ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

СОГЛАСОВАНО
Доцент базовой кафедры
«Системное программирование»
Института системного программирования
им В.П. Иванникова РАН (ИСП РАН)
факультета компьютерных наук,
кандидат физико-математических наук

_____ А. С. Камкин
«01» апреля 2023 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы факультета
компьютерных наук
«Программная инженерия»
профессор департамента программной
инженерии, кандидат технических наук

_____ В. В. Шилов
«7» апреля 2023 г.

ПРОТОТИП ОНЛАЙН-ГЕНЕРАТОРА ТЕСТОВЫХ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ

Руководство системного программиста

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.04-01 32 01-1-ЛУ

Исполнитель
студент группы БПИ201
_____ / М. Ю. Литвинов /
«01» апреля 2023 г.

Москва, 2023

УТВЕРЖДЕН

RU.17701729.04.04-01 32 01-1-ЛУ

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРОТОТИП ОНЛАЙН-ГЕНЕРАТОРА ТЕСТОВЫХ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ

Руководство системного программиста

RU.17701729.04.04-01 32 01-1-ЛУ

Листов 28

Москва, 2023

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ 5
Наименование программы 5
Наименование программы на английском языке 5
Функциональное назначение 5
Эксплуатационное назначение 5
Состав выполняемых программой функций 5
Требования к составу и параметрам технических средств 6
Требования к составу и параметрам программных средств 6
Описание используемого математического аппарата 7
Описание общего алгоритма работы 10
2. СТРУКТУРА ПРОГРАММЫ 12
Описание файла «main.c» 12
Директория «core/common» 13
Описание файла «types.h» 13
Описание файлов «utility.h» и «utility.c» 13
Директория «core/random» 13
Описание файлов «rand.h» и «rand.c» 13
Директория «core/decls» 13
Описание файла «ins_decls.h » 13
Описание файла «tests_decls.h» 13
Описание файла «mut_decls.h» 14
Описание файла «predef_ins_macros.h» 14
Директория «core/memory» 14
Описание файла «memutils.h» и memutils.c» 14
Описание файла «ins_memutils.h» и описание файла «ins_memutils.c» 14
Директория «core/generation» 14
Описание файла «generator_context.h» 14
Описание файлов «generator.h» и «generator.c» 14

Москва, 2023

Директория «core/template_block» 14
Описание файла «template_block_decls.h» 14
Описание файла «template_block.h» 15
Описание файла «block_process_info.h» 15
Описание файла «block_process_result.h» 15
Описание файлов «template_block_type.h» и «template_block_type.c» 15
Директория «core/execution» 15
Описание файла «runner.h» и «runner.c» 15
Директория файлов описания ISA 15
Описание файлов «instruction.h» и «instruction.c» 15
Директория файлов описания частично заданных и строго зафиксированных тестов 16
Описание файлов «tests.h» и «tests.c» 16
Директория файлов описания мутаций созданных последовательностей инструкций 16
Описание файла «mutator.c» 16
3. НАСТРОЙКА ПРОГРАММЫ 17
Подготовка программного окружения 17
Описание формата входных данных 17
Описание файлов «instruction.h» и «instruction.c» 17
Описание файлов «tests.h» и «tests.c» 19
Описание файла «mutator.c» 21
Сборка проекта 21
Описание формата выходных данных 23
4. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ 24
5. СООБЩЕНИЯ СИСТЕМНОМУ ПРОГРАММИСТУ 25
Отображение информации компилируемого онлайн-генератора 25
Отображение сопроводительной информации 26
6. СПИСОК ИСПОЛЬЗОВАВШИХСЯ ИСТОЧНИКОВ 27
7. ПРИЛОЖЕНИЯ 28
Приложение 1: Словарь использующихся терминов и понятий 28

Москва, 2023

Москва, 2023

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Наименование программы

Прототип онлайн-генератора тестовых программ для микропроцессоров.

Наименование программы на английском языке

Prototype of Online Test Program Generator for Microprocessors.

Функциональное назначение

Программа предоставляет возможности генерации и исполнения тестовых последовательностей из инструкций целевой микропроцессорной архитектуры (Instruction Set Architecture, ISA) верифицируемой системы (по умолчанию предоставляется реализация RISC-V RV32I-совместимой модели [1] микропроцессора), а также вынесения вердикта о корректности работы использующихся в тестах инструкций на основе сравнения результатов выполнения двух функционально эквивалентных в рамках заданных правил последовательностей.

Эксплуатационное назначение

Программа может являться компонентом более обширного комплекса программных средств для верификации микропроцессоров [3], ускоряя работу разработчиков микропроцессоров посредством автоматической генерации тестовых последовательностей из инструкций целевой ISA (с их непосредственным исполнением) и соответствующих им входных данных по установленным заранее принципам.

В частности, программа может применяться в отношении программируемых логических интегральных схем (ПЛИС), настроенной на работу с целевой ISA, при этом самостоятельно реализуя необходимые ей сервисы операционной системы по работе с памятью, если таковые понадобятся.

Состав выполняемых программой функций

- Использование пользовательских описаний ISA тестируемой модели микропроцессора, логики изменения (мутаций) генерируемых последовательностей инструкций, логики сравнения результатов выполнения оригинальной и измененной последовательностей инструкций для определения специфики конкретного генератора, а также моделей (шаблонов) частично заданных тестов и строго зафиксированных тестов;

- Последовательная генерация последовательностей из инструкций целевой архитектуры набора команд по предоставленным моделям частично заданных тестов (шаблонов) с подготовкой требуемых для их полного определения входных данных;
- Последовательная генерация последовательностей из инструкций целевой архитектуры набора команд по описаниям строго зафиксированных тестов;
- Использование пользовательского описания логики мутаций последовательностей генерируемых инструкций для их изменения в границах исходного уровня функциональной эквивалентности и дальнейшего сравнения результатов выполнения каждой сгенерированной последовательности инструкций целевой архитектуры команд с ее измененной версией;
- Подготовка и вывод итоговых вердиктов об успешности выполнения тестируемым прототипом микропроцессора каждой сгенерированной последовательности инструкций и ее измененной версии по заданным правилам сравнения.

Требования к составу и параметрам технических средств

Для успешной компиляции программы требуется следующий состав технических средств:

- Компьютер с одной из установленных операционных систем – Linux (x86-64) или Linux (ARM64);
- Наличие не менее 100МБ свободного места на жестком диске компьютера.

Иные требования совпадают с системными требованиями указанных операционных систем.

Поскольку скомпилированный прототип генератора будет использовать статическую память в глобальной области видимости в зависимости от требований самого пользователя (описывается набором использующихся регистров в рамках тестируемой системы, количеством тестов и итераций для отдельных тестов, объемом памяти для выгрузки результатов отдельных тестов и так далее), итоговое максимальное ограничение на объем оперативной памяти не может быть сформулировано однозначно.

В случае запуска генератора на уровне bare-metal системы (например, с использованием ПЛИС), могут потребоваться условия совместимости аппаратного обеспечения с существующими программными решениями для выделения регионов оперативной памяти, используемой в генераторе, а также для возможности изучить выгруженные генератором данные вердиктов о выполненных тестах.

Требования к составу и параметрам программных средств

Для успешной компиляции программы требуется следующий состав программных средств:

- Кросс-компилятор языка Си (стандарт C99) с генерацией кода, совместимого с тестируемым прототипом микропроцессора на уровне его ISA;
- Система CMake[2] с открытым исходным кодом для кроссплатформенного описания процесса компиляции исходного кода на языке Си версии 3.16 или более поздней.

Если же запуск генератора будет производиться в рамках эмулятора микропроцессора с опциональным флагом вывода сопровождающих и отладочных сообщений, то вдобавок к вышеупомянутым требованиям добавится и требование совместимости с набором интерфейсов системных вызовов в рамках стандарта C99.

Описание используемого математического аппарата

В контексте поставленной задачи разделяются две основные группы тестов: частично заданные (шаблоны) и строго зафиксированные. В рамках динамического тестирования гораздо эффективнее иметь большой объем разнообразных тестов, где тестовые последовательности инструкций используют разные входные данные. Существует возможность описать некоторым образом набор тестов, что в совокупности с подстановкой псевдослучайных входных данных (если такое требуется) и большим количеством их итераций предоставляет возможность получить существенное количество рассмотренных случаев поведения тестируемого прототипа микропроцессора, требующих корректного выполнения используемых операций – подобные наборы тестов представляется шаблонами.

С другой стороны, некоторые отдельные случаи ожидаемого поведения микропроцессора достаточно трудно сгенерировать средствами генератора псевдослучайных последовательностей чисел за счет малой вероятности создания их точных копий, а потому некоторые случаи достаточно рассмотреть ровно один раз – именно в такую категорию тестов попадают строго зафиксированные тесты, задающие четкую последовательность инструкций без пропущенных входных данных.

Заметим некоторую систематичность, что любая тестовая последовательность инструкций представима в виде некоторой перестановки над элементами большей совокупности инструкций с определенными правилами извлечения инструкций для перестановки, а та совокупность, в свою очередь, определяется на основе обработки других совокупностей инструкций – в рамках выработанной терминологии взаимодействия с частично заданными тестами подобные совокупности инструкций получили название *шаблонных блоков*, а процесс генерации перестановки инструкций блока с определенными правилами их извлечения получил название *обработки*.

Таким образом, можно прийти к основной идее для описания шаблонов генерации тестовых последовательностей инструкций, которая используется по аналогии, например, в программном комплексе

MicroTESK: шаблон генерации тестовых инструкций описывается через иерархическую структуру специфических шаблонных блоков (*дерево шаблона*), где для обработки конкретного блока в первую очередь требуется обработать все его вложенные блоки (или осуществить непосредственную обработку инструкций самого конкретного блока, если какие-либо вложенные блоки отсутствуют), на основе которых будет генерироваться последовательность инструкций для текущего блока – это означает, что при обработке самого внешнего блока (для которого все остальные блоки будут считаться вложенными и что никакой блок не имеет самый внешний блок в числе своих вложенных блоков) все остальные блоки будут уже обработаны.

На уровне обработки блока считается, что если блок не обладает вложенными блоками, а обладает непосредственным набором инструкций (в древовидной структуре сверху вниз от самого внешнего блока такие блоки будут располагаться в самом низу) для обработки, то этот набор инструкций будет считаться неявно принадлежащим некоторому (единственному) вложенному блоку.

На момент написания настоящего документа шаблонные блоки могут соответствовать одному из двух типов: *sequence* или *pool* соответственно.

Тип шаблонного блока sequence предписывает, что инструкции, находящиеся непосредственно на уровне самого sequence-блока или в любом обработанном блоке из числа вложенных, будут зафиксированы в итоговой перестановке инструкций точно так же, как они и были зафиксированы на момент начала обработки sequence-блока с сохранением исходной очередности. Неформально говоря, тип sequence описывает упорядоченное (мульти[1])множество инструкций. Любые вспомогательные флаги и параметры, которыми может обладать sequence-блок, игнорируются.

Тип шаблонного блока pool предписывает, что инструкции, находящиеся непосредственно на уровне самого pool-блока или в любом обработанном блоке из числа вложенных, будут включены в итоговую перестановку в определенном количестве на основе определенных правил, зафиксированных на уровне блока вспомогательными флагами.

К таким вспомогательным флагам относятся:

Выбор готовых блоков – вместо того, чтобы включать отдельные инструкции в перестановку, будут выбираться сразу обработанные блоки инструкций;

[1] Зависит от того, были ли в изначальном блоке полностью идентичные инструкции без незаполненных полей.

Возможность повторно выбирать элементы (блоки/инструкции) – предоставляется возможность включать одну и ту же инструкцию или блок в итоговую перестановку два и более раз;

Отключение учета относительного порядка (блоков/инструкций) – вместо того, чтобы выбирать элементы с учетом их относительного расположения, они будут выбираться независимо от него (выбрав элемент, никакой элемент до него не будет выбран впоследствии).

Все вышеупомянутые флаги по умолчанию не используются, а значит, в таком случае будут выбираться инструкции без повторов и с учетом их относительного следования. Можно утверждать, что выбирая столько же инструкций блоков или самих блоков, сколько и было всего из числа вложенных блоков, с сохранением относительного порядка и отсутствием повторов обработка будет происходить аналогично sequence-блоку.

Неформально говоря, тип pool описывает (упорядоченное) (мульти)множество инструкций. За исключением параметров количества выбираемых элементов и используемых флагов любые другие параметры игнорируются.

При обработке вложенных блоков с общим внешним блоком порядок их обработке соответствует тому порядку, который использовался при описании блоков изначально.

Следующее представление демонстрирует пример описания некоторого тестового шаблона (число в скобках обозначает некоторый уникальный номер вершины, сокращения «seq» и «pool» соответствуют типам шаблонным блоков sequence и pool; параметр «n» описывает количество элементов (блоков/инструкций), выбираемых в блоке pool; «BLOCKS», «WITHREPEATS» и «NOORDER» соответствуют флагам выбора блоков вместо инструкций, разрешения повторов при выборе элементов и отсутствии следования относительному порядку элементов при обработке pool-блока; многоточие показывает, наличие непосредственного набора инструкций блока вместо вложенных для него блоков).

$$(1)seq\begin{cases}(2)seq\{\dots \\ (3)pool(n=1,BLOCKS)\begin{cases}(4)seq\{\dots \\ (5)pool(n=3,WITHREPEATS,NOORDER)\{\dots \\ (6)seq\{\dots\end{cases}\end{cases}$$

Таким образом, для обработки вершины 1 требуется обработать вершины 2, 3 и 6, а для обработки вершины 3 требуется обработать вершины 4 и 5, тогда итоговым порядок обработки вершин будет 2, 4, 5, 3, 6, 1.

Описание общего алгоритма работы

Все основные параметры для работы с онлайн-генератором задаются на этапе его компиляции и на этапе его исполнения дополнительной обработки входных параметров не производится.

Онлайн-генератор начинает выполнение своих задач с инициализации тех рабочих областей оперативной памяти, с которыми будет происходить взаимодействие при генерации и выполнении тестов — для этого вызываются определенные процедуры, спецификация которых задается на пользовательском уровне (подгружаются инструкции ISA, сами тесты и иные вспомогательные данные).

Далее, вызывается процедура генерации тестов по описанным шаблонам: каждый шаблон имеет определенное количество итераций, которые для него необходимо провести, на каждой итерации заново формируя тестовую последовательность из заданного шаблона.

Так как для обработки конкретного шаблонного блока требуется провести обработку всех вложенных для него блоков, в иерархической структуре шаблона подобная обработка схожа со стандартным алгоритмом обхода графа данных в глубину.

Для каждой сгенерированной в итоге инструкции вызывается процедура, также определяемая на пользовательском уровне и определенная либо для конкретного шаблона, либо определенная по умолчанию для шаблонных тестов, позволяющая заполнить пропущенные данные в инструкциях с использованием логики генерации псевдослучайных последовательностей данных.

Далее, все инструкции транслируются в их двоичное представление и размещаются с учетом их выравнивания по длине самих инструкций в заготовленной заранее области памяти для инструкций.

Вызываются предварительно описанные на пользовательском уровне с помощью препроцессорных средств языка Си процедуры выгрузки информации о содержании регистров на момент вызова процедуры в заготовленную заранее область памяти для регистров – так как при оформлении и непосредственном вызове подобных процедур с помощью функций языка Си могут быть перезаписаны и утеряны данные отдельных регистров, то требуется использовать именно пользовательские макроопределения языка.

Упомянутая область памяти для инструкций трактуется на уровне программы как область инструкций самой программы, которые должны быть исполнены, и управление передается на нее – по завершении исполнения инструкций содержимое регистров выгружается в отдельную область памяти регистров, а также восстанавливается сохраненное до выполнения инструкций содержимое регистров.

Далее, происходит функционально эквивалентное преобразование изначальной тестовой последовательности, которые также описывается на пользовательском уровне, и для новой последовательности выполняются шаги из трех предыдущих абзацев настоящего документа.

С использованием объявленного на пользовательском уровне функционала определяются затронутые в рамках выполнения двух тестовых последовательностей регистры процессора и производится сравнение их сохраненного содержимого и выносится вердикт об отсутствии и наличии различий среди данных затронутых регистров – в случае наличия расхождений необходимая контекстная информация о тесте выгружается в заранее подготовленную для этого область памяти, также определенную на пользовательском уровне, а также при указании соответствующего флага на этапе компиляции генератора производится подробный вывод информации о текущем тесте в стандартный поток вывода языка Си.

В случае со строго зафиксированными тестами производятся практически все вышеупомянутые шаги, только уже не потребуются процедуры генерации тестовой последовательности по заданному шаблону и заполнения недостающих параметров инструкций, а также появляется возможность задавать на пользовательском уровне специфические процедуры проверки корректности выполнения тестов без расчета затронутых в рамках выполнения тестовых последовательностей регистров.

При возникновении расхождений среди данных затронутых регистров выполнение как шаблонных, так и строго зафиксированных тестов прекращается.

2. СТРУКТУРА ПРОГРАММЫ

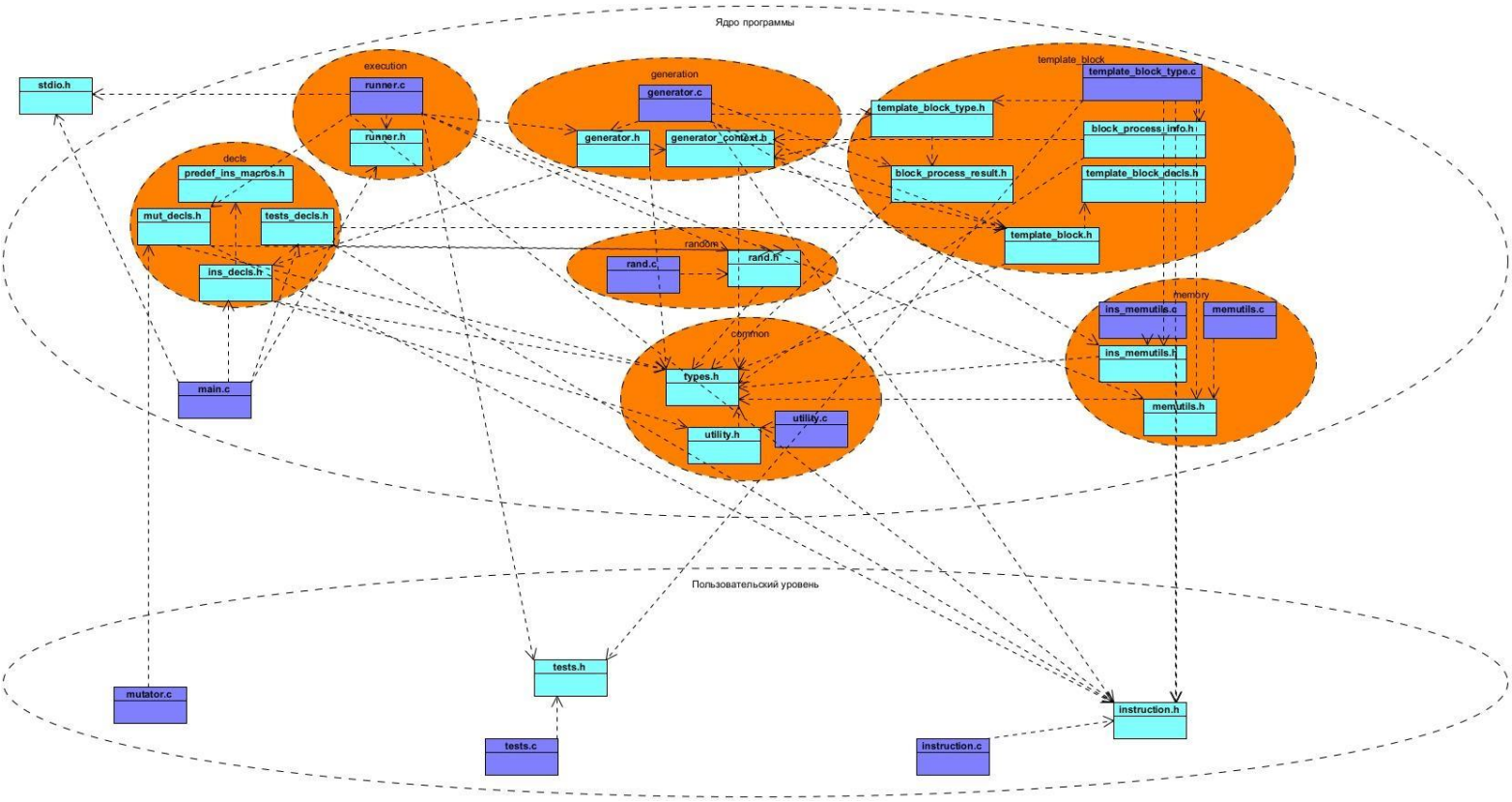

Рисунок 1 - схема зависимостей файлов проекта

Онлайн-генератор проектировался в рамках подхода, что любая зависимость, выражаемая через директиву #include языка Си, содержит самый необходимый минимум для успешной сборка проекта – в частности, это означает использование лишь прототипов структур данных, имеющихся в сигнатуре некоторого метода, в то время как само определение структуры данных находится в другом разделе проекта.

Описание файла «main.c»

Содержит одноименную функцию для спецификации стартовой точки работы онлайн-генератора с опциональным выводом сопроводительной информации (заданной конфигурации и сообщений-уведомлений о старте и завершении работы соответственно) в стандартный поток вывода.

Директория «core/common»

Описание файла «types.h»

Содержит описание использующихся фундаментальных типов, преждевременных деклараций (прототипов) определяемых пользователем структур данных, а также другие полезные макроподстановки.

Описание файлов «utility.h» и «utility.c»

Содержат описание и реализацию доступных вспомогательных функций, наиболее полезных на пользовательском уровне – в частности, двух версий функции подстановки данных в частично-заполненное двоичное представление некоторой инструкции целевой ISA.

Директория «core/random»

Описание файлов «rand.h» и «rand.c»

Содержат описание и реализацию одного из существующих алгоритмов генерации псевдослучайных последовательностей чисел на основе зафиксированных опорных значений («зерна»). В частности, предоставляются возможности получения чисел из диапазонов от [0; n) или [a; b).

Директория «core/decls»

Описание файла «ins_decls.h »

Содержит объявление определяемых пользователем глобальных переменных (указателей на области памяти) и функций для работы с некоторой моделью целевой ISA – в частности, областей памяти для хранения информации об имеющихся в ISA группах инструкций, инструкций и псевдоинструкций и функций их инициализации, а также областей памяти под хранение информации о данных регистров после выполнения тестовых последовательностей.

Также включает в себя все необходимые для работы с ISA зависимости.

Описание файла «tests_decls.h»

Содержит объявление определяемых пользователем глобальных переменных (указателей на области памяти) и функций для работы с некоторым набором частично заданных и строго зафиксированных тестов – в частности, областей памяти для хранения информации о самих тестах и выгрузки данных в процессе обработки частично заданных и строго зафиксированных тестов.

Также включает в себя все необходимые для работы с тестами зависимости.

Описание файла «mut_decls.h»

Содержит объявление определяемого пользователем алгоритма изменения существующей тестовой последовательности инструкций целевой микропроцессорной архитектуры с сохранением исходной функциональной эквивалентности

Описание файла «predef_ins_macros.h»

Содержит объявление стандартных макроподстановок-битовых флагов для спецификации инструкций и их операндов, которые могут быть полезны пользователю.

Директория «core/memory»

Описание файла «memutils.h» и memutils.c»

Содержат объявление и реализацию функций для работы с памятью на уровне отдельных байт – реализация аналогична используемой в актуальной на момент разработки проекта версии компилятора из коллекции GCC.

Описание файла «ins_memutils.h» и описание файла «ins_memutils.c»

Содержат объявление и реализацию функций для работы с памятью некоторой последовательности инструкций – реализация аналогична используемой в файлах «memutils.h» и «memutils.c».

Директория «core/generation»

Описание файла «generator_context.h»

Содержит объявление структуры данных для инкапсуляции данных, требуемых в процессе генерации тестовой последовательности на основе частично заданного шаблона.

Описание файлов «generator.h» и «generator.c»

Содержат объявление и реализацию функций для управления процессом обработки дерева конкретного шаблона – в частности, функции для непосредственной генерации тестовой последовательности и заполнения пропущенных входных данных имеющихся в ней инструкций.

Директория «core/template_block»

Описание файла «template_block_decls.h»

Содержит объявление стандартных макроподстановок-битовых флагов для спецификации шаблонных блоков (в частности, для типов sequence и pool).

Описание файла «template_block.h»

Содержит объявление структуры данных для репрезентации одного шаблонного блока, используемого для обработки некоторого частично-заданного теста.

Описание файла «block_process_info.h»

Содержит объявление структуры данных для инкапсуляции информации, необходимой для обработки одного шаблонного блока при условии уже обработанных блоков, являющихся вложенными для рассматриваемого – в частности, используемого контекста генератора, а также типа обрабатываемого блока и использующихся в его отношений параметров и флагов.

Описание файла «block_process_result.h»

Содержит объявление структуры данных для инкапсуляции информации об обработанном шаблонном блоке – количестве обработанных инструкций и количестве инструкций, сгенерированных на их основе.

Описание файлов «template_block_type.h» и «template_block_type.c»

Содержат объявление и реализацию функций для обработки конкретных типов шаблонных блоков, а также функции, которая делегирует эту обязанность упомянутым функциям.

Директория «core/execution»

Описание файла «runner.h» и «runner.c»

Содержат объявление и реализацию функций для выполнения тестов с предварительными вызовами процедур генерации тестовых последовательностей и входных данных для их инструкций там, где это требуется (частично заданные тесты) с опциональным выводом сопроводительной информации (о пройденных и не пройденных тестах) в стандартный поток вывода.

Директория файлов описания ISA

Описание файлов «instruction.h» и «instruction.c»

Содержат определение глобальных переменных и функций, объявленных в файле «ins_decls.h», а также необходимых для работы генератора макроподстановок.

Полное описание этих файлов находится в пункте 3.2.1 «Описание файлов «instruction.h» и «instruction.c»» настоящего документа.

Директория файлов описания частично заданных и строго зафиксированных тестов

Описание файлов «tests.h» и «tests.c»

Содержат определение глобальных переменных и функций, объявленных в файле «tests_decls.h», а также необходимых для работы генератора макроподстановок.

Полное описание этих файлов находится в пункте 3.2.2 «Описание файлов «tests.h» и «tests.c»» настоящего документа.

Директория файлов описания мутаций созданных последовательностей инструкций

Описание файла «mutator.c»

Содержит реализацию пользовательского алгоритма изменения существующей тестовой последовательности инструкций целевой микропроцессорной архитектуры с сохранением исходной функциональной эквивалентности, объявленного в файле «mut_decls.h».

Полное описание этого файла находится в пункте 3.2.3 «Описание файла «mutator.c»» настоящего документа.

3. НАСТРОЙКА ПРОГРАММЫ

Подготовка программного окружения

Предполагается соответствие системы для сборки онлайн-генератора требованиям, указанным в пункте 1.6 «Требования к составу и параметрам технических средств» настоящего документа.

Процесс загрузки исходного кода онлайн-генератора реализуется посредством использования возможностей распределенной системы контроля версий git для клонирования исходных файлов из удаленного репозитория проекта (https://forge.ispras.ru/projects/otpg/) на текущее устройство (команда *git clone --recursive https://forge.ispras.ru/git/otpg.git*).

Описание формата входных данных

Описание файлов «instruction.h» и «instruction.c»

В рамках описания ISA предполагается следующее разделение: существуют группы инструкций, включающие в себя набор инструкций с одинаковым форматом кодирования в двоичное представление, которое также описывается на уровне группы; любая инструкция без параметров содержит в себе некоторую ссылку на ту группу, к которой она принадлежит, а также ту информацию, которая остается неизменной при любых входных данных инструкции (например, код самой операции); каждая инструкция может иметь свое сокращенное представление – псевдо-инструкцию, в рамках которой одно или несколько полей инструкции-прообраза, на которую в псевдо-инструкции хранится ссылка, уже имеет зафиксированные данные (например, что инструкция прибавления значения к одному и тому же регистру имеет сокращенное представление с двумя операндами вместо трех); любая инструкция в рамках тестов представляется параметризованной инструкцией, которая может ссылаться как на псевдо-инструкцию, так и напрямую на инструкцию без параметров и которая может обладать своими собственными зафиксированными данными (оставшиеся данные должны быть сгенерированы в рамках обработки шаблонов).

Важно: файлы «instruction.h» и «instruction.c» должны обладать соответствующими header guard’ами «GEN_INSTRUCTION_H» и «GEN_INSTRUCTION_C»; заголовок должен иметь директиву #include “ins_decls.h”; если для определения указателей глобальной области видимости используются статически инициализированные массивы, тогда необходимо их пометить квалификатором extern.

Файлы с описанием ISA для верифицируемого прототипа должны включать в себя следующий минимальный набор аспектов:

- [Заголовок] Описание количеств использующихся регистров: регистров общего назначения, а также регистров для работы с вещественными числами, если те используются – используется макроподстановка «GEN_REG_COUNT»;
- [Заголовок] Описание количеств присутствующих в описании обычных инструкций и групп, на которые их можно поделить, а также псевдо-инструкций, реализуемых через имеющиеся обычные инструкции – используются макроподстановки «GEN_INS_COUNT», «GEN_INS_GROUP_COUNT» и «GEN_PSEUDO_INS_COUNT» соответственно;
- [Заголовок] Описание реализации зафиксированного интерфейса для способов выгрузки значений регистров из/в статически выделенные области оперативной памяти без изменения до/во время/после операции выгрузки соответствующих регистрам значений – для этого используются макроподстановки «LOAD_REG_DUMP» и «STORE_REG_DUMP» соответственно, которые принимают строковый аргумент-название массива в глобальной области видимости и которые не должны до/во время/после своей соответствующей операции менять значения регистров, а следовательно, требуется использовать синтаксис ассемблерных вставок языка Си;
- [Заголовок] Описание представлений групп инструкций, не-параметризованных инструкций, псевдо-инструкций и параметризованных инструкций – пользователем определяются соответствующие структуры данных «InstructionGroup», «Instruction», «PseudoInstruction» и «ParametrizedInstruction»;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализации зафиксированного интерфейса для получения длины конкретной инструкции в байтах – функции с сигнатурой *uint8_t get_ins_len(struct InstructionGroup *, struct Instruction *, struct PseudoInstruction *, struct ParametrizedInstruction)*;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализации зафиксированного интерфейса для кодирования (ассемблирования) инструкции по их имеющимся данным – функции с сигнатурой *ins_t encode_ins(struct ParametrizedInstruction *)*;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализации зафиксированного интерфейса для включения тех или иных регистров, изменяемых в рамках выполнения инструкции, в список таких регистров в рамках проведения одной итерации

некоторого теста – функции с сигнатурой *void include_regs(struct ParametrizedInstruction *, bool_t *)*;

- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализаций зафиксированных интерфейсов для инициализации областей памяти регистров , групп инструкций, инструкций, псевдо-инструкций – функций с сигнатурами void init_reg_dumps(), void init_groups(), void init_instructions() для указателей *struct InstructionGroup *GEN_INS_GROUPS, struct Instruction *GEN_INS, struct PseudoInstruction *GEN_PSEUDO_INS, reg_t *GEN_FIRST_REG_DUMP, reg_t *GEN_SECOND_REG_DUMP, reg_t *GEN_BUF_REG_DUMP, bool_t *GEN_REG_INCLUDE_IN_DUMP*.

Описание файлов «tests.h» и «tests.c»

При описании строго зафиксированных тестов требуется вручную задавать инструкции на уровне той области памяти, которая была под них выделена. В случае частично заданных тестов помимо этого требуется также задавать информацию о шаблонных блоках, на основе которых будет генерироваться итоговая тестовая последовательность.

Важно: любой тест должен включать в себя хотя бы одну достижимую инструкцию возврата (*return*) для передачи управления для корректного функционирования программы, и в любом шаблоне самый внешний блок должен иметь ID 0.

Важно: файлы «tests.h» и «tests.c» должны обладать соответствующими header guard’ами «GEN_TESTS_H» и «GEN_TESTS_C»; заголовок должен иметь директиву #include “tests_decls.h”; если для определения указателей глобальной области видимости используются статически инициализированные массивы, тогда необходимо их пометить квалификатором extern.

Файлы с описанием частично заданных и строго зафиксированных тестов для верифицируемого прототипа должны включать в себя следующий минимальный набор аспектов:

- [Заголовок] Описание максимальных допустимых количеств инструкций в рамках одной генерируемой последовательности (обязательный параметр) и инструкций, добавляемых в процессе мутаций последовательностей (необязательный параметр – при его отсутствии количество генерируемых дополнительно инструкций будет зависеть от первого упомянутого параметра) - используются макроподстановки «GEN_MAX_INS_COUNT» и «GEN_MAX_MUT_COUNT соответственно;

- [Заголовок] Описание количеств частично заданных и строго зафиксированных тестов - используются макроподстановки «GEN_TEST_TEMPLATES_COUNT» и «GEN_ASM_TESTS_COUNT соответственно;
- [Заголовок] Описание максимально допустимого количества блоков в рамках одного шаблона – используется макроподстановка «GEN_MAX_TREE_BLOCK_COUNT»;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание частично заданных тестов через конкретный для всех таких тестов интерфейс – реализация набора функций с сигнатурой *void …(struct TemplateBlock *, uint8_t struct ParametrizedInstruction *, uint16_t)*;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание строго зафиксированных тестов через конкретный для всех таких тестов интерфейс – реализация набора функций с сигнатурой *uint16_t …(struct ParametrizedInstruction *, uint16_t)*;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализации зафиксированного интерфейса для заполнения инструкций по умолчанию в рамках одной тестовой последовательности, если для обрабатываемого шаблона не задавалась специфическая функция – реализация функции с сигнатурой *void default_fill_ins(struct ParametrizedInstruction *)*;
- [Заголовок (с квалификаторами static/inline) / Файл исходного кода] Описание реализаций зафиксированных интерфейсов для инициализации областей памяти под частично заданные тесты, количества их последовательных запусков и пользовательские функции заполнения сгенерированных тестовых последовательностей, строго зафиксированные тесты и их пользовательские функции сравнения эквивалентности результатов выполнения двух тестовых последовательностей, иерархическую структуру шаблонных блоков, стеки обхода шаблонных блоков и размещения сгенерированных инструкций, а также вывод вердиктов об ошибках во время выполнения тестов и иные буферные области – функции с сигнатурами *void init_stacks(), void init_tree(), void init_test_patterns(), void init_asm_tests(), void init_misc()* для указателей *void (**GEN_TEST_TEMPLATES)(struct TemplateBlock *, uint8_t , struct ParametrizedInstruction *, uint16_t), uint8_t *GEN_TEST_TEMPLATES_ITERS, void (**GEN_TEST_TEMPLATES_FILL_INS)(struct ParametrizedInstruction *), uint16_t (**GEN_ASM_TESTS)(struct ParametrizedInstruction *, uint16_t), bool_t (**GEN_ASM_TESTS_DUMP_EQ)(ins_t *, ins_t *, uint8_t), bool_t *GEN_USED_ELEMENTS, byte_t *GEN_OUTPUT, struct ParametrizedInstruction *GEN_TEST_INSTRUCTIONS, struct*

*ParametrizedInstruction *GEN_BLOCK_INSTRUCTIONS_STACK, struct ParametrizedInstruction *GEN_INSTRUCTIONS_BUF, uint8_t *GEN_BLOCK_ORDER_STACK, struct TemplateBlock *GEN_TEMPLATE_TREE.*

Описание файла «mutator.c»

Файл с описанием мутаций (с сохранением функциональной эквивалентности) созданных последовательностей инструкций должен включать в себя реализацию интерфейса для добавления конкретного количества дополнительных строк в выделенную под это область памяти на основании сгенерированной ранее тестовой последовательности инструкций – функцию с сигнатурой *uint16_t mutate(struct ParametrizedInstruction *, uint16_t, uint16_t, struct ParametrizedInstruction *).*

Важно: файл «mutator.c» должен обладать соответствующим header guard'ом «GEN_MUTATOR_C» и должен иметь директиву #include "mut_decls.h".

Сборка проекта

Сборка настоящего проекта осуществляется посредством использования средств CMake напрямую или скриптов, автоматизирующих работу с CMake - compile.sh как общий скрипт и default_isas/riscv/rv32i/riscv.sh как скрипт специально под ISA RISC-V RV32I (в качестве эталона рассматривался компилятор RISC-V из коллекции GCC).

При работе с общим описанием CMakeLists.txt или скриптом compile.sh потребуются следующие параметры:

1. Относительный или абсолютный путь в файловой системе до файла (кросс-)компилятора языка Си с поддержкой стандарта C99, который будет использоваться в процессе компиляции;
2. Совокупность флагов, применяемых к выбранному ранее компилятору, разделяемая символами точки с запятой;
3. Используемая верифицируемой системой длина машинного слова (32 или 64);
4. Относительный или абсолютный путь в файловой системе до файлов описания ISA, использующейся в процессе генерации и исполнения тестовых цепочек;
5. Относительный или абсолютный путь в файловой системе до файлов описания тестов, необходимых для генерации тестовых последовательностей по заданным шаблонам и для исполнения строго зафиксированных тестов;
6. Относительный или абсолютный путь в файловой системе до файла описания логики функционально эквивалентного преобразования одной последовательности тестовых инструкций в другую;

7. Относительный или абсолютный путь в файловой системе до директории вывода итогового результата статической компиляции – исполняемого файла онлайн-генератора тестовых программ.

Следует уточнить, что расчет относительных путей производится относительно точки вызова compile.sh, средств CMake или скрипта riscv.sh .

Если работа производится со скриптом compile.sh, то требуется лишь запустить его с указанием всех упомянутых выше параметров в порядке их уточнения. Например:

./compile.sh "~/RISCV/compiler/bin/riscv32-unknown-linux-gnu-gcc" "–march=rv32i;-mabi=ilp32" 32 "instructions" "tests" "mutators""bin"

Если же указанная директория параметра №7 существовала до вызова скрипта, она будет очищена от своего содержимого и в ней будут располагаться исполняемый файл «Generator» и директория «otpg_build», являющимися выходными данными программы.

Если же работа производится через средства CMake, то также требуется знать расположение корневой директории самого проекта онлайн-генератора – обозначим его восьмым параметром.

Тогда в первую очередь требуется удалить директорию, в которую планируется разместить готовые файлы онлайн-генератора и его непосредственный двоичный образ, если таковая существует. Затем требуется провести генерацию файлов сборки всего проекта – для этого вызывается команда следующая команда (вместо записи вида ***$i*** подставляется значение *i*-го аргумента из списка выше): *cmake –DCC_PATH:FILEPATH=**$1** –DCC_PARAMS=**$2** –DWSIZE=**$3** –DISA_DIR=**$4** –DTESTS_DIR=**$5** –DMUT_DIR=**$6** –DOUT_DIR=**$7** –S **$8** –B **$7***

Далее, необходимо произвести саму сборку проекта:

cmake –build $7

Если же работа производится со скриптом riscv.sh, то для него требуется указать лишь два параметра:

1. Абсолютный или относительный путь в файловой системе до директории с компилятором riscv32-unknown-linux-gnu-gcc;
2. Относительный или абсолютный путь в файловой системе до директории вывода итогового результата статической компиляции – исполняемого файла онлайн-генератора тестовых программ.

Например, *./riscv.sh "~/RISCV/compiler/bin" "bin"* .

С учетом всех остальных параметров, будет скомпилирован онлайн-генератор под ISA RISC-V RV32I с двоичным интерфейсом приложения ilp32.

Важно: при работе со скриптом riscv.sh подразумевается использование дополнительного флага “–p” (см. раздел 4 «Дополнительные возможности»).

Описание формата выходных данных

На этапе компиляции выходными данными считаются статически-скомпилированный исполняемый файл самого онлайн-генератора тестовых программ, а также набор файлов использовавшихся при сборке.

На этапе исполнения в качестве основных выходных данных предоставляются вердикты об успешности/безуспешности корректности работы использовавшихся в тестах инструкций на основании сравнения результатов выполнения каждой сгенерированной тестовой последовательности инструкций и ее измененной версии, а также должны предоставляться некоторые дополнительные данные, необходимые для воспроизведения тестов с некорректным завершением работы – по адресу для вывода вердиктов (указатель «GEN_OUTPUT») первые два байта памяти используются для записи количества вердиктов неуспешно завершенных тестов, далее для каждого вердикта первый байт, равный 0 для строго заданных тестов или 1 для частично заданных тестов, а затем в зависимости от этого значения считывается от одного до двух байт, где первый всегда отвечает за индекс теста, на котором произошла ошибка, а второй байт, если тест был частично заданным, говорит об итерации, на которой произошла ошибка. Поскольку система генерации псевдослучайных чисел имеет фиксированные опорные значения, то генерируемые тестовые последовательности и входные данные для них остаются неизменными от запуска к запуску генератора, а потому существует возможность повторно выполнить предыдущие тестовые последовательности.

Если при компиляции генератора использовался дополнительный флаг использования стандартного потока вывода, описанный в пункте 4 «Дополнительные возможности», то информация о вердиктах выполнения тестовых последовательностей инструкций будет дополнительно выводиться в упомянутый стандартный поток вывода.

4. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

В дополнение к другим параметрам сборки проекта, описанным в пункте 3.3 «Сборка проекта», стоит отметить возможность использования дополнительного флага “-p” как параметра скрипта compile.sh или аргумента “-DPRINT=1” как параметра при использовании средств CMake (для riscv.sh этот параметр стоит активированным по умолчанию) – данный флаг предоставляет возможности вывода информации об использующейся конфигурации скомпилированного генератора при его запуске, а также вывода информации об успешном или неуспешном завершении конкретных тестов с выводом состояний данных регистров на момент после исполнения тестовых последовательностей. Вся эта информацию выводится в стандартный поток вывода языка Си, а потому использование данного функционала является возможным только при наличии возможности осуществлять системные вызовы на целевой системе.

5. СООБЩЕНИЯ СИСТЕМНОМУ ПРОГРАММИСТУ

Отображение информации компилируемого онлайн-генератора

При генерации файлов сборки и непосредственной компиляции онлайн-генератора выводятся те параметры, которые передавались на сборку.

```
user@debian:~/Downloads/OTPGenerator$ default_isas/riscv/rv32i/riscv.sh ~/RISCV/compiler/bin bin
-- The C compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
---------------------------------------
Chosen C compiler: /home/user/RISCV/compiler/bin/riscv32-unknown-linux-gnu-gcc
Chosen C compiler parameters: -march=rv32i;-mabi=ilp32;-Wl,-nostdlib
Chosen target system machine word size: 32
Chosen ISA directory: /home/user/Downloads/OTPGenerator/default_isas/riscv/rv32i
Chosen tests directory: /home/user/Downloads/OTPGenerator/default_isas/riscv/rv32i/tests
Chosen mutation functionality directory: /home/user/Downloads/OTPGenerator/default_isas/riscv/rv32i/mutators
Chosen binary executable output directory: /home/user/Downloads/OTPGenerator/bin
---------------------------------------
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/Downloads/OTPGenerator/bin/otpg_build
Scanning dependencies of target Generator
[  8%] Building C object CMakeFiles/Generator.dir/core/main.c.o
[ 16%] Building C object CMakeFiles/Generator.dir/core/common/utility.c.o
[ 25%] Building C object CMakeFiles/Generator.dir/core/generation/generator.c.o
[ 33%] Building C object CMakeFiles/Generator.dir/core/execution/runner.c.o
[ 41%] Building C object CMakeFiles/Generator.dir/core/random/rand.c.o
[ 50%] Building C object CMakeFiles/Generator.dir/core/template_block/template_block_type.c.o
[ 58%] Building C object CMakeFiles/Generator.dir/core/memory/memutils.c.o
[ 66%] Building C object CMakeFiles/Generator.dir/core/memory/ins_memutils.c.o
[ 75%] Building C object CMakeFiles/Generator.dir/default_isas/riscv/rv32i/instruction.c.o
[ 83%] Building C object CMakeFiles/Generator.dir/default_isas/riscv/rv32i/tests/tests.c.o
[ 91%] Building C object CMakeFiles/Generator.dir/default_isas/riscv/rv32i/mutators/mutator.c.o
[100%] Linking C executable ../Generator
[100%] Built target Generator
user@debian:~/Downloads/OTPGenerator$
```

Рисунок 2 - отображение выбранных параметров сборки генератора

Отображение сопроводительной информации

При использовании дополнительного флага, упомянутого в разделе 4 «Дополнительные возможности», вся сопроводительная информация выводится в стандартный поток вывода языка Си.

```
user@debian:~/Downloads/OTPGenerator/bin$ spike --isa=RV32GC ~/RISCV/pk/build/pk Generator
bbl loader
Generator ver. 0.0.1 starting...
Configured with ->
Hardware architecture: 32 bit
//////////Running template tests...
Test #0, iter. #0: OK
Test #1, iter. #0: OK
Test #2, iter. #0: OK
Test #3, iter. #0: OK
Test #4, iter. #0: OK
Test #5, iter. #0: OK
Test #6, iter. #0: OK
Test #7, iter. #0: OK
Test #8, iter. #0: OK
Test #9, iter. #0: OK
Test #9, iter. #1: OK
Test #10, iter. #0: Error
Core dumped:
x0(!): 0x00000000(0x00000000)   x1: 0x000111ec(0x00010fd4)      x2: 0x7fffdd00(0x7fffdd00)
x3: 0x00073e84(0x00073e84)      x4: 0x000926e0(0x000926e0)      x5: 0x00011de4(0x00011de4)
x6(!): 0x00000628(0x00000027)   x7(!): 0x00000031(0x00000031)   x8(!): 0x7fffdd7b(0x7fffdd7b)
x9(!): 0x0007c9aa(0x0007c9aa)   x10(!): 0x7fffde21(0x7fffeda5)  x11(!): 0x7fffdd09(0x7fffdd09)
x12(!): 0x00000020(0x00000020)  x13(!): 0x7fffde1d(0x7fffeda1)  x14(!): 0x00000048(0x00000041)
x15(!): 0x7fffdda5(0x7fffed45)  x16(!): 0x00000010(0x00000010)  x17(!): 0x00000028(0x00000028)
x18(!): 0x00000016(0x00000016)  x19(!): 0x7ffffdbc(0x7ffffdbc)  x20(!): 0x000926e0(0x000926e0)
x21(!): 0x0000000f(0x0000000f)  x22(!): 0x0001044a(0x0001044a)  x23(!): 0x00000031(0x00000031)
x24(!): 0x00000029(0x00000029)  x25(!): 0x0000000b(0x0000000b)  x26(!): 0x00000031(0x00000031)
x27(!): 0x0000000d(0x0000000d)  x28(!): 0x0000061c(0x0000061c)  x29(!): 0x64383fbb(0x64383fbb)
x30(!): 0x0000000a(0x0000000a)  x31(!): 0x0000040f(0x0000040f)

//////////Finished running template tests.
//////////Running ASM-tests...
Test #0: OK
Test #1: Error
Core dumped:
x0: 0x00000000(0x00000000)      x1: 0x000117b4(0x0001159c)      x2: 0x7fffdd40(0x7fffdd40)
x3: 0x00073e84(0x00073e84)      x4: 0x000926e0(0x000926e0)      x5: 0x00011de4(0x00011de4)
x6: 0x00000019(0x00000019)      x7: 0x00000000(0x0000001c)      x8: 0x7fffdd70(0x7fffdd70)
x9: 0x00073cd8(0x00073cd8)      x10: 0x7fffdd8c(0x7fffed18)     x11: 0x7fffdd24(0x7fffdd24)
x12: 0x00000004(0x00000004)     x13: 0x7fffdd8b(0x7fffed17)     x14: 0x00000006(0x00000001)
x15: 0x7fffdd78(0x7fffed18)     x16: 0x00000000(0x7efefeff)     x17: 0x00000001(0x00000040)
x18: 0x00000001(0x00000001)     x19: 0x7ffffda4(0x7ffffda4)     x20: 0x000926e0(0x000926e0)
x21: 0x00000001(0x00000001)     x22: 0x00010434(0x00010434)     x23: 0x00000000(0x0000001c)
x24: 0x00000000(0x00000000)     x25: 0x00000000(0x00000000)     x26: 0x00000000(0x00000000)
x27: 0x00000000(0x00000000)     x28: 0xffffffff(0xffffffff)     x29: 0x64383fa3(0x64383fa3)
x30: 0x00000000(0x00000000)     x31: 0x000003e8(0x000003e8)

//////////Finished running ASM-tests.
Failures: 2.
Failure #0: template #10, iter. #0.
Failure #1: ASM-test #1.
Generator stopped.
user@debian:~/Downloads/OTPGenerator/bin$
```

Рисунок 3 - отображение сопроводительной информации

6. СПИСОК ИСПОЛЬЗОВАВШИХСЯ ИСТОЧНИКОВ

1) Andrew Waterman, Krste Asanovic. The RISC-V Instruction Set Manual [Статья в Интернет] Режим доступа: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf , свободный. (дата обращения: 14.11.2022)
2) CMake cross-platform build-process managing system. [Интернет-ресурс] Режим доступа: https://cmake.org/ , свободный. (дата обращения: 14.12.2022)
3) А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.Д. Татарников, М.М. Чупилко, А.А. Сортов. Средства функциональной верификации микропроцессоров [Статья в Интернет] Режим доступа: Сборник трудов ИСП РАН, https://ispranproceedings.elpub.ru/jour/article/view/771, свободный. (дата обращения: 13.10.2022)

7. ПРИЛОЖЕНИЯ

Приложение 1: Словарь использующихся терминов и понятий

Таблица 1 – словарь использующихся терминов и понятий

Термин	**Значение**
Функциональная верификация	Процесс проверки соответствия разрабатываемой RTL-модели микропроцессора документам, описывающих требования к ней.
Динамическое (функциональное) тестирование	Способ проведения верификации, подразумевающий проверку соответствия реализации требованиям путем исполнения тестов на целевой (тестируемой) системе.
Оффлайн-тестирование	Процесс генерации набора тестовых ситуаций без использования системы, подлежащей тестированию.
Онлайн-тестирование	Процесс генерации набора тестовых ситуаций и их непосредственного исполнения на системе, подлежащей тестированию.
Архитектура набора команд (Instruction Set Architecture, ISA)	Совокупность правил и свойств, по которым происходят вычисления на уровне микропроцессора. ISA регламентирует использование регистров микропроцессора, а также существующие типы данных, модель памяти и другие аспекты.
Bare-metal система	Комплекс аппаратных средств без выстроенного программного окружения, ответственного за контроль используемой памяти, планируемых задач и других взаимодействий в рамках системы и необходимых для этого ресурсов – операционной системы.
Граф	Математический объект, описывающий пару из множеств вершин и ребер, где множество ребер задается как некоторое подмножество возможных пар вершин.