Author : Irtaza Hyder

Date : 2026-02-06

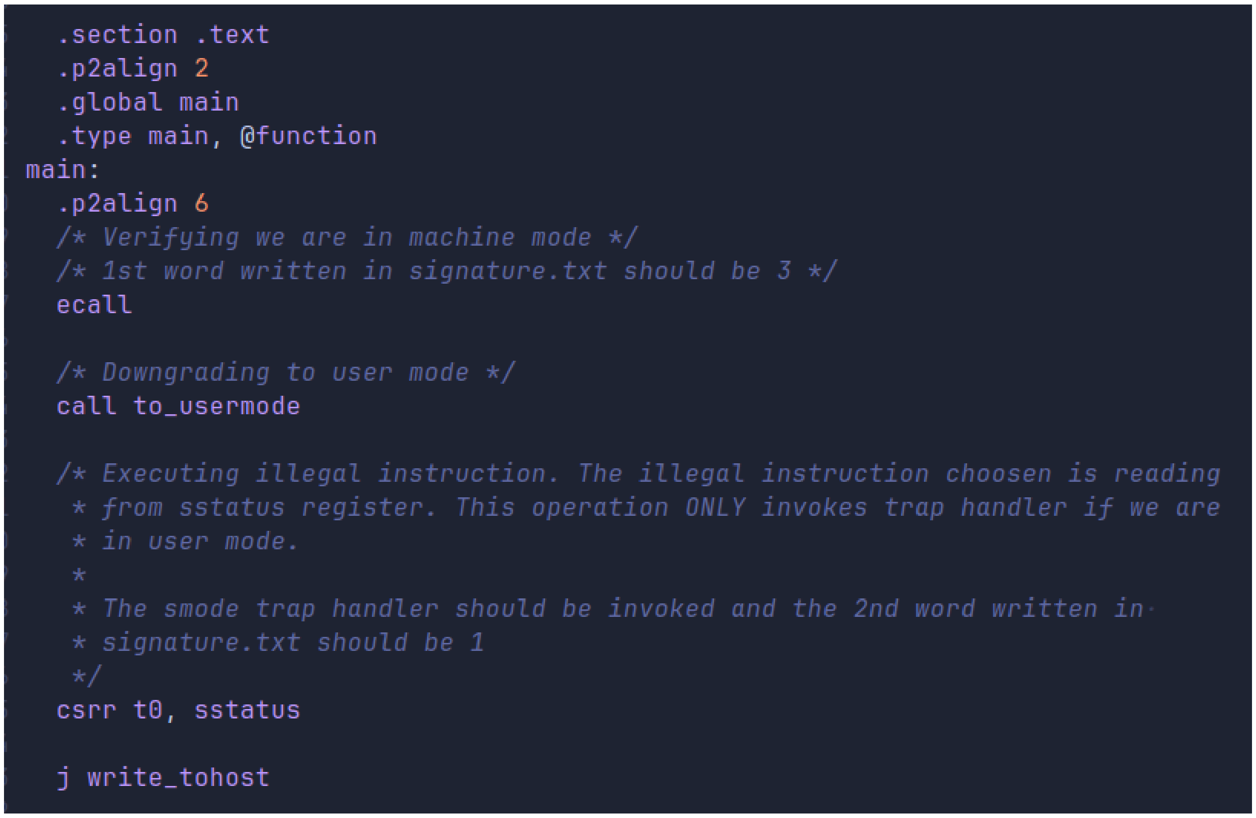
Module : RISC-V Arch Test

Section : RISC-V Arch Test

Task Name : Task 3

Description: Write an assembly test to handle exceptions in S mode

Test Description



```
  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  .p2align 6
  /* Verifying we are in machine mode */
  /* 1st word written in signature.txt should be 3 */
  ecall

  /* Downgrading to user mode */
  call to_usermode

  /* Executing illegal instruction. The illegal instruction choosen is reading
   * from sstatus register. This operation ONLY invokes trap handler if we are
   * in user mode.
   *
   * The smode trap handler should be invoked and the 2nd word written in
   * signature.txt should be 1
   */
  csrr t0, sstatus

  j write_tohost
```

test.S: lines 3-25.

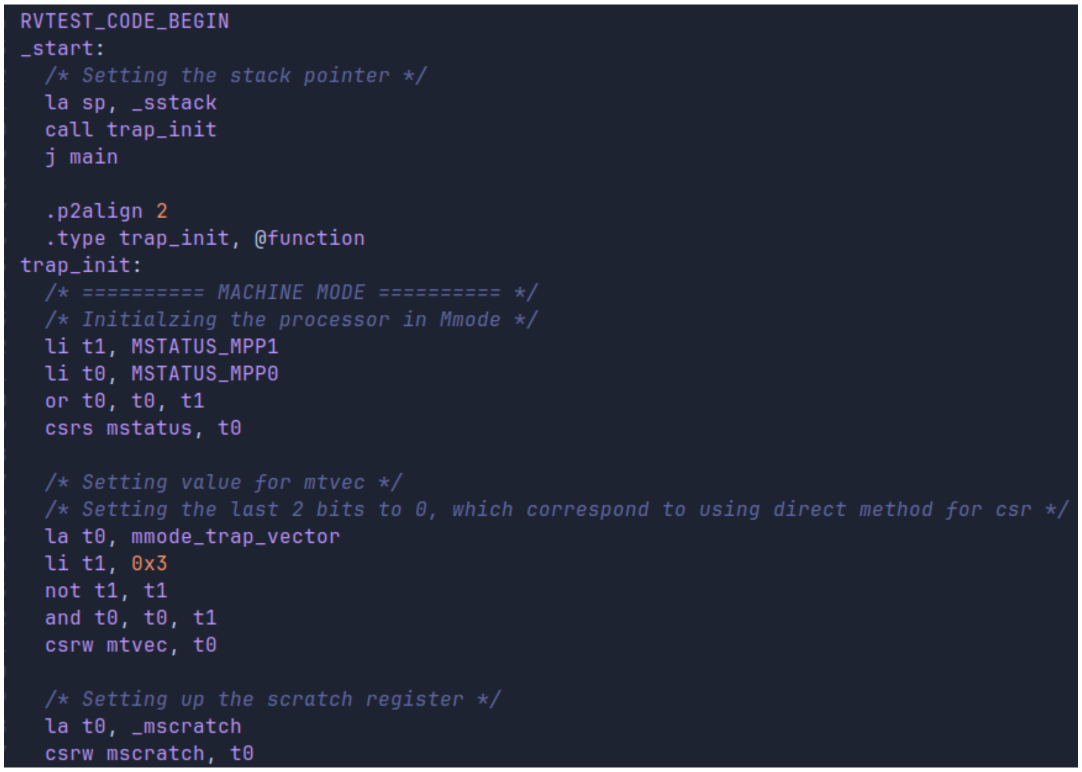
The machine mode and supervisor mode write a single byte ‘03’ and ‘01’ to the signature file respectively, whenever their respective trap handlers are invoked. In the test shown above we first invoke the machine handler by using ecall (in this case generating a MACHINE ECALL).

After that we switch to user mode and then generate an illegal instruction. The CSR sstatus is used to show that we are in user mode. As the instruction is legal in both supervisor and machine mode, Illegal instruction exception is ONLY generated if our program is in user mode.

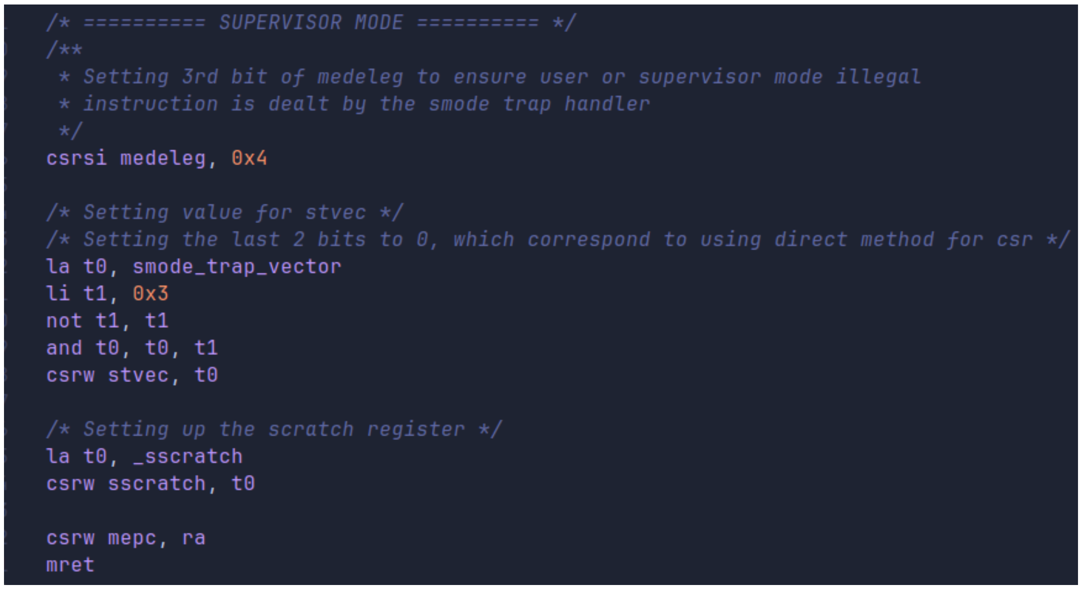
No explicit trap handling routines are programmed. They only write to signature.txt ‘03’ or ‘01’ based on which trap handler has been invoked.

Output and Q&A

1. **Start your test in M mode and then switch to U mode.**



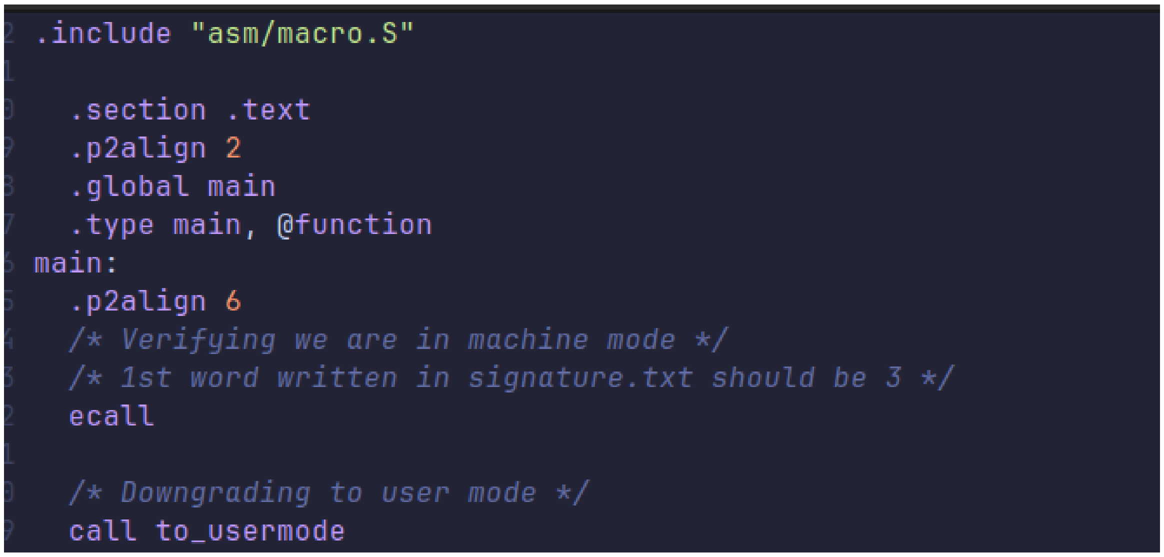
```
RVTEST_CODE_BEGIN
_start:
  /* Setting the stack pointer */
  la sp, _sstack
  call trap_init
  j main

  .p2align 2
  .type trap_init, @function
trap_init:
  /* ========== MACHINE MODE ========== */
  /* Initialzing the processor in Mmode */
  li t1, MSTATUS_MPP1
  li t0, MSTATUS_MPP0
  or t0, t0, t1
  csrs mstatus, t0

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

  /* Setting up the scratch register */
  la t0, _mscratch
  csrw mscratch, t0
```



```
  /* ========== SUPERVISOR MODE ========== */
  /**
   * Setting 3rd bit of medeleg to ensure user or supervisor mode illegal
   * instruction is dealt by the smode trap handler
   */
  csrsi medeleg, 0x4

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

  /* Setting up the scratch register */
  la t0, _sscratch
  csrw sscratch, t0

  csrw mepc, ra
  mret
```

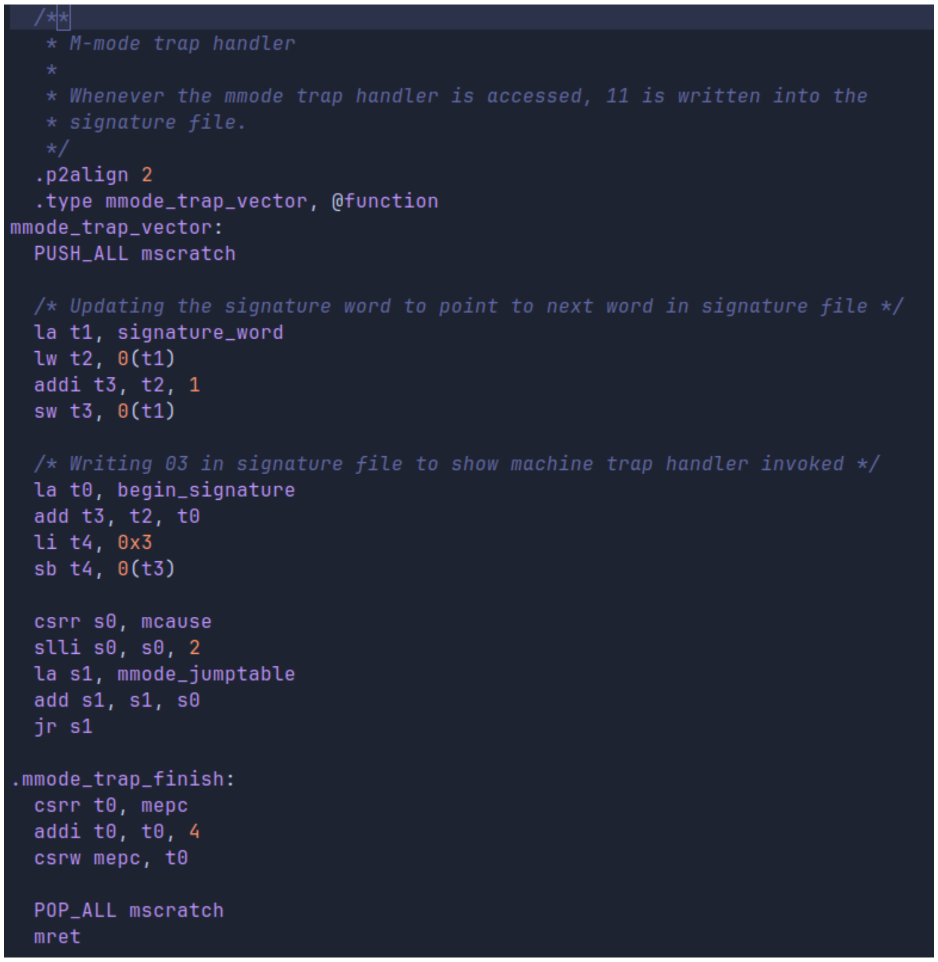
init.S: lines 3-52

The required machine mode and smode registers are set as per requirement of the question. mstatus MPP is set to 11 to ensure the program starts in machine mode. Note the initialization of CSR medeleg in line 37. By setting the 3rd bit to 1, we enable illegal instruction exceptions in user and supervisor modes to be dealt with using the supervisor trap handler.



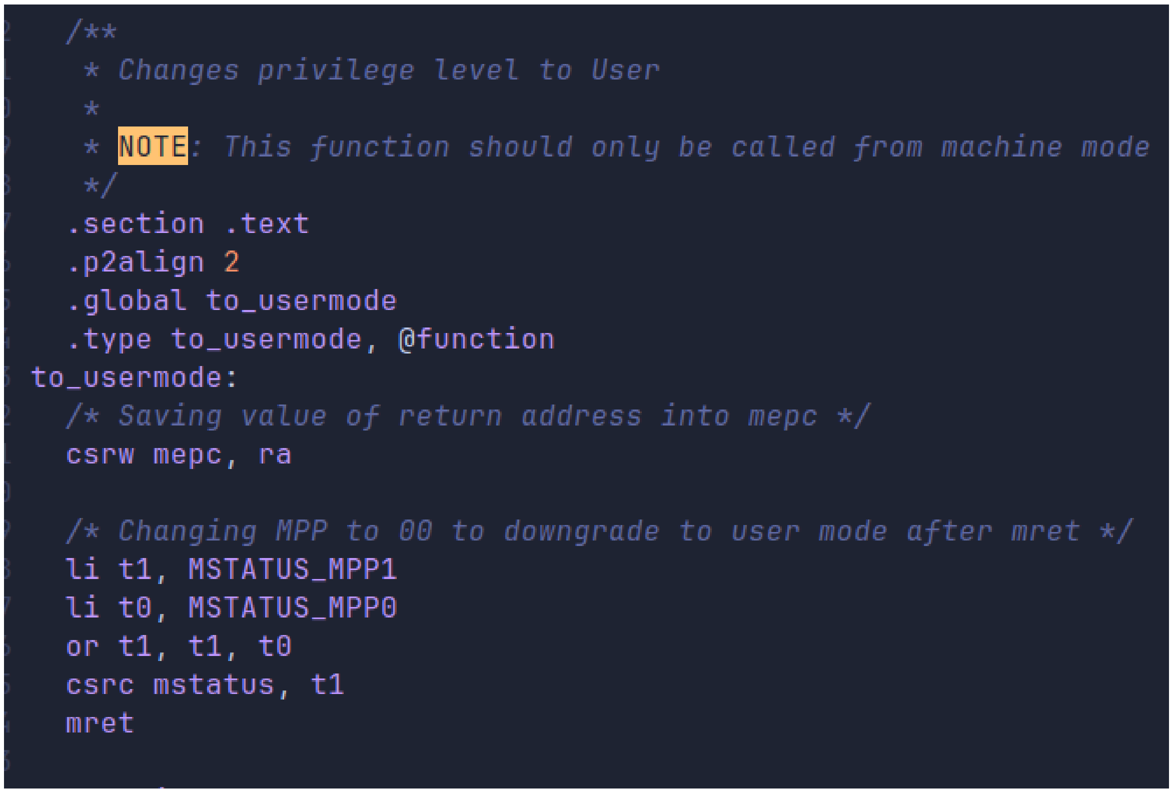
```
.include "asm/macro.S"

  .section .text
  .p2align 2
  .global main
  .type main, @function
main:
  .p2align 6
  /* Verifying we are in machine mode */
  /* 1st word written in signature.txt should be 3 */
  ecall

  /* Downgrading to user mode */
  call to_usermode
```

Main function invoking to_usermode function. test.S: line 14. Note the first ecall invokes the machine mode trap handler.



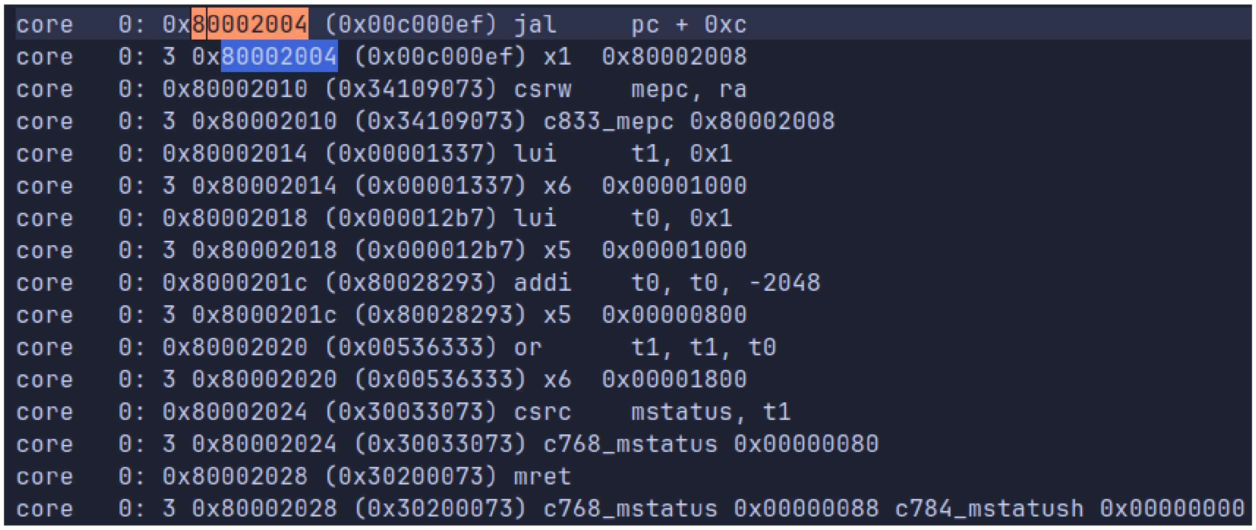
```
  /**
   * M-mode trap handler
   *
   * Whenever the mmode trap handler is accessed, 11 is written into the
   * signature file.
   */
  .p2align 2
  .type mmode_trap_vector, @function
mmode_trap_vector:
  PUSH_ALL mscratch

  /* Updating the signature word to point to next word in signature file */
  la t1, signature_word
  lw t2, 0(t1)
  addi t3, t2, 1
  sw t3, 0(t1)

  /* Writing 03 in signature file to show machine trap handler invoked */
  la t0, begin_signature
  add t3, t2, t0
  li t4, 0x3
  sb t4, 0(t3)

  csrr s0, mcause
  slli s0, s0, 2
  la s1, mmode_jumptable
  add s1, s1, s0
  jr s1

.mmode_trap_finish:
  csrr t0, mepc
  addi t0, t0, 4
  csrw mepc, t0

  POP_ALL mscratch
  mret
```

M mode trap handler. Whenever it is invoked it writes byte ‘03’ in signature.txt.



```
  /**
   * Changes privilege level to User
   *
   * NOTE: This function should only be called from machine mode
   */
  .section .text
  .p2align 2
  .global to_usermode
  .type to_usermode, @function
to_usermode:
  /* Saving value of return address into mepc */
  csrw mepc, ra

  /* Changing MPP to 00 to downgrade to user mode after mret */
  li t1, MSTATUS_MPP1
  li t0, MSTATUS_MPP0
  or t1, t1, t0
  csrc mstatus, t1
  mret
```

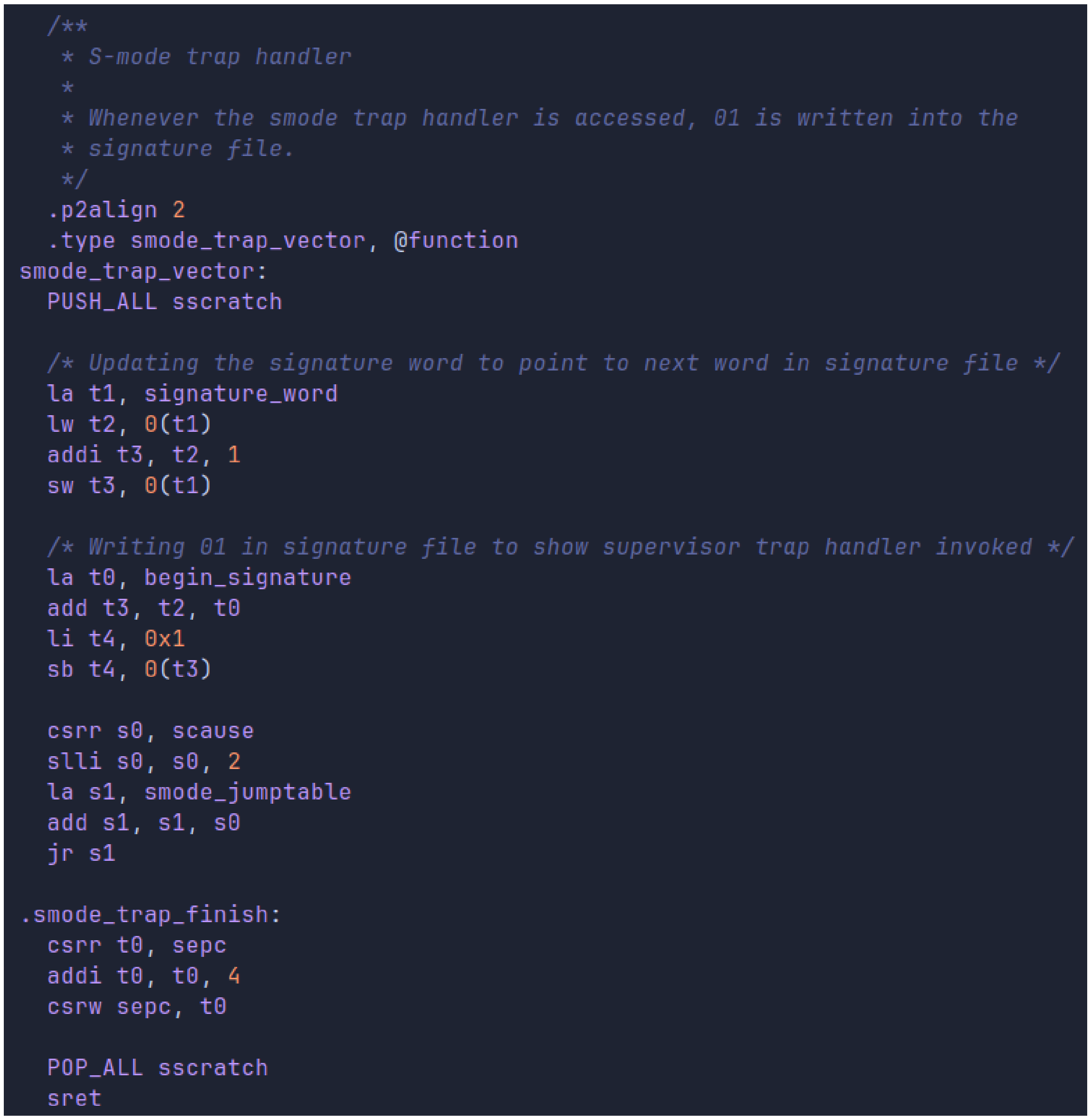
to_usermode function: test.S: lines 27-45. The value of MPP is updated to 00, and then we call mret. This switches the program execution to user mode.



```
core   0: 0x80002004 (0x00c000ef) jal     pc + 0xc
core   0: 3 0x80002004 (0x00c000ef) x1  0x80002008
core   0: 0x80002010 (0x34109073) csrw    mepc, ra
core   0: 3 0x80002010 (0x34109073) c833_mepc 0x80002008
core   0: 0x80002014 (0x00001337) lui     t1, 0x1
core   0: 3 0x80002014 (0x00001337) x6  0x00001000
core   0: 0x80002018 (0x000012b7) lui     t0, 0x1
core   0: 3 0x80002018 (0x000012b7) x5  0x00001000
core   0: 0x8000201c (0x80028293) addi    t0, t0, -2048
core   0: 3 0x8000201c (0x80028293) x5  0x00000800
core   0: 0x80002020 (0x00536333) or      t1, t1, t0
core   0: 3 0x80002020 (0x00536333) x6  0x00001800
core   0: 0x80002024 (0x30033073) csrc    mstatus, t1
core   0: 3 0x80002024 (0x30033073) c768_mstatus 0x00000080
core   0: 0x80002028 (0x30200073) mret
core   0: 3 0x80002028 (0x30200073) c768_mstatus 0x00000088 c784_mstatush 0x00000000
```

spike.log: lines 249-264. It shows that mstatus is now configured in user mode.

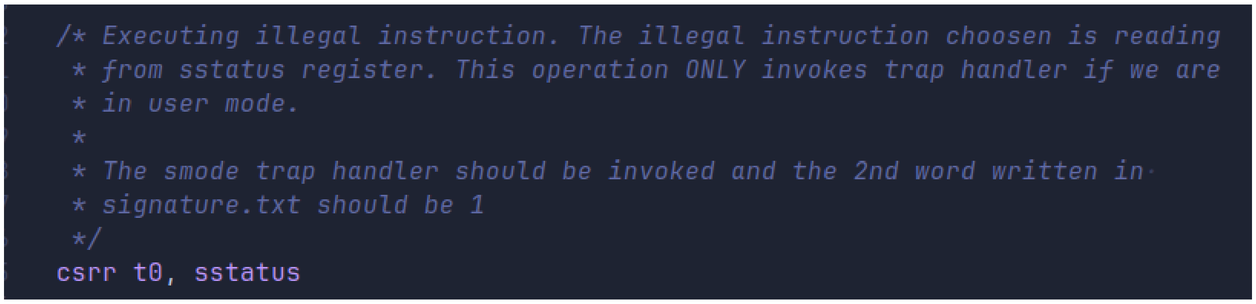
2. **Implement a trap handler for S mode to handle exceptions.**



```
  /**
   * S-mode trap handler
   *
   * Whenever the smode trap handler is accessed, 01 is written into the
   * signature file.
   */
  .p2align 2
  .type smode_trap_vector, @function
smode_trap_vector:
  PUSH_ALL sscratch

  /* Updating the signature word to point to next word in signature file */
  la t1, signature_word
  lw t2, 0(t1)
  addi t3, t2, 1
  sw t3, 0(t1)

  /* Writing 01 in signature file to show supervisor trap handler invoked */
  la t0, begin_signature
  add t3, t2, t0
  li t4, 0x1
  sb t4, 0(t3)

  csrr s0, scause
  slli s0, s0, 2
  la s1, smode_jumptable
  add s1, s1, s0
  jr s1

.smode_trap_finish:
  csrr t0, sepc
  addi t0, t0, 4
  csrw sepc, t0

  POP_ALL sscratch
  sret
```

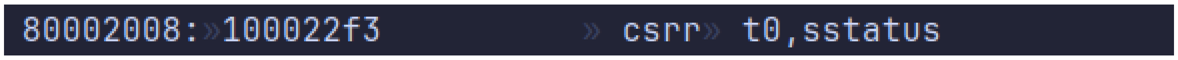
init.S: lines 91-126. Whenever the smode trap handler is invoked, it writes byte '01' in the signature.txt.

3. **Generate an illegal instruction exception and handle this exception in S mode (all the exceptions are by default handled in M mode).**



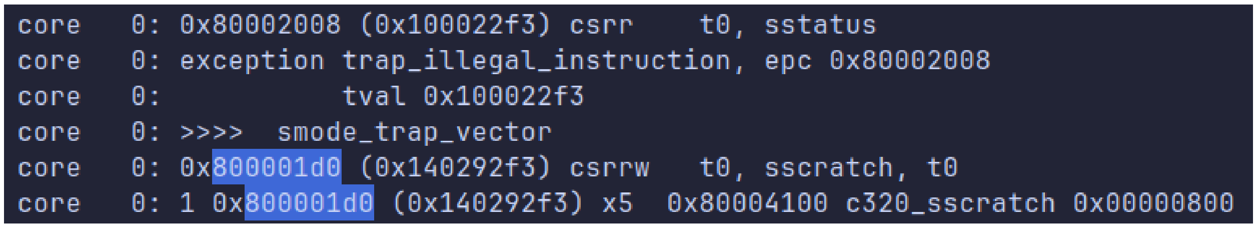
```
/* Executing illegal instruction. The illegal instruction choosen is reading
 * from sstatus register. This operation ONLY invokes trap handler if we are
 * in user mode.
 *
 * The smode trap handler should be invoked and the 2nd word written in
 * signature.txt should be 1
 */
csrr t0, sstatus
```

test.S: lines 23. Generating an illegal instruction in user mode. Reading sstatus is illegal due to downgraded privileges. If this does not generate an exception, we know that we are not in USER mode.



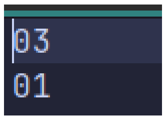
```
80002008:»100022f3          » csrr» t0,sstatus
```

test.disass: line 318



```
core   0: 0x80002008 (0x100022f3) csrr    t0, sstatus
core   0: exception trap_illegal_instruction, epc 0x80002008
core   0:           tval 0x100022f3
core   0: >>>>  smode_trap_vector
core   0: 0x800001d0 (0x140292f3) csrrw   t0, sscratch, t0
core   0: 1 0x800001d0 (0x140292f3) x5  0x80004100 c320_sscratch 0x00000800
```

spike.log: lines 265-270. It shows upon hitting the csrr t0, status instruction, smode_trap_vector is as expected.



```
03
01
```

signature.txt. The 1st line shows '03', which is due to the mmode trap handler being invoked first (due to the first ecall command in main). The 2nd line shows '01', which tells us that the supervisor trap handler was invoked afterwards (due to the illegal instruction exception generated in user mode).