УДК 004.23 DOI 10.31114/2078-7707-2018-2-2-8

Генератор тестовых программ для архитектуры RISC-V на основе инструмента MicroTESK

А.С. Камкин[1, 2, 3, 4], А.С. Проценко[1], С.А. Смолов[1], А.Д. Татарников[1, 4]

[1]Институт системного программирования им. В.П. Иванникова РАН, г. Москва

[2]Московский государственный университет имени М.В. Ломоносова, г. Москва

[3]Московский физико-технический институт, г. Москва

[4]Национальный исследовательский университет «Высшая школа экономики», г. Москва

{kamkin, protsenko, smolov, andrewt}@ispras.ru

***Аннотация* — В работе рассматривается генератор тестовых программ, предназначенный для верификации микропроцессоров с архитектурой RISC-V. Генератор разработан на основе инструмента MicroTESK и состоит из формальных спецификаций архитектуры RISC-V и архитектурно независимого ядра. Спецификации задают синтаксис и семантику команд. Ядро реализует техники построения последовательностей команд и генерации данных. Генерация осуществляется на основе шаблонов, описывающих структурные и поведенческие свойства программ. Инструмент позволяет расширять систему команд и поддерживает случайные, комбинаторные и основанные на ограничениях техники генерации.**

***Ключевые слова* — микропроцессор; система команд; формальная спецификация; верификация; генерация тестовых программ; RISC-V; nML; MicroTESK.**

I. Введение

RISC-V – это открытая архитектура, основанная на концепции RISC [1]. Архитектура разработана в Калифорнийском университете в Беркли в 2010 г. В настоящее время основным институтом ее развития и стандартизации является RISC-V Foundation [2]. Ключевыми особенностями архитектуры RISC-V являются открытость и расширяемость. Над созданием микропроцессоров с архитектурой RISC-V работают такие компании, как NVIDIA [3] и Samsung [4]; совместные исследования в этой области ведутся в Швейцарской высшей технической школе Цюриха и в Болонском университете [5].

При проектировании микропроцессоров неизбежны ошибки, поэтому верификация является приоритетной задачей. Одним из основных подходов к верификации является исполнение *тестовых программ* на RTL-модели микропроцессора и сравнение результатов исполнения с данными, полученными на эталонном симуляторе [6]. Тестовые программы создаются с помощью *генераторов тестовых программ*. Важным требованием к генераторам для микропроцессоров RISC-V является возможность адаптации к различным версиям архитектуры, включающим разные наборы расширений. На наш взгляд, наиболее подходящее решение основано на использовании *формальных спецификаций* [7]. При таком подходе описание команд отделено от логики генерации тестов и может быть модифицировано без изменения ядра генератора.

В работе рассматривается генератор MicroTESK for RISC-V [8], разработанный с помощью инструмента MicroTESK (Microprocessor TEsting and Specification Kit) [9-10]. Генератор состоит из двух частей: архитектурно независимого ядра (библиотечных компонентов) и формальных спецификаций команд RISC-V на языке nML [11]. Такое устройство генератора снижает затраты на разработку и упрощает адаптацию к изменениям в архитектуре.

Оставшаяся часть статьи организована следующим образом. В разделе II рассматриваются существующие решения в области генерации тестовых программ для микропроцессоров. Раздел III посвящен генератору MicroTESK for RISC-V; здесь описываются базовые принципы используемого подхода и особенности его применения к архитектуре RISC-V. В разделе IV приводятся данные о генераторе: поддерживаемые команды, объем спецификаций, трудоемкость разработки. Раздел V резюмирует работу и описывает направления дальнейших исследований.

II. Обзор существующих решений

Сотрудниками Калифорнийского университета в Беркли, участвующими в разработке RISC-V, был создан набор тестовых программ для проверки правильности реализации системы команд [12]. Эти программы тестируют каждую команду в отдельности и не покрывают ситуации, связанные с особенностями микроархитектуры. Для проверки разных вариантов совместного исполнения команд на конвейере был разработан генератор RISC-V Torture Test Generator (далее – Torture) [13]. Он строит тестовые программы случайным образом, комбинируя описанные вручную последовательности команд небольшой длины и выбирая используемые ими регистры.

Существуют универсальные генераторы тестовых программ, применимые к разным архитектурам. Наиболее известным из них является Genesys-Pro [7] от IBM Research. Генератор использует следующие входные данные: *модель микропроцессора* и *шаблоны*, описывающие последовательности команд и задающие для них распределения вероятностей и ограничения. При генерации инструмент предсказывает состояние регистров и памяти путем исполнения построенного кода на внешнем симуляторе. Что используется для создания *самопроверяющих тестовых программ*.

Другой генератор, RAVEN (Random Architecture Verification Machine) [14], разработан в компании Obsidian Software, поглощенной ARM. Инструмент основан на том же принципе, что и Genesys-Pro – логика генерации отделена от модели, описывающей архитектуру. RAVEN генерирует программы на основе шаблонов и может создавать случайные и нацеленные тесты. Как и Genesys-Pro, RAVEN использует внешний симулятор для определения корректного состояния микропроцессора.

Рассмотренные генераторы тестовых программ имеют следующие недостатки. Инструмент Torture ориентирован на случайную генерацию и не умеет строить тесты по пользовательским сценариям; кроме того, для расширения набора поддерживаемых команд нужно модифицировать код генератора. Инструменты RAVEN и Genesys-Pro лишены указанных недостатков, но не поддерживают RISC-V. Следует отметить, что создание и поддержка модели архитектуры является непростой задачей; помимо этого, при расширении системы команд требуется менять внешний симулятор.

Подход, реализованный в инструменте MicroTESK, нацелен на упрощение разработки и сопровождения генераторов тестовых программ за счет извлечения всей необходимой информации из единого источника – формальных спецификаций системы команд.

III. Генератор тестовых программ MicroTESK for RISC-V

A. *Инструмент MicroTESK*

Генератор тестов MicroTESK for RISC-V создан с помощью инструмента MicroTESK [9-10]. Инструмент включает две основные части: *среду моделирования* и *среду генерации*. Первая отвечает за построение модели архитектуры микропроцессора на основе формальных спецификаций; вторая – за построение тестовых программ на основе модели архитектуры и тестовых шаблонов. Архитектура MicroTESK показана на рис. 1.

Среда моделирования использует в качестве входных данных формальные спецификации системы команд микропроцессора на языке nML [11]. Она реализует анализаторы и конструкторы, применяемые для построения модели микропроцессора. Построенная модель состоит из следующих частей:

1) *метаданные* — описывают поддерживаемые регистры и команды;

2) *симулятор* — исполняет команды и предоставляет информацию о состоянии регистров и памяти;

3) *модель тестового покрытия* — описывает возможные пути исполнения команд.

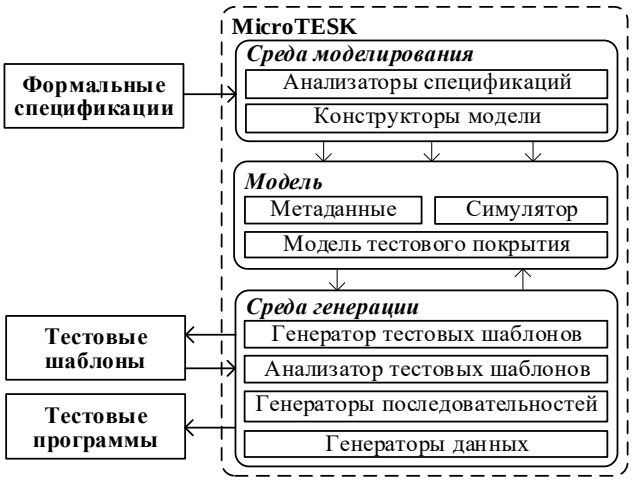

Рис. 1. Архитектура инструмента MicroTESK

Среда генерации получает на вход *тестовые шаблоны* на языке, основанном на Ruby [15]. Язык позволяет описывать сценарии тестирования на абстрактном уровне, где состав команд, их порядок и операнды не фиксируются, а выбираются динамически в зависимости от параметров генерации. В шаблонах можно обращаться к регистрам и вызывать команды, используя синтаксис близкий к языку ассемблера. Все конструкции языка, необходимые для работы с той или иной архитектурой, создаются динамически на основе метаданных.

Тестовые шаблоны разрабатываются инженерами-верификаторами, исходя из своих целей. Создание некоторых типов тестов автоматизируется средствами MicroTESK:

1) тесты на отдельные команды, использующие случайные значения операндов;

2) тесты на отдельные команды, использующие граничные значения операндов;

3) тесты на отдельные команды, покрывающие все пути исполнения, описанные в спецификациях;

4) тесты на короткие цепочки команд, полученные перебором команд разного типа.

Процесс генерации тестовых программ по шаблону состоит из следующих стадий:

1) анализ тестового шаблона и построение его внутреннего представления;

2) построение абстрактных последовательностей команд (без указания конкретных тестовых данных); для каждой из них применяются следующие действия:

   a) конкретизация команд:
      - выбор регистров;
      - генерация данных;
      - построение инициализирующего кода;

b) исполнение команд на внутреннем симуляторе MicroTESK;

c) при генерации самопроверяющих программ: вставка в последовательность проверок на основе данных симулятора;

3) печать тестовой программы на языке ассемблера.

B. Спецификации системы команд

Архитектура RISC-V описана в руководстве «*The RISC-V Instruction Set Manual*» [16-17]. Система команд состоит из базового набора команд и расширений. Предусмотрены 32-, 64- и 128-битные реализации RISC-V. Для 32- и 64-битных версий определены следующие подмножества системы команд:

1) **RV32I** (базовый набор) — содержит команды целочисленной арифметики, ветвления, доступа к памяти и системных вызовов, общие для 32- и 64-битной версий;

2) **RV64I** (дополнение к **RV32I**) — содержит дополнительные команды целочисленной арифметики и доступа к памяти для 64-битной версии;

3) **RV32M** (стандартное расширение) — содержит команды целочисленного умножения и деления, общие для 32- и 64-битной версий;

4) **RV64M** (дополнение к **RV32M**) — содержит дополнительные команды целочисленного умножения и деления для 64-битной версии;

5) **RV32A** (стандартное расширение) — содержит команды атомарных операций над данными из памяти для 32- и 64-битной версий;

6) **RV64A** (дополнение к **RV32A**) — содержит дополнительные команды атомарных операций над данными из памяти для 64-битной версии;

7) **RV32F** (стандартное расширение) — содержит команды арифметики с плавающей точкой одинарной точности и команды конвертации в целочисленный формат, общие для 32- и 64-битной версий;

8) **RV64F** (дополнение к **RV32F**) — содержит дополнительные команды конвертации для 64-битной версии;

9) **RV32D** (стандартное расширение) — содержит команды арифметики с плавающей точкой двойной точности и команды конвертации в целочисленный формат, общие для 32- и 64-битной версий;

10) **RV64D** (дополнение к **RV32D**) — содержит дополнительные команды конвертации для 64-битной версии;

11) **RV32C** (стандартное расширение) — содержит сжатые (16-битные) команды, общие для 32- и 64-битной версий;

12) **RV64C** (дополнение к **RV32C**) — содержит дополнительные сжатые (16-битные) команды для 64-битной версии;

13) привилегированные системные команды.

Состояние микропроцессоров RISC-V описывается следующими регистрами:

1) счетчик команд PC размером 32 или 64 бита;

2) 32 регистра общего назначения x0-x31 размером 32 или 64 бита (RV32I и RV64I);

3) 32 регистра для хранения чисел с плавающей точкой одинарной точности f0-f31 (RV32F и RV64F);

4) 32 регистра для хранения чисел с плавающей точкой двойной точности f0-f31 (RV32D и RV64D);

5) управляющий регистр для операций с плавающей точкой FCSR (Floating-Point Control and Status Register) размером 32 бита (RV32F, RV64F, RV32D и RV64D);

6) управляющие системные регистры CSR (Control and Status Registers) размером 32 или 64 бита (RV32I и RV64I) в количестве до 4096.

Микропроцессоры RISC-V поддерживают три уровня привилегий, переключение между которыми осуществляется системными командами:

1) M (Machine) — высший уровень привилегий;

2) S (Supervisor) — уровень операционной системы;

3) U (User) – уровень пользовательских приложений.

На языке nML были описаны все перечисленные подмножества системы команд. Версия архитектуры (32 или 64 бита) задается при помощи директивы.

В приведенном ниже примере описываются регистры общего назначения, режим адресации для доступа к ним и команды целочисленного сложения.

```
// Разрядность слова (зависит от конфигурации)
let XLEN = #ifdef RV64I 64 #else 32 #endif
type XWORD = card(XLEN)

// Регистры общего назначения
reg XREG [32, XWORD]

// Режим адресации для регистров общего назначения
mode X (i: card(5)) = XREG [i]
  syntax = format("x%d", i) // Ассемблерный формат
  image  = format("%5s", i) // Бинарный формат

// Команда сложения регистра и 12-битного значения
op addi(rd: X, rs1: X, imm: card(12))
  syntax = format("addi %s, %s, 0x%x",
                  rd.syntax, rs1.syntax, imm)
  image  = format("%12s%s000%s0010011",
                  imm, rs1.image, rd.image)
  action = { // Осуществляемое действие
    rd = rs1 + sign_extend(XWORD, imm);
  }
```

В описании команд может быть несколько путей исполнения. В качестве примера ниже приведена спецификация команды целочисленного деления DIV.

Для всех путей исполнения MicroTESK строит описывающие их ограничения (формулы). С помощью оператора *mark* этим ограничениям можно назначить имена (см. пример ниже), позволяющие ссылаться на ограничения из тестовых шаблонов.

```
op div(rd: X, rs1: X, rs2: X)
 syntax = format("div %s, %s, %s",
                 rd.syntax, rs1.syntax, rs2.syntax)
 image  = format("0000001%s%s100%s0110011",
                 rs2.image, rs1.image, rd.image)
 action = {
  if rs2 == 0 then
   mark("div_by_zero");
   rd = -1;
  elif rs1 == 1 << (XLEN - 1) && rs2 == -1 then
   mark("min_xint_negation");
   rd = rs1;
  else
   mark("normal");
   rd = cast(XINT, rs1) / cast(XINT, rs2);
  endif;
 }
```

C. Генерация тестовых программ

После того как разработаны спецификации и по ним автоматически построена модель архитектуры, можно использовать среду генерации MicroTESK для построения тестовых программ. Для этого на вход инструменту подаются тестовые шаблоны на языке Ruby, описывающие сценарии тестирования с помощью специальных конструкций.

При построении тестовых программ задействуются разные техники: рандомизация, перебор, разрешение ограничений. Например, приведенный ниже тестовый шаблон порождает 10 тестовых воздействий, каждое из которых состоит из команды ADD со случайными номерами регистров.

```
# Класс на Ruby, определяющий тестовый шаблон
class RandomTemplate < RISCVBaseTemplate

 # Главный метод тестового шаблона
 def run

  # Распределение вероятностей для входных данных
  int_dist = dist(
   range(:value => 0, :bias => 25),
   range(:value => 1..2, :bias => 25),
   range(:value => 0xffffFFFE..0xffffFFFF, :bias => 50))

  # Описание структуры тестового воздействия
  sequence {
   add x(_), x(_), x(_) do
    # Задание способа генерации входных данных
    testdata('random_biased', :dist => int_dist)
   end
  }.run 10 # Число тестовых воздействий
 end
end
```

Ниже приведен фрагмент шаблона, описывающий последовательности команд ADD и SUB, которые принимают в качестве входных данных декартово произведение значений из диапазона [1, 3]. Всего получается 81 последовательность (3 ^ 4, где 3 – число значений, а 4 – число входных регистров).

```
sequence(:data_combinator => 'product') {
 add x8, x9, x10 do
  testdata('range', :min => 1, :max => 3)
 end
 sub x11, x12, x13 do
  testdata('range', :min => 1, :max => 3)
 end
}.run
```

Следующий пример демонстрирует совместное применение комбинаторной генерации и генерации на основе ограничений. Входные данные для команды DIV строятся таким образом, чтобы покрыть все пути исполнения, заданные в спецификации. При этом каждый вариант дополняется как командой SLT, так и командой MUL, что дает 6 последовательностей.

```
# Комбинирует результаты вложенных блоков
block(:combinator => 'product', :compositor => 'random'){
 iterate { # Итерирует по вложенным командам
  div x8, x9, x10 do situation('div_by_zero') end
  div x8, x9, x10 do situation('min_xint_negation') end
  div x8, x9, x10 do situation('normal') end
 }
 iterate { # Итерирует по вложенным командам
  slt x(_), zero, x8
  mul x(_), x8, x10
 }
}.run
```

Для каждого тестового воздействия строится *инициализирующий код*, помещающий тестовые данные в регистры и память. Для этого используются так называемые *препараторы*. Можно определить несколько препараторов, которые будут применяться в зависимости от формата тестовых данных. Ниже приведены препараторы, записывающие данные в регистры общего назначения.

```
# Препаратор по умолчанию для регистров X
preparator(:target => 'X') {
  # Записывает значение value в регистр target
  li target, value
}

# Препаратор для случая, когда value равно 0
preparator(:target => 'X', :mask => '0000_0000') {
  or target, zero, zero
}
```

IV. Характеристики разработанного генератора тестовых программ

Инструмент MicroTESK for RISC-V позволяет генерировать следующие виды тестов:

1) случайные тесты:
   a) рандомизация последовательностей команд;
   b) рандомизация тестовых данных:
      i. задание распределений вероятностей;
      ii. случайный выбор значений из диапазонов;

2) комбинаторные тесты:
   a) комбинирование последовательностей команд;
   b) перебор тестовых данных:
      i. перебор граничных значений;
      ii. перебор значений из диапазонов;
3) комбинаторные тесты на команды ветвления:
   a) перебор графов потока управления;
   b) перебор трасс исполнения;
4) тесты на покрытие путей исполнения команд:
   a) перебор путей исполнения;
   b) разрешение ограничений;
5) тесты со встроенными проверками;
6) комбинированные тесты, сочетающие свойства перечисленных выше типов тестов.

Пакет MicroTESK for RISC-V включает тестовые шаблоны, демонстрирующие возможности генератора. Также он содержит шаблоны, описывающие тесты из набора Университета Калифорнии в Беркли [12].

Таблица 1

Специфицированные команды RISC-V

Подмножество	**Описание**	**Число**
RV32I	целочисленная арифметика, доступ к памяти, ветвления	37
RV32I	доступ к управляющим регистрам	6
RV32I	системные команды	9
RV64I	целочисленная арифметика, доступ к памяти	12
RV32M	целочисленное умножение и деление	8
RV64M	целочисленное умножение и деление	5
RV32A	атомарные операции	11
RV64A	атомарные операции	11
RV32F	плавающая арифметика	26
RV64F	плавающая арифметика	4
RV32D	плавающая арифметика	26
RV64D	плавающая арифметика	6
RV32C	сжатые команды	35
RV64C	сжатые команды	7
Псевдокоманды	псевдокоманды	59
Всего		**262**

В работе по созданию формальных спецификаций системы команд RISC-V на языке nML участвовало 2 человека; общие трудозатраты составили около 4 человеко-месяцев. Всего описано 262 команды (включая 59 псевдокоманд, используемых ассемблером). Объем спецификаций составил около 3500 строк кода. В таблице 1 приведены данные по специфицированным командам.

При разработке спецификаций возможны ошибки, которые выражаются в некорректном ассемблерном формате команд или некорректных результатах их исполнения на встроенном симуляторе. В результате тестовые программы либо не компилируются, либо приводят к ошибочным результатам. Чтобы избежать подобных проблем для всех тестовых шаблонов, входящих в MicroTESK for RISC-V, при сборке дистрибутива осуществляются следующие действия.

1) По шаблонам генерируются тестовые программы. При генерации программа исполняется на встроенном симуляторе, создающем *трассы исполнения* (трассы фиксируют такие события, как исполнение команды, запись в регистр и обращение к памяти).

2) Построенные инструментом тестовые программы компилируются и исполняются на таких симуляторах, как QEMU for RISC-V [18] и Spike [19], создающих трассы того же формата.

3) Трассы исполнения одной и той же программы сравниваются с помощью средства Trace Matcher [20], которое генерирует отчет о найденных различиях.

V. Заключение

В работе описан генератор тестовых программ для микропроцессоров с архитектурой RISC-V, созданный в ИСП РАН с помощью инструмента MicroTESK. Инструмент позволяет генерировать тесты разных типов, в частности аналогичные тем, что входят в тестовый набор Университета Калифорнии в Беркли, и тем, что создаются инструментом Torture. Важными преимуществами MicroTESK for RISC-V являются открытость и расширяемость, что позволяет пополнять систему команд без модификации ядра генератора.

В ближайшем будущем мы планируем описать подсистему памяти микропроцессоров RISC-V, а также команды из появляющихся стандартных расширений (векторные команды и другие). Также мы планируем реализовать следующие вспомогательные средства инструмента MicroTESK:

1) генераторы отчетов о тестовом покрытии уровня команд, достигаемом сгенерированными программами;

2) инструмент автоматизированного создания online-генераторов тестов – генераторов, работающих на ПЛИС-прототипах или опытных образцах СБИС;

3) инструмент статического анализа и дедуктивной верификации бинарного кода.

ЛИТЕРАТУРА

[1] URL: https://en.wikipedia.org/wiki/RISC-V (дата обращения: 28.03.2018)
[2] URL: https://riscv.org (дата обращения: 28.03.2018)
[3] URL: https://www.phoronix.com/scan.php?page=news_item&px=NVIDIA-RISC-V-Next-Gen-Falcon (дата обращения: 28.03.2018)
[4] URL: https://www.electronicsweekly.com/blogs/mannerisms/dilemmas/samsung-defection-arm-risc-v-2016-11 (дата обращения: 28.03.2018)
[5] URL: https://www.pulp-platform.org (дата обращения: 28.03.2018)
[6] Камкин А.С. Генерация тестовых программ для микропроцессоров. Труды ИСП РАН, Т. 14, Ч. 2, 2008. С. 23-64.
[7] Adir A., Almog E., Fournier L., Marcus E., Rimon M., Vinov M., and Ziv A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification // IEEE Design & Test of Computers, Volume 21, Issue 2, 2004, P. 84–93.
[8] URL: https://forge.ispras.ru/projects/microtesk-riscv (дата обращения: 28.03.2018)
[9] URL: https://forge.ispras.ru/projects/microtesk (дата обращения: 28.03.2018)
[10] Chupilko M., Kamkin A., Kotsynyak A., Tatarnikov A. MicroTESK: Specification-Based Tool for Constructing Test Program Generators. Hardware and Software: Verification and Testing. Proceedings of 13th International Haifa Verification Conference, 2017. P. 217-220.
[11] Freericks M. The nML Machine Description Formalism. Techical Report, TU Berlin, FB20, Bericht 1991/15.
[12] URL: https://github.com/riscv/riscv-tests (дата обращения: 28.03.2018)
[13] URL: https://github.com/ucb-bar/riscv-torture (дата обращения: 28.03.2018)
[14] URL: http://www.slideshare.net/DVClub/introducing-obsidian-software-and-ravengcs-for-powerpc (дата обращения: 28.03.2018)
[15] Tatarnikov A.D. Language for Describing Templates for Test Program Generation for Microprocessors. Trudy ISP RAN (Proceedings of ISP RAS), Volume 28, Part 4, 2016. P. 81-102.
[16] Waterman A., Asanovic K. The RISC-V Instruction Set Manual. Volume I: User-Level ISA. Version 2.2. University of California, Berkeley. May 7, 2017. 145 P.
[17] Waterman A., Asanovic K. The RISC-V Instruction Set Manual. Volume II: Privileged Architecture. Version 1.10. University of California, Berkeley. May 7, 2017. 91 P.
[18] URL: https://forge.ispras.ru/projects/qemu-riscv (дата обращения: 28.03.2018)
[19] URL: https://github.com/riscv/riscv-isa-sim (дата обращения: 28.03.2018)
[20] URL: https://forge.ispras.ru/projects/traceutils (дата обращения: 28.03.2018)

MicroTESK-Based Test Program Generator for the RISC-V Architecture

A.S. Kamkin[1, 2, 3, 4], A.S. Protsenko[1], S.A. Smolov[1], A.D. Tatarnikov[1, 4]

[1]Ivannikov Institute for System Programming of the Russian Academy of Sciences, Moscow

[2]Lomonosov Moscow State University, Moscow

[3]Moscow Institute of Physics and Technology, Moscow

[4]National Research University Higher School of Economics, Moscow

{kamkin, protsenko, smolov, andrewt}@ispras.ru

***Abstract* — In this paper, a specification-based test program generator for functional verification of RISC-V microprocessors is presented. The tool is based on the MicroTESK framework and consists of two main parts: (1) the formal specifications of the RISC-V ISA and (2) the ISA-independent generation core. Test programs are generated on the basis of the ISA specifications and test templates that are high-level descriptions of test scenarios. The RISC-V ISA specifications are written in nML language. They describe the syntax and semantics of the instructions. The information extracted from the specifications is used in multiple ways: to get instruction signatures to be used in test templates; to build the test coverage model that holds constraints describing execution paths of the instructions; to construct the instruction set simulator that serves as a reference model. Test templates are created using a Ruby-based domain-specific language and describe how instruction sequences are composed and what constraints and generation methods are applied. The generation core provides random, combinatorial, and constraint-based test generation methods. The built-in instruction set simulator allows executing instructions in the process of test program generation. This allows predicting the microprocessor state to ensure the validity of the tests, to create self-checks, and to solve constraints that use state information. MicroTESK for RISC-V supports the following instruction subsets: RV32I, RV64I, RV32M, RV64M, RV32A, RV64A, RV32F, RV64F, RV32D, RV64D, RV32C, and RV64C. In total, the specifications cover 262 instructions. The effort required to develop the specifications constituted about 4 man-months. The specifications can be easily modified to support more instructions. MicroTESK for RISC-V includes a set of test templates that provide basic ISA coverage and demonstrate the generator facilities.**

***Keywords* — microprocessors; formal specifications; instruction set architecture; functional verification; test program generation; RISC-V; nML; MicroTESK.**

REFERENCES

[1] URL: https://en.wikipedia.org/wiki/RISC-V (access date: 28.03.2018)
[2] URL: https://riscv.org (access date: 28.03.2018)
[3] URL: https://www.phoronix.com/scan.php?page=news_item&px=NVIDIA-RISC-V-Next-Gen-Falcon (access date: 28.03.2018)
[4] URL: https://www.electronicsweekly.com/blogs/mannerisms/dilemmas/samsung-defection-arm-risc-v-2016-11 (access date: 28.03.2018)
[5] URL: https://www.pulp-platform.org (access date: 28.03.2018)
[6] Kamkin. A. Generatsiya testovykh programm dlya mikroprotsessorov (Test Program Generation for Microprocessors). Trudy ISP RAN (Proceedings of ISP RAS), Volume 14, Part 2, 2008, P. 23-64.
[7] Adir A., Almog E., Fournier L., Marcus E., Rimon M., Vinov M., and Ziv A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification // IEEE Design & Test of Computers, Volume 21, Issue 2, 2004, P. 84–93.
[8] URL: https://forge.ispras.ru/projects/microtesk-riscv (access date: 28.03.2018)
[9] URL: https://forge.ispras.ru/projects/microtesk (access date: 28.03.2018)
[10] Chupilko M., Kamkin A., Kotsynyak A., Tatarnikov A. MicroTESK: Specification-Based Tool for Constructing Test Program Generators. Hardware and Software: Verification and Testing. Proceedings of 13th International Haifa Verification Conference, 2017. P. 217-220.
[11] Freericks M. The nML Machine Description Formalism. Techical Report, TU Berlin, FB20, Bericht 1991/15.
[12] URL: https://github.com/riscv/riscv-tests (access date: 28.03.2018)
[13] URL: https://github.com/ucb-bar/riscv-torture (access date: 28.03.2018)
[14] URL: http://www.slideshare.net/DVClub/introducing-obsidian-software-and-ravengcs-for-powerpc (access date: 28.03.2018)
[15] Tatarnikov A.D. Language for Describing Templates for Test Program Generation for Microprocessors. Proceedings of ISP RAS, Volume 28, Part 4, 2016. P. 81-102.
[16] Waterman A., Asanovic K. The RISC-V Instruction Set Manual. Volume I: User-Level ISA. Version 2.2. University of California, Berkeley. May 7, 2017. 145 P.
[17] Waterman A., Asanovic K. The RISC-V Instruction Set Manual. Volume II: Privileged Architecture. Version 1.10. University of California, Berkeley. May 7, 2017. 91 P.
[18] URL: https://forge.ispras.ru/projects/qemu-riscv (access date: 28.03.2018)
[19] URL: https://github.com/riscv/riscv-isa-sim (access date: 28.03.2018)
[20] URL: https://forge.ispras.ru/projects/traceutils (access date: 28.03.2018)