Генератор тестовых программ для архитектуры ARMv8 на основе инструмента MicroTESK

[1, 2, 3] *А.С. Камкин <kamkin@ispras.ru>*
[1] *А.М. Коцыняк <kotsynyak@ispras.ru>*
[1] *А.С. Проценко <protsenko@ispras.ru>*
[1] *А.Д. Татарников <andrewt@ispras.ru>*
[1] *М.М. Чупилко <chupilko@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, г. Москва, Ленинские горы, д. 1
[3] *Московский физико-технический институт,*
141700, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9

Аннотация. ARM — это семейство микропроцессорных архитектур, разработанных в одноименной компании. Новейшая архитектура этого семейства, ARMv8, содержит большое число команд разных типов и отличается сложной организацией виртуальной памяти (включающей аппаратную поддержку многоуровневой трансляции адресов и виртуализации); все это делает функциональную верификацию микропроцессоров этой архитектуры крайне трудной технической задачей. Неотъемлемой частью верификации микропроцессора является генерация тестовых программ — программ на языке ассемблера, создающих разнообразные ситуации (исключения, блокировки конвейера, неверные предсказания переходов, вытеснения данных из кэш-памяти и т.п.). В статье описываются требования, предъявляемые к промышленным генераторам тестовых программ, и представляется генератор для микропроцессоров архитектуры ARMv8, разработанный с использованием инструмента MicroTESK (Microprocessor TEsting and Specification Kit). Генератор поддерживает подмножество команд, характерное для мобильных приложений (около 400 команд) и состоит из двух основных частей: (1) архитектурно независимого ядра и (2) формальной спецификации ARMv8 (точнее, модели, автоматически построенной по формальным спецификациям). При такой организации процесс разработки генератора тестовых программ состоит преимущественно в создании формальных спецификаций, что экономит усилия за счет повторного использования архитектурно независимых компонентов. Архитектура описывается на языках nML и MMUSL: первый язык позволяет описывать регистры микропроцессора, синтаксис и семантику команд; второй — устройство подсистемы памяти (адресные пространства, разного рода буферы и таблицы, алгоритмы

трансляции адресов и т.п.). В статье приводятся характеристики разработанного генератора и делается сравнение с существующими аналогами.

Ключевые слова: микропроцессоры; спецификация системы команд; функциональная верификация; генерация тестовых программ; ARM; nML; MicroTESK.

DOI: 10.15514/ISPRAS-2016-28(6)-6

Для цитирования: Камкин А.С., Коцыняк А.М., Проценко А.С., Татарников А.Д., Чупилко М.М. Генератор тестовых программ для архитектуры ARMv8 на основе инструмента MicroTESK. Труды ИСП РАН, том 28, вып. 6, 2016, стр. 87-102. DOI: 10.15514/ISPRAS-2016-28(6)-6

1. Введение

ARM (изначально Acorn RISC Machine, сейчас Advanced RISC Machine) — это семейство микропроцессорных архитектур, разработанных в одноименной компании [1]. Микропроцессоры этого семейства чрезвычайно популярны: с 1990 года было выпущено более 86 миллиардов чипов [2]; 95% смартфонов базируются на ARM (данные за 2010 год) [3]; микросхемы ARM находят применение во встроенных системах и центрах обработки данных. Разработкой ARM-совместимых ядер занимается не только ARM, но и другие компании, включая Qualcomm, Apple, Samsung, NVIDIA и Huawei [4]; очевидно, что эффективная реализация микропроцессора есть важнейший инструмент конкурентной борьбы. Новая архитектура ARMv8(-A) отличается большим числом команд (около 1000) и сложной организацией виртуальной памяти (VMSA, Virtual Memory System Architecture) [5].

Проектирование ARMv8-совместимого микропроцессора, оптимизированного по производительности, энергопотреблению и стоимости, чревато ошибками. Для их выявления применяют разные методы: инспекция кода, тестирование, формальная верификация. Основным подходом является исполнение на модели микропроцессора тестовых программ и сравнение результатов исполнения (трасс) с результатами, полученными при исполнении тех же программ на программном эмуляторе (эталонной модели) [6]. Общепринятая практика создания тестовых программ базируется на рандомизированной генерации на основе шаблонов: пользователь определяет шаблон — описание структуры программы и ограничений на используемые данные, — генератор же строит тест, рандомизируя элементы шаблона, не заданные жестко [7].

Задача генерации тестовых программ для микропроцессора не так проста, как может показаться на первый взгляд. При конструировании тестов нужно учитывать не только формат, но семантику команд: сгенерированные данные не должны приводить к неопределенному поведению (такие ситуации особо выделяются в руководствах по архитектурам); программы не должны зацикливаться; поток управления не должен переходить в секцию данных и

т.д. и т.п. Кроме того, генератор должен принимать в расчет такие механизмы микропроцессора, как конвейер, обработка исключений, виртуальная память, многоядерность.

В работе рассматривается генератор тестовых программ для подмножества ARMv8 (около 400 команд), разработанный с использованием инструмента MicroTESK (Microprocessor TEsting and Specification Kit) [8]. Генератор состоит из двух основных частей: архитектурно независимого ядра (библиотечной части) и формальной спецификации ARMv8 на языках nML (система команд) [9] и MMUSL (подсистема памяти) [10]. При такой организации создание генератора состоит в основном в написании спецификаций, что, во-первых, экономит усилия за счет использования общих компонентов, а во-вторых, упрощает адаптацию генератора к модификациям архитектуры (добавлению новых команд, изменениям в подсистеме памяти и т.п.); помимо прочего, использование формальных спецификаций позволяет автоматизировать решение проблем, обозначенных выше.

Оставшаяся часть статьи структурирована следующим образом. В разделе 2 рассматриваются существующие решения в области генерации тестовых программ для микропроцессоров: RIS (ARM), RAVEN (ARM) и Genesys-Pro (IBM). Раздел 3 посвящен генератору тестовых программ MicroTESK-ARMv8; здесь описываются базовые принципы используемого подхода (MicroTESK) и особенности его применения к архитектуре ARMv8. В разделе 4 приводятся характеристики разработанного генератора (поддерживаемые команды, объем спецификаций и т.п.), а также сведения о трудоемкости разработки. Раздел 5 резюмирует работу и обрисовывают направления дальнейших исследований.

2. Обзор существующих решений

Компания ARM разрабатывает собственные инструменты генерации тестовых программ. К настоящему времени разработано несколько генераторов, получивших название RIS (Random Instruction Sequence) [11]. Это набор узкоспециализированных средств, предназначенных для тестирования таких механизмов, как многоядерность [12] и управление памятью [13]. Настройка RIS осуществляется путем задания используемых команд, их весов (распределений вероятности), ограничений на операнды, способа размещения кода и данных в памяти, а также цепочек команд, решающих специальные задачи (вытеснение данных из кэш-памяти и т.п.).

Другим инструментом, используемым в ARM, является RAVEN (Random Architecture Verification Machine) [14], разработанный в компании Obsidian Software, поглощенной ARM в 2011 году. Это универсальное средство, применимое для широкого спектра архитектур, в котором ядро, реализующее логику генерации, отделено от конфигурации для конкретной архитектуры. RAVEN позволяет создавать как случайные, так и нацеленные тесты. В процессе работы инструмент отслеживает состояние микропроцессора путем симуляции построенных программ на внешней эталонной модели.

Еще одним универсальным средством генерации тестовых программ является Genesys-Pro [7] от IBM Research. Этот инструмент использует входные данные двух типов: модель, описывающая архитектуру микропроцессора, и шаблоны, формулирующие задачи тестирования. Шаблоны создаются на выразительном языке, имеющем конструкции для описания цепочек команд, распределений вероятностей и ограничений. Как и в RAVEN, состояние микропроцессора в Genesys-Pro отслеживается с использованием внешней эталонной модели. Возможности инструмента могут быть расширены средством DeepTrans [15], предназначенным для верификации механизмов трансляции адресов.

Подводя итог, отметим следующие недостатки существующих решений. Инструменты типа RIS жестко привязаны к архитектуре, что делает их непригодными для верификации микропроцессоров с новой системой команд. Инструменты типа RAVEN и Genesys-Pro этого недостатка лишены — их можно адаптировать к новой архитектуре; однако эта работа не является простой и требует определенной квалификации. Так, при расширении архитектуры необходимо добавить знание о новых командах в генератор, а также реализовать эти команды в эталонной модели. Предлагаемый подход нацелен на максимальное упрощение процессов разработки и сопровождения генераторов тестовых за счет извлечения всей необходимой информации из единого источника — формальных спецификаций.

3. Генератор тестовых программ MicroTESK-ARMv8

3.1 Инструмент MicroTESK

Описываемый в статье генератор тестовых программ для микропроцессоров архитектуры ARMv8 реализован с помощью инструмента MicroTESK [8], разрабатываемого в ИСП РАН. Инструмент включает в себя две основные части: (1) *среду моделирования* и (2) *среду генерации*. Среда моделирования отвечает за построение модели микропроцессора на основе формальных спецификаций. Задача среды генерации — построение тестовых программ для микропроцессора по сценариям, описанным в тестовых шаблонах. Структура инструмента MicroTESK изображена на рис. 1.

В настоящее время среда моделирования поддерживает два типа *формальных спецификаций*: (1) *спецификации системы команд микропроцессора* на языке nML [9] и (2) *спецификации подсистемы памяти микропроцессора* на языке MMUSL [10]. Для каждого типа поддерживаются свои анализаторы и генераторы кода модели (их набор может быть расширен). Модель микропроцессора, генерируемая по спецификациям, включает в себя: (1) *метаданные*, хранящие информацию о доступных регистрах и командах; (2) *симулятор*, позволяющий программно эмулировать исполнение команд и получать информацию о состоянии микропроцессора; (3) *модель тестового покрытия*, содержащая информацию о возможных путях исполнения команд.

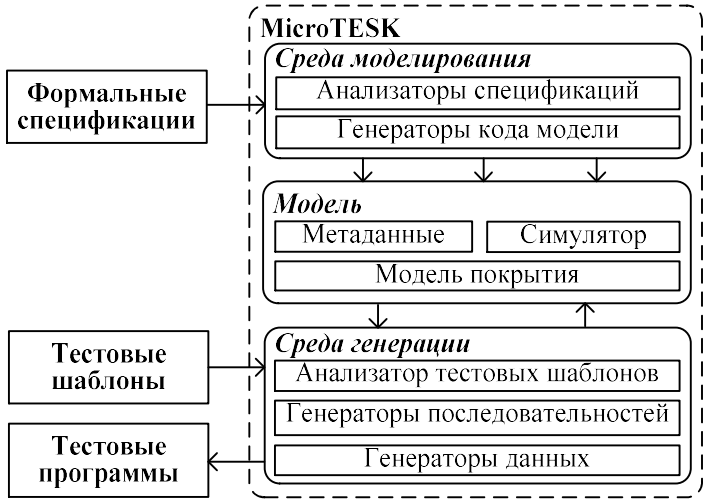

Рис. 1. Архитектура инструмента MicroTESK

Fig. 1. Architecture of the MicroTESK instrument

Среда генерации использует *тестовые шаблоны* на специальном языке, основанном на Ruby [16]. Язык позволяет описывать сценарии тестирования на достаточно абстрактном уровне, когда состав, порядок и операнды команд не фиксируются, а выбираются динамически в зависимости от поставленной задачи. В шаблонах можно «обращаться» к регистрам и «вызывать» команды микропроцессора, как это делается на языке ассемблера — для этого на основе метаданных инструмент автоматически создается оберточные функции.

Процесс генерации тестовых программ (потока команд) по шаблону состоит из следующих стадий:

1. анализ тестового шаблона и построение внутреннего представления;
2. перебор абстрактных последовательностей команд и осуществление следующих действий для каждой из них:
    a. конкретизация последовательности команд: выбор регистров, генерация данных, построение инициализирующего кода;
    b. исполнение построенных команд на симуляторе;
    c. вставка в последовательность проверок на основе данных из симулятора (при генерации самопроверяющих программ);
    d. добавление последовательности в поток команд;
3. разбиение потока на программы и их печать в ассемблерном формате.

3.2 Спецификация команд ARMv8

Архитектура ARMv8 описывается на примерно 4000 страницах руководства «*ARM Architecture Reference Manual*» [5]. Описание включает архитектурные особенности (уровни исключений и т.п.), систему команд (около 1000 команд разных типов) и организацию подсистемы памяти (см. раздел 3.3). Всю совокупность команд можно разбить на следующие группы:

- *команды ветвления* (условные и безусловные переходы, исключения, возврат из обработчика исключений);
- *команды загрузки и сохранения данных* (обычные загрузки/сохранения для различных типов данных, загрузки/сохранения с блокировками);
- *вычислительные команды* (инициализация регистров, целочисленная арифметика и арифметика с плавающей точкой);
- *векторные команды* (SIMD — Single Instruction, Multiple Data),
- *системные команды* (работа с системными регистрами).

Состояние микропроцессора (точнее, одного его ядра) описывается набором значений регистров, соответствующих текущему уровню исполнения команд (EL — Exception Level). Всего поддерживается четыре уровня, переключение между которыми осуществляется специальными системными командами:

- EL0 — уровень пользовательских приложений (непривилегированное исполнение);
- EL1 — уровень ядра операционной системы (привилегированное исполнение);
- EL2 — уровень гипервизора;
- EL3 — уровень защищенного монитора.

Архитектурой определены следующие регистры:

- R0-R30 — 64-битные регистры общего назначения (с доступом ко всем 64 битам или только к младшим 32);
- SP — указатель стека текущего EL;
- PC — счетчик команд (недоступный программно);
- V0-V31 — 128-битные регистры для хранения чисел с плавающей точкой (с поддержкой разных форматов и возможностью векторной интерпретации одного регистра);
- FPCR и FPSR — контрольный и статусный регистры для операций над числами с плавающей точкой;
- несколько сотен системных регистров.

Все регистры ARMv8 и около 40% команд (см. раздел 4), характерных для мобильных устройств (все команды за исключением SIMD и арифметики с плавающей точкой), были специфицированы на языке nML [9].

Ниже в качестве иллюстрации приведено описание команды MOVZ (Move Wide with Zero), перемещающей 16-битные данные в регистр и обнуляющей 48 оставшихся битов.

```
// Константы, соответствующие разным режимам перемещения
let MoveWideOp_N = 0b00 // Перемещение с инверсией всех битов
let MoveWideOp_Z = 0b10 // Перемещение с обнулением остальных битов
let MoveWideOp_K = 0b11 // Перемещение без изменения остальных битов

// Общая операция (функция) для команд перемещения 16-битных данных
op MovWideImmGeneral (rd: REG, imm: HWORD, shift: card(2), opcode: card(2))
  action = {
    // Учет режима перемещения (обнуление битов)
    if opcode != MoveWideOp_K then rd = 0; endif;

    // Выбор поля регистра для записи
    if shift == 0 then
      rd<15..0> = imm;
    elif shift == 1 then
      rd<31..16> = imm;
    elif shift == 2 then
      rd<47..32> = imm;
    elif shift == 3 then
      rd<63..48> = imm;
    endif;

    // Учет режима перемещения (инверсия битов)
    if opcode == MoveWideOp_N then rd = ~rd; endif;
  }

// Команда перемещения 16-битных данных с обнулением остальных битов
op movz (rd: REG, imm: HWORD, shift: card(2))
  // Ассемблерный синтаксис
  syntax = format("movz %s, #0x%x, LSL #%d",
             rd.syntax, imm, coerce(BYTE, shift) * 16)
  // Двоичный формат
  image = format("%s10100101%2s%16s%5s", coerce(BIT, 1), shift, imm, rd.image)
  // Совершаемые командой действия
  action = {
    MovWideImmGeneral(rd, imm, shift, MoveWideOp_Z).action;
  }
```

3.3 Спецификация подсистемы памяти ARMv8

Описание виртуальной памяти ARMv8 (VMSAv8-64) занимает около 600 страниц. Значительная часть документации посвящена механизму трансляции адресов. В VMSAv8-64 определены четыре режима трансляции: защищенный EL3, незащищенный EL2, EL1&0 в защищенном и незащищенном вариантах. Оба варианта режимов ELl&0 состоят в традиционном преобразовании виртуального адреса в физический. Режимы EL3 и EL2 включают два этапа:

сначала виртуальный адрес преобразуется в промежуточный физический, а затем промежуточный физический адрес — в физический. Каждое преобразование вызывает обход таблиц трансляции и может привести к четырем дополнительным обращениям к памяти. Для ускорения этого процесса таблицы трансляции кэшируются в ассоциативном буфере трансляции (TLB — Translation Lookaside Buffer).

Для спецификации подсистемы памяти MicroTESK использует специальный язык MMUSL, расширяющий возможности nML средствами описания типов адресов, сегментов, буферов и таблиц, а также логики обработки запросов к памяти. Были специфицированы TLB, таблицы трансляции, двухуровневая кэш-памяти и логика обработки запросов к памяти во всех режимах.

Ниже приведен пример, описывающий тип виртуального адреса — структуру, содержащую адрес и дополнительные данные о запросе к памяти.

```
// Тип виртуального адреса
address VA (
  // Виртуальный адрес
  addr : 64,
  // Дополнительные данные
  acctype: 4, iswrite: 1, wasaligned: 1, size: 6
)
```

Сегмент памяти задает отображение диапазона адресов некоторого типа на множество адресов другого типа. В следующем примере описан сегмент VA_LO_UNMAPPED, преобразующий виртуальные адреса в физические напрямую (без использования TLB и таблиц трансляции).

```
// Сегмент памяти (адресное пространство)
segment VA_LO_UNMAPPED (va: VA) = (pa: PA)
  // Диапазон отображаемых виртуальных адресов
  range = (0x0000000000000000, 0x0000ffffFFFFffff)
  // Способ преобразования виртуальных адресов в физические
  read = { pa.addrdesc.paddress.physicaladdress = va.vaddress<47..0>; }
```

Буферы (TLB, кэш-память, таблицы страниц и т.п.) характеризуются объемом, ассоциативностью, стратегией вытеснения данных и другими параметрами. Ниже приведен фрагмент спецификации TLB. Формат записи определен архитектурой, в то время как значения других параметров реализационно-зависимы. Ключевое слово **register** говорит о том, что буфер отображается на регистры, что делает его доступным из спецификации системы команд.

```
// Буфер ассоциативной трансляции, отображаемый на регистры
register buffer TLB (va: VA)
  // Ассоциативность
  ways = 64
  // Число множеств
  sets = 1
  // Формат записи
  entry = (addr: 36, nG: 1, contiguous: 1, level: 2, blocksize: 64,
           perms: Permissions, addrdesc: AddressDescriptor)
  // Предикат, проверяющий попадание в буфер
  match = (addr == va.addr<47..12> && addrdesc.fault.type == Fault_None)
  // Политика вытеснения данных
  policy = LRU
```

С помощью ключевого слова **memory** можно задать буферы, отображаемые на память: каждое обращение к ним вызывает «рекурсивный» доступ в память. Примером таких буферов являются таблицы трансляции VMSAv8-64.

```
// Таблица трансляции, отображаемая на память
memory buffer TranslationTable(va: VA)
  entry = (IGNORED: 9, XN: 1, PXN: 1, Contiguous: 1, RES0: 4, pa: 36, nG: 1,
           AF: 1, SH: 2, AP: 2, NS: 1, AttrIndx: 3, page: 1, valid: 1)
```

Логика загрузки и сохранения данных описывается запросами к сегментам и буферам. Синтаксис схож с nML, но допускает конструкции вида B(addr).**hit** (буфер B содержит запись для адреса addr), E = B(addr) (запись для адреса addr считывается из буфера B и сохраняется в E) и т.п. Ниже приведен фрагмент спецификации подсистемы памяти VMSAv8-64.


```
// Управляющая логика подсистемы памяти
mmu pmem (va: VA) = (data: 64)
  var pa: PA;
  var l1Entry: L1.entry;
  var line: 256;
  // Обработка запросов загрузки данных
  read = {
    if va.acctype != AccType_PTW then
     pa.addrdesc = AArch64TranslateAddress(va.vaddress, va.acctype, 0, 1, 8);
    else
      pa.addrdesc.paddress.physicaladdress = va.vaddress<47..0>;
      pa.size = va.size;
      pa.acctype = va.acctype;
    endif;
    if (va.acctype == AccType_AT) then update_PAR_EL1(pa); endif;
    if L1(pa).hit then
      l1Entry = L1(pa);
      line = l1Entry.DATA;
    else
      if va.acctype != AccType_PTW && va.acctype != AccType_AT then
        line = AArch64MemSingleRead(pa.addrdesc, 8, va.acctype, 1);
      else
        line = M(pa);
      endif;
      l1Entry.TAG = pa.addrdesc.paddress.physicaladdress<47..12>;
      l1Entry.DATA = line;
      L1(pa) = l1Entry;
    endif;
    data = get_word_from_line(va.vaddress<4..3>, line);
    if (va.acctype == AccType_IFETCH) then
      data<47..0> = pa.addrdesc.paddress.physicaladdress;
      data<63..48> = 0;
    endif;
  }

  // Обработка запросов сохранения данных
  write = {...}
```

3.4. Генерация тестовых программ для ARMv8

После того как разработаны спецификации и по ним автоматически построена модель микропроцессора, MicroTESK можно использовать для генерации тестовых программ. По сути, результатом анализа спецификаций является генератор тестовых программ, настроенный на конкретную архитектуру, или даже на конкретный микропроцессор. Программы строятся по шаблонам на языке Ruby, описывающим сценарии тестирования с помощью специально введенных в язык конструкций [16]. Важно отметить, что для команд

микропроцессора создается оберточные функции, что позволяет использовать в шаблонах нотацию, максимально приближенную к ассемблерной.

MicroTESK использует разные техники построения тестовых программ: случайные, комбинаторные, основанные на разрешении систем ограничений. Например, приведенный ниже шаблон порождает 10 тестовых воздействий, каждый из которых состоит из одной команды ADD (add_sh_reg), номера регистров которой и их значения генерируются случайно.

```
// Класс на языке Ruby, определяющий тестовый шаблон
class RandomTemplate < ArmV8BaseTemplate
  // Главный метод тестового шаблона
  def run
    // Описание распределения вероятностей для значений регистров
    integer_dist = dist(range(:value => 0x00000000..0x0000000F, :bias => 50),
                        range(:value => 0xffffFFF0..0xffffFFFF, :bias => 50))

    // Описание структуры тестового воздействия
    sequence {
      add_sh_reg reg(_), reg(_), reg(_), LSL, 0
        // Задание способа генерации тестовых данных
        do situation('random_biased', :dist => integer_dist) end
    }.run 10 // Число тестовых воздействий данного типа
  end
end
```

Следующий пример демонстрирует комбинаторную генерацию и генерацию на основе ограничений. Входные данные для команд ADD (adds_sh_reg) и SUB (subs_sh_reg) строятся таким образом, чтобы при их исполнении либо гарантированно не возникало никаких исключений (ситуация normal), либо гарантированно возникало переполнение (ситуация overflow). Конструкция **block** комбинирует (параметр combinator) и смешивает (параметр compositor) последовательности команд, возвращаемые вложенными блоками.


```
// Блок внешнего уровня, комбинирующий и смешивающий
// последовательности, возвращаемые вложенными блоками
block(:combinator => 'product', :compositor => 'random') {
  // Вложенный блок, возвращающий 2 последовательности,
  // каждая из которых содержит одну команду ADD
  iterate {
    // Команда ADD, не вызывающая исключений
    adds_sh_reg reg(_), x0, x1, LSL, 0 do situation('normal') end
    // Команда ADD, вызывающая переполнение
    adds_sh_reg reg(_), x2, x3, LSL, 0 do situation('overflow') end
  }
  // Вложенный блок, возвращающий 2 последовательности,
  // каждая из которых содержит одну команду SUB
  iterate {
    // Команда SUB, не вызывающая исключений
    subs_sh_reg reg(_), x0, x1, LSL, 0 do situation('normal') end
    // Команда SUB, вызывающая переполнение
    subs_sh_reg reg(_), x2, x3, LSL, 0 do situation('overflow') end
  }
}.run
```

Как отмечалось в разделе 3.1, для каждого тестового воздействия строится инициализирующий код, устанавливающий требуемое состояние регистров, буферов и памяти. Для этого используются так называемые препараторы, определенные в базовом тестовом шаблоне. Препаратор — это фрагмент кода, направленный на достижение той или иной цели: инициализации регистра, буфера или памяти. Как правило, число препараторов невелико; они задаются вручную.

Ниже представлен препаратор для начальной таблицы трансляции уровня EL1. Он размещает специальным образом отформатированную запись (entry) по определенному адресу (address).

```
// Препаратор начальной таблицы трансляции
buffer_preparator (:target => 'TranslationTable', :level => 0) {
  mrs x0, ttbr0_el1
  movz x1, address, 0
  add_sh_reg x0, x0, x1, LSL, 3
  movk x2, entry(0, 15), 0
  movk x2, entry(16, 31), 1
  movk x2, entry(32, 47), 2
  movk x2, entry(48, 63), 3
  stlr x2, x0
}
```

4. Характеристики генератора MicroTESK-ARMv8

Разработанный генератор тестовых программ для ARMv8 отвечает основным требованиями, предъявляемым к промышленным инструментам такого рода:

- поддержка разных техник построения тестовых программ;
- возможность построения самопроверяющих тестов;
- учет архитектурных и микроархитектрных особенностей (условий возникновения исключений, числа вычислительных ядер и потоков, параметров кэш-памяти и т.п.).

В работе по созданию формальных спецификаций ARMv8 участвовало 2 человека; продолжительность работ составила 10 месяцев. За это время были описаны 382 команды (все команды за исключением SIMD и арифметики с плавающей точкой) и подсистема памяти. В целом были специфицированы уровни EL0 и EL1. Объем кода составил около 7800 строк для системы команд (nML) и около 2000 строк для подсистемы памяти (MMUSL). В таблице 1 приведены сведения о специфицированных командах.

Табл. 1. Статистика по специфицированным командам

Table 1. Statistics on specified commands

Класс команд	**Число специфицированных команд**
(1) Передача управления	12
(2) Сохранение в память и загрузка из памяти	85
(3) Арифметика с непосредственными операндами	68
(4) Арифметика с регистровыми операндами	121
(5) Расширенная арифметика	22
(6) Арифметики с плавающей точкой	42 (из 168)
(7) Команды SIMD	0 (из 477)
(8) Системные команды	32
Всего	**382 (из 985)**

В настоящее время MicroTESK-ARMv8 успешно используется в известной международной компании, разрабатывающей собственные микропроцессоры архитектуры ARMv8.

5. Заключение

В работе рассмотрен генератор тестовых программ для архитектуры ARMv8, построенный с использованием инструмента MicroTESK. Решение базируется на формальных спецификациях системы команд и подсистемы памяти. По своим функциям MicroTESK-ARMv8 не уступает аналогам, но превосходит их в гибкости настройки на целевую архитектуру (архитектура ARMv8 является молодой и постоянно развивается: выходят дополнения, например, ARMv8.1-A, которые должны отражаться в генераторе).

В будущем мы планируем реализовать дополнительные средства MicroTESK, полезные при промышленном использовании этого инструмента: генераторы отчетов о тестовом покрытии, достигаемом сгенерированными программами; средства автоматического построения типовых тестов; инструментарий для конструирования online-генераторов тестовых программ, т.е. генераторов,

работающих на «железе» (ПЛИС-прототипе микропроцессора или опытном образце СБИС).

Список литературы

[1]. Сайт компании ARM. http://www.arm.com.
[2]. Mallya H. The Backstory of How ARM Reached a Milestone of 86 Billion Chips in 25 Years. July 19, 2016 (https://yourstory.com/2016/07/arm-holdings-story/).
[3]. Morgan T.P. ARM Holdings Eager for PC and Server Expansion. Record 2010, Looking for Intel Killer 2020. February 1, 2011 (http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/).
[4]. Sims G. Custom Cores versus ARM Cores, What Is It All About? January 7, 2016 (http://www.androidauthority.com/arm-cortex-core-custom-core-kryo-explained-664777/).
[5]. ARM Architecture Reference Manual. ARM DDI 0487A.f, ARM Corporation, 2015. 5886 p.
[6]. Камкин А.С. Генерация тестовых программ для микропроцессоров. Труды ИСП РАН, том 14, часть 2, 2008. с. 23-64.
[7]. Adir A., Almog E., Fournier L., Marcus E., Rimon M., Vinov M., Ziv A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. Design & Test of Computers, 21(2), 2004. pp. 84-93. DOI: 10.1109/MDT.2004.1277900.
[8]. Kamkin A., Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. Spring/Summer Young Researchers' Colloquium on Software Engineering, 2012. pp. 64-69. DOI: 10.15514/SYRCOSE-2012-6-8.
[9]. Freericks M. The nML Machine Description Formalism. Technical Report TR SM-IMP/DIST/08, TU Berlin CS Department, 1993.
[10]. Chupilko M., Kamkin A., Kotsynyak A., Protsenko A., Smolov S., Tatarnikov A. Specification-Based Test Program Generation for ARM VMSAv8-64 Memory Management Units. Workshop on Microprocessor Test and Verification, 2015. pp. 1-6. DOI: 10.1109/MTV.2015.13.
[11]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[12]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014. pp. 4-9. DOI: 10.1109/MTV.2014.28.
[13]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014. pp. 98-101. DOI: 10.1109/MTV.2014.26.
[14]. RAVEN test program generator (http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc).
[15]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006. pp. 102-110.
[16]. Tatarnikov A.D. Language for Describing Templates for Test Program Generation for Microprocessors. Труды ИСП РАН, vol. 28, issue 4, 2016. pp. 81-102. DOI: 10.15514/ISPRAS-2016-28(4)-5

MicroTESK-Based Test Program Generator for the ARMv8 Architecture

[1,2,3] *A.S. Kamkin <kamkin@ispras.ru>*
[1] *A.M. Kotsynyak <kotsynyak@ispras.ru>*
[1] *A.S. Protsenko <protsenko@ispras.ru>*
[1] *A.D. Tatarnikov <andrewt@ispras.ru>*
[1] *M.M. Chupilko <chupilko@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia.*
[3] *Moscow Institute of Physics and Technology, 9 Institutskiy per., Moscow Region, Dolgoprudny, 141700, Russia.*

Abstract. ARM is a family of microprocessor instruction set architectures developed in a company with the same name. The newest architecture of this family, ARMv8, contains a large number of instructions of various types and is notable for its complex organization of virtual memory, which includes hardware support for multilevel address translation and virtualization. All of this makes functional verification of microprocessors with this architecture an extremely difficult technical task. An integral part of microprocessor verification is generation of test programs, i.e. programs in the assembly language, which cause various situations (exceptions, pipeline stalls, branch mispredictions, data evictions in caches, etc.). The article describes the requirements for industrial test program generators and presents a generator for microprocessors with the ARMv8 architecture, which has been developed with the help of MicroTESK (Microprocessor TEsting and Specification Kit). The generator supports an instruction subset typical for mobile applications (about 400 instructions) and consists of two main parts: (1) an architecture-independent core and (2) formal specifications of ARMv8 or, more precisely, a model automatically constructed on the basis of the formal specifications. With such a structure, the process of test program generator development consists mainly in creation of formal specifications, which saves efforts by reusing architecture-independent components. An architecture is described using the nML and MMUSL languages. The first one allows describing the microprocessor registers and syntax and semantics of the instructions. The second one is used to specify the memory subsystem organization (address spaces, various buffers and tables, address translation algorithms, etc.) The article describes characteristics of the developed generator and gives a comparison with the existing analogs.

Keywords: microprocessors; instruction set specification; functional verification; test program generation; ARM; nML; MicroTESK.

DOI: 10.15514/ISPRAS-2016-28(6)-6

For citation: Kamkin A.S., Kotsynyak A.M., Protsenko A.S., Tatarnikov A.D., Chupilko M.M. MicroTESK-Based Test Program Generator for the ARMv8 Architecture. *Trudy ISP RAN/Proc. ISP RAS*, vol. 28, issue 6, 2016, pp. 87-102 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-6

References

[1]. ARM site. http://www.arm.com.
[2]. Mallya H. The Backstory of How ARM Reached a Milestone of 86 Billion Chips in 25 Years. July 19, 2016 (https://yourstory.com/2016/07/arm-holdings-story/).
[3]. Morgan T.P. ARM Holdings Eager for PC and Server Expansion. Record 2010, Looking for Intel Killer 2020. February 1, 2011 (http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/).
[4]. Sims G. Custom Cores versus ARM Cores, What Is It All About? January 7, 2016 (http://www.androidauthority.com/arm-cortex-core-custom-core-kryo-explained-664777/).
[5]. ARM Architecture Reference Manual. ARM DDI 0487A.f, ARM Corporation, 2015. 5886 p.
[6]. Kamkin A. Test Program Generation for Microprocessors. Trudy ISP RAN / Proc. ISP RAS, volume 14, part 2, 2008, pp. 23-64 (in Russian).
[7]. Adir A., Almog E., Fournier L., Marcus E., Rimon M., Vinov M., Ziv A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. Design & Test of Computers, 21(2), 2004. pp. 84-93. DOI: 10.1109/MDT.2004.1277900.
[8]. Kamkin A., Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. Spring/Summer Young Researchers' Colloquium on Software Engineering, 2012. pp. 64-69. DOI: 10.15514/SYRCOSE-2012-6-8.
[9]. Freericks M. The nML Machine Description Formalism. Technical Report TR SM-IMP/DIST/08, TU Berlin CS Department, 1993.
[10]. Chupilko M., Kamkin A., Kotsynyak A., Protsenko A., Smolov S., Tatarnikov A. Specification-Based Test Program Generation for ARM VMSAv8-64 Memory Management Units. Workshop on Microprocessor Test and Verification, 2015. pp. 1-6. DOI: 10.1109/MTV.2015.13.
[11]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[12]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014. pp. 4-9. DOI: 10.1109/MTV.2014.28.
[13]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014. pp. 98-101. DOI: 10.1109/MTV.2014.26.
[14]. RAVEN test program generator (http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc).
[15]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006. pp. 102-110.
[16]. Tatarnikov A.D. Language for Describing Templates for Test Program Generation for Microprocessors. Trudy ISP RAN / Proc. ISP RAS, volume 28, part 4, 2016. pp. 81-102. DOI: 10.15514/ISPRAS-2016-28(4)-5