FORCE-RISCV ISG User Manual

VERSION 0.9

PLEASE FORWARD COMMENTS AND CORRECTIONS TO MIKE VADEN,
mvaden@futurewei.com

Contributing Authors

Jingliang (Leo) Wang

Morris (Gene) Cook

Tana Reeve

Miya Zhang

Weiming Tang

Mike Vaden

Table of Contents

Chapter 1 Introduction

FORCE-RISCV is an instruction sequence generator (ISG) for the RISC-V instruction set architecture. It can be used to generate tests for design verification of RISC-V processors. FORCE-RISCV uses randomization to choose instructions, registers, addresses and data for the tests, and can generate valid test sequences with very little input from the user. However, FORCE-RISCV provides a set of APIs with extensive capabilities which gives the user a high level of control over how the instruction generation takes place.

Test templates (written in Python) are used to control FORCE-RISCV instruction generation by invoking the FORCE-RISCV APIs to define constraints and specify the instructions. Because the test templates are normal Python code, the user has the power of the Python programming language to define the instruction generation sequence and the appropriate constraints.

FORCE-RISCV is integrated with the Handcar instruction simulator to model the behavior of the generated RISC-V instructions. Handcar is based on the Spike open source RISC-V instruction simulator. The format for the generation output is the standard *.ELF file and a disassembled text *.S file.

FORCE-RISCV provides full support of the RISC-V ISA, including:

- RV64G – (RV64I, MAFDC). Support for the V extention is planned.
- RISC-V privileged ISA, including full support for the U, S, and M privilege levels.
- RISC-V traps and exceptions.
- Support for FORCE-RISCV provided exception handlers is provided.
- Full support for the v48 virtual memory systems, including 4KB, 2MB, 1GB, and 512GB page sizes.

Chapter 2 Getting Started

2.1 Building the code

The FORCE-RISCV source code is repository is in GitLab and can be cloned from there. The url is

http://gitlab.futurewei.com/force-riscv/force-riscv.git

The default root directory will be force-riscv unless you renamed it during the clone.

To build FORCE-RISCV, execute these commands. It will take several minutes for the make commands to complete.

```
cd force-riscv
make
make tests
```

After the commands complete, the executable will be at force-riscv/bin/friscv.

2.2 Running the program

FORCE-RISCV is run from the Linux/Unix command line, and you can specify command line options that alter how the test generation is run. You can run this simple "smoke test" to demonstrate that the build was successful.

```
bin/friscv -t utils/smoke/test_force.py
```

You can run FORCE-RISCV from any directory; just adjust the relative path names. For example,

```
cd force-riscv/tmp
../bin/friscv -t ../utils/smoke/test_force.py
```

Note that the file name of a FORCE-RISCV test template always ends with "_force.py"; this serves to indicate that the file is a Python script, and that it is FORCE-RISCV test template.

The FORCE-RISCV program will generate the following files:

```
<templateName>.Default.S
<templateName>.Default.ELF
```

Which in our example is:

```
test_force.Default.S
test_force.Default.ELF
```

The *.ELF files can be the input to an RTL simulation environment defininf what instructions get executed during the simulation. The *.S files are assembly code style files, to help the user see what instructions have been generated.

2.3 Command line arguments

Use help flags to see the available command line arguments.

bin/frisc -h or bin/frisc --help

```
USAGE: bin/friscv [options]
Options:
  --cfg, -c                 Configuration file path.
  --help                    Print usage and exit.
  --log, -l                 Specify logging level.
  --dump, -d                Specify dumping option.
  --dump, -d                dump some images when generation. Support Asm, Elf,
                            Mem and Handlers.
  --noasm,                  Disables creation of the *.S assembly code output file.
  --options, -o             Specify test options.
  --seed, -s                Specify seed for test generation.
  --test, -t                Specify test template file name to run.
  --noiss, -n               Indicate to not simulate during test generation.
  --max-instr, -m           Maximum number of instructions that can be simulated.
  --num-cores, -c           Number of cores per chip.
  --num-threads, -T         Number of threads per core.
  --outputwithseed, -w      Indicate to generate outputs with seed number.

Examples:
  friscv -s 0x123 -t utils/smoke/test_force.py
  friscv -s 0x123 -l info -t utils/smoke/test_force.py
  friscv --unknown -- --this_is_no_option
  friscv –d Asm –t utils/smoke/test_force.py
```

The command line arguments are described in more details in the following table.

Command line argument	Usage
--cfg, or -c	This switch allows the user to specify an ISG config file. By default, it is not needed, and the FORCE-RISCV program will pick up the default config file from config/riscv.config.
--help	Print help information.
--log or -l	Specify ISG logging level, currently supported log levels are trace, debug, info, warn, error, fail, notice.
--noasm	Tell the ISG to not output assembly code (*.S file)
--options or -o	Pass in test options that can be checked in test template by using API such as getOption.
--seed or -s	Specify an initial seed for the ISG.
--test or -t	Specify a test file to run by the ISG.

2.4 Back-End Generator Options

Option	Usage	Default Value
NoHandler	No Exception Handler. If NoHandler=1, no exception handlers are loaded. If this is set, the template must not take any exceptions, or it will fail.	NoHandler=0
NoSkip	If NoSkip=1, if the back end fails to generate, the test will fail; otherwise, if the back end is unable to generate an instruction, it will be skipped.	NoSkip=0
FlatMap	If FlatMap=1, all virtual addresses will be mapped directly to the associated physical address.	FlatMap=0

2.5 Front-End Generator Options

There are also command line options that are read and interpreted by the front-end python code. Any template can use a front-end generator option by using the self.genThread.getOption("keyword") method. By convention, front-end generator option keywords are all lowercase.

Keyword	Description
all_cacheable	If all_cacheable=1 the template will be initialized with all addresses as cacheable Note: In order for this command line option to work, must include the PageMemoryAttributeModifier from ModifierUtils, add a SetUpSequence and gen_thread_initialization step. See example at "**Error! Reference source not found. Error! Reference source not fo und.**"
all_nc	If all_nc=1, the template will be initialized with all the addresses as non-cacheable Note: In order for this command line option to work, must include the PageMemoryAttributeModifier from ModifierUtils, add a SetUpSequence and gen_thread_initialization step. See example at "**Error! Reference source not found. Error! Reference source not fo und.**
regression_count	This option is issued in some of the regressions to be able to specify on the command line the number of times through a loop. For example: regression_count, valid = self.getOption("regression_count") for i in range(0,regression_count):
handlers_set	Exception Handlers. If handlers_set=Fast, the Fast exception handlers will be loaded, otherwise the default handlers_set=Comprehensive will be used.

Chapter 3 FORCE-RISCV ISG

3.1 Front-end interface

The FORCE-RISCV implementation is divided into a front-end portion written in Python and a back-end portion written in C++. The test templates can easily interact with the APIs of the front-end since they are both written in Python which is a popular general-purpose language that is easy to learn and maintain.

FORCE-RISCV ISG front-end provides a full-featured Python interpreter, rich front-end Python APIs, and transparent multi-thread/multi-core programming model. In the test templates, the user has control of a rich set of adjustable knobs on instruction/operand properties, address selection, and paging properties. The resulting test templates can be very flexible, powerful and extendable.

The added benefit of a front-end having a fully functional script interface is when back-end support is not yet fully implemented, the capabilities of the front-end enable many targeted scenarios via the provided APIs. Meanwhile the back-end development can continue allowing for enhancements that are well thought-out, that perform well, and that provide high quality tests.

3.1.1 Variables and flow control

Since the FORCE-RISCV front-end can use all of the features in the Python language, the variables and flow control are part of the Python language. With minor front-end-framework library support, desired sequencing behavior like All_of, One_of, permute, repeat can be readily supported.

Via the FORCE-RISCV API, the user can write a code block that is dependent on a combination conditions of dynamic run-time system states. When such an event's conditions are meet, the associated test generation code block will be triggered.

3.1.2 Sequence

The Sequence class object is a building block of a FORCE-RISCV test template, which itself is made up of a series of FORCE-RISCV API calls. Every test template contains a top-level sequence object. A basic test template might only contain a simple sequence.

But since every sequence can call a lot of other sequences and we can inspect thread group to determine which sequence to use, as well as use Sequence-library a FORCE-RISCV test template can get quite rich and deep.

3.1.3 MP/MT

In the FORCE-RISCV front-end framework, each PE thread is represented as a GenThread Python object and its sequence generation controlled by a Python thread by default.

Thread scheduling is implemented in a pseudo-random way, so that given the same seed and the same FORCE-RISCV ISG version, the exact same pseudo-random thread scheduling will be reproduced.

API calls from all PE controlling threads will be randomly mixed; therefore, the randomness quality of all threads will be equally good.

The threading details are transparent to the DV user. This natural threading view makes it easier for design verification (DV) engineers without much software experience to use FORCE-RISCV ISG. If the user wishes to share addresses between threads/cores, it is important to use the "Shared=1" flag on the genPA instruction when generating addresses for the template to use. This will allow the back-end to keep track of the unpredictable nature of loads from shared memory. For more information about unpredictable registers, please refer to Unpredictable Register Values.

In a sequence mixing scenario, a PE thread will be associated with a Python thread that controls a sub-thread group in the mixing scope, each sub-thread associated with a sequence being mixed in.

If a DV user wants to achieve certain co-operation scenarios among PE threads, FORCE-RISCV ISG APIs such as threadLockingContext, synchronizeWithBarrier, etc. can come in handy.

Thread locking is implemented via a locking context. If a template has a locking context, all of the threads will go through the locking context in random order, one at a time. Information can be passed via a shared object that is set inside the locking context which can be read in the main part of the template.

3.1.4 Exception Handlers

FORCE-RISCV exception handlers are grouped into two categories:

- Comprehensive handlers - FORCE-RISCV uses an OS like approach to service exceptions. Register contents are saved in a stack upon exception entry, restored before returning. The handlers will fix up the page descriptors. Handlers tend to be long in instruction count, but the user does not typically need to write any handler code.
- Fast mode handlers - No stack is used, and the handlers are bare bones. A few basic handlers are provided so the system can run properly. In this mode, it is expected the user will write their own defined exception handlers.

3.2 Back-end characteristics

FORCE-RISCV random test generator has a well thought out back-end design:

- ✓ Efficient, comprehensive, easy to use constraint engine and fine-grained yet fast constraint solving. Clean, efficient data structures.
- ✓ All algorithm carefully designed to never sacrifice randomness while maintain good performance.
- ✓ Efficient, fine-grained back-end, proven to generate very good quality test and easily scale to 100+ threads in production.

3.2.1 ISS integration

The FORCE-RISCV ISG uses an instruction sequence simulator (ISS) to model the behavior of the instructions and determine how the architectural state changes due to the execution of an instruction. After the generation of each instruction, the instruction execution is simulated by the ISS, and FORCE-RISCV will take the results from the ISS and update the appropriate architectural state. This allows FORCE-RISCV to optimize the generation of desired test events.

FORCE-RISCV can also run with the ISS disabled if desired. This mode is useful when the ISS does not have all necessary features implemented, in which case FORCE-RISCV can still generate valid tests that can be run by the RTL test bench. This is also useful in the initial development of features.

Handcar is the ISS that FORCE-RISCV uses. It is based on the open source Spike code, but has significant enhancements. Handcar provides an API that FORCE-RISCV (and potentially other tools) use to interactively step through instructions of a test and retrieve the results of simulating an instruction or multiple instructions. Another enhancement Handcar makes is replacing the Spike memory model with a sparse memory model. This allows tests to access the entire memory space without requiring an inordinate amount of system memory.

After a test is generated, it can be validated by running it through Handcar before using it in RTL simulation. The FORCE-RISCV environment provides a tool called Fpix to do just that. It is run as part of the master_run.py regression script that is also part of the FORCE-RISCV environment.

3.3 Register and memory resource controls

The FORCE-RISCV register and memory resource control capabilities are similar to commercial ISGs, except more flexible.

3.3.1 Register initialization and reservation

The general-purpose registers, floating point registers and system registers, if used in the main sequence, will be initialized as needed in the boot code. System registers can be initialized field by field, when the fields are used in the test generation, allowing as much flexibility as possible.

The user can initialize registers via API interface randomly, with specific value, adjust register initialization settings, or let the back-end randomize by default.

The user can reserve and unreserve registers using APIs. If one register is reserved, then FORCE-RISCV will not pick this register to be used as operand of random instructions unless the user specifically uses the register as operand parameter in the instruction generation API. The reservation can be on a register read, write or both. The API description section will give more details.

3.3.2 Unpredictable Register Values

There are times when register values are unpredictable, meaning their values could vary between different executions of the same instruction sequence. A typical case is when a memory location is shared between multiple threads and/or cores. FORCE-RISCV does not automatically detect shared memory locations, but they can and should be marked by the user with "Shared=1" in the genPA API.

Any register that is the target of a load from shared memory will be marked unpredictable by the back-end FORCE-RISCV code. Any register that is written to by an instruction that reads from an unpredictable register will be marked unpredictable and will remain unpredictable until a predictable value is written to that register. FORCE-RISCV avoids using unpredictable registers as read operands, unless the user explicitly specifies the unpredictable register in the genInstruction command.

There are classes of registers that are always marked unpredictable by the back-end FORCE-RISCV code. Reads from these types of registers could generate unpredictable results:

- Counter Registers (e.g. mhpmcounter3-31 registers)
- Interrupt pending registers (e.g. uip, sip, mip)

3.3.3 Memory initialization and reservation

The user can initialize memory locations directly via APIs, the memory address initialized can be either physical address or virtual address.

Memory locations can be reserved. FORCE-RISCV will not randomly choose reserved memory locations as instruction or data memory. The user can specify the memory location to be used in instruction generation API.

Memory locations can also be unreserved. Example of use case: a FORCE-RISCV test template code block can reserve a large memory location, end up only using part of it, and then release the part that is not used.

Chapter 4 FORCE-RISCV Test Template

The code segment shown in Figure 1 is a basic test template that creates a 2 instruction test case.

The Sequence class object holds the main GenThread object and contains the methods that comprise a large portion of the API interface that the test template uses to communicate with FORCE-RISCV. As shown in Figure 1, the generate() method contains the instructions to FORCE-RISCV on what type of and how many instructions to generate. In this simple test, the genInstruction API is called once to generate an ADD instruction, and a second time to generate an SRA instruction. A jump-to-self is appended to the end of the generated instruction sequence.

The reference assignments at the bottom of the template show the default values. If there were multiple Sequence subclasses, the MainSequenceClass would be assigned a reference to the Sequence subclass that is the main entry point that FORCE-RISCV uses to execute the test template.

```
From riscv.EnvRISV import EnvRISV
From riscv.GenThreadRISCV import GenThreadRISCV
From base.Sequence import Sequence

class MainSequence(Sequence):

      def generate(self, **kargs):
            self.genInstruction("ADD##RISCV")
            self.genInstruction("SRA##RISCV")


## Points to the MainSequence defined in this file
## This will be the entry point when FORCE-RISCV starts to execute the
template
MainSequenceClass = MainSequence

## Using GenThreadRISCV by default, can be overriden with extended classes
GenThreadClass = GenThreadRISCV

## Using EnvRISCV by default, can be overriden with extended classes
EnvClass = EnvRISCV
```

Figure 1: A basic test template. This test template is located in the distribution at utils/smoke/test_force.py.

Additional information on test template contents is provided in the Chapter 5 and **Error! R eference source not found.**.

Chapter 5 FORCE-RISCV front-end APIs

FORCE-RISCV ISG provides a rich set of front-end APIs, so that the user can use them to effectively adjust back-end test generation behaviors.

5.1 Instruction generation APIs

5.1.1 genInstruction

genInstruction is the main API for the user to tell the back-end to generate an instruction for the current PE.

API Name	genInstruction
Brief description	Generate an instruction for the current PE.
Return data	Instruction record ID, which can be used to query the back-end information regarding instruction(s) generated in this record.
Parameters	
Instruction name	Name of the instruction to be generated.
Operand Register overrides	There can be multiple operand overrides. The format is “Operand-name”:override. The override could be a value, a variable, a string representing a range, etc.
Operand Data overrides	There can be multiple operand data overrides. The format is “Operand-name.Data”:override. The override for GPRs is: IF_TYPE(value or ranges). IF_TYPE = <INT16>, <INT32> or <INT64>. If value or ranges are omitted, the back-end will choose an integer value. The override for FP registers is: (1). FP_TYPE(exp=value or ranges)(sign=value)(frac=value or ranges), FP_TYPE = <FP16>, <FP32> or <FP64>; (2). IF_TYPE(value or ranges) IF_TYPE = <INT128>, <INT64>, <INT32>, <INT16> or <FIXED>. If there is no value assignment, the back-end will choose a floating-point value. The override for SIMD register is: [lanes]FP overrides[lanes] integer overrides.
LSTarget	Specify target address or target address range for load/store instructions.

BRTarget	Specify target address or target address range for branch instructions. For more information about conditional branch instructions with unpredictable registers, please refer to Unpredictable Implied Register Operands.
CondTaken	Specify whether the condition is taken or not. Applies to all conditional instructions such as conditional branch instructions, conditional selection instructions, etc.
SpeculativeBnt	Specify whether or not to do speculative BNT processing. Applies to all branch instructions. Default value is false.
NoSkip	Indicate if the test generator should fail when unable to generate the requested instruction, or whether the instruction can be skipped. Default value is zero (skipping permitted).
NoBnt	When set to 1, the back-end will not generate a not-taken code for conditional branch instructions.
NoRestriction	When set to 1, the back-end will not do constraint solving on the requested instruction. Instead, the user accepts the responsibility of making sure the resulting test code block is correct. Default value is 0. For example, when generating B##RISCV, setting NoRestriction to 1 requires the user to explicitly specify the imm26 field value.
UnalignedPC	When set to 1 on a branch or return instruction, FORCE-RISCV will generate an unaligned target address, which will cause a PC alignment fault exception. Default value is 0.
AlignedData	When this attribute is specified, load/store instructions will ensure data is aligned.
AlignedSP	When this attribute is specified, load/store instructions will ensure data is aligned for SP when SP is used as base register.
NoPreamble	When set to 1, the back-end will not generate any preamble instructions to pre-load the operands for this instruction. Only the register values that are already available in the test will be used. This will increase the likelihood of failing to generate the instruction if usable register values are not available. Default value is 0.
LSData	Specify target data or target data range for load instructions, each element should be separated by ";".
Examples	

```
i_rec_id = self.genInstruction("ADD##RISCV", {"rs1":5, "rs2":6})
va_target = 0x0000F800C000
self.genInstruction("LD##RISCV", {"LSTarget":va_target, "NoSkip":0})
```

```
self.genInstruction("ADD##RISCV", {"rs1": "INT32(0x80000000)",
"rs2":"INT32(0x1000-0x2000)"})
self.genInstruction("ADD##RISCV", {"rs1": "INT32(0x80000000)",
"rs2":"INT32"})
self.genInstruction("B#cond#RISCV", {"CondTaken": "1"})
self.genInstruction('BR##RISCV',{"UnalignedPC":1, "SpeculativeBnt":True})
self.genInstruction("LD##RISCV", {"LSData":target_data})
```

5.1.2 queryInstructionRecord

The user can use this API to query details of generated instruction by passing in a unique instruction-record-ID.

API Name	queryInstructionRecord
Brief description	Query details of generate instruction.
Return data	An InstructionRecord dictionary that contains relevant attribute of the generated instruction. Or return None if instruction record id is invalid.
Parameters	
Instruction Record ID	ID of the instruction record. This is usually obtained as the result of genInstruction API call.
Examples	
i_rec_obj = self.queryInstructionRecord(i_rec_ID)	

The returned dict will contain quite a few attributes if i_rec_ID is valid, as shown in the table below.

queryInstructionRecord result InstructionRecord attributes	**Descriptions**
Name	Instruction full-name.
Opcode	The actual opcode of the instruction generated.
VA	Starting virtual address of the instruction.
PA	Starting physical address of the instruction.
Bank	Instruction memory bank information

LSTarget	Target address of load/store, if applicable
BRTarget	Branch target address, if applicable
Dests	A list of destination register indices, if applicable
Srcs	A list of source register indices.
~~**Status**~~	~~A list of status register information. Current only one tuple in the list returned (name, value).~~
Addressing	A list of addressing operand information. Each operand information is represented as a tuple (value, name). The following information are provided: "Base", ~~"Index",~~ "Offset", ~~and "ExtendAmount~~
Imms	A list of immediate indices.
Group	Instruction group, such as general, float, etc.

Not every key is valid for each instruction type. In the figure below, you can see some of the variances for 3 instruction types: ADD, LD and BEQ.

```
[notice]Front-end: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[notice]Front-end: >>>>>>>>>>   Key:  Opcode          Value:  0x1ad0ab3
[notice]Front-end: >>>>>>>>>>   Key:  Name            Value:  ADD##RISCV
[notice]Front-end: >>>>>>>>>>   Key:  Group           Value:  General
[notice]Front-end: >>>>>>>>>>   Key:  PA              Value:  0x80011018
[notice]Front-end: >>>>>>>>>>   Key:  Bank            Value:  0
[notice]Front-end: >>>>>>>>>>   Key:  VA              Value:  0x80011018
[notice]Front-end: >>>>>>>>>>   Key:  Dests           Value:  {'rd': 21}
[notice]Front-end: >>>>>>>>>>   Key:  Srcs            Value:  {'rs1': 26, 'rs2': 26}
[notice]Front-end: >>>>>>>>>>   Key:  Imms            Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Status          Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Addressing      Value:  {}
[notice]Front-end: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[notice]Front-end: >>>>>>>>>>   Key:  Opcode          Value:  0x1d623003
[notice]Front-end: >>>>>>>>>>   Key:  Name            Value:  LD##RISCV
[notice]Front-end: >>>>>>>>>>   Key:  Group           Value:  General
[notice]Front-end: >>>>>>>>>>   Key:  PA              Value:  0x80011034
[notice]Front-end: >>>>>>>>>>   Key:  Bank            Value:  0
[notice]Front-end: >>>>>>>>>>   Key:  VA              Value:  0x80011034
[notice]Front-end: >>>>>>>>>>   Key:  LSTarget        Value:  0x3c0159029fe0
[notice]Front-end: >>>>>>>>>>   Key:  Dests           Value:  {'rd': 0}
[notice]Front-end: >>>>>>>>>>   Key:  Srcs            Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Imms            Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Status          Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Addressing      Value:  {'Base': (4, 'rs1'), 'Offset': (470, 'simm12')}
[notice]Front-end: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[notice]Front-end: >>>>>>>>>>   Key:  Opcode          Value:  0xd1bf8a63
[notice]Front-end: >>>>>>>>>>   Key:  Name            Value:  BEQ##RISCV
[notice]Front-end: >>>>>>>>>>   Key:  Group           Value:  General
[notice]Front-end: >>>>>>>>>>   Key:  PA              Value:  0x800aa04c
[notice]Front-end: >>>>>>>>>>   Key:  Bank            Value:  0
[notice]Front-end: >>>>>>>>>>   Key:  VA              Value:  0x800aa04c
[notice]Front-end: >>>>>>>>>>   Key:  BRTarget        Value:  0x800a9560
[notice]Front-end: >>>>>>>>>>   Key:  Dests           Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Srcs            Value:  {'rs1': 31, 'rs2': 27}
[notice]Front-end: >>>>>>>>>>   Key:  Imms            Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Status          Value:  {}
[notice]Front-end: >>>>>>>>>>   Key:  Addressing      Value:  {'Offset': (2698, 'simm12')}
```

5.2 Sequence Library APIs

The SequenceLibrary class object allows the user to create a list of sequences from which a sequence can be randomly chosen for generation. There are two APIs from Sequence Library that user can use: chooseOne() and getPermutated(). Please note, user must create his own sequence library class, such as “MySequenceLibrary”, which is derived from given SequenceLibrary class, to implement the function of “createSequenceList” where the user defines which sequences should be in the list and their associated weighting. The user initiates his own MySequenceLibrary such as seq_lib = MySequenceLibrary(self.genThread) to call the following APIs.

The SequenceLibrary class object supports nested sequence lists; another SequenceLibrary list can be embedded in the given sequence list.

5.2.1 chooseOne

The user can use chooseOne to randomly pick one sequence instance from the sequence list in sequence library.

API Name	chooseOne
Brief description	According to the given weight given in the sequence list, randomly pick and return a sequence instance in the list.
Return data	An instance of Sequence class
Parameters	
	No parameter needed in the API
Example	
seq_lib = MySequenceLibrary(self.genThread) seq = seq_lib.chooseOne() seq.run()	

5.2.2 getPermutated

The user can use getPermutated API to obtain a sequence instance from a permutated sequence list in the sequence library.

API Name	getPermutated
Brief description	Sequence library prepares for a permutated sequence list from pre-defined sequence list. Return one sequence instance from the permutated list, one at a time, use generator.
Return data	A generator. User can obtain a sequence instance, one at a time.
Parameters	
skip_weight_check	(Optional) default is False. Indicate whether to check the weight or not. If weight check is required, item that has weight of 0 will not be included in the permutated list.
Example	
seq_lib = MySequenceLibrary(self.genThread) for seq in seq_lib.getPermutated(): seq.run()	

5.3 Address request APIs

There are a few address requesting APIs the user can use to setup interesting addressing scenarios.

5.3.1 genPA

The user can use genPA to obtain a valid physical address.

API Name	genPA
Brief description	Obtain a valid random physical address.
Return data	Physical memory address
Parameters	
Size	Size of requested memory block in number of bytes. Optional, defaults to 1.
Align	Alignment of the starting address of the requested memory block. Optional, defaults to 1 and must be a power of 2 – 1, 2, 4, 8, 16, etc…
Range	A string expressing a range or a set of ranges of memory to pick the target address from. Optional.
Bank	Specify which memory to select, Bank=0 for Default Memory.
CanAlias	Supported values: 0-1, defaults to 1. Sets a flag for the Physical Page being allocated to prevent it from being aliased to by future page allocation.
IndexMask	A pair of values in the “index-value/mask-value” format. An index/mask pair defines a set of addresses that when ANDed with the mask will produce the index value.
Shared	Optional, defaults to 0. When set, indicates that the PA being generated references memory that will be shared across multiple threads. All target registers in loads from memory locations marked Shared=1 will be marked as unpredictable by the FORCE-RISCV back-end code. For more information about unpredictable registers, please refer to Unpredictable Register Values.
Type	The type of address: “I” for instruction, “D” for data.
Example	
paddr = self.genPA(Size=64, Align=4, Type=”D”, Range=”0x100000-0x2fffff,0x400000-0x4fffff,0x8000000-0x9fffffff”, IndexMask=”0xfffc/0xfffc”)	

5.3.2 genVA

The user can use genVA to obtain a valid virtual address.

API Name	genVA
Brief description	Obtain a valid random virtual address. Necessary page translations and page/page-table descriptors will be randomly generate based on current settings, if they do not yet exist for the selected virtual address.
Return data	Virtual memory address.
Parameters	
Size	Size of requested memory block in number of bytes. Optional, default to 1.
Align	Alignment of the starting address of the requested memory block. Optional, default to 1. The value specified must be a power of 2.
Range	A string expressing a range or a set of ranges of memory to pick the target address from. Optional
IndexMask	A pair of value in the "index-value/mask-value" format. An index/mask pair define a set of addresses that when ANDed with the mask will produce the index value.
FlatMap	Supported values: 0, 1, Default value: 0 unless a global FlatMap value is specified in the control file, then it will default to that value. When FlatMap=1, forces the VA which is generated to have a flat mapped allocation. This will override any global FlatMap value specified in the control file
ForceAlias	Supported values: 0, 1, Default value: 0 Forces the VA which is generated to be aliased to an existing page. If aliasing is not possible the generation will throw an error.
PhysPageID	Optional. Unsigned Integer Value. Corresponds to targeted PageId. No default value (must specify valid page ID). Used to target a specific Physical Page that the VA mapping should alias to.
CanAlias	Supported values: 0-1, defaults to 1. Sets a flag for the Physical Page being allocated to prevent it from being aliased to by future page allocation.
Shared	Optional, defaults to 0. When set, indicates that the VA being generated references memory that will be shared across multiple threads. All target registers in loads from memory locations marked Shared=1 will be

	marked as unpredictable by the FORCE-RISCV back-end code. For more information about unpredictable registers, please refer to Unpredictable Register Values.
Type	The type of address: "I" for instruction, "D" for data.
Example	
vaddr = self.genVA(Size=64, Align=4, Range="0x100000-0x2fffff,0x400000-0x4fffff,0xff8000000-0xff9ffffff", IndexMask="0xfffc/0xfffc")	

5.3.3 genVAforPA

Map a given PA to a valid VA in the current address space. A physical address can then be shared between threads and among ELs and ASIDs of the same thread.

API Name	genVAforPA
Brief description	Given a physical address, obtain a valid virtual address in the current address space context, very useful in sharing cases. Necessary page translations and page/page-table descriptors will be randomly generate based on current settings, if they do not yet exist for the specified PA.
Return data	Virtual memory address.
Parameters	
PA	The physical address to be mapped. Required parameter. Unsigned Integer Value.: No default value (must specify address that is valid inside of the supported physical memory ranges).
Bank	Supported values: 0.
FlatMap	Supported values: 0, 1, Default value: 0 unless a global FlatMap value is specified in the control file, then it will default to that value. When FlatMap=1, forces the VA which is generated to have a flat mapped allocation. This will override any global FlatMap value specified in the control file
CanAlias	Supported values: 0-1, defaults to 1. Sets a flag for the Physical Page being allocated to prevent it from being aliased to by future page allocation.
ForceNewAddr	Supported values: 0, 1, defaults to 0.

	Used to specify that the VA generation should create a new VA to PA mapping as opposed to returning an existing mapping if possible. Useful to ensure a new mapping to the specified PA.
Type	The type of address: “I” for instruction, “D” for data.
Size	Size of requested memory block in number of bytes. Optional, default to 1.
Example	
vaddr = self.genVAforPA(PA=paddr_target, Size=0x100)	

5.3.4 verifyVirtualAddress

Verify a given virtual address is usable or not. Return true if the virtual address is free and can be used by application.

API Name	verifyVirtualAddress
Brief description	Verify a virtual address range is useable or not.
Return data	Bool. True: the address is free; False: the address is occupied.
Parameters	
VA	Virtual address to verify
Size	Size of virtual address in number of bytes
is_instr	The address is for Instruction Or Data
Example	

If self.verifyVirtualAddress(0x80001000, 4, True): // check whether virtual address 0x80001000-0x80001003 is usable for instruction.
Self.notice(“the virtual address is free”)	

5.4 Page request APIs

5.4.1 genFreePagesRange

Obtain a virtual address range that has not been mapped. This is usually useful when the user wants to obtain multiple pages next to each other and setting up interesting page attribute combinations.

API Name	genFreePagesRange
Brief description	Obtain a virtual address range that has not been mapped in the current address space to setup interesting paging conditions.
Return data	A tuple consisted of the start address and start-end string of the virtual address range, as well as a flag variable indicating if the API call was successful, then followed by all the page sizes picked.
Parameters	
Number	Number of pages requested. It has to be at least 2. Required parameter.
PageSize	Page size specification of the requested pages. Optional parameter. If omitted page sizes will be randomly chosen. The user can also specify page size for some of the pages, but leave other pages random by leave the position in the comma separated list empty.
Range	A string expressing a range or a set of ranges of memory to pick the target page from. Optional
Example	
(is_valid, start_addr, start_end_range, psize1, psize2, psize3) = self.genFreePagesRange(Number=3, PageSize="4K,,1M")	

After obtaining the address range, the user can create the pages one by one, by looping through the page sizes creating each page with desired page attributes, starting from the returned VA range start address. An example will be given on how to do this later in this document.

5.4.2 getPageInfo

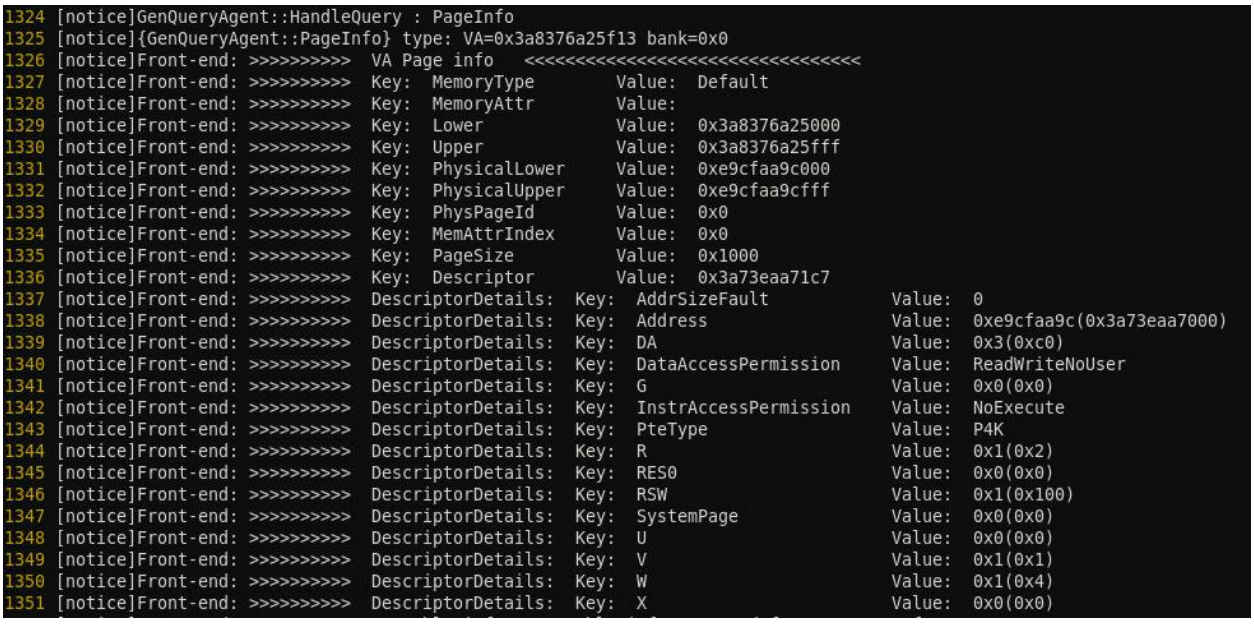
The user can use this API to retrieve page information for a given address.

API Name	getPageInfo
Brief description	Get page information for a given address
Return data	A page information dictionary
Parameters	
addr	The given address
Addr_type	Type is "VA"
Example	
page_info = self.getPageInfo(0x80001000, "VA", 0)	

The following shows the returned page information dictionary:

- First level - Note: dictionary could be empty if given page is not found
    - Page – “Page”:page_dict – see “Second level”
- Second level
    - Page based dictionary
        - PhysicalLower – page physical lower boundary
        - PhysicalUpper – page physical upper boundary
        - Lower – page virtual address lower boundary
        - Upper – page virtual address upper boundary
        - MemoryType – page memory type such as “Default”
        - Descriptor – page descriptor value
        - DescriptorDetails – see “Third level”
        - PageSize
- Third level
    - DescriptorDetails – list the “name”:”value string” retrieved from back-end

The figure below shows test output printing the detailed output from the getPageInfo api.



5.5 Register control APIs

Register control APIs can be used to adjust register resource usage in the back-end.

5.5.1 getRandomRegisters

Obtain a number of randomly usable registers, this refer to registers that have not been reserved by any means, and can be randomly used by random instructions.

API Name	genRandomRegisters
Brief description	Get a few randomly usable registers.
Return data	A tuple consisted of the indices of the randomly selected registers.
Parameters	
Number	Number of registers requested. Required parameter.
Type	Type of register, such as GPR, FPR, etc.
Exclude	A string containing one or more register indices that the user would like to avoid using. Optional parameter.
Example	
(reg1, reg2, reg3) = self.getRandomRegisters(3, ”GPR”, ”0”) reg1 = self.getRandomRegisters(1, ”GPR”, ”0”)[0]	

5.5.2 getRandomGPR

Obtain one of randomly usable GPR, this refer to registers that have not been reserved by any means, and can be randomly used by random instructions. This API is based on getRandomRegisters, for user convenience

API Name	genRandomGPR
Brief description	Get one randomly usable GPR.
Return data	The register index of the randomly selected GPR.
Parameters: None	
Example	
random_gpr = self.getRandomGPR()	

5.5.3 reserveRegister

Reserve a register so that it cannot be randomly accessed by other instruction generation. reserveRegister and unreserveRegister APIs are usually used in pairs inside a sequence scope. Remembering to call unreserveRegister at the end of the scope is very important. Forgetting to do so will result in failure to generate random instructions since FORCE-RISCV will run out of available registers needed for instruction generation.

API Name	reserveRegister
Brief description	Reserve register so that it cannot be randomly accessed by random instructions.
Return data	None
Parameters	
Name	Name of the register to be reserved.
Access	Type of access to be reserved, default to “Write”, meaning the register will be protected from updates from random instructions, but can still be used as source operand. The other available options are “Read” and “ReadWrite”. For “Read” and “ReadWrite” reservations, the register will be protected from updates from random instructions, and will not be used as a source operand.
Example	
self.reserveRegister(name=”x1”, access=”Write”)	

5.5.4 reserveRegisterByIndex

Uses the index (integer value) instead of the name, i.e. 12 versus ‘x12’.

API Name	**reserveRegisterByIndex**
Brief description	Reserve register by its register index and type so that it cannot be randomly accessed.
Return data	None
Parameters	
Size	Register bit size: 8,16,32,64 and etc.
Index	Index of the register to be reserved.

Type	Type of register, such as GPR, FPR, etc.
Access	Type of access to be reserved, default to “Write”, meaning the register will be protected from updates from random instructions, but can still be used as source operand.
Example	
self.reserveRegister(32, index=0, type=”GPR”, access=”Write”)	

5.5.5 unreserveRegister

API Name	unreserveRegister
Brief description	Unreserve register so that it can be randomly accessed by random instructions.
Return data	None
Parameters	
Name	Name of the register to be reserved.
Access	Type of access to be unreserved, default to “W”, meaning removing the write protection which was the default behavior of reserveRegister call. Note that when calling the unreserveRegister API, the user should use the same attribute used when reserving the register. For example, if the user reserves a register with access=”ReadWrite”, and then unreserves with attribute=”Write”, the register is still reserved for Read accesses since it was originally reserved for both Read and Write accesses.
Example	
self.unreserveRegister(name=”x12”, attribute=”W”)	

Unreserve a register so that it will again be available to be used by other instruction generation.

API Name	unreserveRegisterByIndex
Brief description	Unreserve register by its register index and type so that it can be randomly accessed
Return data	None
Parameters	
Size	Register bit size: 8,16,32,64 and etc.
Index	Index of the register to be reserved
Type	Type of register, such as GPR, FPR, etc.
Access	Type of access to be unreserved, default to “W”, meaning removing the write protection which was the default behavior of reserveRegister call.
Example	
self.unreserveRegisterByIndex(64, index=12, type=”GPR”, attribute=”W”)	

5.5.6 isRegisterReserved

Check if a register is reserved for a given access mode (Read/Write).

API Name	isRegisterReserved
Brief description	Check if a given register is reserved for a given access mode (Read/Write)
Return data	True if register is reserved for the given access mode False if register is not reserved for the given access mode
Parameters	
Name	Name of the register that will be checked for the given access mode reservation.
Access	Type of access to be checked - either “Read” or “Write”. Defaults to “Write”.
Example	
self.isRegisterReserved(name=”x22”, access=”Write”)	

5.5.7 readRegister

Get current value of a register/field. This can be very useful when the user wants to choose next step action based on one or more register states.

API Name	readRegister
Brief description	Obtain the current value of a register.
Return data	A tuple consisted of the register value, and a flag variable indicating whether the value is valid. The value is invalid if the register value is unknown.
Parameters	
Name	Name of the register
Field	Field of the register, usually useful when we are only interested in a certain field of a system register. Optional parameter.
Example	
(reg_value, value_valid) = self.readRegister(”x14”)	

5.5.8 writeRegister

When there is no ISS integration, this is a good way to communicate a new register value to the back-end.

API Name	writeRegister
Brief description	Inform register value update to the back-end when running in mode with no ISS.
Return data	None.
Parameters	
Name	Name of the register.
Value	The value to be written.
Field	Field of the register. Optional parameter.
Example	
self.writeRegister(”x6”, 0x7FF00000)	

5.5.9 initializeRegister

Initialize a register or register field with certain register value.

API Name	initializeRegister

Brief description	Initialize register or register field to a specified value, if it has already been initialized, a warning will be send to the generator log.
Return data	None.
Parameters	
Name	Name of the register.
Value	The initial value. Optionally, a list of values can be supplied for registers greater than 64 bits which are classified as a 'LargeRegister' in the register definition. If a list of values is provided the Field parameter is ignored.
Field	Field of the register. Optional parameter. Ignored if passing a list of values for 'Value'.
Example	
self.initializeRegister(name="fcsr", value=1, field="FRM") self.initializeRegister("D8", 0xFEDCBA9876543210)	

5.5.10 initializeRegisterFields

Initialize a list of registers of specified register with specified field value.

API Name	initializeRegisterFields
Brief description	Initialize a list of register fields, if a field has already been initialized, a warning will be send to the generator log.
Return data	None.
Parameters	
register_name	Name of the register.
field_value_map	The map of field name and its field value to be initialized. The key is field name and the value is the field value to be initialized.
Example	
self.initializeRegisterFields("FCSR", {"frm":1})	

5.5.11 randomInitializeRegister

Initialize a register or register field randomly. The purpose of this API is usually to make sure a register/register-field is initialized when you don't care what it is initialized to.

API Name	randomInitializeRegister
Brief description	Initialize register or register field to a random value based on the current settings, if it has not been initialized.
Return data	None.
Parameters	
Name	Name of the register.
Field	Field of the register. Optional parameter.
Example	
self.randomInitializeRegister(Name="fcsr", Field="FRM")	

5.5.12 randomInitializeRegisterFields

Initialize a list of fields from specified register randomly. The purpose of this API is usually to make sure a list of register-fields are initialized, but don't care what it is initialized to.

API Name	randomInitializeRegisterFields
Brief description	Initialize a list of fields from specified register randomly. If the register has been initialized, ignore it.
Return data	None.
Parameters	
register_name	Name of the register.
field_list	List of fields to be randomly initialized.
Example	
self.randomInitializeRegister(Name="fcsr", Field=["UF", "OtherField"])	

5.5.13 getRegisterIndex

Given the register name, get register index. The purpose of this API is to allow the front user easy access to the register index.

API Name	getRegisterIndex
Brief description	Return register index from the given register name.
Return data	Register index of the given register name.
Parameters	
register_name	Name of the register.
Example	
self.getRegisterIndex(”fcsr”)	

5.5.14 getRegisterReloadValue

Given the register name, and custom field value constraints as optional, get a valid reload value; this value is not the current register value. The purpose of this API is to allow the front-end user to get a valid value that can be used to set that register.

API Name	getRegisterReloadValue
Brief description	Return a valid reload value given a register name and optional custom field value constraints.
Return data	Reload value as uint64
Parameters	
register_name	Name of the register.
field_constraints	A dictionary as custom constraints, with field name as key and constraint string as value, i.e., {“Len”, “1,3-5”}
Example:	
self.getRegisterReloadValue(“fcsr”, {“FRM”, “0-4,7”}) Valid values for the frm are binary 000, 001, 010, 011, 100, 111 -> 0-4, 7. The user can specify a subset of those values if desired. If the range is not specified, FORCE-RISCV knows what the valid field values are, and will randomly select from those field values.	

5.5.15 getRegisterInfo

Given the register name and index, such as “xn” and 4, get register information. The purpose of this API is to allow the front user retrieve register information, such as type, width and value.

API Name	getRegisterInfo
Brief description	Return register information from the given register name and index
Return data	A dictionary of register information, such as {“Type” : “GPR”, “Value” : 0x12345678, “Width” : 32} If the register in question is classified as a ‘LargeRegister’ in the register definition file and has a width > 64 bits, a “LargeValue” key will be present in the dictionary which contains a list of 64 bit values corresponding to the full value of the register in order from least significant 64 bits to most significant 64 bits.
Parameters	
name	Name of the register, such as x4
index	Register index value, such as 4
Example	
self.getRegisterInfo(“x4”, 4)	

5.5.16 getRegisterFieldMask

Given the register name, list of field names to get the actual fields mask. The purpose of this API is to allow the front-end user to retrieve given field masks in the register in order to operate on the given fields accordingly.

API Name	getRegisterFieldMask
Brief description	Return actuals fields mask from the given register and list of fields
Return data	A 64-bit integer that represents the given fields mask (location in the register)
Parameters	
regName	Name of the register, such as “FCSR”
fieldList	List of field names, such as [“frm”]
Example	
self.getRegisterFieldMask(“fcsr”, [“FRM”])	

5.5.17 getRegisterFieldInfo

Given the register name and field constraints, get mask and value. The purpose of this API is to generate a random value based on given register name and field constraints.

API Name	getRegisterFieldInfo
Brief description	Return register mask and value from given register and field constraints
Return data	A tuple: (mask, reverse_mask, value) to indicates the actual register mask (fields location), reverse mask (opposite of the mask), and register value generated based on field contraints
Parameters	
registerName	Name of the register, such as “FCSR”
fieldConstraints	Dictionary of field constraints such as {“frm”:“0-1”}
Example	
self.getRegisterFieldInfo(“fcsr”, {“FRM”:“0-1”}	

5.5.18 SystemRegisterUtils

A SystemRegisterUtils utility is provided to allow front-end user to operate upon system registers more easily. Currently, two classes are provided (AccessSystemRegister and UpdateSystemRegister). It can be expanded to add more upon request in the future.

5.5.18.1 AccessSystemRegister

A derived class from Sequence, AccessSystemRegister, allow front-end user to access (read or write) a given system register without writing the actual assembly code. Instead, user simply use the function access(reg_name, gpr_index, for_write), where reg_name is the given system register name, gpr_index is the selected GPR register index, and for_write indicates whether to write value to system register or read from.

Here is a simple example to illustrate how to use it.

```
accessReg = AccessSystemRegister(self.genThread)
gpr1 = self.getRandomGPR()
accessReg.access("fcsr", gpr1, for_write=False)  # read from FCSR to GPR
accessReg.access("fcsr", gpr1, for_write=True)   # write to FCSR from GPR
```

5.5.18.2 UpdateSystemRegister

A derived class from Sequence, UpdateSystemRegister, allow the front-end user to manipulate a given list of field values on a given system register. The front-end user can request a system register update by calling function update(reg_name, field_name_dict, isb=False), where reg_name is the given system register name, field_name_dict is the field name dictionary, and isb indicates where an ISB instruction is needed at the end.

Here is a simple example to illustrate how to use it:

```
Update_reg = UpdateSystemRegister(self.genThread)
Update_reg.update("fcsr", {"FRM" : "0-1"}, isb=True)
```

5.6 Exception related APIs

5.6.1 queryExceptionVectorBaseAddress

Obtain the Vector Base address for the exception handlers based on exception level and other settings. This works for both comprehensive and fast exception handlers.

API Name	queryExceptionVectorBaseAddress
Brief description	Get the vector base address for the specified exception level
Return data	64-bit vector base address value
Parameters	
ExceptionLevel	Name of the exception level being queried. Options are: "DefaultMachineModeVector" and DefaultSupervisorModeVector"
Example	
base_address = self.queryExceptionVectorBaseAddress("DefaultSupervisorModeVector")	

5.6.2 queryExceptionRecordsCount

Obtain the number of exceptions that have occurred thus far in the test based on the EC (Exception Class). This works for both comprehensive and fast exception handlers.

API Name	queryExceptionRecordsCount
Brief description	Returns the number of exceptions based on the EC
Return data	Integer
Parameters	
ExceptionClass	Number of the exception class being queried. All valid exception classes are supported except EC=9 and EC=26.
Example	
num_svc = self.queryExceptionRecordsCount(21)	

5.6.3 queryExceptionRecords

Obtain the number of exceptions that have occurred thus far in the test based on the EC (Exception Class). This works for both comprehensive and fast exception handlers.

API Name	queryExceptionRecords
Brief description	Returns a list of tuples showing the (EC, EL) for each exception class
Return data	List of tuples showing (EC,EL), or empty list if no exceptions of that class have occurred.
Parameters	
ExceptionClass	Number of the exception class being queried. All valid exception classes are supported except EC=9 and EC=26.
Example	
svc_exception_hist = self.queryExceptionRecords(21) returns list of tuples, the first tuple is the first EC=21 exception taken, the second tuple is the second EC=21 … up to the total number of EC=21 exceptions that have been taken svc_exception_hist = [(21,3),(21,2),(21,1)….]	

5.6.4 queryExceptions

Obtain the number of exceptions that have occurred thus far in the test based on the EC (Exception Class) based on search criterion. Can be searched by EC, PC (Program Counter), lastOnly, Source EL, and Target EL, and any of their combinations.

This works for both comprehensive and fast exception handlers.

API Name	queryExceptions
Brief description	An advanced universal search to returns a list of tuples showing the (EC, PC, SRC_PL, TGT_PL) for each exception
Return data	List of tuples showing (EC, PC, SRC_PL, TGT_PL), or empty list if no exceptions satisfies the criterion have occurred.
Parameters	
EC	exception class is an integer 0-64, able to accept integer or string in constraint Set format
PC	address stored in Program Counter is an 64bit integer, able to accept integer or string in constraint Set format
SRC_PL	source privilege level, an integer 0-3, able to accept integer or string in constraint Set format
TGT_PL	target privilege level, an integer 0-3, able to accept integer or string in constraint Set format
Last	True or False, False by default. When True, only the latest exception satisfies the criterion returns as the only tuple in the returned list.
Example	

exception_hist = self.queryExceptions(EC=23, Last=False) returns list of tuples, the first tuple is the first EC=23 exception taken, the second tuple is the second EC=23 … up to the total number of EC=23 exceptions that have been taken exception_hist = [(23, 12e0efcace10, 0, 3), (23, 910812c7a1c8, 0, 3) ….]

exception_hist = self.queryExceptions(EC=23, Last=True) returns list of one tuple, the tuple is the last EC=23 exception taken exception_hist = [(23, 5dccbab3e218, 0, 3)]

exception_hist = self.queryExceptions(EC = "21-23,51,52", PC=0x5dccbab3e218, TGT_PL="1,2", SRC_PL="1-2", Last=True) returns list of one tuple, the tuple is the last exception satisfies the Dictionary. exception_hist = [(21, 5dccbab3e218, 0, 3)]

exception_hist = self.queryExceptions(Last=False) returns a list of all exceptions in chronological order. Same as self.queryExceptions(TGT_PL = "0,1,2,3", SRC_PL = "0-2")

exception_hist = self.queryExceptions(TGT_PL = "0,1,2", SRC_PL = "0-2")

5.7 PE states related APIs

5.7.1 getPEstate

Obtain important PE states like PC, EL, ASID, etc.

API Name	getPEstate
Brief description	Query important PE states.
Return data	The PE state value.
Parameters	
State name	Name of the state being queried, supported states are PC, EL, ASID, VMID, etc.
Example	
current_pc = self.getPEstate(“PC”)	

5.7.2 setPEstate

Set important PE states during generation, to avoid bad test, use with care. For example, the user can set PC to a location and generate a subroutine there, then set PC back to resume normal test generation.

API Name	setPEstate
Brief description	Set important PE states.
Return data	None
Parameters	
State name	Name of the state being queried, supported states are PC, EL, ASID, VMID, etc.
State value	Value to be set.
Example	
self.setPEstate("PC", new_pc_value)	

5.8 Memory control APIs

5.8.1 initializeMemory

Initialize memory with a specified value.

API Name	initializeMemory
Brief description	Initialize memory location.
Return data	None
Parameters	
Address	Starting address of the memory location. This can be virtual or physical address, depending on the last parameter of the API call.
Size	Number of bytes to be initialized, this is limited to 8 or less bytes. If more bytes need to be initialized, the user can call this API multiple times.
Value	Initial value.
Endian	Byte ordering. Optional. If not specified, will default to big-endian.
Virtual	Indicates whether the address is a virtual address or physical address.

Example
Positional arguments, no keywords: self.initializeMemory(Address (#), Bank (0), Size (# bytes), mem_value (#), IsInstruction(True/False), isVirtual (True/False))

5.9 ChoicesModifier APIs

The choices modifier APIs are provided through a python class ChoicesModifier. The user should define a sub class to do all choices modifications in the API apply(). Refer to the example in Section 7.10 for details.

The choices available to be modified are given in specific xml files in the "force-riscv/riscv/arch_data" directory. The relevant files are listed in the description of each API.

5.9.1 modifyOperandChoices

Modify operand choices. It is used to adjust weights for operand choices so that they can be selected with higher or lower possibility.

Valid tree_names are given in xml <choices> records within the operand_choices.xml file. For example,

File: force-riscv/riscv/arch_data/operand_choices.xml

```
<choices name="GPRs" type="RegisterOperand">
<choices name="Nonzero GPRs" type="RegisterOperand">
<choices name="64-bit SIMD/FP registers" type="RegisterOperand">
```

The highlighted strings are valid tree_names. The strings for the register names which serve as the dictionary keys are given in the <choice> records underneath each <choices> record. For example the GPRs have:

```
<choices name="GPRs" type="RegisterOperand">
    <choice name="x0" value="0" weight="10">
    <choice name="x1" value="1" weight="10">
    <choice name="x2" value="2" weight="10">
```

Consult the force-riscv/riscv/arch_data/operand_choices.xml file for other tree names that can have their weighting adjusted.

API Name	modifyOperandChoices
Brief description	Modify operand choices
Return data	None

Parameters	
tree_name	Name of operand choices tree. Example tree_names: “GPRs”, “Integer data pattern”, “Floating-point data pattern”
mod_dict	Modification dictionary. The key is choice name, the value is weight
Example	
Self.modifyOperandChoices(tree_name=”GPRs”, mod_dict={“x7”:40, “x14”:30}) self.modifyOperandChoices(tree_name=”64-bit SIMD/FP registers”, mod_dict={“D0”: 100, “D31”:100})	

5.9.2 modifyRegisterFieldValueChoices

Modify weights for register field value choices for the current (inner most) scope.

Valid tree_names are given in xml <choices> records within the register_field_choices.xml file. For example,

File: force-riscv/riscv/arch_data/register_field_choices.xml

```
<choices name=”sstatus.SUM” type=”RegisterFieldValue”>
<choices name=”sstatus.MXR” type=”RegisterFieldValue”>
```

The highlighted strings are valid setting names. The strings for the setting description names which serve as the dictionary keys are given in the <choice> records underneath each <choices> record. For example:

```
<choices name=”sstatus.SUM” type=”RegisterFieldValue”>
    <choice name=”No Supervisor access to User Memory” value=”0x0” weight=”10”>
    <choice name=”Supervisor access to User Memory” value=”0x1” weight=”0”>
```

Consult the force-riscv/riscv/arch_data/register_field_choices.xml file for other settings that can have their weighting adjusted.

API Name	modifyRegisterFieldValueChoices
Brief description	Modify weights of choices for a register field value choices tree.
Return data	None
Parameters	
Setting name	Name of the RegisterFieldValueChoice to be modified. Example values:

	“sstatus.SUM” “sstatus.MXR
Weight dict	A Python dict describing the new weight values for various choices.
Example	
self.modifyRegisterFieldValueChoices(“sstatus.SUM”, **{ “0x0”:10, “0x1”: 90})	

5.9.3 modifyPagingChoices

Modify weights for paging choices for the current (inner most) scope.

Valid setting names are given in xml <choices> records within the paging_choices.xml file. For example,

File: force-riscv/riscv/arch_data/paging_choices.xml

```
<choices name=”Page Allocation Scheme” type=”Paging”>
<choices name=”Data Page Aliasing” type=”Paging”>
```

The highlighted strings are valid setting names. The strings for the setting description names which serve as the dictionary keys are given in the <choice> records underneath each <choices> record. For example:

```
<choices name=”Page Allocation Scheme” type=”Paging”>
    <choice description=”Random Free Allocation” name=”RandomFreeAlloc” value=”0x0”
    weight=”100”>
    <choice description=”Flat mapped Allocation” name=”FlatMapAlloc” value=”0x1”
    weight=”0”>
```

Consult the force-riscv/riscv/arch_data/paging_choices.xml file for other settings that can have their weighting adjusted.

API Name	modifyPagingChoices
Brief description	Modify weights of value choices for a paging choices tree.
Return data	None
Parameters	
Setting name	Name of the paging choice to be modified, such as: “Page size#4K granule#S#stage 1”, and “Data Page Aliasing”

Weight dict	A Python dict describing the new weight values for various choices.
Example	
self.modifyPagingChoices(“Page size#4K granule#S#stage 1”, **{ “4K”:50, “2M”: 50})	

5.9.4 modifyGeneralChoices

Modify weights for general choices for the current (inner most) scope.

Valid setting names are given in xml <choices> records within the general_choices.xml file. For example,

File: force-riscv/riscv/arch_data/general_choices.xml

```
<choices name=”Starting jump” type=”General”>
<choices name=”Privilege level switch to lower or same level” type=”General”>
```

The highlighted strings are valid setting names. The strings for the setting description names which serve as the dictionary keys are given in the <choice> records underneath each <choices> record. For example:

```
<choices name=”Starting jump” type=”General”>
    <choice description=”Use Branch” name=”Branch” value=”0x0” weight=”0”>
    <choice description=”Use RET” name=”RET” value=”0x1” weight=”10”>
```

Consult the force-riscv/riscv/arch_data/general_choices.xml file for other settings that can have their weighting adjusted.

API Name	modifyGeneralChoices
Brief description	Modify weights of value choices for a general choices tree
Return data	None
Parameters	
Setting name	Name of the general choices tree to be modified.
Weight dict	A Python dict describing the new weight values for various choices.
Example	
self.modifyGeneralChoices(“Privilege Switch - Target Privilege”, **{ “S”:10, “M”: 10000})	

5.9.5 modifyDependenceChoices

Modify weights for dependence choices for the current (inner most) scope.

Valid setting names are given in xml <choices> records within the dependence_choices.xml file. For example,

File: force-riscv/riscv/arch_data/dependence_choices.xml

```
<choices name="Register Dependency" type="Dependence">
<choices name="Source Dependency" type="Dependence">
```

The highlighted strings are valid setting names. The strings for the setting names which serve as the dictionary keys are given in the <choice> records underneath each <choices> record. For example:

```
<choices name="Source Dependency" type="General">
    <choice name="Source after source" value="0x0" weight="10">
    <choice name="Source after Target" value="0x1" weight="10">
    <choice name="No source dependence" value="0x2" weight="10">
```

Consult the force-riscv/riscv/arch_data/dependence_choices.xml file for other settings that can have their weighting adjusted.

API Name	modifyGeneralChoices
Brief description	Modify weights of value choices for a general choices tree
Return data	None
Parameters	
Setting name	Name of the general choices tree to be modified.
Weight dict	A Python dict describing the new weight values for various choices.
Example	
self.modifyDependenceChoices("Register Dependency", **{ "No dependency":10, "Inter-dependency": 20, "Intra-dependency":10})	

5.9.6 commitSet

Commit all modify choices to affect instruction generation. This method is mostly intended to be used in the apply method of ChoicesModifier.

API Name	commitSet
Brief description	Commit modify choices
Return data	Return commit ID
Parameters	
None	
Example	
self.commitSet()	

5.9.7 registerSet

Register all modify choices which will not affect instruction generation until the modification is applied on back-end. This method is mostly intended to be used in the register method of ChoicesModifier.

API Name	registerSet
Brief description	Register modify choices
Return data	Return register ID
Parameters	
None	
Example	
self.registerSet()	

5.9.8 apply

Apply a series of modifications. User/template-class-library should implement the details in a subclass of ChoicesModifier.

API Name	apply
Brief description	Apply a series of modification
Return data	Return apply ID
Parameters	
kwargs	Key word parameter for modifications
Example	

dict1 = {“GPRs” : {“x1”: 40}} self.apply(arg1=dict1)

5.9.9 register

register a series of modifications. User/template-class-library should implement the details in a subclass of ChoicesModifier.

API Name	register
Brief description	Register a series of modification
Return data	Return register ID
Parameters	
Kwargs	Key word parameter for modifications
Example	
dict1 = {“GPRs” : {“x1”: 40, “x12”:20} } self.register(arg1=dict1)	

5.9.10 update

Update a series of modifications by calling modify choices method. User/template-class-library should implement the details in a subclass of ChoicesModifier.

API Name	update
Brief description	Update a series of modification
Return data	None
Parameters	
Kwargs	Key word parameter for modifications
Example	
dict1 = {“GPRs” : {“x1”: 30, “x0”:50} } for tree_name, value in dict1.item(): self.modifyOperandChoices(tree_name, value)	

5.9.11 revert

Revert all modify choices and they will NOT affect instruction generation any longer

API Name	revert
Brief description	Revert all chocies modification
Return data	None
Parameters	
Apply_ID	Revert to the version identified by Apply ID
Example	
self.revert(apply_id)	

5.9.12 getChoicesTreeInfo

Get all choices information on a choices tree.

API Name	getChoicesTreeInfo
Brief description	Get all choices tree information
Return data	Dictionary. Its key is choice name, its value is choice weight
Parameters	
treeName	Choice tree name
treeType	Choice tree type like OperandChoices, RegisterFieldValueChoices, PagingChoices, GeneralChoices and DependenceChoices force-riscv/riscv/arch_data/operand_choices.xml force-riscv
Example	
choices = self.getChoicesTreeInfo(“GPRs”, “OperandChoices”) for (name, weight) in sorted(choices.items()) self.notice(“%s:%d”%(name, weight))	

5.10 Variable APIs

FORCE-RISCV provides variable module to configure some issues dynamically. Two types are supported so far: Value and Choice. When type is Value, the variable is an integer. When the type is Choice, the variable is range-weight string. For example: <Variable name="looking up window" value="12" type="Value" />;

<Variable name="Aging choices" value="0-5 :30, 6-9: 50, 10-18:20" type="Choice">

5.10.1 modifyVariable

Modify variables dynamically, which will be applied by the end of test case.

API Name	modifyVariable
Brief description	Modify a variable to set its value.
Return data	None
Parameters	
name	Variable name to modify
value	The value to set after modification
type	Variable type. "Value" or "Choice" supported so far.
Example	
self.modifyVariable("Aging choices", "0-5:10,6-9:70,10-12:20", "Choice") Modify the variable "Aging chocies" to 0-5 with weight 10, 6-9 with weight 70, 10-12 with weight 20.	

5.10.2 getVariable

Get variable value in string.

API Name	getVariable
Brief description	Get variable value in string
Return data	Variable value
Parameters	
Name	Variable name to modify
Type	Variable type. "Value" or "Choice" supported so far.
Example	
var = self.getVariable("Aging choices", "Choice")	

self.notice(“variable value %s” % var)

5.11 Bnt Sequence APIs

There are two types of Bnt sequences: the default Bnt sequence which is applied when no customized sequence is defined, and a customized Bnt Sequence which is applied to a group of specific conditional branch instructions.

5.11.1 setBntHook

Hook customized Bnt sequence. If no bnt sequence is set, then the default Bnt sequence is used.

API Name	setBntHook
Brief description	Hook customized Bnt Sequnce
Return data	hook id
Parameters	
**kargs	Dictionary arguments. The keys supported are “Seq” and “Func”.
Example	
Hook_id = self.setBntHook(Seq = “MyBntSequence”, Func = ”my_func”)	

5.11.2 revertBntHook

Revert Bnt hook to the scenario before ID.

API Name	revertBntHook
Brief description	Revert Bnt hook to the scenario before ID.
Return data	No
Parameters	
Hook id	
Example	
self.revertBntSequence(hook_id) // revert bnt sequence to the last one.	

5.12 Misc APIs

Various other useful APIs.

5.12.1 getOption

The user can create various options to be passed in from test generator command line. Depending on the value associated with the option, the user can write sequence that behave differently based on the option value passed in.

API Name	getOption
Brief description	Get value for option that the user might have set.
Return data	Value for the option.
Parameters	
Name of the option	The name of the option that the sequence behavior is dependent on.
Example	
opt_value = self.getOption(“dbg_step_enable”)	

5.12.2 pickWeighted

Use this API to pick a random choice from a Python dict of weighted choices.

API Name	pickWeighted
Brief description	Randomly pick a choice from a Python dict of weighted choices.
Return data	The randomly selected option.
Parameters	
Weighted choice dict	A Python dict with weighted choices.
Example	
self.pickWeighted(choice_dict)	

5.12.3 pickWeightedValue

Use this API to pick a random value from a Python dict of weighted ranges.

API Name	pickWeighted
Brief description	Randomly pick a value from a Python dict of weighted ranges.
Return data	The randomly selected option.
Parameters	

Weighted choice dict	A Python dict with weightedrange
Example	
mem_dict = {“0x0-0x7fffffff”: 0, “0x80000000-0xffffffff”: 100} mem_addr = self.pickWeightedValue(mem_dict) // pick a value from the ranges 0x80000000-0xffffffff.	

5.12.4 getPermutated

Use this API to pick one from a Python permutated list of dict of weighted choices. Note: this API returns a Python generator.

API Name	getPermutated
Brief description	Get one item from the permutated list from a Python dict of weighted choices.
Return data	The generator of the item in the permutated list
Parameters	
Weighted choice dict	A Python dict with weighted choices.
skip_weight_check	(Optional) default is False. Indicate whether to skip the weight check or not. If it is False, the weight of 0 choice will not be included in the permutated list
Example	
sequence.getPermutated (choice_dict, True) # skip the weight check sequence.getPermutated (choice_dict) # do not skip the weight check	

5.12.5 sample

Use this API to randomly pick a sub-list from given item list based on the given sample size.

API Name	sample
Brief description	Randomly pick up a sub-list from given item list based on the given sample size
Return data	The randomly picked sub-list
Parameters	

Items	A Python item list which could be list, tuple or string. It can NOT be a dictionary, however.
Sample_size	The sample size
Example	
sequence.sample (items, sample_size)	

5.12.6 choice

Use this API to randomly pick one from a given Python item list, which could be dictionary, list, tuple or string.

API Name	choice
Brief description	Randomly pick up an item from given item list, which could be dictionary, list, tuple or string
Return data	The randomly picked item from the given item list
Parameters	
items	A Python item list which could be dictionary, list, tuple or string
Example	
sequence.choice (items)	

5.12.7 choicePermutated

Use this API to pick one from a Python permutated list of given item list. The item list could be dictionary, list, tuple or string. Note: this API returns a Python generator.

API Name	choicePermutated
Brief description	Get one item from the permutated list from a Python item list, which could be dictionary, list, tuple or string
Return data	The generator of the item in the permutated list
Parameters	
items	A Python item list which could be dictionary, list, tuple or string
Example	
sequence.choicePermutated (items)	

5.12.8 genInstrOrSequence

While calling choice or choicePermutated, if user provides a list with mixed items, such as instruction string and “Sequence” sub-class, the return could be either instruction string or sequence subclass. genInstrOrSequence allows user to pass the returned item, no matter whether it is an instruction string or sequence subclass.

genInstrOrSequence examines the given item. If it is an instruction string, it calls “genInstrunction”. If it is a sequence subclass, it creates an instance of the given class and calls instance.run to execute the given sequence instance.

API Name	genInstrOrSequence
Brief description	Either call genInstruction or create a given sequence subclass instance and execute it
Return data	The instruction record if genInstruction is called. Otherwise, None
Parameters	
Items	An instruction string or sequence subclass
Example	
item = choice([SequenceSubClass, “ADD##RISCV”]) ret_id = genInstrOrSequence(item)	

5.12.9 random32

Use this API to pick a 32bit random number.

API Name	random32
Brief description	Randomly pick a choice from a 32bit random number.
Return data	32bit random value.
Parameters	
Example	
self.random32(0, 0xFFFFFFFF)	

5.12.10 random64

Use this API to pick a 64bit random number.

API Name	random64
Brief description	Randomly pick a choice from a 64bit random number.

Return data	64bit random value.
Parameters	
Example	
self.random64(0, 0xFFFFFFFFFFFFFFFF)	

5.12.11 error

Use this API to report an error to the back end with an error message. When the API is called, the ISG will exit with the error message reported.

Example:

```
self.error("failed to setup scenario at address 0x%x" % addr)
```

5.12.12 notice, warn, debug, info and trace

Use this set of APIa to report a log to the back end, which calls the same back-end log utility.

Example:

```
self.notice("Start generating instruction %s..." % instr)      # notice log level
self.warn("Start generating instruction %s..." % instr)        # warn log level
self.debug("Start generating instruction %s..." % instr)       # debug log level
self.info("Start generating instruction %s..." % instr)        # info log level
self.trace("Start generating instruction %s..." % instr)       # trace log level
```

5.12.13 bitstream

FORCE-RISCV provides a bitstream class to support bitstream operation. The front side bitstream is located at py/base/Bitstream.py. It provides a class called "Bitstream" with following functions:

- **__init__(self, bitstream = "")** – when user creates Bitstream, user can pass an initial bitstream or not.
- **append(self, bitstream)** – append given bitstream to the back of self.bitstream and return Bitstream object so user can call additional method on it
- **prepend(self, bitstream)** – prepend given bitstream to the front of self.bitstream and return Bitstream object so user can call additional method on it
- **value(self)** – return the converted value from self.bitstream
    - For example if self.bitstream is "1101 X 0000 xx01", then value returned is 1101000000000001 in integer

- **mask(self)** – return the converted mask from self.bitstream
    - For example if self.bitstream is “1101 X 0000 xx01”, then mask returned is 1111000011110011 in integer
- **valueMask(self)** – return the converted value and mask from self.bitstream as string of “value/mask”
    - For example if self.bitstream is “1101 X 0000 xx01”, the returned string is “1101000000000001 (hex string format)/1111000011110011 (hex string format)”. The actual string looks like “0xd001/0xf0f3”.
- **stream(self)** – return a formatted bitstream from self.bitstream
    - For example, if current self.bitstream is “1101 X 0000 xx01”, the returned bitstream is “1101xxxx0000xx01”
- **bits(self, bits_string)** – return a substring of self.bitstream based on given bits_string
    - For example, if self.bitstream is “1101 X 0000 xx01”
        - If bits_string is “0-2,5”, then returned string is “10x0”
        - If bits_string is “15-12, 8, 3-1”, then returned string is “1101xxx0”
- **__getitem__(self, args)**
    - Enable Bitstream to support indexing, such as my_stream[“1-3, 15”]
    - First check if args is an instance of “str”, if so, return call of bits(…) shown above
    - Otherwise, return “”

Example:

```
my_stream = Bitstream()        # self.bitstream is “” by default
my_stream.append(“ X ”)        # self.bitsrtream is “ X ” now
my_stream.append(“0000 xx01”).prepend(“1101”) # since append and prepend returns Bitstream object, user can cascade the calls while building the bitstream
my_stream[“1-3, 15”]           # return a substring of bit 1 to bit 3 and bit 15 in order
                               # the return string is “0xx1” in this example
my_stream[“15, 3-1”]           # return a substring of bit 15, and bit 3 to bit 1 in order
                               # the return string is “1xx0” in this example
my_stream.value()              # return an integer, which is 53249 in this example
my_stream.mask()               # return an integer, which is 61683 in this example
my_steam.valueMask()           # return a hex string, which is “0xd001/0xf0f3” in this example
```

5.12.14 genData

Use this API to random one data using user specified data pattern.

API Name	genData

Brief description	Generate one data using user specified data pattern.
Return data	The data that generated by user specified data pattern.
Parameters	
aPattern	The pattern of data, the format is same as “OperandData” format. (1) Integer format : IF_TYPE(value or ranges). IF_TYPE = <INT16> or <INT32> or <INT64>. (2) Floating point format : FP_TYPE(exp=value or ranges)(sign=value)(frac=value or ranges), FP_TYPE = <FP16> or <FP32> or <FP64>; (3) Simd format: [index]TYPE()[index]TYPE(), TYPE = IF_TYPE or FP_TYPE;
Example	
int32 = self.genData(“INT32(0x55555-0x66666)”) fp64 = self.genData(“FP64(sign=1)(exp=0x100-0x200)(frac=0x300)”) vec64 = self.genData(“[0]INT32(0x1)[1]INT32(0x2)”) sve256 = self.genData(“[1,2,3,4]INT64”)	

Chapter 6 Sequence examples

Complicated random/directed-random test templates can be developed using FORCE-RISCV APIs and sequence libraries while maintaining readability and maintainability of the test template.

Even a simple test sequence can provide good coverage of a large design space due to the good quality of the randomness in the back-end.

The well presented view of threading makes it very easy to design tests with cooperation between threads. The clean interface makes it very enjoyable for DV users to use and easy to achieve high productivity.

Sequences are the focal points of a FORCE-RISCV test template. The examples below show sequences without other minor details in a test template.

6.1 Basic Sequences

Example 1 shows a basic sequence that generates 100 random instructions from an instruction tree dictionary. The API is accessible by subclassing the Sequence.py class and overriding the 'generate' method. From the generate method in the subclass, Sequence methods can be called to interface with FORCE-RISCV to generate instructions and specify constraints on how the instructions get generated.

This example uses an instruction tree dictionary that available in

py/DV/riscv/trees/instruction_tree.py

You can create your own instruction tree dictionary to define the set of instructions that you want FORCE-RISCV to choose from when it generates instructions and the corresponding frequency weightings. See Section XXXX for more details.

The self.pickWeighted() method randomly chooses an instruction from the specified instruction dictionary. See Section XXXX for a description of the API.

```
# examples/riscv/um_itree_01_force.py
#
class I100Sequence(Sequence):
    def generate(self, **kwargs):
        for _ in range(100):
            instr = self.pickWeighted(ALU_Int32_instructions)
            self.genInstruction(instr)
            self.notice(“>>>>>  The instruction:  {}”.format(instr))
```

Example 01: From examples/riscv/um_itree_01_force.py. Random instruction selected, then generated. The dictionary ALU_Int32_instructions is in py/DV/riscv/trees/instruction_tree.py. 100 integer 32-bit ALU ops are generated with the instructions chosen randomly and the operands and data for each instruction chosen randomly.

```
# examples/riscv/um_itree_02_force.py
#
class F100Sequence(Sequence):
    def generate(self, **kwargs):
        for _ in range(100):
            instr = self.pickWeighted(ALU_Float_Double_instructions)
            self.genInstruction(instr)
            self.notice(“>>>>>  The instruction:  {}”.format(instr))
```

Example 2: From examples/riscv/um_itree_02_force.py. Random instruction selected, then generated. The dictionary ALU_Float_Double_instructions is in py/DV/riscv/trees/instruction_tree.py. 100 Floating point 64-bit ALU instructions are generated with the instructions chosen randomly and the operands and data for each instruction chosen randomly.

6.2 Slightly more complicated sequences

This next sequence uses a couple of additional APIs and a little bit of Python logic to generate an instruction sequence that has 100 randomly selected load or store instructions. The target address for these loads and stores is set to be one of the addresses near the very end of a page such that the access will frequently, but not always, cross a page boundary. The min_addr value can be tweaked if it is desired that the target access of every generated non-byte sized load/store will cross a page boundary.

```
# examples/riscv/um_pageCrossing_01_force.py
#
from DV.riscv.trees.instruction_tree import LDST_All_instructions
from DV.riscv.trees.instruction_tree import LDST_Byte_instructions
from DV.riscv.trees.instruction_tree import LDST_Half_instructions
from DV.riscv.trees.instruction_tree import LDST_Word_instructions
from DV.riscv.trees.instruction_tree import LDST_Double_instructions

class MyMainSequence(Sequence):
    def generate(self, **kwargs):
        for _ in range(100):
            instr = self.pickWeighted(LDST_All_instructions)

            # Get two adjacent 4K pages.
            self.genVA(Size=0x2000, Align=0x1000)

            # Calculate a target address for the load/store
            # instruction that will generate frequent page
            # crossings.

            if instr in LDST_Byte_instructions:
                min_addr = 0xFFC
            elif instr in LDST_Half_instructions:
                min_addr = 0xFFC
            elif instr in LDST_Word_instructions:
                min_addr = 0xFFA
            elif instr in LDST_Double_instructions:
                min_addr = 0xFF6
            else:
                self.error(">>>>>  Hmmm…  {} is an unexpected \
                instruction.".format(instr))

            target_addr = page_addr + self.random32(min_addr, 0xFFF)

            self.genInstruction(instr, {"LSTarget":target_addr})
```

Example 3: From examples/riscv/um_pageCrossing_01_force.py. The genVA API is used to ask the FORCE_RISCV to provide a virtual address that can be used. Options are provided to define a size of 2x4K pages and an alignment on a 4K page boundary. The self.random32 can be used to find a value between the min and max values – in this case, chosen to produce a page offset that is near the end of a page. A dictionary of options is specified in the genInstruction call to tell FORCE-RISCV to use a specific value for the target address of the load/store instruction that gets generated.

The following example demonstrates that you can build a hierarchy of sequences with a main sequence that runs other sequences. Those sequences, in turn, can also call other sequences. The main sequence is shown instantiating two other sequences which are also contained within the file (I100Sequence and F100Sequence). These are t

```
from DV.riscv.trees.instruction_tree import ALU_Int32_instructions
from DV.riscv.trees.instruction_tree import ALU_Float_Double_instructions

class MyMainSequence(Sequence):

      # Main sequence which calls the other sequences.
      def generate(**kwargs):

         i100_seq = I100Sequence(self.genThread)
         i100_seq.run()

         f100_seq = F100Sequene(self.genThread)
         f100_seq.run()


class I100Sequence(Sequence):
      # generate 100 random integer 32 ALU ops
      def generate(self, **kargs):
            for _ in range(100):
                  instr = self.pickWeighted(ALU_Int32_instructions)
                  self.genInstruction(instr)
                  self.notice(">>>>>  The instruction:  {}".format(
                        instr))

class F100Sequence(Sequence):
      # generate 100 random integer 32 ALU ops
      def generate(self, **kargs):
            for _ in range(100):
                  instr = self.pickWeighted(ALU_Float_Double_instructions)
                  self.genInstruction(instr)
                  self.notice(">>>>>  The instruction:  {}".format(
                        instr))
```

Example 4: From examples/riscv/um_sequences_01_force.py. MyMainSequence calls the other two sequences and uses the run() method to execute the generate methods in the other sequences.

6.3 Sequence library

Sequences can be put into a sequence library which has a list containing the sequences and two methods for selecting a sequence from the list. Each sequence in the list is assigned a weighting for priority. chooseOne() selects one sequence randomly from the list. getPermutated() returns a generator that selects each sequence in the list in a random order.

The user can create his own sequence library such as “MySequenceLibrary” that is derived from the base “SequenceLibrary” to implement the “createSequenceList” method which creates the actual sequence list in the sequence library. Additional derived sequence libraries can be added into the sequence list as well.

```
from riscv.EnvRISCV import EnvRISCV
from riscv.GenThreadRISCV import GenThreadRISCV
from base.Sequence import Sequence
from base.SequenceLibrary import SequenceLibrary

class MainSequence(Sequence):
    def generate(self, **kwargs):
        seq_library = MySequenceLibrary(self.genThread)
        # 4 iterations of selecting a sequence randomly from the
        # sequence list
        for _ in range(4):
            current_sequence = seq_library.chooseOne()
            current_sequence.run()

        # In a random order, select and run with each sequence in the
        # sequence list

        for current_sequence in seq_library.getPermutated():
            current_sequence.run()

class  MySequenceLibrary(SequenceLibrary):
    def createSequenceList (self):
        # sequence list with only 2 sequences – equally weighted
        Self.seqList = [\
        (“Bunch_of_ALU_Int”, “DV.riscv.sequences.BasicSequences”,\
        “Your description”, 20),\
        (“Bunch_of_LDST”, “DV.riscv.sequences.BasicSequences”,\
        “Your description”, 20)]

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

Example 5a: From examples/riscv/um_seqLibrary_01_force.py. Subclassing SequenceLibrary to create a sequence list. The file containing the sequences being added to the list is py/DV/riscv/sequences/BasicSequences.py.

```
from riscv.EnvRISCV import EnvRISCV
from riscv.GenThreadRISCV import GenThreadRISCV
from base.Sequence import Sequence
from DV.riscv.trees.instruction_tree import ALU_Int_All_instructions
from DV.riscv.trees.instruction_tree import LDST_All_instructions

class Bunch_of_ALU_Int(Sequence):

    def generate(self, **kwargs):

            self.notice("Generating in 'Bunch_of_ALU_Int'")
            for _ in range(self.random32(5, 20)):
                  instr = self.pickWeighted(ALU_Int_All_instructions)
                  self.genInstruction(instr)


class Bunch_of_ALU_Int(Sequence):

    def generate(self, **kwargs):

            self.notice("Generating in 'Bunch_of_ALU_Int'")
            for _ in range(self.random32(5, 20)):
                  instr = self.pickWeighted(ALU_Int_All_instructions)
                  self.genInstruction(instr)
```

Example 5b: From py/DV/riscv/sequences/BasicSequences.py. The template in example 5a creates a sequence list by pointing to the two sequences contained in this separate file.

6.4 Creating user defined instruction set or tree

Instruction_trees are valuable because they can be used with the pickWeighted API to randomly choose an instruction. Each instruction has an integer weighting number to adjust the probability of it being chosen relative to all of the other instructions in the tree. Examples can be found in py/DV/riscv/trees/instruction_tree.py.

```
# each instruction has a weight that can be adjusted
random_iset = { "ADD##RISCV":10,
                "SD##RISCV":10,
                "BEQ##RISCV":10,
                "FMADD.D#Double-precision#RISCV":10 }
# increase weight for SD
random_iset["SD##RISCV"] = 100
# randomly choose an instruction - SD will be chosen more often
picked_instr = self.pickWeighted(random_iset)
```

Example 6: You can create your own instruction tree by building your own Python dictionary as shown in the following example.

A user can define an instruction tree using a hierarchy if desired. An example is shown below.

```
# example/riscv/um_itree_03_force.py
# defining your own instruction tree and building and using a
# hierarchical instruction tree.
from base.InstructionMap import InstructionMap
from DV.riscv.trees.instruction_tree import LDST_All_map
from DV.riscv.trees.instruction_tree import ALU_Int_All_map
from DV.riscv.trees.instruction_tree import ALU_Float_Double_map

class MainSequence(Sequence):
     def generate(self, **kargs):
          for _ in range(100):
               # define a single level instruction tree
               random_iset1 = {
                    "FMADD.S#Single-precision#RISCV":10,
                    "FMAX.S##RISCV":20,
                    "FMIN.S##RISCV":30 }
               instr = self.pickWeighted(random_iset1)
               self.genInstruction(instr)

               ## define a multilevel instruction tree
               random_iset2 = {
                    "ADD##RISCV":10,
                    "SD##RISCV":10,
                    "BEQ##RISCV":10,
                    "FMADD.D#Double-precision#RISCV":10 }
               random_iset3 = {
                    "JAL##RISCV":70,
                    "LUI##RISCV":50,
                    "FENCE##RISCV":30,
                    "ORI##RISCV":10 }
               # create maps
               iset1_map = InstructionMap("random_iset1,
                              random_iset1)
               iset2_map = InstructionMap("random_iset2,
                              random_iset2)
               iset3_map = InstructionMap("random_iset3,
                              random_iset3)
               # build the tree with the maps
               random_itree = { iset1_map:10,
                               iset2_map:10,
                               iset3_map:10 }
               instr = self.pickWeighted(random_itree)
               self.genInstruction(instr)
```

Example 7: From examples/riscv/um_itree_03_force.py. A hierarchical instruction tree using the InstructionMap class.

6.5 Controlled way to setup instruction dependency

An example is to use the target of last instruction and use it as source of the next instruction. We can use queryInstructionRecord API to query information regarding last instruction, obtain DEST register index, among other things, then we can specify the next instruction to use it as one of its sources.

```
Class MyMainSequence(Sequence):
     def generate(self, **kargs):
          for _ in range(10):

               instr_rec_id1 = self.genInstruction("ADD##RISCV")

               # The object returned is a dictionary.  The value
               # of some keys are themselves dictionaries.
               instr_obj = self.queryInstructionRecord(instr_rec_id1)

               # The key "Dests" indicates the write target of the
               # instruction.
               # get dict of target regs, then the id for "rd"
               target_regs = instr_obj["Dests"]
               target_reg_id = target_regs["rd"]

               # if rd is 0, then it is not being written so
               # skip to the next pair of instructions
               if target_reg_id == 0

               # generate new instr with rs1 = target_reg of the
               # previous instr
               instr_rec_id2 = self.genInstruction("SUB##RISCV",
                         {"rs1":target_reg_id})

               instr_obj2 = self.queryInstructionRecord(instr_rec_id2)
               source_regs = instr_obj2["Srcs"]
               source_reg_id = source_regs["rs1"]

               # resulting instructions will look something like…
               # add  x5,x7, x2
               # sub  x17, x5, x28
               # with the value of "rd" of the ADD matching the "rs1"
               # of the SUB.
```

Example 8. From examples/riscv/um_regDependency_01_force.py.

The next example shows an alternative method for forcing the generation of a register dependency. In stead of using the queryInstructionRecord to determine which register was used in the generated instruction, this approach finds an available register and then specifies the usage of that register when the writer and reader instructions are generated.

```
Class MyMainSequence(Sequence):
      def generate(self, **kargs):

            for _ in range(100):

                  # request the id of an available GPR – avoid "x0"
                  reg_id = 0
                  while(reg_id ==0):
                        reg_id = self.getRandomGPR()

                  # generate the instructions using that register
                  instr_rec_id1 = self.genInstruction("ADD##RISCV",
                                    {"rd":reg_id})
                  instr_rec_id2 = self.genInstruction("SUB"##RISCV",
                                    {"rs2":reg_id})

                  # check the results: instr1 rd should = instr2 rs2
                  instr_obj1 = self.queryInstructionRecord(instr_rec_id1)
                  instr_obj2 = self.queryInstructionRecord(instr_rec_id2)

                  instr1_rd_index = instr_obj1["Dests"]["rd"]
                  instr2_rs2_index = instr_obj2["Srcs"]{"rs2"]

                  if instr1_rd_index == instr2_rs2_index:
                        self.notice(">>>>>>>>>>>  It worked.  ")
                  else:
                        self.error(">>>>>>>>>>>  FAIL – reg indexes \
                                    did not match.")
```

Example 9: From examples/riscv/um_regDependency_02_force.py.

6.6 Create user defined choices modifier and control choices during generation

FORCE-RISCV uses weightings to determine how often to select a particular choice when multiple choices are available. For example, when selecting a GPR to use when generating an instruction, the GPR choices are often evenly weighted making one GPR just as likely to be chosen as another. Sometimes the user may want to increase (or decrease) the likelihood of choosing a certain GPR. This can be done do using the ChoicesModifier class object. The

following example shows the weightings being changed at the start of the test using the GenThreadInitialization interface.

```
Class MyMainSequence(Sequence):
      # Change the "choices" settings at the start of the test by using
      # the gen_thread_initialization entry point.

      def generate(self, **kwargs):
            usage_count = {}
            for _ in range(1000):
                  # the selection of GPRs to use for generation will be
                  # affected by the fact that the weightings were
changed
                  # in the gen_thread_initialization function.
                  instr = self.pickWeighted(RV_G_instructions)
                  instr_rec_id = self.genInstruction(instr)

                  # get the indexes for the GPRs that were used.
                  instr_obj - self.queryInstructionRecord(instr_rec_id)

            # ====================================================
            # Some code not shown for brevity.  Please see test
            # template in the repository to view the full source code.
            # ====================================================


# Modify GPR wightings.  A higher weighting means force-riscv is more
# likely to choose that value during a weighted random selection when
# generating a new instruction.
# This will be run before the MyMainSequence.generate()
def gen_thread _initialization(gen_thread):

      choices_mod = ChoicesModifier(gen_thread)

      # Increase the likelihood of using GPRs x10, x11, x12, x13 by
      # increasing the wighting.  The default weighting in the
      # operand_choices.xml file is 10 for each GPR.
      choices_mod.modifyOperandChoices("GPRs", {"x10":40, "x11":40,
                        "x12":60, "x13":60})

      choices_mod.commitSet()

GenThreadInitialization = gen_thread_initialization
```

Example 10: From examples/riscv/um_choicesMod_01_force.py.

6.7 Creating a custom entry point and control flow for a utility sequence

The following example shows how to override the normal control flow of the run() function of a sequence. This can be useful inside of utility sequences or subsequences of a test that do not necessarily need to follow the setup(), generate(), and cleanUp() default control flow. By setting the sequence parameter 'entryFunction' to the function you'd like to use as your entry point, run will now execute that sequence instead of the default control flow. Note that the keyword parameters to run() must now contain and match the names of the entry points positional parameters, and additional keyword parameters will be retained inside of the keyword argument dictionary.

```
class testUtil(Sequence):
  def __init__(self, gen_thread):
    super(testUtil, self).__init__(gen_thread)
    self.entryFunction = self.load
    self.testValue = 0

  def load(self, arg1, arg2, **kargs):
    self.testValue = arg1 + arg2
    self.generate(**kargs)

  def generate(self, **kargs):
    if kargs is not None:
      for key, value in kargs.items():
        self.notice("%s %s" % (key, value))
    self.notice("test value=%d" % self.testValue)

class MainSequence(Sequence):
  def generate(self, **kargs):
    test_util = testUtil(self.genThread)
    # Run with entry function specified as test_util.load
    test_util.run(arg1=1, arg2=4, arg3=423)
    test_util.entryFunction = None
    # Run with default control flow, in this case test_util.generate
    test_util.run(arg1=1, arg2=9, arg3=32)
    # Execute load function directly, without using run()
    test_util.load(2,4)
```

Chapter 7 Choices and variables

FORCE-RISCV has defined some variables and choices for the user to control the behavior of the ISG either for the whole test or in a specific scope.

This section will describe some of the choices and/or variables that need some explanation to facilitate their usage.

7.1 Register dependency controls

Quite a few register dependency controls are defined in the variable.xml file.

7.1.1 "Inter-Dependency Window" variable

This variable let the user define a desired distribution instruction window to base the register dependency on. The default value is currently: *"1-20:90,21-30:10"*. This string defines a distribution of window of *"1-20"* with weight of 90 and window *"21-30"* with weight of 10.

With the default distribution, most of the time we will be picking "1-20" and selection a random number in this range. Let's say we picked 10, then we will be selecting register dependency candidates from last 1 to 10 instructions.

The "Inter-" prefix here indicates the dependency is between instructions, not among the register operands of the same instruction.

The user can specify a different distribution to replace the default distribution in a test template. The window value can also just be a constant.

7.2 Register reloading controls

In order to reduce the preamble instructions, the generator automatically adds some available address to GPRs before generating load/store instructions when address shortage is detected.

7.2.1 "Enable register reloading" variable

This variable let the user can enable/disable the reloading register mechanism in different templates according to the required. The default value is *"1"* means that it is enabled.

If user want to disable the mechanism, user can specify a *"0"* to replace the default value in a test template.

7.2.2 “Reloading registers number” variable

For the number of reloading registers, user can define a desired register number of each batch reloading sequence. The default value is currently: *“20-15”*, this string means the maximum number of reloading registers is between 20 and 15.

This range constraint is not hard, the back-end will try to satisfy the number if there are enough unreserved registers.

7.3 Address table allocation controls

7.3.1 “Address table memory size” variable

This variable is control the memory size of address table. In general, this value should be not changed.

7.4 Page table allocation controls

The page table allocation provides flexibility in terms of creating, using, and expanding the page table. User can control the page table regions by change the following controls variables in the variable.xml file.

7.4.1 “Page tables number per block allocating” variable

Page table constraint allocate one continuous block and then allocate one page table in this block each time. If the block is used up all the available space, it will request other block. This variable is define the number of page tables for per block allocating. The default value is currently: *“0x200”*, means 512 page tables will be allocated when per block allocating.

Note: this value should be in an appropriate interval, too small will make page tables too scattered, so it will slow down the generation performance. Opposite, too large will reduce randomness of page tables, most of page tables may be at a continuous address region.

Appendix A Architecture version compliance

FORCE-RISCV ISG supports these versions of the base architecture and architectural extensions.

Base	Version	Supported?	Enabled by Default?	Comments
RVWMO	2.0			
RV32I	2.1			
RV64I	2.1			
RV32E	1.9	Not supported		
RV128I	1.7	Not supported		
Extension				
M	2.0			
A	2.0			
F	2.2			
D	2.2			
Q	2.2	Not supported		
C	2.0			
Counters	2.0			
N		Not supported		
L	0.0	Not supported		
B	0.0	Not supported		
J	0.0	Not supported		
T	0.0	Not supported		
P	0.2	Not supported		
V	0.8			
Zicsr	2.0			
Zifencei	2.0			
Zam	0.1	Not supported		
Ztso	0.1	Not supported		