main label, our program is executing in machine mode. In the startup we are also initializing the stack pointer and mtvec register.



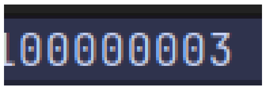
```
RVTEST_CODE_BEGIN
_start:
  /* Setting the stack pointer */
  la sp, _sstack
  call trap_init
  j main

  /**
   * Trap initializer
   *
   * Initializes the mstatus and mtvec registers
   */
  .p2align 2
  .type trap_init, @function
trap_init:
  /* Initialzing the processor in Mmode */
  li t0, MSTATUS_MPP0
  li t1, MSTATUS_MPP1
  or t0, t0, t1
  csrs mstatus, t0

  /* Setting value for mtvec */
  la t0, trap_vector
  /* Setting the last 2 bits to 0, which correspond to using direct method for csr */
  li t1, 0x3
  not t1, t1
  and t0, t0, t1
  csrw mtvec, t0

  csrw mepc, ra
  mret
```

init.S: lines 4-43



```
core   0: 0x80000010 (0x000012b7) lui     t0, 0x1
core   0: 3 0x80000010 (0x000012b7) x5  0x00001000
core   0: 0x80000014 (0x80028293) addi    t0, t0, -2048
core   0: 3 0x80000014 (0x80028293) x5  0x00000800
core   0: 0x80000018 (0x00001337) lui     t1, 0x1
core   0: 3 0x80000018 (0x00001337) x6  0x00001000
core   0: 0x8000001c (0x0062e2b3) or      t0, t0, t1
core   0: 3 0x8000001c (0x0062e2b3) x5  0x00001800
core   0: 0x80000020 (0x3002a073) csrs    mstatus, t0
core   0: 3 0x80000020 (0x3002a073) c768_mstatus 0x00001800
core   0: 0x80000024 (0x00000297) auipc   t0, 0x0
core   0: 3 0x80000024 (0x00000297) x5  0x80000024
core   0: 0x80000028 (0x01c28293) addi    t0, t0, 28
core   0: 3 0x80000028 (0x01c28293) x5  0x80000040
core   0: 0x8000002c (0x00300313) li      t1, 3
core   0: 3 0x8000002c (0x00300313) x6  0x00000003
core   0: 0x80000030 (0xfff34313) not     t1, t1
core   0: 3 0x80000030 (0xfff34313) x6  0xfffffffc
core   0: 0x80000034 (0x0062f2b3) and     t0, t0, t1
core   0: 3 0x80000034 (0x0062f2b3) x5  0x80000040
core   0: 0x80000038 (0x30529073) csrw    mtvec, t0
core   0: 3 0x80000038 (0x30529073) c773_mtvec 0x80000040
core   0: 0x8000003c (0x00008067) ret
```

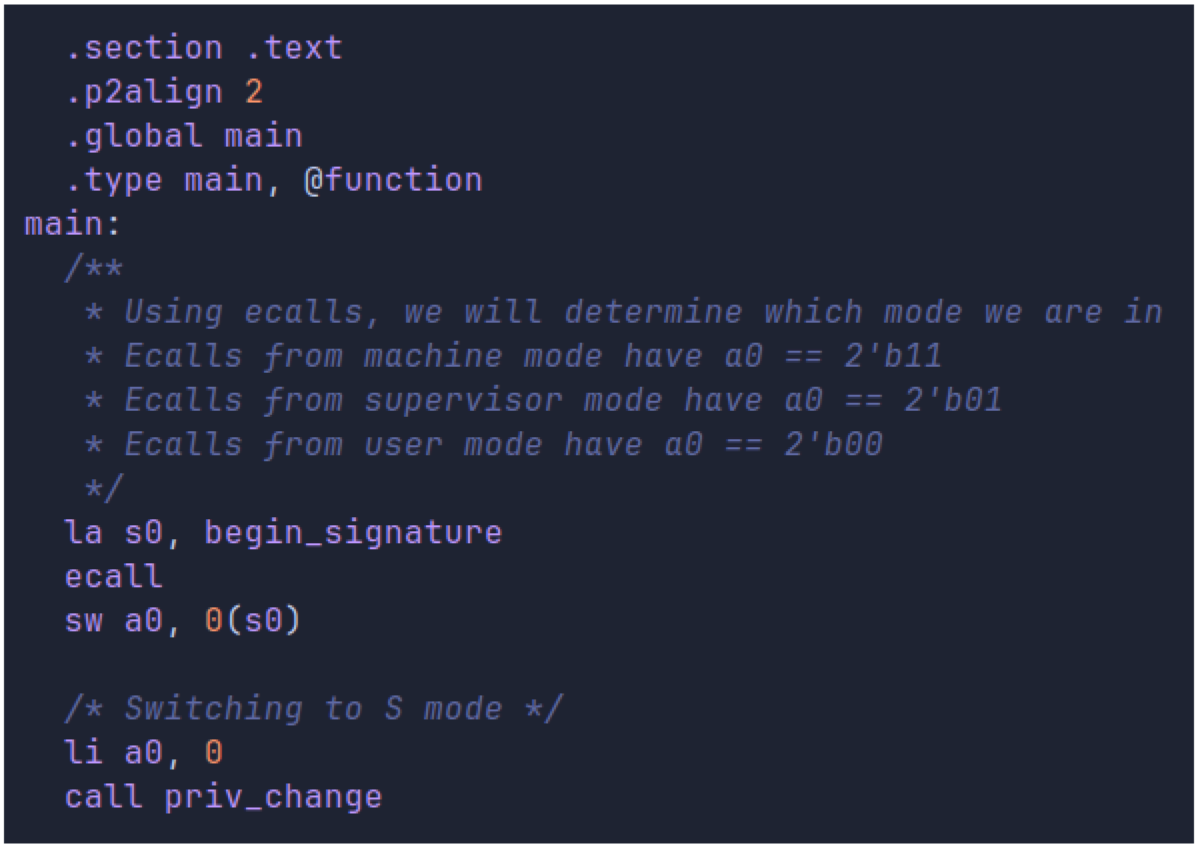
spike.log: lines 18-40



```
100000003
```

Signature.txt: line 1. These are saved after the first ecall returned by main. It shows that mmode_ecall took place as it saves the MPP bits to the signature file.

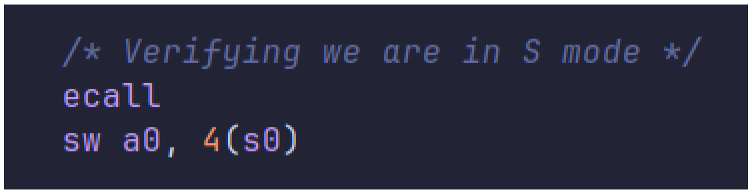
ii. Try switching to S mode



```
  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  /**
   * Using ecalls, we will determine which mode we are in
   * Ecalls from machine mode have a0 == 2'b11
   * Ecalls from supervisor mode have a0 == 2'b01
   * Ecalls from user mode have a0 == 2'b00
   */
  la s0, begin_signature
  ecall
  sw a0, 0(s0)

  /* Switching to S mode */
  li a0, 0
  call priv_change
```

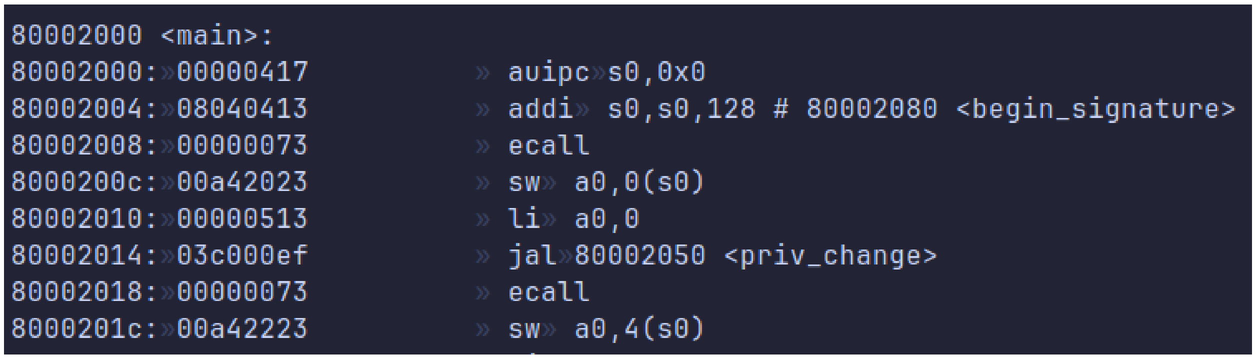
test.S: lines 3-20

iii. Verify that you have switched to the required mode(Figure out How to verify this)



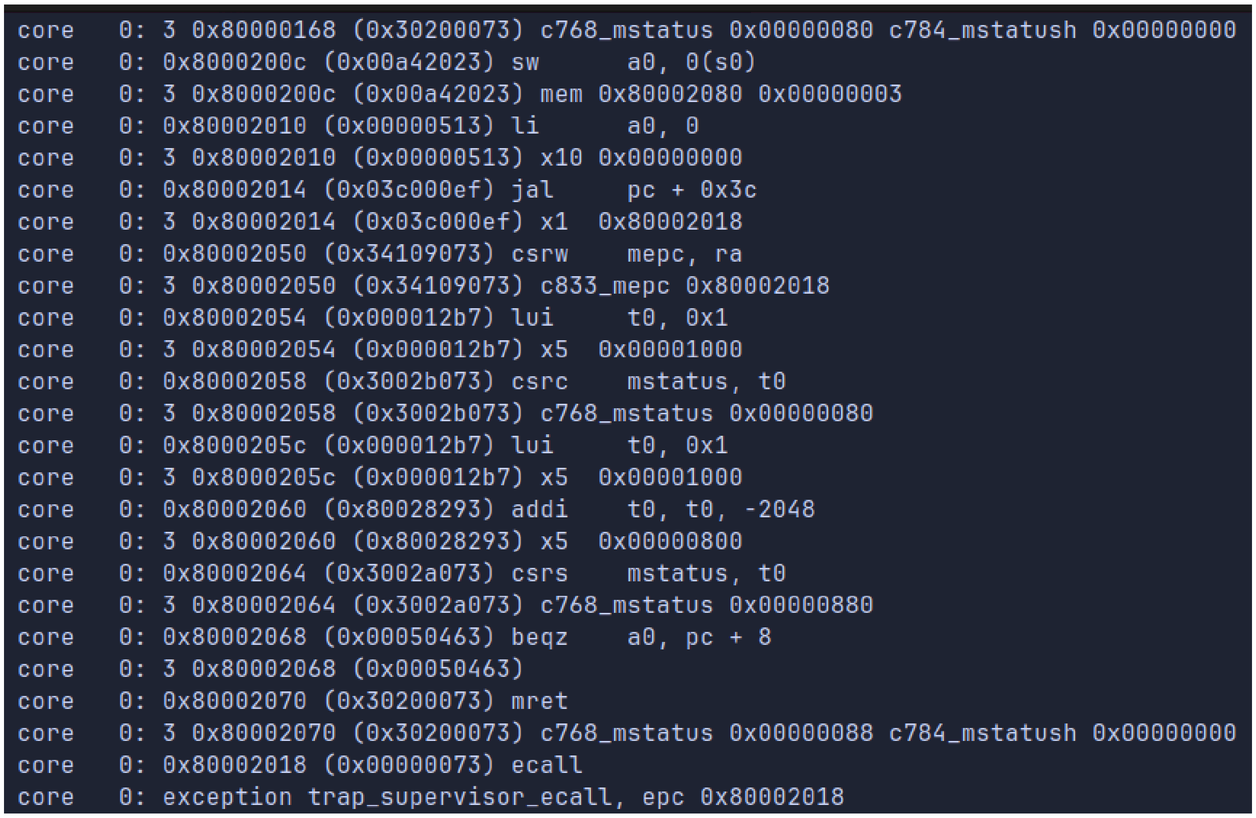
```
  /* Verifying we are in S mode */
  ecall
  sw a0, 4(s0)
```

test.S: lines 30-31



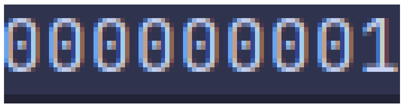
```
80002000 <main>:
80002000:»00000417          »  auipc»s0,0x0
80002004:»08040413          »  addi»  s0,s0,128 # 80002080 <begin_signature>
80002008:»00000073          »  ecall
8000200c:»00a42023          »  sw»  a0,0(s0)
80002010:»00000513          »  li»  a0,0
80002014:»03c000ef          »  jal»80002050 <priv_change>
80002018:»00000073          »  ecall
8000201c:»00a42223          »  sw»  a0,4(s0)
```

test.disass: lines 200-216



```
core   0: 3 0x80000168 (0x30200073) c768_mstatus 0x00000080 c784_mstatush 0x00000000
core   0: 0x8000200c (0x00a42023) sw      a0, 0(s0)
core   0: 3 0x8000200c (0x00a42023) mem 0x80002080 0x00000003
core   0: 0x80002010 (0x00000513) li      a0, 0
core   0: 3 0x80002010 (0x00000513) x10 0x00000000
core   0: 0x80002014 (0x03c000ef) jal     pc + 0x3c
core   0: 3 0x80002014 (0x03c000ef) x1  0x80002018
core   0: 0x80002050 (0x34109073) csrw    mepc, ra
core   0: 3 0x80002050 (0x34109073) c833_mepc 0x80002018
core   0: 0x80002054 (0x000012b7) lui     t0, 0x1
core   0: 3 0x80002054 (0x000012b7) x5  0x00001000
core   0: 0x80002058 (0x3002b073) csrc    mstatus, t0
core   0: 3 0x80002058 (0x3002b073) c768_mstatus 0x00000080
core   0: 0x8000205c (0x000012b7) lui     t0, 0x1
core   0: 3 0x8000205c (0x000012b7) x5  0x00001000
core   0: 0x80002060 (0x80028293) addi    t0, t0, -2048
core   0: 3 0x80002060 (0x80028293) x5  0x00000800
core   0: 0x80002064 (0x3002a073) csrs    mstatus, t0
core   0: 3 0x80002064 (0x3002a073) c768_mstatus 0x00000880
core   0: 0x80002068 (0x00050463) beqz    a0, pc + 8
core   0: 3 0x80002068 (0x00050463)
core   0: 0x80002070 (0x30200073) mret
core   0: 3 0x80002070 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
core   0: 0x80002018 (0x00000073) ecall
core   0: exception trap_supervisor_ecall, epc 0x80002018
```

spike.log: lines 208-232. Lines corresponding test.diass shown above. Note the last line: trap_supervisor_ecall.

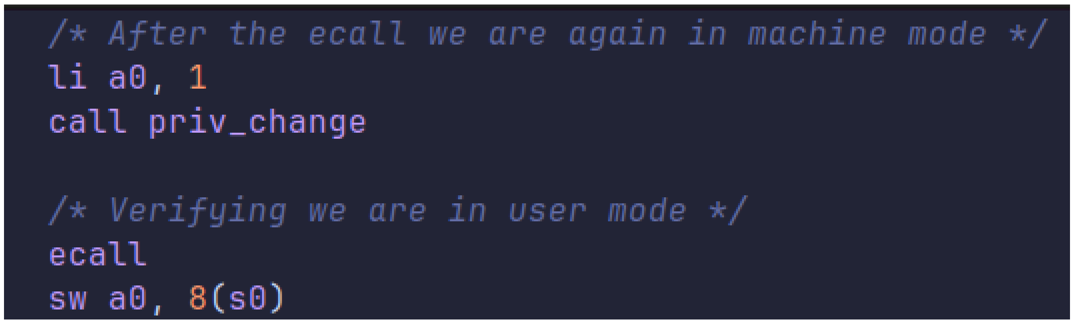


```
00000001
```

signature.txt: 2nd word (line 2). It shows that after the ecall 01 was saved in the 2nd word of signature, showing us that program was in supervisor mode before ecall.

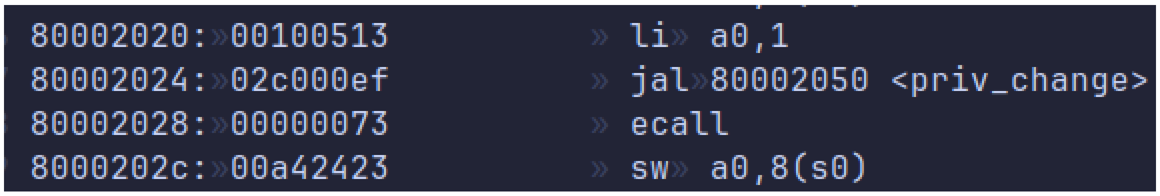
iv. **Try switching to U mode. (figure out how you can do this using standard way) Note: You are currently in S mode.**

By using the priv_change function, we are able to go to user mode.



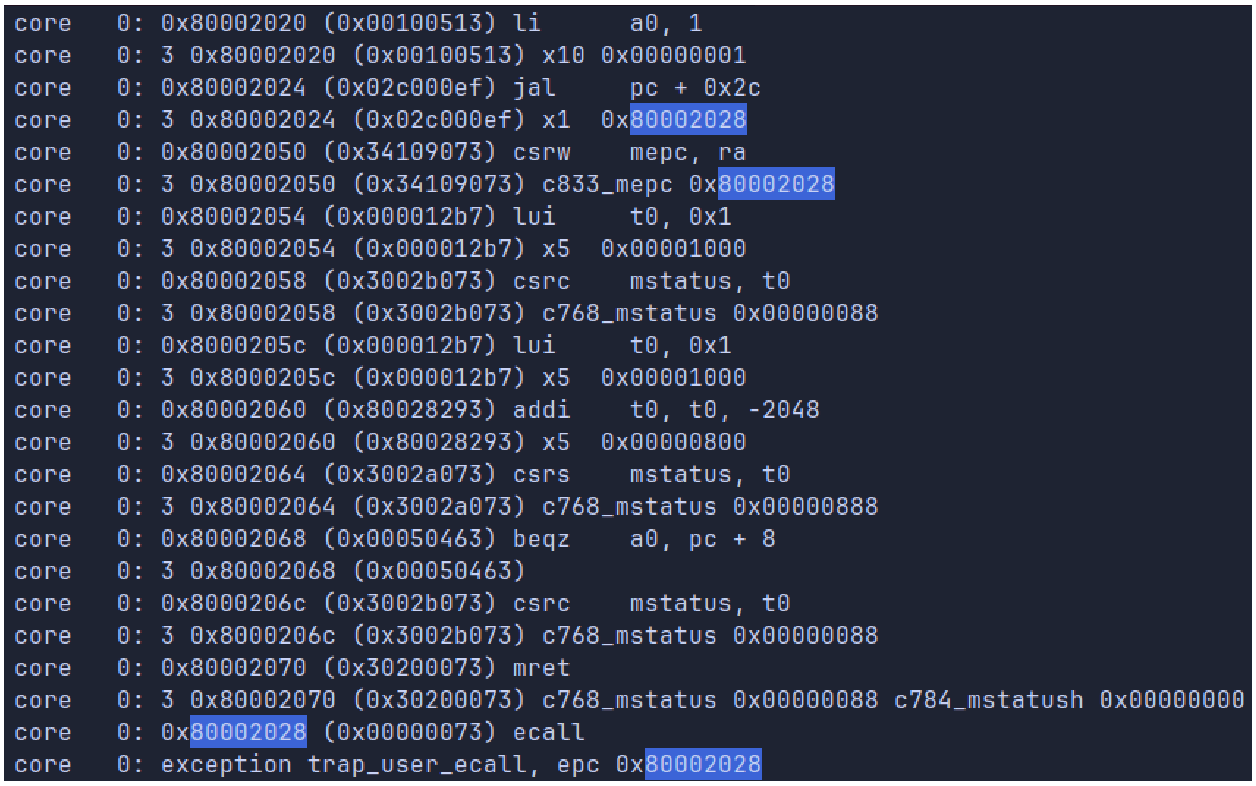
```
/* After the ecall we are again in machine mode */
li a0, 1
call priv_change

/* Verifying we are in user mode */
ecall
sw a0, 8(s0)
```

test.S: lines 26-32

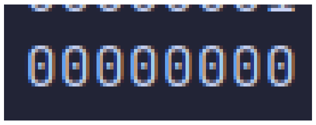


```
80002020:»00100513          » li» a0,1
80002024:»02c000ef          » jal»80002050 <priv_change>
80002028:»00000073          » ecall
8000202c:»00a42423          » sw» a0,8(s0)
```

test.disass: lines 209-213



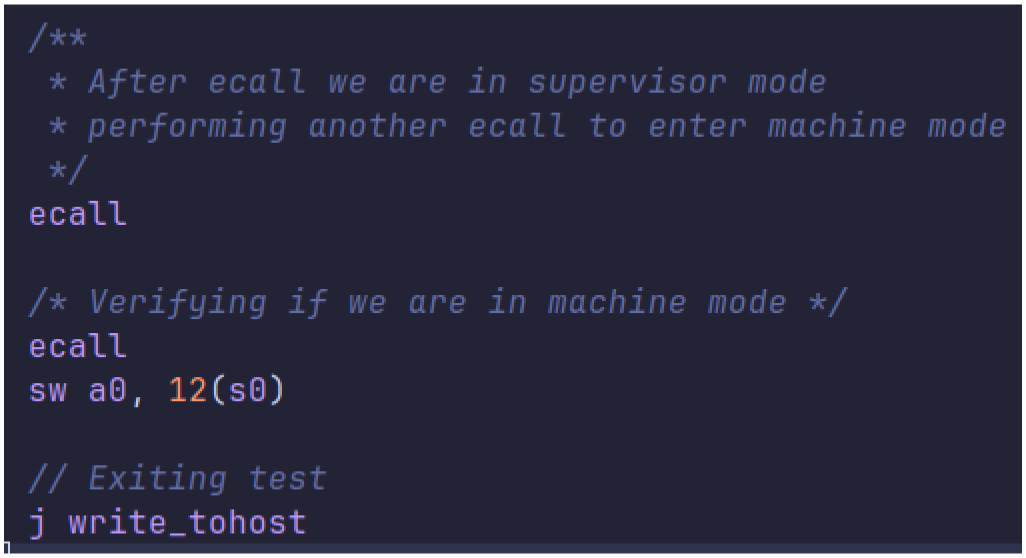
```
core   0: 0x80002020 (0x00100513) li      a0, 1
core   0: 3 0x80002020 (0x00100513) x10 0x00000001
core   0: 0x80002024 (0x02c000ef) jal     pc + 0x2c
core   0: 3 0x80002024 (0x02c000ef) x1  0x80002028
core   0: 0x80002050 (0x34109073) csrw    mepc, ra
core   0: 3 0x80002050 (0x34109073) c833_mepc 0x80002028
core   0: 0x80002054 (0x000012b7) lui     t0, 0x1
core   0: 3 0x80002054 (0x000012b7) x5  0x00001000
core   0: 0x80002058 (0x3002b073) csrc    mstatus, t0
core   0: 3 0x80002058 (0x3002b073) c768_mstatus 0x00000088
core   0: 0x8000205c (0x000012b7) lui     t0, 0x1
core   0: 3 0x8000205c (0x000012b7) x5  0x00001000
core   0: 0x80002060 (0x80028293) addi    t0, t0, -2048
core   0: 3 0x80002060 (0x80028293) x5  0x00000800
core   0: 0x80002064 (0x3002a073) csrs    mstatus, t0
core   0: 3 0x80002064 (0x3002a073) c768_mstatus 0x00000888
core   0: 0x80002068 (0x00050463) beqz    a0, pc + 8
core   0: 3 0x80002068 (0x00050463)
core   0: 0x8000206c (0x3002b073) csrc    mstatus, t0
core   0: 3 0x8000206c (0x3002b073) c768_mstatus 0x00000088
core   0: 0x80002070 (0x30200073) mret
core   0: 3 0x80002070 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
core   0: 0x80002028 (0x00000073) ecall
core   0: exception trap_user_ecall, epc 0x80002028
```



```
00000000
```

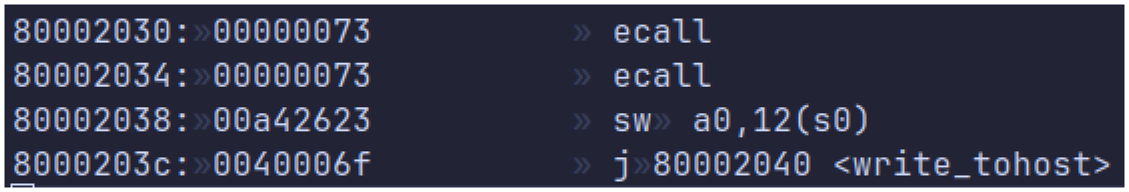
signature.txt: line 3. It shows that all 0's was saved. Telling us that before the ecall we were in user mode.

v. Now switch to M mode and exit the test. (always exit your test in M mode)



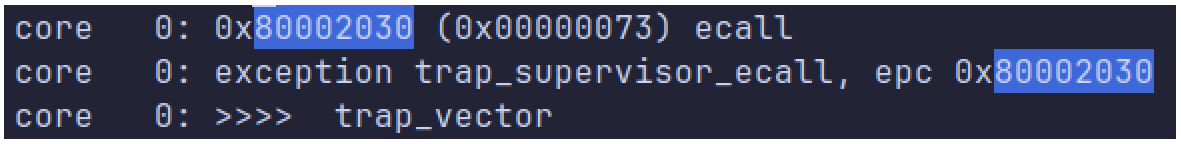
```
/**
 * After ecall we are in supervisor mode
 * performing another ecall to enter machine mode
 */
ecall

/* Verifying if we are in machine mode */
ecall
sw a0, 12(s0)

// Exiting test
j write_tohost
```

test.S: lines 34-45



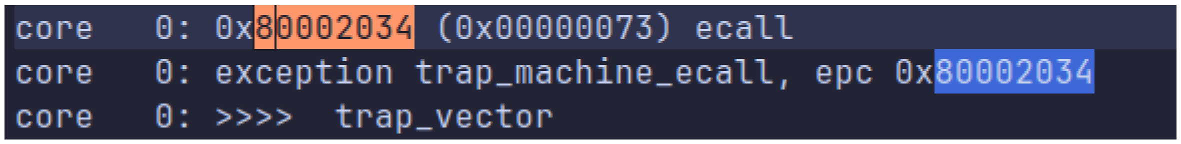
```
80002030:»00000073          »  ecall
80002034:»00000073          »  ecall
80002038:»00a42623          »  sw»  a0,12(s0)
8000203c:»0040006f          »  j»80002040 <write_tohost>
```

test.disass: lines 211-216



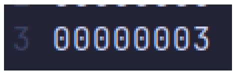
```
core   0: 0x80002030 (0x00000073) ecall
core   0: exception trap_supervisor_ecall, epc 0x80002030
core   0: >>>>  trap_vector
```

spike.log: lines 585 - 587. This is the situation after the first ecall, where we change mode from supervisor to machine mode



```
core   0: 0x80002034 (0x00000073) ecall
core   0: exception trap_machine_ecall, epc 0x80002034
core   0: >>>>  trap_vector
```

spike.log: lines 748-750. The second ecall to write into signature that the mode we are in is machine mode.



```
3 00000003
```

signature.txt: line 4. It shows that the mode before the ecall is machine mode. As the machine mode ecall does not change MPP, mret also returns in machine mode. Ensuring that before we enter the write host loop, we are in Machine Mode.