На правах рукописи

Татарников Андрей Дмитриевич

АВТОМАТИЗАЦИЯ КОНСТРУИРОВАНИЯ ГЕНЕРАТОРОВ ТЕСТОВЫХ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРОВ НА ОСНОВЕ ФОРМАЛЬНЫХ СПЕЦИФИКАЦИЙ

Специальность 05.13.11 –
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Москва
2017

Работа выполнена в Федеральном государственном бюджетном учреждении науки Институт системного программирования им. В.П. Иванникова Российской академии наук.

Научный руководитель: **Камкин Александр Сергеевич**, кандидат физико-математических наук, ведущий научный сотрудник Федерального государственного бюджетного учреждения науки Институт системного программирования им. В.П. Иванникова Российской академии наук

Официальные оппоненты: **Лацис Алексей Оттович,** доктор физико-математических наук, заведующий сектором Федерального государственного учреждения «Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук»

Чибисов Петр Александрович, кандидат технических наук, заведующий сектором Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук»

Ведущая организация: АО «МЦСТ»

Защита диссертации состоится « 19 » октября 2017 г. в 16 часов на заседании диссертационного совета Д 002.087.01 при Федеральном государственном бюджетном учреждении науки Институт системного программирования им. В.П. Иванникова Российской академии наук по адресу: 109004, Москва, ул. А. Солженицына, д. 25.

С диссертацией можно ознакомиться в библиотеке Федерального государственного бюджетного учреждения науки Институт системного программирования им. В.П. Иванникова Российской академии наук.

Автореферат разослан «____» _____ 2017 г.

Ученый секретарь
диссертационного совета Д 002.087.01, **Зеленов С.В.**
кандидат физико-математических наук

Общая характеристика работы

Актуальность темы

Микропроцессоры лежат в основе большинства электронных устройств. Высокая сложность современных микропроцессоров, вызванная оптимизацией производительности и энергопотребления, может приводить к ошибкам проектирования.

Для обеспечения правильности работы микропроцессоров применяется комплекс мер, одной из важнейших составляющих которого является *функциональная верификация*. Наиболее часто применяемым на практике подходом к функциональной верификации микропроцессоров является *имитационная верификация* (*simulation-based verification*), также называемая *тестированием*. Она осуществляется следующим образом: создаются *тестовые программы* на языке ассемблера; программы запускаются на проектной модели микропроцессора; в результате получаются *трассы исполнения*, содержащие информацию о событиях, которые произошли в процессе исполнения программ; эти трассы сравниваются с эталонными, полученными в результате исполнения этих же программ на эталонном программном эмуляторе; на основе данного сравнения делается вывод о правильности работы микропроцессора.

Создание тестовых программ осуществляется при помощи специальных программных инструментов, известных как *генераторы тестовых программ*. Они используют различные *техники генерации* для обеспечения максимальной полноты тестирования: от случайной генерации до нацеленной генерации, основанной на формальных методах. Ни одна из них не является универсальным решением для всех задач верификации, поэтому на практике применяется множество дополняющих друг друга техник. Общепринятым подходом является генерация на основе *шаблонов*, описывающих структурные и поведенческие свойства тестовых программ. Обработка шаблона состоит в

применении той или иной техники генерации для удовлетворения того или иного свойства.

Как правило, генераторы тестовых программ предназначены для конкретных микропроцессорных архитектур и основаны на конкретных техниках генерации. Однако так как микропроцессорные архитектуры и техники генерации эволюционируют, возникает задача расширения возможностей существующих генераторов. Трудность ее решения заключается в том, что знание об архитектуре микропроцессора зачастую неотделимо от реализации техник генерации. Поддержка новых архитектур и техник генерации требует существенных изменений в реализации генератора. Чтобы этого избежать, инженеры-верификаторы вынуждены одновременно использовать несколько генераторов, каждый из которых решает какую-то отдельную задачу. При этом совместное использование различных техник генерации для решения общей задачи оказывается невозможным.

Таким образом, актуальной является задача разработки метода, позволяющего создавать для любых микропроцессорных архитектур генераторы тестовых программ, интегрирующие в себе разные техники генерации. Перспективным решением данной задачи видится автоматизированное конструирование генераторов тестовых программ на основе формальных спецификаций архитектуры микропроцессора. При этом формальные спецификации будут выступать в качестве источника знания об архитектуре тестируемого микропроцессора, используемого компонентами генератора, реализующими техники генерации. Генерация будет осуществляться на основе шаблонов, разработанных на специальном языке, позволяющем описывать свойства тестовых программ в терминах формальных спецификаций и задавать техники генерации, применяемые для удовлетворения этих свойств.

Цель и задачи работы

Цель работы — разработка метода автоматизации конструирования генераторов тестовых программ для микропроцессоров. Метод должен быть применимым к широкому спектру микропроцессорных архитектур. Генераторы должны создавать тестовые программы на языке ассемблера по шаблонам, описывающим структурные и поведенческие свойства этих программ. Генераторы должны реализовывать разные техники генерации и быть расширяемыми. Для достижения цели работы были поставлены следующие задачи:

1. Провести анализ существующих методов и средств генерации тестовых программ для микропроцессоров.
2. Разработать метод автоматизации конструирования генераторов тестовых программ для микропроцессоров на основе формальных спецификаций.
3. Разработать язык описания шаблонов тестовых программ, позволяющий описывать их структурные и поведенческие свойства.
4. Разработать архитектуру расширяемого генератора тестовых программ для микропроцессоров, позволяющую интегрировать разные техники генерации.
5. Разработать программный инструмент, реализующий предложенный метод автоматизации конструирования генераторов тестовых программ.
6. Оценить характеристики предложенного метода на основе опыта применения разработанного программного инструмента для конструирования генераторов тестовых программ для нескольких микропроцессорных архитектур.

Научная новизна работы

Научной новизной обладают следующие результаты работы:

1. Метод автоматизации конструирования генераторов тестовых программ для микропроцессоров на основе формальных спецификаций.

2. Язык описания шаблонов тестовых программ, позволяющий описывать их структурные и поведенческие свойства.
3. Архитектура генератора тестовых программ для микропроцессоров, позволяющая интегрировать разные техники генерации.

Теоретическая и практическая значимость

В работе предложен метод автоматизации конструирования генераторов тестовых программ для микропроцессоров. В основе предложенного метода лежат архитектурно-независимые техники генерации, для применения которых к конкретной микропроцессорной архитектуре используется информация, полученная в результате анализа формальных спецификаций этой архитектуры. Результаты проведенного исследования могут послужить основой для разработки архитектурно-независимых техник генерации тестовых программ и создания программных инструментов, основанных на анализе формальных спецификаций. Кроме этого, результаты работы могут использоваться в исследовательских проектах и учебных курсах по проектированию и верификации микропроцессоров.

На основе предложенного метода разработан программный инструмент MicroTESK, позволяющий автоматизировать на основе формальных спецификаций конструирование генераторов тестовых программ для микропроцессоров. Разработанный инструмент был применен для создания генераторов тестовых программ для архитектур MIPS64, ARMv8, PowerPC и RISC-V. Генераторы тестовых программ для MIPS64 и ARMv8 используются в отечественных и зарубежных компаниях. Помимо этого, разработанный инструмент может быть использован для создания генераторов тестовых программ для широкого спектра других микропроцессорных архитектур.

Методология и методы исследования

Методологическую основу исследования составляют теория компиляторов, теория формальных языков, теория автоматов, теория множеств, теория графов, теория алгоритмов и математическая логика.

Положения, выносимые на защиту

1. Метод автоматизации конструирования генераторов тестовых программ для микропроцессоров на основе формальных спецификаций.
2. Язык описания шаблонов тестовых программ, позволяющий описывать их структурные и поведенческие свойства.
3. Архитектура генераторов тестовых программ для микропроцессоров, позволяющая интегрировать разные техники генерации и допускающая расширение множества поддерживаемых техник.
4. Программный инструмент, использующий предложенный метод для конструирования генераторов тестовых программ с предложенной архитектурой.

Апробация работы

Основные положения работы обсуждались на следующих конференциях и семинарах:

- Международная конференция «Design Automation Conference», выставка University Booth (г. Остин, США, 2-7 июня 2013 г. и г. Сан-Франциско, США, 2-5 июня 2014 г.);
- Международная конференция «Design, Automation & Test in Europe», выставка University Booth (г. Гренобль, Франция, 18-22 марта 2013 г.; г. Дрезден, Германия, 24-28 марта 2014 г.; г. Гренобль, Франция, 10-12 марта 2015 г.; г. Лозанна, Швейцария, 28-30 марта 2017 г.);
- Международный коллоквиум молодых исследователей в области программной инженерии «Spring/Summer Young Researchers' Colloquium on Software Engineering» (г. Пермь, 30-31 мая 2012 г. и д. Красновидово, 30 мая-1 июня 2016 г.);
- Международная конференция «A.P. Ershov Informatics Conference» (г. Москва, 27-29 июня 2017 г.);
- Открытая конференция ИСП РАН (г. Москва, 1-2 декабря 2016 г.);

- Международный симпозиум «IEEE East-West Design & Test Symposium» (г. Ереван, Армения, 14-17 октября 2016 г.);
- Всероссийская научно-техническая конференция «Проблемы разработки перспективных микро- и наноэлектронных систем» МЭС-2016 (г. Москва, 3-7 октября 2016 г.);
- Международная конференция «Новые информационные технологии в исследовании сложных структур» (г. Екатеринбург, 6-10 июня 2016 г.);
- Международная летняя школа молодых ученых «Новые информационные технологии в исследовании сложных структур» (г. Анапа, 8-12 июня 2015 г.);
- Совместный семинар АО «МЦСТ» и ИСП РАН «Проблемы верификации микропроцессоров» (г. Москва, 10 апреля 2014 г.);
- Научно-техническая конференция студентов, аспирантов и молодых специалистов НИУ ВШЭ им. Е.В. Арменского (г. Москва, 4 февраля 2015 г.);
- Семинар отдела технологий программирования ИСП РАН (г. Москва, 2013, 2014 и 2016 гг.).

Публикации

По теме диссертации автором опубликовано 15 работ, в том числе 7 научных статей [1–7] в рецензируемых журналах, входящих в перечень журналов, рекомендованных ВАК РФ. В работе [2] автором описывается архитектура инструмента MicroTESK. В статье [5] вклад автора заключается в описании средств системной верификации микропроцессоров. В работах [6, 14] автору принадлежит описание концепции и архитектуры расширяемой среды генерации тестовых программ. В работах [7, 10, 11] вклад автора состоит в разработке средств моделирования подсистемы памяти и конструкций языка описания шаблонов тестовых программ, позволяющих создавать тесты на подсистему памяти. В работе [13] автором дается обзор существующих подходов и формулируются требования для системы хранения информации,

используемой для построения тестов. В статье [15] автором описываются предлагаемый подход к моделированию архитектуры микропроцессора и концепция языка описания шаблонов тестовых программ.

Личный вклад

Все представленные в диссертации результаты получены лично автором.

Структура и объем диссертации

Работа состоит из введения, четырех глав, заключения, списка литературы (122 наименования) и одного приложения. Основной текст диссертации (без списка литературы и приложения) занимает 148 страниц.

Краткое содержание диссертации

Во **введении** обосновывается актуальность темы работы, определяются ее цели и задачи, раскрывается теоретическая и практическая значимость.

В **первой главе** дается обзор процесса проектирования микропроцессоров и методов функциональной верификации, применяемых параллельно данному процессу. После этого рассматриваются существующие техники и инструменты генерации тестовых программ. При этом основное внимание уделяется следующим свойствам инструментов генерации:

- *Реконфигурируемость* – возможность адаптации к тестированию новых микропроцессорных архитектур.
- *Расширяемость* – возможность интеграции компонентов, реализующих новые техники генерации.
- *Контролируемость генерации* – возможность отслеживать состояние микропроцессора на каждом шаге генерации путем исполнения команд на программном эмуляторе, что позволяет гарантировать корректность построенных программ, создавать тесты со встроенными проверками и применять техники генерации, использующие информацию о текущем состоянии эмулятора.

- *Целенаправленность тестирования* – способность генерировать тесты, нацеленные на покрытие конкретных ситуаций в работе микропроцессора или классов ситуаций.

Таблица 1 сравнивает возможности существующих генераторов тестовых программ по перечисленным критериям.

Таблица 1. Сравнение возможностей существующих генераторов тестовых программ

Инструмент	**Реконфигурируемость**	**Расширяемость**	**Контролируемость генерации**	**Целенаправленность тестирования**
MicroTESK 1.0	Да	Нет	Да	Высокая
INTEG	Нет	Нет	Нет	Средняя
RIS	Нет	Нет	Нет	Средняя
RAVEN	Да	Нет	Да	Средняя
Genesys-Pro, FPGen, DeepTrans	Да	Частично	Да	Высокая
RDG	Да	Нет	Нет	Низкая
MA^2TG	Да	Нет	Нет	Высокая
UFL/UCI	Да	Нет	Нет	Высокая
μGP	Да	Нет	Нет	Средняя

На основе сравнительного анализа существующих решений делается вывод об отсутствии генератора тестовых программ, сочетающего все перечисленные выше свойства, и выдвигается тезис об актуальности задачи его создания.

Во **второй главе** предлагается автоматически конструировать генераторы тестовых программ для микропроцессоров на основе формальных спецификаций их архитектуры. При этом информация, извлеченная из формальных спецификаций, будет использоваться для обеспечения целенаправленности тестирования и отслеживания состояния микропроцессора в процессе генерации. Построение тестовых программ будет осуществляться на основе шаблонов, разработанных на специальном расширяемом языке, позволяющем описывать их структурные и поведенческие свойства. Сконструированные генераторы тестовых программ будут включать в себя

компоненты, реализующие техники генерации, обеспечивающие удовлетворение заданных структурных (генераторы последовательностей команд) и поведенческих (генераторы входных данных для команд) свойств. Расширяемая архитектура генераторов позволит интегрировать компоненты, реализующие разные техники генерации. На рисунке 1 показана архитектура инструмента конструирования генераторов тестовых программ, реализующего предлагаемый метод, и генераторов, сконструированных с помощью этого инструмента.

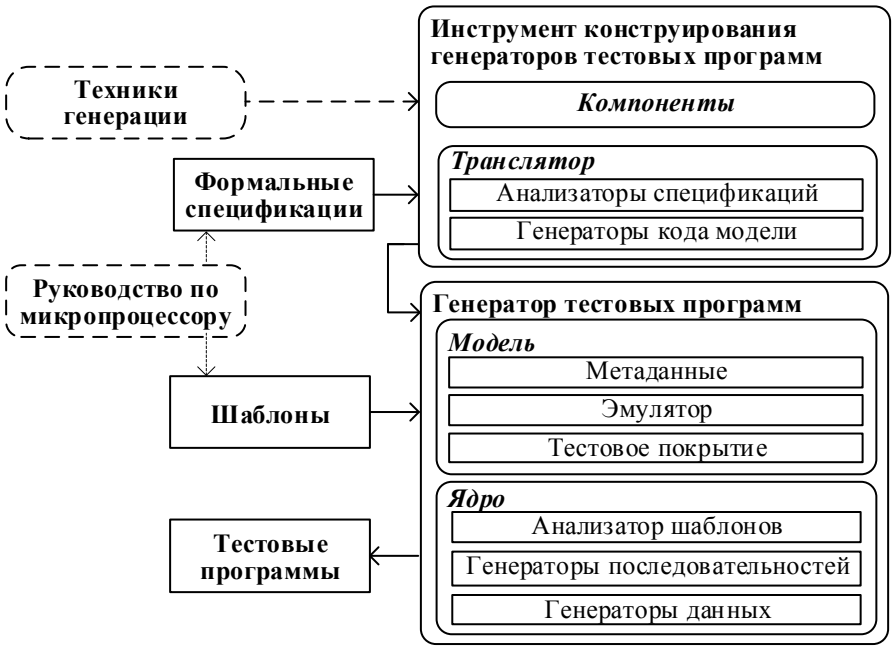

Рисунок 1. Инструмент, реализующий предлагаемый метод, и результат его работы

В **разделе 2.1** описывается метод автоматизации конструирования генераторов тестовых программ. Предлагается конструировать генераторы тестовых программ для микропроцессоров на основе формальных спецификаций их архитектуры. Спецификации архитектуры взяты за основу, так как при использовании большинства техник генерации требуется информация о синтаксисе и семантике команд. При необходимости спецификации архитектуры могут быть дополнены спецификациями, описывающими особенности микроархитектуры.

Для создания формальных спецификаций архитектуры микропроцессора предлагается использовать язык nML. Спецификации на этом языке включают в себя описания следующих сущностей: типы данных, константы, регистры,

режимы адресации, команды и память. Семантика команд задается в виде присваиваний значений переменным, описывающим регистры и память. На рисунке 2 показан пример спецификации команды ADD микропроцессора MIPS64.

```
01:   op add (rd: R, rs: R, rt: R)
02:     syntax = format("add %s, %s, %s", rd.syntax, rs.syntax, rt.syntax)
03:     image  = format("000000%5s%5s%5s00000100000", rs.image, rt.image, rd.image)
04:     action  = {
05:       if sign_extend(WORD, rs<31>) != rs<63..32> || sign_extend(WORD, rt<31>) != rt<63..32> then
06:         unpredicted;
07:       endif;
08:       temp33 = rs<31>::rs<31..0> + rt<31>::rt<31..0>;
09:       if temp33<32> != temp33<31> then
10:         exception("IntegerOverflow");
11:       else
12:         rd = sign_extend(DWORD, temp33<31..0>);
13:       endif;
14:     }
```

Рисунок 2. Спецификация команды ADD архитектуры MIPS64 на nML

Данный пример разработан на основе взятого из руководства по архитектуре MIPS64 описания команды ADD, которое приведено на рисунке 3.

```
01:   if NotWordValue(GPR[rs]) or NotWordValue(GPR[rt]) then
02:     UNPREDICTABLE
03:   endif
04:   temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0)
05:   if temp32 ≠ temp31 then
06:     SignalException(IntegerOverflow)
07:   else
08:     GPR[rd] ← sign_extend(temp31..0)
09:   endif
```

Рисунок 3. Описание семантики команды ADD в руководстве по архитектуре MIPS64

Язык nML не позволяет описывать логику управления памятью. При обращении к массиву памяти не осуществляется кэширование данных и трансляция адресов, а виртуальный адрес считается равным физическому. Для поддержки генерации тестов для *подсистемы управления памятью (MMU, memory management unit)* предлагается использовать дополнительные спецификации на языке MMUSL, описывающие механизмы трансляции адресов и кэширования данных.

Далее предлагается конструировать генераторы тестовых программ, состоящие из двух частей: (1) *модели*, содержащей информацию об архитектуре тестируемого микропроцессора, и (2) *ядра*, интегрирующего в себе архитектурно-независимые компоненты, реализующие различные техники

генерации. При этом ядро является общим для всех конструируемых генераторов. Таким образом, конструирование генератора сводится к построению модели микропроцессора.

Конструируемые генераторы строят тестовые программы на основе шаблонов, задающих их свойства. Для создания шаблонов используется специальный язык, позволяющий описывать эти программы в терминах команд, предоставляемых моделью, и техник генерации, реализуемых ядром. Обработка тестовых шаблонов заключается в применении компонентов ядра, реализующих техники генерации, к сущностям, определяемым моделью, для построения тестовых программ, обладающих заданными свойствами.

В **разделе 2.2** описывается предлагаемый язык описания шаблонов тестовых программ. Данный язык позволяет описывать свойства тестовых программ для любых микропроцессорных архитектур и применять расширяемый набор техник генерации для достижения этих свойств.

Генерируемые *тестовые программы* имеют следующую структуру. Они состоят из *тестовых воздействий*, то есть последовательностей команд, исполнение которых приводит к возникновению некоторых событий в работе микропроцессора, называемых *тестовыми ситуациями*. Тестовые воздействия предваряются *инициализирующим кодом*, который подготавливает необходимое начальное состояние микропроцессора. После тестовых воздействий в программы могут добавляться *встроенные проверки (self-checks)*, основанные на сравнении текущих значений регистров и памяти с эталонными. Тестовое воздействие вместе с инициализирующим кодом и встроенными проверками называют *тестовым примером (test case)*. В простейшем случае тестовые примеры не зависят друг от друга и исполняются последовательно. Чтобы обеспечить их исполнение в определенном порядке или параллельное исполнение, в тестовые программы добавляется *код диспетчеризации (dispatching code)*. Кроме этого каждая тестовая программа включает в себя *пролог*, который осуществляет подготовку микропроцессора к работе, и *эпилог*, который завершает работу программы. Таким образом, структуру тестовых

программ можно описать формулой $П = П_{start} \cdot \{\langle \pi_{dispatch} \rangle \cdot \langle \pi_{start}, x_{i,} \pi_{stop} \rangle\}_{i=1,n} \cdot П_{stop}$, где:

- $П_{start}$ – пролог тестовой программы;
- $\pi_{dispatch}$ – код диспетчеризации;
- $\langle \pi_{start}, x_{i,} \pi_{stop} \rangle$ – тестовый пример, включающий в себя:
  - π_{start} – инициализирующий код;
  - x_i – тестовое воздействие;
  - π_{stop} – встроенные проверки;
- $П_{stop}$ – эпилог тестовой программы;
- n – число тестовых примеров в программе.

На основе одного шаблона можно построить множество тестовых программ, удовлетворяющих заданным свойствам. Это позволяет обеспечить покрытие всевозможных комбинаций тестовых ситуаций. Шаблоны описывают пролог, эпилог и код диспетчеризации в виде фиксированных последовательностей команд, так как выполняемые ими действия могут быть заранее определены. Свойства тестовых примеров описываются следующим образом: задается способ построения последовательностей команд (*структурные свойства*) и тестовые ситуации, которые должны возникнуть в процессе их исполнения (*поведенческие свойства*). Свойства тестовых примеров, описываемые при помощи шаблонов, можно разделить на следующие типы:

- способы построения последовательностей команд;
- способы выбора регистров, используемых командами;
- способы комбинирования тестовых ситуаций;
- способы генерации данных для тестовых ситуаций;
- способы построения инициализирующего кода;
- способы построения встроенных проверок.

Для удовлетворения данных свойств применяются разнообразные *техники генерации*, реализуемые генератором.

Шаблоны тестовых программ предлагается разрабатывать на специальном предметно-ориентированном языке, основанном на языке Ruby. Использование в качестве основы существующего языка высокого уровня имеет следующие преимущества: (1) все предоставляемые им возможности можно использовать при разработке шаблонов тестовых программ и (2) изучение языка описания шаблонов становится проще. Ruby был выбран из-за наличия средств метапрограммирования, при помощи которых в язык добавляются конструкции для описания команд тестируемого микропроцессора и директив ассемблера.

Возможности языка описания шаблонов тестовых программ сводятся к следующему. Шаблон тестовой программы представляет собой Ruby-класс, который реализует методы, внутри которых описываются пролог, эпилог, тестовые примеры и код диспетчеризации. Последовательности команд, составляющие тестовые примеры, описываются в виде *блоков*, которые можно определять рекурсивно. Отдельный блок представляет собой или *элементарный блок* или упорядоченный набор *вложенных блоков*. Атрибуты блоков задают техники генерации, используемые для построения тестовых примеров на основе вложенных элементов. Регистры, используемые командами тестовых примеров, могут задаваться жестко или выбираться по определенным правилам. С командами ассоциируются тестовые ситуации, для которых должны быть сгенерированные тестовые данные. Также в шаблонах определяются правила построения инициализирующего кода и кода встроенных проверок. Выбор конкретных техник генерации определяется значениями атрибутов соответствующих конструкций. Это позволяет наращивать множество поддерживаемых техник без внесения изменений в язык.

На рисунке 4 показано шаблонное описание тестовых примеров, заданных как декартово произведение команд ADD и SUB микропроцессора MIPS, исполнение которых вызывает ситуации Normal и Overflow, и один из вариантов построенного на его основе кода. Используемые регистры

выбираются случайным образом и инициализируются в соответствии с правилом, заданным конструкцией preparator.

```
01:   class MyTemplate < Template
02:     def run
03:       preparator(:target => 'R') {
04:         lui target, value(16, 31)
05:         ori target, target, value(0, 15)
06:       }
07:       block(:combinator => 'product') {
08:         iterate {
09:           add r(_), r(_), r(_) do situation('Normal') end
10:           add r(_), r(_), r(_) do situation('Overflow') end
11:         }
12:         iterate {
13:           sub r(_), r(_), r(_) do situation('Normal') end
14:           sub r(_), r(_), r(_) do situation('Overflow') end
15:       }
16:       }
17:     end
18:   end
```

Инициализирующий код:

```
lui r14, 0xa9c9
ori r14, r14, 0x7025
lui r10, 0xe6f6
ori r10, r10, 0x78de
lui r12, 0x5db7
ori r12, r12, 0xfa2c
lui r13, 0xfafc
ori r13, r13, 0x3700
```

Тестовое воздействие:

```
add r8, r14, r10
sub r15, r12, r13
```

Рисунок 4. Шаблонное описание и один из тестовых примеров, построенных на его основе

В **разделе 2.3** предлагается расширяемая архитектура конструируемых генераторов тестовых программ, которая позволяет интегрировать разные техники генерации.

Генераторы состоят из *модели*, содержащей информацию об архитектуре тестируемого микропроцессора, и *ядра*, реализующего архитектурно-независимые техники генерации. Модель включает в себя три основных части: (1) *метаданные*, хранящие каталог команд, регистров и элементов памяти микропроцессора; (2) *эмулятор*, позволяющий исполнять команды и получать информацию о результатах их исполнения; (3) *модель тестового покрытия*, содержащая информацию об условиях возникновения тестовых ситуаций, связанных с командами. Взаимодействие с моделью осуществляется через архитектурно-независимые интерфейсы, при этом метаданные используются для обращения к элементам архитектуры. Ядро осуществляет генерацию тестовых программ на основе шаблонов, взаимодействуя с моделью. Процесс генерации включает следующие стадии:

1. Анализ шаблона, в процессе которого определяется структура тестовой программы и строятся блоки, описывающие свойства ее отдельных

частей (пролога, эпилога, тестовых примеров и секций диспетчеризации), которые затем будут обрабатываться по отдельности.

2. Построение и размещение в памяти эмулятора кода пролога и секций диспетчеризации, имеющих фиксированные адреса, а также всех глобальных данных.
3. Исполнение построенного кода на эмуляторе с целью подготовки начального состояния, необходимого для генерации тестовых примеров. Исполнение приостанавливается при достижении конца одной из построенных секций кода.
4. Построение кода тестового примера, который следует за секцией кода, при достижении конца которой исполнение было приостановлено. В рамках данной стадии выполняются следующие действия:
    a. Построение последовательностей команд.
    b. Выбор регистров, используемых командами.
    c. Построение комбинаций вариантов решений тестовых ситуаций, связанных с командами.
    d. Генерация данных для тестовых ситуаций и построение инициализирующего кода.
5. Размещение в памяти эмулятора и исполнение кода построенного тестового примера.
6. Построение кода встроенных проверок для тестового примера на основе информации о текущем состоянии регистров и памяти эмулятора. При этом построенный код размещается в памяти эмулятора и исполняется.
7. Построение, размещение в памяти эмулятора и исполнение кода секции диспетчеризации, следующей за построенным тестовым примером. Код этой секции не был построен ранее, так как его адрес не был зафиксирован. После этого выполняется переход к стадии 4. Если за построенным тестовым примером следует эпилог, выполняется переход к стадии 8.
8. Построение, размещение в памяти эмулятора и исполнение кода эпилога.

9. Печать построенного кода в виде программы на языке ассемблера.

Такой подход к генерации тестовых программ дает следующие преимущества: (1) проверяется корректность построенных программ; (2) дается возможность создавать тесты со встроенными проверками; (3) при построении тестовых примеров учитывается текущее состояние эмулятора.

Ядро включает в себя компоненты, обеспечивающие построение тестовых примеров при помощи различных техник генерации. Эти техники ориентированы на удовлетворение структурных и поведенческих свойств, заданных шаблонами. Удовлетворение структурных свойств сводится к построению последовательностей команд заданного вида. Удовлетворение поведенческих свойств заключается в генерации для этих команд входных данных, которые приводят к возникновению заданных тестовых ситуаций в процессе их исполнения. Поддержка новых техник генерации осуществляется путем добавления соответствующих компонентов.

Основная идея алгоритма построения тестовых примеров заключается в следующем. При помощи различных случайных и комбинаторных техник, реализуемых ядром, строятся последовательности команд, описывающие тестовые сценарии. После этого осуществляется выбор используемых регистров, за который отвечают реализованные ядром техники выбора регистров. После этого осуществляется генерация входных данных для команд. С командами связаны тестовые ситуации, условия возникновения которых описаны моделью микропроцессора. Эти условия могут иметь разную природу и для их удовлетворения могут использоваться разные техники. Поэтому тестовые ситуации обрабатываются по отдельности. Их обработка происходит в порядке исполнения команд, с которыми они связаны. Способы обработки тестовых ситуаций определяется их типом: для каждого типа тестовой ситуации ядро реализует соответствующую технику.

В **третьей главе** описывается реализация предложенного метода автоматизации конструирования генераторов тестовых программ. Метод нашел свое воплощение в инструменте с открытым исходным кодом

MicroTESK (Microprocessor TEsting and Specification Kit) версии 2.0, разработанном на языке Java. Данный инструмент на основе формальных спецификаций на языках nML и MMUSL конструирует генераторы тестовых программ, состоящие из модели микропроцессора и архитектурно-независимого ядра.

В **разделе 3.1** описывается реализации инструмента MicroTESK. Данный инструмент осуществляет построение модели микропроцессора на основе формальных спецификаций и специальных библиотек. Обработка формальных спецификаций осуществляется при помощи транслятора, который поддерживает языки nML и MMUSL, а также позволяет добавлять поддержку других языков. Транслятор строит модель путем генерации кода на языке Java. Построенная модель включает в себя *метаданные*, *эмулятор* и *модель тестового покрытия*. Метаданные описывают свойства команд, поддерживаемых тестируемым микропроцессором. Эмулятор отвечает за исполнение команд, описанных при помощи метаданных. Тестовые ситуации, составляющие модель покрытия, описываются в виде ограничений, которые автоматически извлекаются из формальных спецификаций. Транслятор можно расширять дополнительными средствами анализа, позволяющими извлекать новую информацию о свойствах команд микропроцессора.

В **разделе 3.2** описывается реализация архитектурно-независимого ядра генераторов тестовых программ. Ядро решает задачи построения внутреннего представления шаблона и генерации тестовых программ на его основе.

Для анализа шаблонов и построения их внутреннего представления используется интерпретатор JRuby. С его помощью исполняется код на языке Ruby, взаимодействующий с ядром генератора для конструирования сущностей внутреннего представления. При этом языковые конструкции, позволяющие описывать архитектурно-зависимые сущности, создаются динамически при помощи средств метапрограммирования языка Ruby. Языковые конструкции, связанные с архитектурно-независимыми техниками генерации, реализуются в виде библиотек. Для поддержки новых конструкций требуется разработать

дополнительные Ruby библиотеки и компоненты ядра, отвечающие за конструирование соответствующих сущностей внутреннего представления.

Построенное внутреннее представление обрабатывается компонентами ядра, осуществляющими генерацию последовательностей команд и их входных данных. Для каждого типа ограничений, описываемых моделью покрытия, предоставляется компонент, отвечающий за их разрешение. Расширение возможностей генератора достигается путем добавления новых компонентов.

В **четвертой главе** описываются результаты применения предложенного метода автоматизации конструирования генераторов тестовых программ для микропроцессоров MIPS64, ARMv8, PowerPC и RISC-V. Для данных архитектур были разработаны формальные спецификации системы команд (на языке nML) и подсистемы управления памятью (на языке MMUSL), на основе которых были сконструированы генераторы тестовых программ. Статистика по разработанным спецификациям приводится в таблице 2.

Таблица 2. Статистика по разработанным формальным спецификациям

Архитектура	**MIPS64**	**ARMv8**	**PowerPC**	**RISC-V**
Число заспецифицированных команд	235	1015	34	62
Размер nML спецификаций в строках кода	3999	18178	935	816
Размер MMUSL спецификаций в строках кода	267	2643	0	0
Общие трудозаты в человеко-месяцах	4	30	1	0.75
Трудоемкость описания одной команды в человеко-днях	0.37	0.65	0.64	0.27

Практический опыт показал, что трудоемкость разработки формальных спецификаций находится в линейной зависимости от количества команд, поддерживаемых микропроцессором. Трудоемкость описания одной команды зависит от сложности системы команд и в среднем составляет 0.4 человеко-дня. Для сравнения: при использовании инструмента MicroTESK 1.0, который предполагает разработку спецификаций на языке Java, трудоемкость описания одной команды архитектуры MIPS64 составляет около 0.8 человеко-дня. Таким образом, предложенный метод позволяет сократить трудозатраты на создание генератора тестовых программ примерно в 2 раза. Сравнение с другими инструментами генерации не приводится, так как данные о трудоемкости их конфигурирования отсутствуют в открытых источниках.

Сконструированные генераторы тестовых программ отвечают основным требованиям, предъявляемым к промышленным инструментам такого рода:

- поддержка различных техник построения тестовых программ (случайные, комбинаторные, на основе ограничений);
- возможность построения тестов со встроенными проверками;
- учет архитектурных и микроархитектурных особенностей (условий возникновения исключений, числа вычислительных ядер и потоков, параметров кэш-памяти и т.п.).

Генераторы тестовых программ для MIPS64 и ARMv8 применяются в отечественных и зарубежных компаниях.

В **заключении** перечисляются основные результаты работы.

Основные результаты работы

Основные научные и практические результаты, полученные в диссертационной работе, состоят в следующем:

1. Разработан метод автоматизации конструирования генераторов тестовых программ для микропроцессоров на основе формальных спецификаций.
2. Разработан язык описания шаблонов тестовых программ, позволяющий описывать их структурные и поведенческие свойства.
3. Разработана архитектура генераторов тестовых программ для микропроцессоров, позволяющая интегрировать разные техники генерации и допускающая расширение множества поддерживаемых техник.
4. Разработан программный инструмент, использующий предложенный метод для конструирования генераторов тестовых программ с предложенной архитектурой.

Разработанный программный инструмент применяется в промышленных проектах по верификации микропроцессоров.

Работы автора по теме диссертации

Работы, опубликованные в изданиях, рекомендуемых ВАК:

1. Татарников А.Д. Обзор методов и средств генерации тестовых программ для микропроцессоров / Татарников А.Д. // Труды ИСП РАН. - т. 29. - в. 1. - 2017. - С. 167-194.
2. Татарников А.Д. Генератор тестовых программ для архитектуры ARMv8 на основе инструмента MicroTESK / Камкин А.С., Коцыняк А.М., Проценко А.С., Татарников А.Д., Чупилко М.М. // Труды ИСП РАН. - т. 28. - в. 6. - 2016. - С. 87-102.
3. Татарников А.Д. Комбинаторная генерация тестовых программ для микропроцессоров на основе формальных спецификаций системы команд / Татарников А.Д. // Сборник трудов конференция «Проблемы разработки перспективных микро- и наноэлектронных систем». - 2016. - Часть II. - С. 38-45.
4. Tatarnikov A. Language for Describing Templates for Test Program Generation for Microprocessors / Tatarnikov A. // Proceedings of the Institute for System Programming. - Volume 28. - Issue 4. - 2016. - P. 77-98.
5. Татарников А.Д. Средства функциональной верификации микропроцессоров / Камкин А.С., Коцыняк А.М., Смолов С.А., Сортов А.А., Татарников А.Д., Чупилко М.М. // Труды ИСП РАН. - т. 26. - в. 1. - 2014. - С. 149-200.
6. Татарников А.Д. Расширяемая среда генерации тестовых программ для микропроцессоров / Камкин А.С., Сергеева Т.И., Смолов С.А., Татарников А.Д., Чупилко М.М. // Программирование. - №1. - 2014. - С. 3-14.
7. Tatarnikov A. An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms/ Kamkin A., Protsenko A., Tatarnikov A. // Proceedings of the Institute for System Programming. - Volume 27. - Issue 3. - 2015. - P. 125-138.

Работы, опубликованные в других изданиях:

8. Татарников А.Д. Построение поведенческих моделей микропроцессоров для генерации тестовых программ / Татарников А.Д. // Известия высших учебных заведений. Физика. - Том 59. - No 8/2. - 2016. - С. 97-100.
9. Tatarnikov A. An Approach to Instruction Stream Generation for Functional Verification of Microprocessor Designs / Tatarnikov A. // Proceedings of 14th IEEE East-West Design & Test Symposium (EWDTS'2016). - 2016. - P. 270-273.
10. Tatarnikov A. Specification-Based Test Program Generation for ARM VMSAv8-64 Memory Management Units / Chupilko M., Kamkin A., Kotsynyak A., Protsenko A., Smolov S., Tatarnikov A. // Proceedings of 16th International Workshop on Microprocessor and SOC Test and Verification (MTV 2015). - 2015. - P. 1-7.
11. Татарников А.Д. Генерация тестовых программ для микропроцессоров на основе спецификаций подсистем памяти / Камкин А.С., Проценко А.С., Татарников А.Д. // Известия высших учебных заведений. Физика. - Том 58. - No 11/2. - 2015. - С. 70-74.
12. Татарников А.Д. Инструмент автоматизации разработки генераторов тестовых программ для микропроцессоров на основе формальных спецификаций / Татарников А.Д. // Материалы научно-технической конференции студентов, аспирантов и молодых специалистов НИУ ВШЭ им. Е.В. Арменского. - 2015. - С. 53.
13. Tatarnikov A. Generic Knowledgebase for Test Generation / Kotsynyak A., Tatarnikov A. // Proceedings of SYRCoSE 2014: 8th Spring/Summer Young Researchers' Colloquium on Software Engineering. - 2014. - P. 114-117
14. Tatarnikov A. MicroTESK: An Extensible Framework for Test Program Generation / Kamkin A., Sergeeva T., Tatarnikov A., Utekhin A. // Proceedings of SYRCoSE 2013: 7th Spring/Summer Young Researchers' Colloquium on Software Engineering. - 2013. - P. 51-57.
15. Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors / Kamkin A., Tatarnikov A. // Proceedings of

SYRCoSE 2012: 6th Spring/Summer Young Researchers' Colloquium on Software Engineering. - 2012. - P. 64-69.