Московский государственный университет имени М. В. Ломоносова

На правах рукописи

Корныхин Евгений Валерьевич

Построение тестовых программ для проверки подсистем управления памяти микропроцессоров

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва – 2010

Работа выполнена на кафедре системного программирования факультета вычислительной математики и кибернетики Московского государственного университета имени М. В. Ломоносова.

Научный руководитель:	*доктор физико-математических наук, старший научный сотрудник* ***Петренко Александр Константинович***.
Официальные оппоненты:	*доктор физико-математических наук, профессор* ***Смелянский Руслан Леонидович***; *доктор физико-математических наук* ***Лацис Алексей Оттович***.
Ведущая организация:	*Научно-исследовательский институт системных исследований РАН*

Защита состоится 26 ноября 2010 года в 11 часов на заседании диссертационного совета *Д 501.001.44 Московского государственного университета имени М.В. Ломоносова* по адресу: *119991, ГСП-1, Москва, Ленинские горы, МГУ имени М.В.Ломоносова, 2-й учебный корпус, факультет ВМК, ауд. 685.*

С диссертацией можно ознакомиться в библиотеке *факультета ВМК МГУ*. С текстом автореферата можно ознакомиться на официальном сайте ВМК МГУ http://cs.msu.ru в разделе «Наука» — «Работа диссертационных советов» — «Д 501.001.44»

Автореферат разослан «14» сентября 2010 г.

Ученый секретарь
диссертационного совета
профессор *Н.П. Трифонов*

Общая характеристика работы

Актуальность темы

Современные микропроцессоры — сложные многокомпонентные системы. Размеры современных микропроцессоров оцениваются как $10^7 - 10^8$ вентилей. Естественно при разработке таких сложных систем в проекты микропроцессоров вносятся ошибки, порой довольно критичные. Поэтому для обнаружения этих ошибок в цикл разработки микропроцессора в обязательном порядке входят этапы функциональной верификации.

Чем позднее будут обнаружены ошибки в микропроцессорах, тем дороже обойдётся исправление ошибок: сделать это в готовой микросхеме, тем более выпущенной на рынок, практически невозможно. Тем актуальнее становятся методы обнаружения ошибок на ранних этапах разработки микропроцессоров. Цикл разработки предполагает подготовку микропроцессоров в виде исполнимых программных моделей на языках Verilog или VHDL. Это делает возможным проведение функциональной верификации на таких моделях (т.е. до производства самих микропроцессоров) и актуальным исследование методов такой верификации. Целью функциональной верификации программных моделей микропроцессоров является обнаружение ошибок реализации функциональности в программных моделях микропроцессоров.

Выделяют следующие виды функциональной верификации: экспертизу, имитационное тестирование и формальную верификацию. Экспертиза предполагает анализ текстов моделей экспертами с целью оценки их корректности и обнаружения ошибок. Этот вид функциональной верификации эффективно применяется на ранних стадиях разработки. Однако, ввиду наличия человеческого фактора, после экспертизы ошибки в микропроцессоре всё же остаются. Методы формальной верификации позволяют дать исчерпывающий ответ на вопрос о корректности отдельных модулей и всего микропроцессора. Однако трудоемкость формальной верификации чрезвычайно велика. Например, при разработке Intel Pentium 4 были формально верифицированы модуль работы с плавающей точкой (FPU), модуль декодирования инструкции и логика вне-

очередного выполнения (out-of-order), было найдено порядка 20 новых ошибок, однако трудоемкость этого проекта составила порядка 60 человеко-лет.

Имитационное тестирование позволяет ценой меньших усилий обнаружить значительную часть ошибок, в том числе критичных ошибок. Имитационное тестирование проводят для отдельных модулей (тогда оно называется *модульным тестированием*) и для всего микропроцессора в целом (тогда оно называется *системным тестированием*). Модули тестируются подачей на их входы специальных сигналов (*модульных тестов*) со снятием выходных сигналов и последующим анализом выходных сигналов. Входом при системном тестировании являются программы на машинном языке (*тестовые программы*). Проведение модульного тестирования требует кроме подготовки самих входных данных еще и подготовку тестирующей установки (testbench), выделение тестируемого модуля из всего проекта микропроцессора и т.п. Системное тестирование избавлено от этой необходимости. Поскольку размер и сложность отдельного модуля всегда меньше размера и сложности микропроцессора в целом, потенциально качество модульного тестирования может быть выше, чем системного. Однако для достижения высокого качества тестирования как число модульных тестов, так и совокупная трудоемкость их изготовления, получаются очень большими. Это вынуждает часть проверок проводить на модульном уровне, а другую часть на системном. Невысокая стоимость подготовки и проведения системного тестирования определила его наибольшую востребованность среди других методов функциональной верификации. Практически все разработчики микропроцессоров проводят системное тестирование.

Ключевым вопросом, определяющим качество тестирования, является вопрос выбора тестовых программ. Поскольку современные микропроцессоры обладают множеством инструкций (порядка сотен), длины конвейеров имеют порядок десятка стадий, количество различных состояний и ситуаций, в которых надо протестировать микропроцессор, измеряется десятками тысяч. Поэтому для тщательного системного тестирования нужно подобное же и количество тестовых программ. Это определяет актуальность задачи автома-

тического построения тестовых программ для системного тестирования.

Сложность микропроцессоров растет (увеличивается количество функциональных требований, количество ситуаций, в которых поведение микропроцессора должно обладать заданной спецификой). Это требует тестовых программ для проверки функциональных требований, которые не проверяются имеющимися тестовыми программами, и делает актуальными дальнейшие исследования в области построения тестов.

К числу наиболее сложных механизмов современных процессоров (поэтому наиболее подверженных ошибкам), использующих конвейеры и многоуровневые буферы типа кэш-памяти, относится механизм доступа к памяти. Поэтому актуальной является задача построения тестовых программ для проверки подсистем управления (механизмами) памяти микропроцессоров.

Цель диссертационной работы

Целью диссертационной работы является исследование и разработка методов и программных средств построения тестовых программ для проверки подсистем управления памяти микропроцессоров.

Для достижения этой цели были поставлены следующие задачи:

1) исследовать описанные в научной литературе методы построения тестовых программ на предмет их применимости для системного тестирования подсистем управления памяти микропроцессоров;

2) разработать методы построения тестовых программ для системного функционального тестирования подсистем управления памяти.

Научная новизна

Научной новизной обладают следующие результаты работы:

1) предложен подход к построению тестовых программ для проверки подсистем управления памяти микропроцессоров, сочетающий формализацию документации и технику ограничений;

2) предложен метод моделирования устройств подсистемы управления памяти, использующий конечные автоматы специального вида;

3) предложена формальная модель инструкций, описывающая отдельные пути их выполнения в виде утверждений о свойствах параметров инструкций и модельном состоянии устройств;

4) в рамках предложенного подхода разработан метод формализации механизма вытеснения данных при помощи построения ограничений, эффективно разрешаемых современным инструментарием.

Практическая значимость

Разработанные модели и методы могут быть использованы коллективами, занимающимися разработкой микропроцессоров, для автоматизации построения тестовых программ. Разработанный прототип системы построения тестовых программ использовался для генерации тестов подсистем управления памяти ряда микропроцессоров архитектуры MIPS64. Результаты работы могут быть использованы в исследованиях, которые ведутся в Институте системного программирования РАН, Московском государственном институте электроники и математики, НИИ системных исследований РАН, Институте точной механики и вычислительной техники им. С.А. Лебедева РАН, Институте проблем информатики РАН и других научных и промышленных организациях.

Апробация работы и публикации

По материалам диссертации опубликовано одиннадцать работ [1–11], в том числе одна [4] в издании, входящем в перечень ведущих рецензируемых научных журналов и изданий ВАК. Основные положения докладывались на следующих конференциях и семинарах:

1) на втором и третьем весеннем коллоквиуме молодых исследователей в области программной инженерии (SYRCoSE) (2008 и 2009 гг.);

2) на шестнадцатой и семнадцатой международной конференции студентов, аспирантов и молодых ученых «Ломоносов» (2009 и 2010 гг.);

3) на шестнадцатой всероссийской межвузовской научно-технической конференции студентов и аспирантов «Микроэлектроника и информатика - 2009» (2009 г.);

4) на седьмом международном симпозиуме по проектированию и тестированию под эгидой IEEE (EWDTS) (2009 г.);

5) на российско-ирландской летней школе по научным вычислениям (2009 г.);

6) на научной конференции «Тихоновские чтения» (2009 г.);

7) на научной конференции «Ломоносовские чтения» (2010 г.);

8) на объединенном научно-исследовательском семинаре имени М.Р. Шура-Бура (2010 г.);

9) на семинаре Лаборатории вычислительных комплексов факультета вычислительной математики и кибернетики МГУ имени М.В.Ломоносова (2010 г.);

10) на семинаре отдела Технологий программирования института системного программирования РАН (2009, 2010 гг.).

Структура и объем диссертации

Работа состоит из введения, трех глав, заключения, списка литературы и приложений. Общий объем основной части диссертации составляет 125 страниц. Список литературы содержит 71 наименование.

Содержание работы

Во Введении обоснована актуальность диссертационной работы, сформулирована цель и задачи исследований, сформулированы полученные результаты и показана их практическая значимость.

Первая глава содержит обзор существующих методов построения тестовых программ, обзор подсистем управления памяти и уточнение задач исследования по результатам этих обзоров.

В разделе 1.1 дается схема системного тестирования микропроцессора, описываются его отдельные этапы, в том числе и этап построения тестовых программ. Рассматриваются методы псевдослучайного и целенаправленного автоматического построения тестовых программ. В разделе 1.2 кратко описываются функции и типичный состав подсистем управления памяти, способы повышения их эффективности и классы ошибок. В разделе 1.3 сделан обзор методов целенаправленного построения тестовых программ. Выделены методы на основе массовой генерации тестовых программ и методы непосредственного построения тестовых программ. В разделе 1.4 сделан анализ методов целенаправленного построения тестовых программ по применимости этих методов для тестирования подсистем управления памяти, масштабируемости, возможности «нацеливания» на функциональность. Наиболее перспективными являются методы непосредственного построения тестовых программ по тестовым ситуациям, формализованным в виде *тестовых шаблонов* — цепочек инструкций с указанием вариантов исполнения инструкций — и методы, включающие разрешение ограничений (constraint satisfaction). В разделе 1.5 уточнены задачи в соответствии с проведенным исследованием методов построения тестовых программ.

Во второй главе предложен подход целенаправленного построения нацеленных тестовых программ, сочетающий особенности перспективных методов построения тестовых программ. В разделе 2.1 описаны этапы построения те-

стовых программ согласно предлагаемому подходу: чтение документации по архитектуре, формализация архитектуры, выделение и формализация тестовых ситуаций (в виде шаблонов тестовых программ, далее *тестовых шаблонов*), построение и решение системы ограничений для каждого тестового шаблона, конструирование текста тестовой программы на основе решения каждой совместной системы ограничений.

Тестовый шаблон содержит последовательность инструкций, для каждой из которых указан вариант исполнения (т.е. некоторый путь выполнения инструкции). Для алгоритмического построения ограничений и затем тестовых программ варианта инструкций должны быть формализованы в рамках этапа формализации микропроцессора. Раздел 2.2 более детально освещает этот этап . В этом разделе предложены модель функционирования устройств подсистемы управления памяти и модель вариантов инструкций. В рамках этого этапа следует выделить и формализовать те варианты инструкций, которые входят в тестовый шаблон, и задействованные в них устройства подсистемы управления памяти. Моделью состояния устройства подсистемы управления памяти (кэш-памяти, таблицы страниц и т.п.) предлагается последовательность ассоциативных массивов (далее эта последовательность будет называться *таблицей*, а отдельный ассоциативный массив — *регионом*). Каждый регион состоит из *строк*, каждая строка состоит из набора *полей*, поля делятся на *поля ключа* и *поля данных*. Поля ключа задают ключи в ассоциативном массиве, поля данных — значения. В модели устройства определены следующие операции: успешного обращения, успешного обращения с изменением, неуспешного обращения с замещением. На входе операции успешного обращения — выражение, задающие ключ, и выражение, задающее номер ассоциативного массива из состояния устройства. Операция определена на тех входных данных и состоянии модели, при которых в соответствующем ассоциативном массиве есть строка, поля ключей которой *соответствуют* аргументу-ключу этой операции. Операция возвращает поля данных соответствующей аргументу-ключу строки. Операция успешного обращения с изменением отличается от операции успешного обращения дополнительным аргу-

ментом — полями данных, которые нужно заменить в строке, соответствующей аргументу-ключу. На входе операции неуспешного обращения с замещением 3 аргумента: выражение, задающие ключ, выражение, задающее номер ассоциативного массива, и выражения для полей данных строки. Операция определена на тех входных данных и состоянии модели, при которых в соответствующем ассоциативном массиве нет строки, поля ключей которой *соответствуют* аргументу-ключу этой операции. Эффект операции заключается в замене полей данных одной из строк в ассоциативном массиве, номер которого был передан в качестве одного из аргументов операции, на переданные операции поля данных. Выбор строки для замены определяется на основе стратегии вытеснения так же, как это делается в устройствах подсистемы управления памяти (например, в кэш-памяти). В рамках этапа формализации следует указать набор атрибутов, задающих модели устройств тестируемой подсистемы управления памяти: стратегия вытеснения, количество строк в массиве («ассоциативность»), двоичный логарифм количества массивов, имена и битовые длины полей ключа и полей данных, предикат соответствия строки некоторой битовой строке (эта строка является аргументом-ключом в операциях над моделью). Алгоритмы построения систем ограничений параметризованы этими атрибутами. Таким образом моделируются кэш-память различных уровней, таблицы страниц, буферы трансляции адресов (TLB) и даже оперативная память.

В том же разделе предложены модели вариантов инструкций. Варианты инструкций соответствуют отдельным путям выполнения инструкций. Модель варианта инструкции формализует следующие виды требований: допустимые значения операндов инструкции, вычисление значений операндов-результатов инструкции, условия возникновения исключительных ситуаций (если вариант инструкции состоит в возникновении такой ситуации), вычисление адресов (физических, виртуальных, эффективных) в инструкции, обращения в устройства подсистемы управления памяти и успешность этих обращений, какие данные загружаются или сохраняются. При формализации для варианта инструкции объявляются операнды инструкции и последова-

тельность операторов 4-х видов: оператор утверждения истинности свойства над битовыми строками (`assume`), оператор введения новой переменной (`let`), оператор попадания (`hit`), оператор промаха (`miss`). С помощью этой последовательности операторов задается набор условий на значения операндов инструкции и для моделей устройств, задействованных в этом варианте инструкции, условия на их состояние и изменение этого состояния в рамках данного варианта. Операторы `let` и `assume` имеют ту же семантику, которая используется при формализации императивных языков программирования. Операторы `hit` и `miss` специфичны инструкциям, оперирующим с памятью. Оператор попадания `hit<B>(k;R){loaded(d); [storing(d');] }`, где `k, R, d, d'` — выражения над определенными ранее переменными-битовыми строками, означает, что в данном варианте инструкции должно осуществляться успешное обращение в устройство `B` по ключу `k` в массив с номером `R`, причем ключу соответствуют данные `d` (если полей данных несколько, то соответствующие выражения для них перечисляются в скобках у `loaded`). Если указана секция `storing`, то в варианте инструкции должно осуществляться успешное обращение с изменением на поля данных `d'`. Оператор промаха `miss<B>(k;R){[replacing(d);]}`, где `k, R, d` — выражения над определенными ранее переменными-битовыми строками, означает, что в данном варианте инструкции должно быть неуспешное обращение в устройство `B` по ключу `k` в массив с номером `R`. Секция `replacing` задает поля данных `d'` вытесняющей строки. Если секция `replacing` отсутствует, изменение состояния устройства `B` не должно происходить.

Формализовав устройства подсистемы управления памяти и инструкции и составив тестовый шаблон, нужно определить те значения регистров и хранящиеся данные устройств, при которых исполнение инструкций, указанных в тестовом шаблоне, будет проходить согласно указанным там вариантам. Для этого составляется система ограничений, выражающая все необходимые условия на такие значения. В результате разрешения этой системы определяются искомые значения. В разделе 2.3 описывается предлагаемый алгоритм построения ограничений для тестового шаблона. Исполнение инструкций кон-

вейеризовано, поэтому расположенные рядом инструкции в действительности будут выполняться с существенной долей параллелизма. Однако в алгоритме генерации ограничений считается, что инструкции выполняются последовательно, а тестовые шаблоны составлены таким образом, чтобы при работе соответствующих им тестовых программ проявились все нужные параллельные эффекты.

По каждому оператору алгоритм строит свою часть ограничений, которые выражают семантику этого оператора. Операторы обращений в устройства транслируются в ограничения без моделирования состояний устройств, несмотря на то, что определение этих операторов включало состояние устройства. Это позволяет существенно уменьшить количество переменных-битовых строк и размер ограничений и, тем самым, ускорить разрешение ограничений. Специальное представление выбрано и для начального содержимого устройств, а именно, последовательность обращений в это устройство. Поэтому в число переменных в ограничениях входят переменные, задающие аргументы этих, *инициализирующих*, обращений: ключи, номера ассоциативных массивов. Для операторов `hit` и `miss` строятся ограничения на аргументы-ключи `k` и номера регионов `R` (эти ограничения должны гарантировать успешное обращение для `hit` и неуспешное — для `miss`) и ограничения на аргументы-данные `d` и `d'` (обращения по одинаковым адресам должны давать одинаковые данные, если они не были изменены). Ограничения на аргументы-ключи и аргументы-номера регионов строятся согласно следующим определениям операторов: `hit(`$\mathtt{k}_i$`;`$\mathtt{R}_i$`)` / `miss(`$\mathtt{k}_i$`;`$\mathtt{R}_i$`)` происходит, если

(α) перед ним есть обращение по тому же ключу ($\mathtt{k}_i$) в тот же регион ($\mathtt{R}_i$),

(β) после которого и до этого обращения соответствующая строка не была вытеснена / была вытеснена из таблицы.

Трансляция свойства α достаточно очевидна, трансляция свойства β рассматривается в разделе 2.5.

В этом же разделе диссертации сформулированы и доказаны теоремы корректности и полноты алгоритмов, гарантирующие, что ограничений, постро-

енные согласно алгоритму, не дают решений, не соответствующих тестовому шаблону, и задают все решения, соответствующие тестовому шаблону. Значение атрибута модели устройства, задающего стратегию вытеснения, равное `none`, означает, что при неуспешном обращении в это устройство замещение не производится. Здесь и далее символами $t_1, t_2, ..., t_m$ обозначаются аргументы-ключи инициализирующих обращений, $r_1, r_2, ..., r_m$ — аргументы-регионы инициализирующих обращений, $k_1, k_2, ..., k_n$ — аргументы-ключи обращений в инструкциях тестового шаблона, $R_1, R_2, ..., R_n$ — аргументы-регионы обращений в инструкциях тестового шаблона, $S_1, S_2, ..., S_n$ — успешности обращений в инструкциях тестового шаблона ($S_i = hit$ или $S_i = miss$).

Теорема 1 (Корректность алгоритма генерации ограничений на ключи обращений для таблицы, стратегия вытеснения которой не `none`). *Если система ограничений, построенная для последовательности обращений к таблице* (S_1, k_1, R_1), (S_2, k_2, R_2), ..., (S_n, k_n, R_n) *с дополнительным предикатом* $P(k_1, k_2, ..., k_n, R_1, R_2, ..., R_n)$, *является совместной, то ее решение* $t_1, t_2, ..., t_m$, r_1, r_2, ..., r_m, k_1, k_2, ..., k_n, $R_1, R_2, ..., R_n$, *удовлетворяет последовательности* $S_1, ..., S_n$ *и* P *при любом начальном состоянии таблицы.*

Теорема 2 (Полнота алгоритма генерации ограничений на ключи обращений для таблицы, стратегия вытеснения которой есть `none`). *Фиксируем некоторое начальное состояние* L *таблицы со стратегией вытеснения* `none`. *Если при нем для последовательности обращений к таблице* (S_1, k_1, R_1), (S_2, k_2, R_2), ..., (S_n, k_n, R_n) *с дополнительным предикатом* $P(k_1, k_2, ..., k_n, R_1, R_2, ..., R_n)$ *существуют значения переменных* k_1, k_2, ..., k_n *и* R_1, $R_2, ..., R_n$, *которые удовлетворяют последовательности* S_1, S_2, ..., S_n *и* P, *то система ограничений, построенная согласно алгоритму генерации ограничений на ключи обращений для таблицы со стратегией вытеснения* none, *будет совместной.*

Стратегию вытеснения будем называть *существенно вытесняющей*, если w промахов в один регион полностью вытесняют его предыдущее содержимое (w — значение параметра `lines` таблицы).

Теорема 3 (Полнота алгоритма генерации ограничений на ключи обращений для таблицы, стратегия вытеснения которой не `none` и является **существенно вытесняющей**). *Фиксируем некоторое начальное состояние L таблицы со стратегией вытеснения не* `none`*. Если при нем для последовательности обращений к таблице (S_1, k_1, R_1), (S_2, k_2, R_2), ..., (S_n, k_n, R_n) с дополнительным предикатом $P(k_1,\ k_2, ..., k_n, R_1,\ R_2, ..., R_n)$ существуют значения переменных $k_1,\ k_2,\ ...,\ k_n$ и $R_1,\ R_2, ..., R_n$, которые удовлетворяют последовательности $S_1, S_2, ..., S_n$ и P, то система ограничений, построенная согласно алгоритму генерации ограничений на ключи обращений для таблицы со стратегией вытеснения не none, будет совместной, если стратегия вытеснения является «существенно вытесняющей».*

В разделе 2.3.2 рассматриваются *таблицы вытеснения*, которые позволяют формализовать стратегии вытеснения. Таблицы вытеснения были предложены в 2008 году исследователями из Университета Саарланда. С использованием таблиц вытеснения в разделе 2.3.3 сформулированы и доказаны теоремы о том, что стратегии вытеснения LRU, FIFO и Pseudo-LRU являются существенно вытесняющими:

Теорема 4. *Стратегия вытеснения LRU является существенно вытесняющей.*

Теорема 5. *Стратегия вытеснения FIFO является существенно вытесняющей.*

Теорема 6. *Стратегия вытеснения Pseudo-LRU является существенно вытесняющей.*

В этом же разделе исследуется вопрос длины инициализирующей последовательности. Сформулирована и доказана верхняя оценка достаточного количества инициализирующих обращений в устройство подсистемы управления памяти:

Утверждение 1 (Верхняя оценка количества инициализирующих обращений)**.**

$$m \leqslant n \cdot (n + 2w)$$

где w — значение параметра `lines` *таблицы, n — количество обращений в таблицу в шаблоне.*

Для стратегии вытеснения LRU в этом же разделе сформулирована и доказана более сильная оценка количества инициализирующих обращений:

Теорема 7 (Верхняя оценка количества инициализирующих обращений для стратегии вытеснения LRU)**.** *Фиксируем некоторое начальное состояние L таблицы со стратегией вытеснения LRU. Если при нем для последовательности обращений к таблице (S_1, k_1, R_1), (S_2, k_2, R_2), ..., (S_n, k_n, R_n) с дополнительным предикатом $P(k_1,\ k_2, ..., k_n, R_1,\ R_2, ..., R_n)$ существуют значения переменных $k_1, k_2, ..., k_n$ и $R_1,\ R_2, ..., R_n$, которые удовлетворяют последовательности $S_1, S_2, ..., S_n$ и P, то система ограничений, построенная согласно алгоритму генерации ограничений на ключи обращений будет совместной для любого $m > n \cdot w + M$, где M – количество элементов последовательности $S_1, S_2, ..., S_n$, равных miss.*

В разделе 2.4 предложено новое определение стратегии вытеснения Pseudo-LRU. Обычно при определении Pseudo-LRU предлагают рассматривать для региона упорядоченное бинарное дерево с пометками в нелистовых вершинах, листовым вершинам дерева соответствуют строки региона. Как и прежде, обращения осуществляются к одной из листовых вершин дерева, что приводит к изменению пометок вершин пути в неё из корневой вершины. Вытесняемая строка определяется также на основе пометок вершин дерева. Новое определение задает стратегию вытеснения Pseudo-LRU как изменение битовых векторов (*Pseudo-LRU-ветвей*), сопоставленных строкам региона. Это определение позволит сократить количество ограничений и ускорить их разрешение. Сформулирована и доказана теорема, задающее изменение Pseudo-LRU-ветвей и показывающая эквивалентность нового определения Pseudo-LRU старому ($W = \log_2 w$, для стратегии вытеснения Pseudo-LRU допустимы лишь

те w, которые являются степенями двойки, *абсолютная позиция*(или просто, *позиция*) — это номер строки в регионе, *относительная позиция* позиции i относительно позиции j будем назыаать $i \oplus j$ и обозначать как π_j^i):

Теорема 8 (Инвариантность преобразования Pseudo-LRU-ветвей относительными позициями)**.** *Пусть* $(\alpha_1\ \alpha_2\ \ldots\ \alpha_W)$ *— Pseudo-LRU-ветвь некоторой позиции* i*. Тогда изменение этой ветви согласно стратегии вытеснения Pseudo-LRU определяется только относительной позицией (относительно* i*) и происходит следующим образом при обращении к ключу с (абсолютной) позицией* j*: если* $\pi_j^i \in [\frac{w}{2^k},\ \frac{w}{2^{k-1}})$ *для некоторого* $k = 1, 2, \ldots, W$*, то происходит изменение* $\alpha_1 := 0,\ \alpha_2 := 0,\ \ldots,\ \alpha_{k-1} := 0,\ \alpha_k := 1$*; если* $\pi_j^i = 0$*, то происходит изменение* $\alpha_1 := 0,\ \alpha_2 := 0,\ \ldots,\ \alpha_W := 0$*; вытеснение ключа на позиции* i *происходит в том случае, когда* $\alpha_1 = 1 \wedge \alpha_2 = 1 \wedge \ldots \wedge \alpha_W = 1$*.*

В разделе 2.5 предложен метод построения ограничений для свойства «быть вытесненным к моменту заданного обращения в таблицу» — *метод полезных обращений*. С помощью него это свойство выражается в виде ограничений в операциях над битовыми строками и целочисленной линейной арифметике. В разделе предложена формализация понятия *полезной для вытеснения* (или просто, *полезной*) инструкции. *Формулой полезного обращения* будем называть предикат, истинный для всех обращений, являющихся полезными, и ложный на всех обращениях, не являющихся полезными. В рамках метода полезных обращений свойство «быть вытесненным» рассматривается как ограничение снизу количества предыдущих обращений, являющихся полезными.

В разделе 2.5.1 предложено и формально обосновано представление свойства «быть вытесненным» для стратегии вытеснения LRU в виде ограничений, составленное по методу полезных обращений («||» — операция битовой конкатенации, выражение $[\phi]$ равно 1, если ϕ истинно, и равно 0 в противном случае):

Теорема 9 (Выражение свойства «быть вытесненным» для LRU)**.** *Пусть* (t_1, r_1)*,* (t_2, r_2)*, ...,* (t_m, r_m) *– ключи и регионы инициализирующих обраще-*

ний, (k_i, R_i) *– ключ и регион обращения, для которого описывается вытеснение (будем его называть «вытесняемым»), причем* $(k_i||R_i) \in \{(t_1||r_1)$, ..., $(t_m||r_m), (k_1||R_1)$, ..., $(k_{i-1}||R_{i-1})\}$ *и* $\{(t_1||r_1), ..., (t_m||r_m)\}$ *— все разные. Тогда* k_i *не вытеснен из региона* R_i *согласно определению LRU на перестановках тогда и только тогда, когда*

$$\sum_{j=1}^{m+n} [u_{k_i,R_i}(s_j)] < w$$

где последовательность $s \equiv \langle (t_1||r_1), ..., (t_m||r_m), (k_1||R_1), ..., (k_n||R_n)\rangle$,
$R(s_i)$ *— вторая компонента* s_i *(регион), а формула полезного обращения такая:*

$$u_{k_i,R_i}(s_j) \equiv ((k_i||R_i) \notin \{s_j, ..., s_{m+n}\} \wedge R_i = R(s_j) \wedge s_j \notin \{s_{j+1}, ..., s_{m+n}\})$$

В разделе 2.5.2 предложено и формально обосновано представление свойства «быть вытесненным» для стратегии вытеснения FIFO в виде ограничений, составленное по методу полезных обращений.

В разделе 2.5.3 предложено и формально обосновано представление свойства «быть вытесненным» для стратегии вытеснения Pseudo-LRU в виде ограничений, составленное по методу полезных обращений:

Теорема 10 (Выражение свойства «быть вытесненным» для Pseudo-LRU)**.** *Пусть* (t_1, r_1), (t_2, r_2), ..., (t_m, r_m) *– ключи и регионы инициализирующих обращений, а* (k_i, R_i) *– ключ и регион обращения, для которого описывается вытеснение (будем его называть «вытесняемым»), причем* $(k_i||R_i) \in \{(t_1||r_1)$, ..., $(t_m||r_m)$, $(k_1||R_1)$, ..., $(k_{i-1}||R_{i-1})\}$ *и* $\{(t_1||r_1), ..., (t_m||r_m)\}$ *— все разные. Тогда* k *не вытеснен из региона* R *согласно трактовке Pseudo-LRU в терминах ветвей бинарного дерева тогда и только тогда, когда*

$$\sum_{j=1}^{m+n} [u_{k_i,R_i,\pi_i'}(s_j)] < W$$

где последовательность $s \equiv \langle (t_1||r_1), ..., (t_m||r_m), (k_1||R_1), ..., (k_n||R_n)\rangle$,
$R(s_i)$ *— вторая компонента* s_i *(регион), а формула полезного обращения*

такая:

$$u_{k_i,R_i,\pi'_i}(s_j) \equiv \begin{cases} (k_i||R_i) \notin \{s_j, s_{j+1}, ..., s_{m+n}\} \\ R_i = R(s_j) \\ (\pi'_i||R_i) \in \{(\pi_j||R_j), (\pi_{j+1}||R_{j+1}), ..., (\pi_{m+n}||R_{m+n})\} \\ \bigwedge\limits_{k=j+1}^{m+n} (R_j = R_k \ \wedge \ (\pi'_i||R_i) \notin \{(\pi_j||R_j), (\pi_{j+1}||R_{j+1}), ..., (\pi_k||R_k)\} \\ \qquad\qquad \rightarrow P(\pi_j \oplus \pi'_i, \pi_k \oplus \pi'_i)) \end{cases}$$

$$P(x, y) \equiv (y > x \quad \wedge \quad x \oplus y > x)$$

π'_i определено следующим образом: если $S_i = hit$, то

$$\begin{aligned} &(ite \ ((k_i||R_i) = (k_{i-1}||R_{i-1})) \ \ (\pi'_i = \pi_{i-1}) \\ &(ite \ ((k_i||R_i) = (k_{i-2}||R_{i-2})) \ \ (\pi'_i = \pi_{i-2}) \\ &... \ (\pi'_i = 0)))) \ldots) \end{aligned}$$

если $S_i = miss$, то ($\{\pi_i, ..., \pi_j\}_m$ обозначает подмножество множества позиций от i'го до j'го обращения из неуспешных обращений):

$$\begin{aligned} &(ite \ ((k_i||R_i) = (k_{i-1}||R_{i-1})) \ \ (\pi'_i = \pi_{i-1}) \\ &(ite \ ((k_i||R_i) = (k_{i-2}||R_{i-2})) \ \ (\pi'_i = \pi_{i-2} \wedge \pi'_i \in \{\pi_{i-1}\}_m) \\ &(ite \ ((k_i||R_i) = (k_{i-3}||R_{i-3})) \ \ (\pi'_i = \pi_{i-3} \wedge \pi'_i \in \{\pi_{i-1}, \pi_{i-2}\}_m) \\ &... \ (\pi'_i = 0)))) \ldots) \end{aligned}$$

π_i — позиции — дополнительные переменные, для которых надо построить ограничения: конъюнкцию для каждой пары (s_i, π_i) и (s_j, π_j) при $j > i$ ограничений:

- *если j'е обращение успешное, то* $(\pi_i||R(s_i) = \pi_j||R(s_j) \wedge$

$$\pi_i||R(s_i) \notin \{\pi_{m_1}||R(s_{m_1}), \pi_{m_2}||R(s_{m_2}), \ldots, \pi_{m_n}||R(s_{m_n})\}) \rightarrow s_i = s_j$$

- *в противном случае* $(\pi_i||R(s_i) = \pi_j||R(s_j) \wedge$

$$\pi_i||R(s_i) \notin \{\pi_{m_1}||R(s_{m_1}), \pi_{m_2}||R(s_{m_2}), \ldots, \pi_{m_n}||R(s_{m_n})\}) \rightarrow s_i \neq s_j$$

где $(\pi_{m_1}, R(s_{m_1})), (\pi_{m_2}, R(s_{m_2})), \ldots, (\pi_{m_n}, R(s_{m_n}))$ — позиции и регионы неуспешных обращений, расположенных между i'м и j'м.

В разделе 2.5.4 освещаются некоторые вопросы сложности разрешения ограничений, которые строятся согласно методу полезных обращений.

В разделе 2.6 рассматривается последний этап построения тестовой программы — этап конструирования текста программы. Текст тестовой программы состоит из инициализирующей части и инструкций тестового шаблона. Инициализирующая часть состоит из инструкций для помещения вычисленных на предыдущем этапе начальных значений в соответствующие регистры и последовательности инструкций, обращающихся в соответствующее устройство подсистемы управления памяти по вычисленным ключам в вычисленные регистры с вычисленными данными. Зачастую конструирование таких инструкций не представляет сложности. Пример противоположного случая — инициализация многоуровневой кэш-памяти в случае, если инициализацию отдельных уровней кэш-памяти нельзя осуществить отдельно от остальных уровней. Поэтому в разделе 2.6 предложен и метод конструирования последовательности инициализирующих инструкций, учитывающий особенности многоуровневой кэш-памяти.

В третьей главе дается оценка предлагаемым методам. В разделах 3.1, 3.2 и 3.3 показывается, что подсистемы управления памяти микропроцессоров архитектур MIPS, PowerPC и IA-32 можно представить в виде набора таблиц и формализовать варианты инструкций, оперирующих с памятью. Причем имеется документация по каждой архитектуре, где эти варианты уже описаны на соответствующем псевдокоде. В разделе 3.4 описана автоматизация некоторых этапов предлагаемого во второй главе подхода: генератор ограничений, решатель ограничений и конструктор текстов тестовых программ. Решатель ограничений — это внешний инструмент (Z3) и разрабатывать его для автоматизации подхода не надо, что сильно сокращает трудоемкость автоматизации подхода. Затронут вопрос переиспользования перечисленных компонентов при смене микропроцессора. Был реализован прототип генератора тестовых программ для архитектуры MIPS64. В разделе 3.5 описаны эксперименты над этим прототипом по оценке времени построения тестовых программ

и вероятности завершения построения за 60 секунд. Эксперименты показали увеличение допустимого размера тестовых шаблонов до 9-12 инструкций (по сравнению с 2-3 инструкциями в доступных инструментах). В разделе 3.6 проведено сравнение с инструментом Genesys-Pro (IBM): выделены сходства и отличия, преимущества и недостатки предложенных методов по сравнению с Genesys-Pro. В разделе 3.7 проведено сравнение с работами, проводящимися в Intel.

В Заключении диссертационной работы перечисляются ее основные результаты.

Основные результаты работы

1) Предложен подход к построению тестовых программ для проверки подсистем управления памяти микропроцессоров, позволяющий понизить сложность построения некоторого класса тестовых программ. Этот класс состоит из тестовых программ, в которых имеется цепочка длиной от 6 до 12 инструкций обращения к памяти. Понижение сложности обосновано при помощи ряда экспериментов на прототипе программного средства построения тестовых программ. Теоретически обоснована корректность алгоритмов в рамках предложенного подхода к построению тестовых программ. В рамках предложенного подхода разработаны модели устройств подсистемы управления памяти и модели вариантов исполнения инструкций.

2) В рамках предложенного подхода разработан метод формализации механизма вытеснения данных, позволяющий в отличие от других методов моделирования выразить ряд свойств вытесняемых данных в виде ограничений, причем эти ограничения эффективно разрешаются современным инструментарием. Теоретически обоснована корректность метода формализации для ряда стратегий вытеснения. Показано применение

метода формализации к нескольким важным на практике стратегиям вытеснения.

Публикации по теме диссертации

[1] *Корныхин, Е. В.* Система генерации тестовых программ с использованием ограничений ТЕСЛА / Е. В. Корныхин // *Сборник тезисов конференции Ломоносов.* — 2009. — С. 39.

[2] *Корныхин, Е. В.* Генерация тестовых данных для тестирования арифметических операций центральных процессоров / Е. В. Корныхин // *Труды Института Системного Программирования.* — 2008. — Т. 15. — С. 107–117.

[3] *Корныхин, Е. В.* Система генерации тестовых данных для системного функционального тестирования микропроцессоров ТЕСЛА / Е. В. Корныхин // *Сборник тезисов конференции «Микроэлектроника и информатика».* — 2009. — С. 87.

[4] *Корныхин, Е. В.* Генерация тестовых данных для тестирования механизмов кэширования и трансляции адресов микропроцессоров / Е. В. Корныхин // *Программирование.* — 2010. — Т. 36, № 1. — С. 40–49.

[5] *Корныхин, Е. В.* Метод зеркальной генерации ограничений для построения тестовых программ по тестовым шаблонам / Е. В. Корныхин // *Труды Института Системного Программирования.* — 2010. — Т. 18. — С. 67–80.

[6] *Корныхин, Е. В.* Генерация тестовых данных для системного функционального тестирования микропроцессоров с учетом кэширования и трансляции адресов / Е. В. Корныхин // *Труды Института Системного Программирования.* — 2009. — Т. 17. — С. 145–160.

[7] *Корныхин, Е. В.* Генерация тестовых данных для системного функционального тестирования fifo-кэш-памяти микропроцессоров / Е. В. Корныхин // *Вычислительные методы и программирование.* — 2009. — Т. 10. — С. 107–116.

[8] *Корныхин, Е. В.* Определение стратегии вытеснения pseudolru на ветвях бинарного дерева / Е. В. Корныхин // *Сборник тезисов конференции Ломоносов.* — 2010. — С. 20–21.

[9] *Kornikhin, E.* Smt-based test program generation for cache-memory testing / E. Kornikhin // *Proceedings of 7^{th} East-West Design & Test Symposium.* — 2009. — Pp. 124–127.

[10] *Kornikhin, E.* Test data generation for lru cache-memory testing / E. Kornikhin // *Proceedings of Spring Young Researchers' Colloquium on Software Engineering.* — 2009. — Pp. 88–92.

[11] *Kornikhin, E.* Test data generation for arithmetic subsystem of cpus mips64 / E. Kornikhin // *Proceedings of Spring Young Researchers' Colloquium on Software Engineering.* — 2008. — Vol. 2. — Pp. 43–46.