FORCE-RISCV Support for RISC-V Trap Delegation, Trap Redirection

Introduction

Enhance **FORCE-RISCV** to support the RISC-V *Trap Delegation* feature. Include support for software *Trap Redirection*, as well.

This document describes the **FORCE-RISCV** implemtation for the *Trap Delegation* and the *Trap Redirection* features.

Trap Delegation

Support for the RISC-V hardware *trap delegation*, ie, the Machine-mode *medeleg* register, has been implemented in **FORCE-RISCV***. Register fields and register field choices are in place for the *medeleg* register.

Heres an excerpt from the `riscv/arch_data/system_registers.xml` that displays a few of the *medeleg* register fields added:

```
<register boot="1" index="0x302" name="medeleg" size="64" type="SysReg">
  <register_field name="RESERVED 1" physical_register="medeleg" size="8">
    <bit_field shift="16" size="8"/>
  </register_field>
  <register_field name="CUSTOM USE 1" physical_register="medeleg" size="8">
    <bit_field shift="24" size="8"/>
  </register_field>
  <register_field name="RESERVED 2" physical_register="medeleg" size="16">
    <bit_field shift="32" size="16"/>
  </register_field>
  <register_field name="CUSTOM USE 2" physical_register="medeleg" size="16">
    <bit_field shift="48" size="16"/>
  </register_field>
  <register_field name="Instruction address misaligned" physical_register="medeleg" size="1">
    <bit_field shift="0" size="1"/>
  </register_field>
  <register_field name="Instruction access fault" physical_register="medeleg" size="1">
    <bit_field shift="1" size="1"/>
  </register_field>
  <register_field name="Illegal instruction" physical_register="medeleg" size="1">
    <bit_field shift="2" size="1"/>
  </register_field>
  <register_field name="Breakpoint" physical_register="medeleg" size="1">
    <bit_field shift="3" size="1"/>
  </register_field>
  <register_field name="Load address misaligned" physical_register="medeleg" size="1">
```

As can be inferred from the above, each bit in the medeleg register, starting with the least significate bit, controls a single synchronous exception. When set, an exception normally taken in Machine mode is instead taken in Supervisor mode.

Also, from the `riscv/arch_data/register_field_choices.xml` file:

```
</choices>
<choices name="medeleg.Instruction address misaligned" type="RegisterFieldValue">
  <choice description="Do NOT delegate to S-mode" value="0x0" weight="10"/>
  <choice description="Delegate to S-mode" value="0x1" weight="0"/>
</choices>
<choices name="medeleg.Instruction access fault" type="RegisterFieldValue">
  <choice description="Do NOT delegate to S-mode" value="0x0" weight="10"/>
  <choice description="Delegate to S-mode" value="0x1" weight="0"/>
</choices>
<choices name="medeleg.Illegal instruction" type="RegisterFieldValue">
  <choice description="Do NOT delegate to S-mode" value="0x0" weight="10"/>
  <choice description="Delegate to S-mode" value="0x1" weight="0"/>
</choices>
<choices name="medeleg.Breakpoint" type="RegisterFieldValue">
  <choice description="Do NOT delegate to S-mode" value="0x0" weight="10"/>
  <choice description="Delegate to S-mode" value="0x1" weight="0"/>
</choices>
<choices name="medeleg.Load address misaligned" type="RegisterFieldValue">
  <choice description="Do NOT delegate to S-mode" value="0x0" weight="10"/>
  <choice description="Delegate to S-mode" value="0x1" weight="0"/>
</choices>
<choices name="medeleg.Load access fault" type="RegisterFieldValue">
```

For each *medeleg* register field, there is a corresponding set of choices, to indicate whether or not to enable delegation, ie, set a bit in the *medeleg* CSR.

Existing and new test templates have been implemented that exercise *trap delegation*, for the (synchronous) exceptions that the simulator used by FORCE-RISCV, ie, *spike*, currently supports. Choice modifiers may be used at test/thread initialization time, to control which exceptions to delegate and how often.

Example test thread initialization sequence:

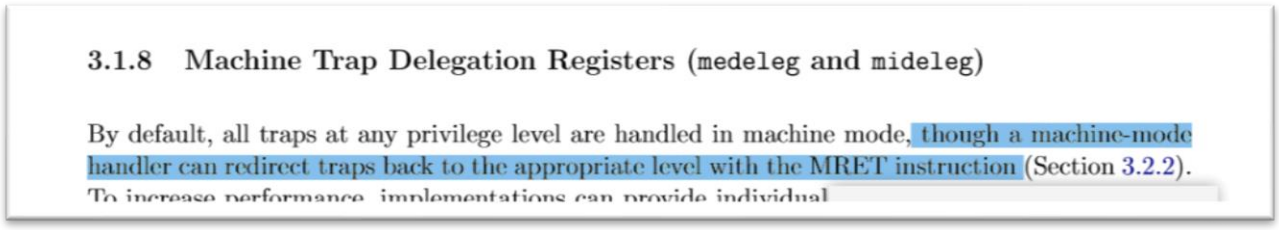
```
def gen_thread_initialization(gen_thread):
    (delegate_opt, valid) = gen_thread.getOption("DelegateExceptions")
    if valid and delegate_opt == 1:
        delegation_enables = ChoicesModifier(gen_thread)
        weightDict = { "0x0":0, "0x1":50 }
        delegation_enables.modifyRegisterFieldValueChoices( 'medeleg.Breakpoint', weightDict )
        delegation_enables.modifyRegisterFieldValueChoices( 'medeleg.Environment call from U-mode', weightDict )
        delegation_enables.commitSet()
```

*note: RISC-V asynchronous (interrupt) *trap delegation* is supported via the *mideleg* register. Support for same in **FORCE-RISCV** is TBD.

The above discussion focuses on setting up trap delegation at test/thread initialization time. A test that starts out generating in Machine mode could of course directly read/modify/write back the *medeleg* CSR to enable/disable trap delegation for select exceptions, and then employ the **FORCE-RISCV** *System Call*, to switch to a subordinate privilege level, before generating exception causing instruction sequences.

Trap Redirection

Trap Redirection, as opposed to trap delegation, is performed by software. From the RISC-V Privileged spec:



3.1.8 Machine Trap Delegation Registers (medeleg and mideleg)

By default, all traps at any privilege level are handled in machine mode, though a machine-mode handler can redirect traps back to the appropriate level with the MRET instruction (Section 3.2.2). To increase performance, implementations can provide individual

Code is issued in Machine mode, as part of an exception handling sequence, to redirect the trap* from Machine mode, to a subordinate privilege level (in our case Supervisor mode) where the exception can itself be handled.

Trap Redirection could be used in conjunction with *Trap Delegation*. Whichever exceptions that are not directly delegated, could be redirected in software.

> *How is a trap redirected? In Machine mode…
> 1. Instructions are issued to copy the Machine mode *exception state* (CSRs mepc, mcause, mtval) to the corresponding Supervisor mode CSRs (sepc, scause, stval).
> 2. The Machine mode return address (mepc CSR – actually the address of the excepting instruction) is overwritten with the Supervisor mode exception base address (stvec CSR).
> 3. The Supervisor mode *previous privilege* field of the mstatus register (mstatus.SPP) is set with the current value of the Machine mode *previous privilege* field (mstatus.MPP) which represents the privilege level the exception was originally taken from.
> 4. The Machine mode *previous privilege* field (mstatus.MPP) is set to 1 (Supervisor mode).
> 5. An MRET instruction is issued, which in effect causes control to passed to the Supervisor *exception dispatcher*. The exception is then processed as if the exception had been taken in Supervisor mode.

Trap Redirection

The *Trap Redirection* code implemented in FORCE-RISCV has been implemented as an exception handler, and can be instantiated on a *per exception basis* when generating the exception handlers in *Machine mode*.

A set of *TrapRedirect* choices has been defined and included in the *General Choices* group, to give control over which traps (exceptions) should be redirected. This choices group is queried as exception handlers are being generated in Machine mode, to determine which exceptions to redirect.

Heres an excerpt from the `riscv/arch_data/general_choices.xml` file, showing a subset of the Trap Redirection specific choices:

```
<choices name="Redirect Trap - Instruction address misaligned" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Instruction access fault" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Illegal instruction" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Breakpoint" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Load address misaligned" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Load access fault" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
</choices>
<choices name="Redirect Trap - Store/AMO address misaligned" type="General">
  <choice description="Do NOT redirect trap to S-mode" name="DoNotRedirect" value="0x0" weight="10"/>
  <choice description="Redirect trap to S-mode" name="DoRedirect" value="0x1" weight="0"/>
```

There is an entry for each synchronous exception (trap) that can be redirected.

The comprehensive handler sets (defined in *py/riscv/exception_handlers/*`default_comprehensive_exception_handlers.json`) currently includes a single entry for the trap redirect exception handler:

```
CULAR PURPOSE. See the License for the specific language governing permissions and limitations
  {
    "ExceptionCode": "ExceptionClassRISCV.TRAP_REDIRECTION",
    "ExceptionHandler":
    {
      "ExceptionHandlerModule": "riscv.exception_handlers.TrapRedirectionHandler",
      "ExceptionHandlerClass": "TrapRedirectionHandlerRISCV"
    }
  },
  {
    "ExceptionCode": "ExceptionClassRISCV.INSTRUCTION_ADDRESS_MISALIGNED",
    "ExceptionHandler":
    {
```

Existing and new test templates have been implemented that exercise *trap redirection*, for the (synchronous) exceptions. Choice modifiers may be used at test/thread initialization time, to control which exceptions to delegate and how often.

Example test thread initialization sequence:

```
def gen_thread_initialization(gen_thread):
    (redirect_opt, valid) = gen_thread.getOption("RedirectTraps")

    if valid and redirect_opt == 1:
        traps_modifier = ChoicesModifier(gen_thread)
        weightDict = { "DoNotRedirect":100 - aWeight, "DoRedirect":aWeight }
        traps_modifier.modifyGeneralChoices('Redirect Trap - Breakpoint', weightDict)
        traps_modifier.modifyGeneralChoices('Redirect Trap - Environment call from U-mode', weightDict)
        traps_modifier.commitSet()
```

Trap Delegation/Redirection Choices Modifier

For convenience, a Choices modifier has been implemented, to allow Trap Delegation and Trap Redirection to be more easily controlled.

Example test template:

```
from riscv.EnvRISCV import EnvRISCV
from riscv.GenThreadRISCV import GenThreadRISCV
from base.Sequence import Sequence
from base.ChoicesModifier import ChoicesModifier
from base.InstructionMap import InstructionMap
from DV.riscv.trees.instruction_tree import *
from riscv.ModifierUtils import TrapsRedirectModifier

import RandomUtils

class MainSequence(Sequence):

    def generate(self, **kargs):
        for i in range(2):
            instruction_group = RV_A_instructions

            for _ in range( RandomUtils.random32(1,10) ):
                the_instruction = self.pickWeighted(instruction_group)
                self.genInstruction(the_instruction)

            if RandomUtils.random32(0,1) == 1:
                self.genInstruction('ECALL##RISCV')
            else:
                self.genInstruction('EBREAK##RISCV')

def gen_thread_initialization(gen_thread):
    traps_modifier = TrapsRedirectModifier(gen_thread)

    traps_modifier.update(ExceptionCode='Breakpoint', TrapChoice="Delegate", Weight = 50)
    traps_modifier.update(ExceptionCode='Environment call from U-mode', TrapChoice="Redirect", Weight = 50)
    traps_modifier.update(ExceptionCode='Environment call from S-mode', TrapChoice="Redirect", Weight = 50)

    traps_modifier.commit()

GenThreadInitialization = gen_thread_initialization

MainSequenceClass = MainSequence
GenThreadClass = GenThreadRISCV
EnvClass = EnvRISCV
```

Note the use of the **FORCE-RISCV** `TrapsRedirectModifier` control. As shown in the above example, an instance of the `TrapsRedirectModifier` control is used to specify both *Trap Delegation* choices and *Trap Redirection* choices. In the above test, approximately 50% of the time the *Breakpoint* exception will be delegated from *Machine-mode* to *Supervisor mode*. In the above test, approximately 50% of the time, the *Environment call from U-mode* and *Environment call from S-mode* exceptions will be redirected from *Machine mode* to *Supervisor mode*.