МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М. В. ЛОМОНОСОВА

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И КИБЕРНЕТИКИ

На правах рукописи

Корныхин Евгений Валерьевич

Исследование методов генерации программ для тестирования модулей управления памяти микропроцессоров

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

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
д.ф-м.н. Петренко Александр Константинович

Москва – 2009

Оглавление

Введение

Актуальность

Современные программные и аппаратные системы зачастую обладают сложным внутренним устройством и логикой работы. Для повышения эффективности работы этих систем применяются в том числе и механизмы кэширования. В микропроцессорах используется иерархическая кэш-память, в операционных системах используется аппарат виртуальной памяти, включающий в себя кэширование и управления страницами виртуальной памяти, системы управления базами данных используют кэширование для ускорения выдачи данных, удовлетворяющих запросам, кэширование применяется и для оптимизации работы в компьютерных сетях. С увеличением сложности таких систем возрастает вероятность ошибки при их реализации.

Для обнаружения ошибок применяются различные методы, такие как статический анализ, тестирование, мониторинг, формальная верификация, верификация на моделях и прочие. Тестирование интерфейса системы осуществляется подачей специальных *тестовых воздействий*, наблюдение за работой системы или, как минимум, фиксация реакции системы на тестовое воздействие, наконец осуществляется вынесение вердикта о соответствии реакции системы требованиям, предъявляемым к системе. Однако для тестирования сложных современных систем зачастую требуются сложные тестовые воздействия, построение которых вручную является нетривиальной и кропотливой задачей. Это повышает актуальность задачи автоматизации построения тестовых воздействий.

Для детерминированных систем требования на тестовые воздействия могут быть сформулированы формально (в виде *тестовых шаблонов*). Однако такие тестовые шаблоны характеризуются большим количеством отношений (ограничений) на элементы отдельных стимулов системы, входящих в него. При этом эффективность применения вероятностных алгоритмов построения тестовых воздействий падает. Тем не менее современные алгоритмы позволяют выражать отношения на объектах и строить эффективные *процедуры разрешения* этих отношений с целью построения объектов, на которых заданные отношения выполнены. Отношения (*ограничения*) могут быть автоматически получены из тестовых шаблонов, а результатом процедуры разрешения станет искомое тестовое воздействие. Представляется перспективным исследование методов автоматической генерации тестовых воздействий по тестовым шаблонам с использованием ограничений.

Однако задача построения эффективных процедур разрешения систем ограничений в общем случае является нетривиальной задачей. Для некоторых классов ограничений исследованы и построены отдельные процедуры разрешения. Поэтому представляется перспективным исследование методов генерации специальных систем ограничений для тестовых шаблонов с целью задействовать имеющиеся процедуры разрешения ограничений и тем самым снизить сложность процесса генерации тестовых воздействий.

Важным классом систем являются микропроцессоры. Тестирование микропроцессоров является неотъемлемой частью процесса их разработки. Тестирование может осуществляться на этапе проектирования микропроцессора, такое тестирование называют *имитационным*. Функциональное тестирование микропроцессоров может проводиться на системном или модульном уровне. При *системном тестировании* тестовым воздействием является программа на языке ассемблера микропроцессора (*тестовая программа*). При *модульном тестировании* тестовым воздействием является сигнал, подаваемый на интерфейсные входы тестируемого модуля. Системное тестирование микропроцессора оказывается дешевле модульного тестирования, поскольку не нужно предварительно выделять модуль из всего микропроцессора, подводить к его входам нужные сигналы и принимать выходные и промежуточные сигналы. Поэтому представляется перспективным уделить особое внимание именно автоматизации построения тестовых программ по тестовым шаблонам с использованием ограничений для системного тестирования микропроцессоров.

Тестовое воздействие включает в себя перевод микропроцессора в некоторое специальное состояние. При этом возможные ошибки могут проявиться уже при этом переводе. Поэтому представляется перспективным исследование методов построения тестовых воздействий небольшой длины и методов построения тестовых программ по тестовым шаблонам с использованием начального состояния микропроцессора, что тоже нацелено на сокращения размера тестового воздействия.

Цели и задачи работы

Целью диссертационной работы является исследование и разработка методов автоматического построения тестовых программ по тестовым шаблонам. Для достижения цели были поставлены следующие задачи:

- провести анализ существующих методов построения тестовых программ;
- разработать новые методы построения тестовых программ по тестовым шаблонам, которые позволяют снизить сложность процедуры генерации ограничений и их разрешения;
- провести апробацию предложенных методов для современных архитектур микропроцессоров.

Основные результаты работы

Основные научные результаты, полученные в рамках диссертационной работы и выносимые на защиту, состоят в следующем:

1. метод генерации ограничений (т.н. *совместная генерация ограничений*) для тестовых шаблонов, нацеленных на тестирование инструкций обращения к памяти;
2. метод *зеркальной генерации* ограничений для тестовых шаблонов, нацеленных на тестирование инструкций обращения к памяти;
3. метод описания стратегий вытеснения в кэш-памяти в виде ограничений *перебором диапазонов вытеснения*;

4. метод описания стратегий вытеснения в кэш-памяти в виде ограничений *с использованием функций полезности.*

Научная новизна работы

Следующие результаты являются новыми:

1. метод генерации ограничений (т.н. *совместная генерация ограничений*) для тестовых шаблонов, нацеленных на тестирование инструкций обращения к памяти;
2. метод *зеркальной генерации* ограничений для тестовых шаблонов, нацеленных на тестирование инструкций обращения к памяти;
3. метод описания стратегий вытеснения в кэш-памяти в виде ограничений *перебором диапазонов вытеснения*;
4. метод описания стратегий вытеснения в кэш-памяти в виде ограничений *с использованием функций полезности.*

Практическая значимость

Практическая значимость подтверждается применимостью предложенных идей к практически значимым архитектурам микропроцессоров и тестовым шаблонам.

Доклады и публикации

Основные результаты диссертации докладывались на следующих конференциях и семинарах:

- Второй весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Санкт-Петербург, 2008 г.);
- XVI Международная конференция студентов, аспирантов и молодых учёных «Ломоносов» (г.Москва, 2009 г.);

- XVI Всероссийская межвузовская научно-техническая конференция студентов и аспирантов «Микроэлектроника и информатика - 2009» (г.Москва, Зеленоград, 2009 г.);

- Третий весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Москва, 2009 г.);

- IV летняя международная школа аспирантов по научным вычислениям (г.Москва, 2009 г.);

- Семинарах Института системного программирования РАН (г. Москва, 2009 г.).

По материалам диссертации опубликованы работы _ _ _ _ _, полно отражающие основные результаты диссертации.

Структура и объем диссертации

Работа состоит из введения, _ _ _ _ глав, заключения и списка литературы (_ _ _ _ наименований). Основной текст диссертации (без приложения и списка литературы) занимает _ _ _ _ страниц.

Краткое содержание работы

Первая глава является обзорной и содержит описание методов построения тестовых программ по тестовым шаблонам, примененных в различных промышленных проектах по тестированию микропроцессоров. Рассматриваются методы ручной генерации, комбинаторные методы, использование задачи ATPG [32] и использование ограничений. Описываются достоинства и ограничения различных методов. Основное внимание уделяется качеству генерируемых тестовых программ, полноте и применимости методов. В конце главы дается анализ существующих методов генерации тестовых программ по тестовым шаблонам, уточняются цели и задачи диссертационной работы.

Во второй главе описываются предлагаемые автором методы генерации ограничений для тестовых шаблонов, нацеленных на тестирование механизмов кэширования. Это методы совместной и зеркальной генерации. В основе совместного метода генерации ограничений лежит возможность сокращения размера ограничений за счет обращения к нескольким подсистемам микропроцессора в процессе исполнения инструкции обращения к памяти. Метод позволяет эффективно выбрать часть начального состояния микропроцессора, действительно влияющую на исполнение инструкции, и тем самым сократить размер ограничений. В основе зеркального метода генерации ограничений лежит внесение и учет дополнительного требования на подготовку состояния микропроцессора к тестовому воздействию, которое заключается в том, что вне зависимости от начального состояния микропроцессора каждый элемент тестового воздействия должен быть предварительно подготовлен последовательностью инструкций, начинающихся к обращения к тому же элементу. Зеркальный метод генерации ограничений дополняет метод совместной генерации ограничений. В отличие от совместной генерации зеркальная генерация дает решение, если оно существует, правда, большей длины, чем длина тестовой программы от совместной генерации. Формулируются и доказываются теоремы о корректности и полноте зеркального метода генерации ограничений. Эти теоремы формально обосновывают применение зеркального метода. Также в разделе формулируется и доказывается теорема о дизъюнктивном представлении тестовых ситуаций. Эта теорема дает способ генерирования ограничений для тестовых ситуаций в кэширующих буферах на основе изменений содержимого кэширующих буферов как множества.

В третьей главе описываются предлагаемые автором методы генерации ограничений, описывающих стратегию вытеснения в кэширующих буферах. Это методы перебора диапазонов вытеснения и метод функций полезности. Метод перебора диапазонов вытеснения позволяет записать ограничения лишь на часть тестового воздействия, непосредственно влияющую на вытеснение в данной инструкции. Метод функций полезности предлагает описать вытеснение как ограничение на количество *полезных к вытеснению* инструкций. В разделе формалируются и доказываются теоремы о корректности и полноте применения предлагаемых методов генерации ограничений, описывающих стратегию вытеснения для стратегий вытеснения LRU, FIFO и Pseudo-LRU.

В четвертой главе главе описывается программная реализация генератора ограничений. Описывается архитектура генератора, процесс подготовки входных данных и формат описания тестовых шаблонов и особенностей архитектуры микропроцессора.

В пятой главе описываются результаты апробации предлагаемых автором методов построения тестовых программ по тестовым шаблонам с использованием ограничений. А именно в главе показывается, как с использованием предлагаемых методов сгенерировать ограничения, описывающие работу MMU микропроцессоров для таких архитектур как PowerPC, Alpha, MIPS и Pentium. Для каждой архитектуры на примере одного из микропроцессоров составляется структура MMU и показываются схемы совместной генерации ограничений. Основным результатом главы является обоснование того, что предлагаемые автором методы достаточны для применения при тестировании современных архитектур микропроцессоров и соответствуют поставленным в работе целям.

Глава 1

Обзорная глава, постановка задачи

1.1 Обзор методов генерации тестовых программ

Тестирование микропроцессоров является важной составляющей частью процесса их разработки. Тестированию может подвергаться как готовый чип, так и модель. Тестирование может проводиться как на модульном, так и на системном уровне. В данной работе речь идет о системном функциональном тестировании. Иными словами, целью тестирования является проверка правильности функционирования микропроцессора целиком. Эта проверка выполняется путем запуска на микропроцессоре специальных машинных программ (далее такие программы будут называться *тестовыми*).

Системное функциональное тестирование включает в себя следующие этапы [2]:

1. определение целей тестирования, тестового покрытия и тестовых ситуаций (структурные – какие инструкции включать в тестирование – и функциональные – как инструкции должны быть исполнены);

2. генерация тестовых программ для тестовых ситуаций;

3. исполнение тестовых программ на микропроцессоре, получение выходных данных (трасса исполнения, финальные значения регистров);

4. вынесение вердикта на основе анализа выходных данных.

Данная работа посвящена этапу генерации тестовых программ. В настоящее время в практике системного функционального тестирования микропроцессоров можно выделить следующие подходы к построению тестовых программ:

- *ручная разработка тестовых программ* хоть и практически неприменима для полного тестирования микропроцессора, всё же может применяться для тестирования особых, крайних случаев;

- *тестирование с использованием кросс-компиляции* применяется часто из-за невысокой сложности его проведения: после согласования спецификации микропроцессора можно начинать делать кросс-компилятор, а код, предназначенный для кросс-компиляции, уже готов. Однако гарантировать полноту такое тестирование не может;

- *случайная генерация тестовых программ* применяется так же часто в силу простоты автоматизации. Сгенерированные таким образом тестовые программы позволяют быстро обнаружить простые ошибки, однако не гарантируют полноты тестирования. Разрабатываются и более сложные варианты случайной генерации [16];

- *генерация тестовых программ на основе тестовых шаблонов* предполагает разделение процесса генерации тестовой программы на два этапа: на первом на основе тестовых ситуаций подготавливаются тестовые шаблоны – абстрактные представления тестовых программ – а на втором этапе по тестовым шаблонам генерируются тестовые программы.

Тестовые шаблоны могут описывать следующие свойства тестовых программ:

- заданная последовательность инструкций (только коды операций или коды операций с аргументами);

- заданная последовательность типов инструкций;

- выборка инструкций заданных типов;

- аргументы инструкций (регистры, непосредственные значения, переменные величины);
- дополнительные ограничения на инструкции;
- дополнительные ограничения на отдельные аргументы инструкций, аргументы разных инструкций;
- дополнительные функциональные ограничения на инструкции (при исполнении должны произойти некоторые заданные события).

Выделяют следующие подзадачи при генерации тестовых программ по тестовым шаблонам (подзадачи могут решаться по отдельности [2] или итеративно для каждой очередной выделяемой инструкции [11]):

1. выбор последовательности инструкций / выбор очередной инструкции;
2. выбор аргументов (не значений, а имен аргументов!) инструкций / выбор аргументов очередной инструкции;
3. построение инициализации микропроцессора для выполнения тестовых ситуаций.

Работа посвящена исследованию методов построения инициализации микропроцессора. Исследователями предложены следующие классы методов решения этой задачи:

1. ручная генерация тестовых программ;
2. комбинаторные методы;
3. использование методов генерации входных векторов (ATPG [34]);
4. использование методов разрешения ограничений.

1.1.1 Ручная генерация тестовых программ

Александром Камкиным разработана технология системного функционального тестирования микропроцессоров с использованием тестовых шаблонов [2]. Построение тестовых шаблонов осуществляется полуавтоматически на основе тестового покрытия по модели системы инструкций микропроцессора. Тестовые шаблоны представляют из себя последовательность инструкций с зависимостями между аргументами (например, «запись-чтение») и тестовыми ситуациями для инструкций.

Для получения тестовых программ по сгенерированным тестовым шаблонам следует реализовать на языке Java *конструкторы тестовых данных*. Под «тестовыми данными» понимаются значения регистров, аргументы инструкций обращения к памяти для инициализации состояния кэш-памяти и ячеек оперативной памяти, если это требуется. Все зависимости в тестовом шаблоне обладают направлением, конструирование аргументов инструкций производится итеративно от инструкций, которые не зависят от остальных инструкций, к инструкциям, которые зависят от уже сгенерированных инструкций. Для выбора независимых значений используется случайная генерация.

1.1.2 Комбинаторные методы генерации тестовых программ

Тестовый шаблон состоит из заданной последовательности инструкций, аргументами которых являются переменные величины. Кроме того для каждой переменной величины указывается конечная область значений. Все значения в области равноправны. Тестовая программа содержит ту же последовательность инструкций, а для каждого аргумента выбрано значение из области значений этого аргумента. В комбинаторных методах инструкции воспринимаются лишь как синтаксические объекты (термы) – у них есть лишь имя и аргументы (возможно типизированные).

Последовательность инструкций может быть задана неявно, но у каждой инструкции всё же будут переменные величины в качестве аргументов и для каждой переменной величины задана область значений. Исследователи из Fujitsu Lab. [20] предлагается описать последовательность инструкций в виде выражений (Test Specification Expressions, TSE), а семантику инструкций – на языке ISDL [22]. Отдаленно TSE могут напоминать регулярные выражения, где бесконечнозначные операции заменены конечными аналогами. ISDL-описание может включать в том числе и параметры исполнения инструкции на конвейере, которые могут быть использованы в TSE. Авторы исследования реализовали специальный генератор, который строит тестовые программы, удовлетворяющие данному TSE.

Kohno и Matsumoto [28] рассматривают задачу верификации конвейерных микропроцессоров, используя для этого генерацию тестовых программ с помощью тестовых шаблонов. Тестовый шаблон явно содержит последовательность типов инструкций, возможно, с использованием конструкций итерирования блоков инструкций. Использование разными инструкциями в шаблоне одной и той же переменной величины должно приводить в тестовой программе к использованию одного и того же значения для этой переменной величины. Областями значений являются заданное в архитектуре множество регистров (GPR – множество регистров общего назначения, CPR – множество регистров сопроцессора).

1.1.3 Генерация тестовых программ с использованием методов решения задачи ATPG

Задача ATPG (Automatic Test Pattern Generation) [34] относится к вопросам модульного тестирования микропроцессоров. Модульное тестирование осуществляется подачей определенных сигналов (возможно, многотактовых) на входы модуля (схемы) и снятие значения выходных сигналов (возможно, также многотактовых). Принятие вердикта осуществляется на основе сравнения ожидаемого выходного сигнала и снимаемого с данной схемы. Тестовым воздействием является сигнал, поданный на входные порты схемы. Моделью ошибки является смена функции некоторых элементов схемы (например, в результате пробоя или замыкания элемент может сменить функцию, которую он реализует, на тождественную константу). ATPG – это задача построения тестовых воздействий для схем, нацеленных на данную модель ошибки. Аргументы инструкций являются входными сигналами некоторых модулей микропроцессора, поэтому решая задачу генерации входных сигналов, можно решать и задачу генерации тестовых программ.

Эту идею использовали исследователи из Politecnico di Milano [32]. Тестовым шаблоном выступает препроцессированная модель этапа декодирования инструкции. Модель написана на языке VHDL [14]. Специальный генератор подставляет на место кода инструкции заданные значения кодов операций и передает получившуюся модель стороннему (коммерческому) ATPG-инструменту. Тот в свою очередь возвращает остальные значения, которые надо передать в модуль декодирования инструкции, т.е. значения аргументов инструкции. Метод был применен к тестированию АЛУ VLIW-микропроцессора.

1.1.4 Генерация тестовых программ с использованием методов разрешения ограничений

Под *ограничением* будет пониматься предикат, в котором переменные принимают значения из конечной области. Например, $x > 0$, если $x \in \{0, 10, 100\}$. Задачей разрешения ограничений (constraint satisfaction problem) является задача поиска значений для переменных из их областей значений, при которых все ограничения выполнены [38]. Для областей значений небольшого размера достаточно перебрать все комбинации значений переменных, пока не встретится комбинация, на которой выполнены все ограничения. В общем случае применяются более сложные алгоритмы (зачастую с привлечением эвристик), сочетающие перебор с возвратом и распространение ограничений (т.е. автоматический вывод ограничений-следствий по данной системе ограничений).

Представление в виде CSP удобно для задач, сформулированных в виде задачи выполнимости некоторого набора условий. Задача генерации тестовых программ по тестовым шаблонам тоже может быть сформулирована в таком виде, поскольку есть связанный набор переменных (инструкций, аргументов инструкций, элементов состояния микропроцессора), причем связи выражаются в виде утверждений, зависимостей. Сама идея построения тестовой программы через формулирование тестового шаблона близка решению задачи с использованием CSP, поскольку этап построения тестового шаблона (формализации требований к тестовому воздействию) по сути является этапом формулирования задачи построения тестового шаблона в виде утверждений, в виде задачи выполнимости. Остается только перевести эту формулировку к виду, используемому в инструментах для решения CSP. Выбор инструментов, метод их решения, а также вида самих ограничений, зависит от того, какие применяются тестовые шаблоны и как описывается семантика инструкций.

С целью упрощения подготовки нужного представления семантики микропроцессора китайские исследователи в своем инструменте MAATG [39] предложили использовать хорошо известный язык описания архитектуры EXPRESSION [24]. Тестовые шаблоны позволяют явно задавать блоки инструкций, задавать ограничения на аргументы разных инструкций (одинаковые регистры, разные регистры, непосредственные значения из некоторого множества констант), а также указывать события, которые могут произойти при исполнении инструкции (например, целочисленное переполнение для инструкции ADD). Специальный генератор строит тестовую программу итеративно. Сначала он упорядочивает инструкции так, чтобы переменные для очередной инструкции зависели только от переменных предыдущих инструкций. Это позволяет разбить задачу генерации тестовой программы на последовательность более простых задач генерации одной инструкции. Однако по доступным публикациям невозможно сделать вывод о том, какие ограничения генерирует MAATG и тем самым оценить эффективность работы этого инструмента.

Еще одно семейство инструментов генерации тестовых программ на основе тестовых шаблонов было разработано в IBM в течение последних 20 лет. Далее будет дано описание последнего на сегодняшний день инструмента в этом семействе – Genesys-Pro [33]. Тестовые шаблоны этого инструмента позволяют описывать как заданные последовательности инструкций, так и всевозможные их композиции. Разработчиками предложен несложный императивный язык, позволяющий задать эту последовательность инструкций.

Для каждой инструкции могут быть указаны ограничения на аргументов пожелания к значениям аргументов инструкции для улучшения тестового покрытия (эта информация называется testing knowledge) [21]. Эти пожелания (по сути особенности семантики инструкций и тестовые ситуации) предлагается описывать с использованием ограничений [10]. Можно задавать ограничения на атрибуты аргументов инструкции (например, значение одного аргумента больше значения другого) и состояние микропроцессора (например, на значения в таблицах и буферах). Для описания механизма трансляции адресов (получения физического адреса по виртуальному) предлагается использовать подход DeepTrans [12]. В этом подходе предлагается пользователю описать структуру строки таблицы, через которую осуществляется трансляция, правило соответствия адреса строке, некоторые другие преобразования, а специальный генератор автоматически построит нужную систему ограничений для использования в Genesys-Pro.

Тестовый шаблон может содержать параметры работы генератора тестовых программ: вероятности выбора тех или иных значений, параметры распределения адресов в памяти и другие – они позволяют управлять выбором некоторого одного значения из множества допустимых.

Требуется описать структуру системы команд (architecture model), задать исполнительную семантику команд (по сути симулятор микропроцессора).

Рассмотрим теперь, как Genesys-Pro генерирует тестовые программы на основе тестовых шаблонов. Для бо́льшей эффективности этапы построения последовательности инструкций и выбора аргументов осуществляются вместе (отдельная инициализация состояния микропроцессора не проводится). На основе параметров генерации, текущего состояния модели микропроцессора и построенной тестовой программы выбирается очередная инструкция (тестовые шаблоны позволяют описывать сложные потоки управления на инструкциях). Далее для этой инструкции генерируются аргументы инструкций. Для этого строится и разрешается система ограничений на основе тестовых ситуаций (testing knowledge) и текущего модельного состояния микропроцессора), в результате чего получаются значения аргументов. Готовая инструкция исполняется на модели микропроцессора (architecture model, он готовится пользователем) с получением нового модельного состояния микропроцессора. На этом генерация инструкции завершается и генерируется следующая инструкция. Ключевым моментом является эффективность работы решателя ограничений. Для этой цели разработчики инструмента самостоятельно написали свой решатель ограничений. Он базируется на хорошо известном семействе алгоритмов разрешения ограничений MAC (Maintaining Arc-Consistency) [38], но заточен под ограничения, генерируемые для тестовых программ [15]. Написание такого решателя является довольно нетривиальной задачей и предметом отдельного исследования. Например, Genesys-Pro позволяет использовать для описания тестовых ситуаций элементы массивов (Memory, таблицы страниц) с переменными индексами.

Тем самым ни один из методов генерации тестовых программ, использующих ограничения, не нацеливается на строго заданную последовательность инструкций, однако потребности в использовании тестовых шаблонов с заданной последовательностью инструкций отмечаются исследователями [2].

1.1.5 Сравнение методов генерации тестовых программ

Сравнение проводилось по следующим критериям:

1. сложность построения генератора тестовых программ;
2. допустимые архитектурые механизмы;
3. полнота метода.

Из сделанного обзора следует, что возможность генерирования тестовых программ для тестовых шаблонов, ориентированных на поведение MMU (тестовые ситуации в кэширующих буферах и таблицах MMU), т.е. поддержку механизмов кэширования в MMU, есть у следующих методов:

- ручное написание генераторов тестовых программ [2];
- вероятностные алгоритмы генерации тестовых программ (как разновидность – примитивные переборные алгоритмы);
- покомандный перебор с возвратом на основе разрешения ограничений [11].

	полный	неполный
простой		переборный алгоритм примитивный вероятностный
сложный	вручную написанный генератор хитрый вероятностный покомандный перебор с возвратом	—

Использование простых переборных или вероятностных алгоритмов генерации тестовых программ позволяют подготовить генератор на основе нехитрых идей, однако они не позволяют добиться хорошей полноты. Это означает, что для произвольного тестового шаблона время генерации тестовой программы может быть очень велико.

Напротив более хитрые варианты вероятностных (переборных) алгоритмов, написанные вручную генераторы или генераторы, основанные на более регулярных алгоритмах (например, покомандный перебор с возвратом, реализованный в системе Genesys-Pro [11]) нацелены на получение высокой полноты. Однако достигается это в основном за счет разработки и применения уникальных идей, которые сложно использовать вновь при тестировании другого микропроцессора. Это усложняет написание таких генераторов и, как результат, увеличивает время подготовки самого генератора.

Тем самым представляется перспективным исследование и разработка регулярных легко переиспользуемых методов построения генераторов тестовых программ по тестовым шаблонам, применимых для тестирования механизмов кэширования, не уступающих в полноте существующим методам.

1.2 Постановка задачи

В диссертации решается задача построения тестовых программ по тестовым шаблонам, обладающим следующими свойствами. Тестовый шаблон представляется последовательностью троек (I_i, A_i, S_i), где I_i – заданная инструкция, A_i – список аргументов инструкции, S_i – тестовая ситуация инструкции. Аргументами являются явно заданные регистры и переменные, не меняющие своего значения (в тестовой программе они станут непосредственными значениями). Тестовая ситуация инструкции – это ограничение на аргументы инструкции и текущее состояние микропроцессора. Примеры тестовых ситуаций: «при исполнении инструкции должно произойти целочисленное переполнение (это ограничение только на аргументы инструкции)», «при исполнении инструкции должно произойти кэш-попадание в кэш-памяти первого уровня» (это ограничение не только на аргументы инструкции, но и на состояние микропроцессора перед ее исполнением, поскольку кэш-память является подсистемой микропроцессора). Кроме тестового шаблона задано начальное состояние модели микропроцессора. Состояние модели микропроцессора включает в себя состояние всех его подсистем. Например, состоянием регистра является значение, которое в нем хранится. Состоянием кэш-памяти является его содержимое.

Требуется построить тестовую программу по тестовому шаблону [6, 8], которая состоит из двух частей: инициализирующие инструкции и инструкции тестового воздействия (см.рис. 1.1). Инициализирующие инструкции переводят модель микропроцессора из заданного начального состояния в состояние, необходимое для тестового воздействия. Инструкции тестового воздействия в точности повторяют последовательность инструкций тестового шаблона с заменой переменных на непосредственные значения. На рисунке 1.2 приведен пример тестового шаблона и возможных инструкций тестового воздействия, построенных по этому шаблону.

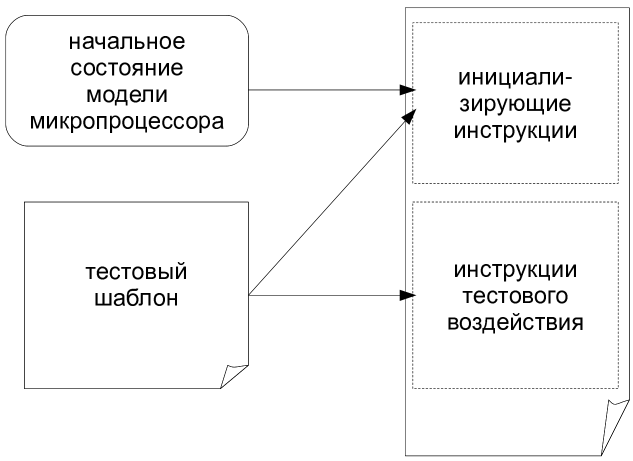


Рис. 1.1. Составление тестовой программы

```
AND r1, r2, r3 @ normal          AND r1, r2, r3
LD r4, r2, c1 @ l1Hit            LD r4, r2, 0x0FA2
SUB r3, r1, r5 @ overflow        SUB r3, r1, r5
```

Рис. 1.2. Тестовый шаблон и возможные соответствующие ему инструкции тестового воздействия

Уже сгенерированная программа может быть позднее дополнена инструкциями проверки состояния микропроцессора после исполнения инструкций тестового воздействия.

Инициализирующие инструкции призваны подготовить модель микропроцессора к исполнению инструкций тестового воздействия. Без инициализирующих инструкций запуск инструкций тестового воздействия даже на корректной модели микропроцессора может приводить к ложным сообщениям об ошибках в модели. В работе рассматривается модель микропроцессора, включающая в себя регистры общего назначения, кэш-память (возможно многоуровневую) и буфер трансляции адресов (TLB, Translation Lookaside Buffer) [27]. Таким образом, инициализирующие инструкции могут включать инструкции изменения значений регистров и ячеек кэш-памяти и TLB.

В данной работе среди методов генерации тестовых программ выбран метод, использующий разрешение ограничений (CSP). Однако по сравнению с существующими аналогами в данной работе поставлена задача исследовать возможности снижения сложности подготовки генератора тестовых программ (по сравнению, например, с мощным Genesys-Pro), не проиграв сильно в масштабируемости генератора. При этом, возможно, придется выделить среди всевозможных архитектурных механизмов наиболее часто использующиеся и требующие тестирование в современных микропроцессорах.

1.3 Предварительные сведения и термины

1.3.1 Типы кэш-памяти

По организации кэш-память делят на *полностью ассоциативную*, *прямого доступа* и *наборно-ассоциативную*. Различие производится на основе двух параметров: количества секций W и количества наборов R. Кэш-память хранит некоторый набор данных. Каждому блоку данных соответствует некоторый адрес (физический или виртуальный). Блоки с адресами организованы в *секции* и *наборы* (см.рис. 1.3).

Каждый адрес может быть разделен на два битовых поля: поле *те-*

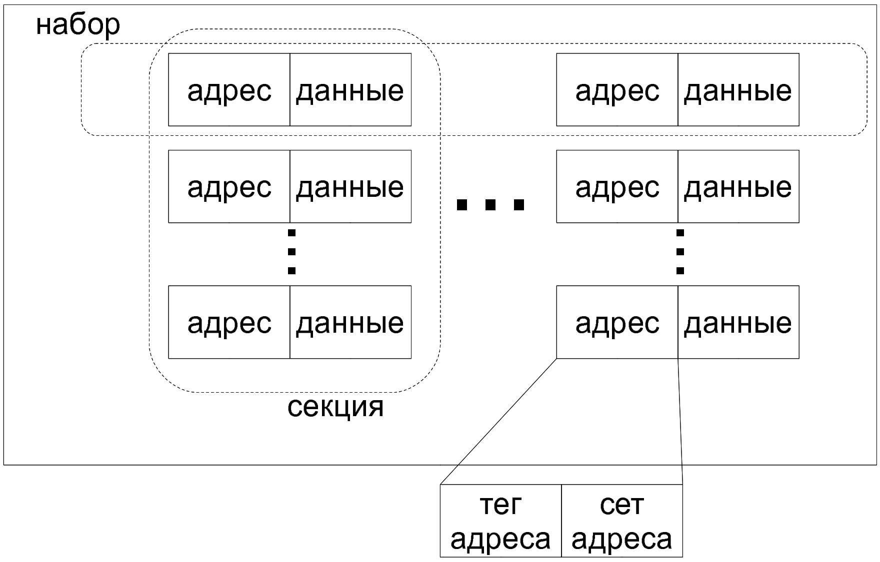


Рис. 1.3. Модель кэш-памяти и адреса данных

га адреса и поле *сет адреса*. Один набор составляют адреса с одинаковым сетом. Кэш-память организована таким образом, что для каждого сета хранится всегда одно и то же количество адресов (равное количеству секций W). Адреса всех данных в кэш-памяти различные. Отсюда следует, что теги адресов одного набора разные. В кэш-памяти представлены все наборы, возможные в рамках битового поля сета адреса.

Кэш-память является полностью ассоциативной, если $R = 1$. Кэш-память является кэш-памятью прямого доступа, если $W = 1$. И кэш-память является наборно-ассоциативной, если $R > 1$ и $W > 1$.

Инструкции обращения в память бывают двух видов: инструкции загрузки данных из памяти по данному адресу и инструкции сохранения данных в памяти по данному адресу. При выполнении этих инструкций может быть задействована кэш-память. Если данные по требуемому адресу присутствуют в кэш-памяти, операция проводится с нею. Такая ситуация называется *кэш-попаданием*. Если данные по требуемому адресу не присутствуют в кэш-памяти, осуществляется подгрузка данных в кэш-память и совершение операции. Такая ситуация называется *кэш-промахом*. В этом случае если кэш-память полностью заполнена, некоторые данные должны быть *вытеснены* из кэш-памяти и на их место будут загружены данные по требуемому адресу. *Стратегия вытеснения* (или *политика замещения*) – это правило, по которому определяются вытесняемые данные. Например, могут быть вытеснены данные, которые дольше всего не были нужны (такая стратегия называется LRU), или данные, которые были внесены в кэш-память раньше остальных (такая стратегия называется FIFO).

1.3.2 Стратегия вытеснения Pseudo-LRU

Стратегия вытеснения Pseudo-LRU стала одним из результатов попыток предложить стратегию вытеснения, близкой по эффективности к LRU , но обладающую меньшими накладными расходами на организацию. ....

Каноническое определение Pseudo-LRU

Во многих книгах приводятся следующее определение стратегии вытеснения Pseudo-LRU для случая $w = 4$ [17] (в этом случае для каждого набора выделяется 3 бита B_1, B_2 и B_3, обычно этот случай поясняется на *бинарном дереве*):

$$\left[\begin{array}{c|ccc} & B_1 & B_2 & B_3 \\ \hline \pi_0 & 0 & 0 & \mathsf{X} \\ \pi_1 & 0 & 1 & \mathsf{X} \\ \pi_2 & 1 & \mathsf{X} & 0 \\ \pi_3 & 1 & \mathsf{X} & 1 \end{array}\right]$$

При кэш-попадании тега, расположенного в секции с номером i, действует i'я строка матрицы (она помечена символом π_i). Биты, напротив которых в i'й строке находится X, не меняются. Биты, напротив которых в i'й строке находится число, принимают значение, равное этому числу.

При кэш-промахе надо определить номер секции, в которой будут заменены данные. Для этого используется инвертированная форма той же матрицы:

$$\left[\begin{array}{ccc|c} B_1 & B_2 & B_3 & \\ \hline 1 & 1 & \mathsf{X} & \to \pi_0 \\ 1 & 0 & \mathsf{X} & \to \pi_1 \\ 0 & \mathsf{X} & 1 & \to \pi_2 \\ 0 & \mathsf{X} & 0 & \to \pi_3 \end{array}\right]$$

Выбирается строка, соответствующая текущему состоянию бит B_1, B_2 и B_3: если напротив бита в строке находится число, бит должен быть равен этому числу – если напротив бита в строке находится X, то требования на соответствующий бит нет. Такая всегда всегда будет единственной.

Формализованное описание для всех допустимых w не приводится. Однако в дальнейшем для формулирования и доказательства утверждений про стратегию вытеснения Pseudo-LRU , такое описание будет необходимо. Для $w = 8$ стратегия будет задаваться следующей матрицей:

$$\left[\begin{array}{c|ccccccc} & B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 \\ \hline \pi_0 & 0 & 0 & \mathsf{X} & 0 & \mathsf{X} & \mathsf{X} & \mathsf{X} \\ \pi_1 & 0 & 0 & \mathsf{X} & 1 & \mathsf{X} & \mathsf{X} & \mathsf{X} \\ \pi_2 & 0 & 1 & \mathsf{X} & \mathsf{X} & 0 & \mathsf{X} & \mathsf{X} \\ \pi_3 & 0 & 1 & \mathsf{X} & \mathsf{X} & 1 & \mathsf{X} & \mathsf{X} \\ \pi_4 & 1 & \mathsf{X} & 0 & \mathsf{X} & \mathsf{X} & 0 & \mathsf{X} \\ \pi_5 & 1 & \mathsf{X} & 0 & \mathsf{X} & \mathsf{X} & 1 & \mathsf{X} \\ \pi_6 & 1 & \mathsf{X} & 1 & \mathsf{X} & \mathsf{X} & \mathsf{X} & 0 \\ \pi_7 & 1 & \mathsf{X} & 1 & \mathsf{X} & \mathsf{X} & \mathsf{X} & 1 \end{array}\right]$$

Следующее утверждение 1 дает алгоритм преобразования списка бит $B_1, B_2, ..., B_{w-1}$. В его формулировке применяется двоичное разложение, биты разложения обозначаются следующим образом, например, $1 = (0\ 0\ 1)$, $6 = (1\ 1\ 0)$.

Утверждение 1 (($w-1$)-представление стратегии вытеснения Pseudo–LRU). *При кэш-попадании тега с позицией $i = (i_1\ i_2\ \ldots\ i_W)$ происходит следующее изменение:*

$B_{k_1} := i_1$	$k_1 = (1)$
$B_{k_2} := i_2$	$k_2 = (1\ i_{W-1})$
$B_{k_3} := i_3$	$k_3 = (1\ i_{W-2}\ i_{W-1})$
...	...
$B_{k_W} := i_W$	$k_W = (1\ i_1\ i_2\ \ldots\ i_{W-1})$

При кэш-промахе тега позиция $i = (i_1\ i_2\ \ldots\ i_W)$ определяется следующим образом:

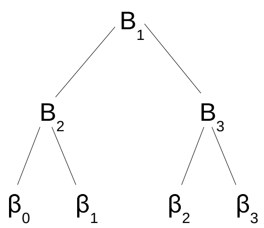
$$
\begin{array}{l|l}
i_1 = \neg B_{k_1} & k_1 = (1) \\
i_2 = \neg B_{k_2} & k_2 = (1\ \neg B_{k_1}) \\
i_3 = \neg B_{k_3} & k_3 = (1\ \neg B_{k_1}\ \neg B_{k_2}) \\
... & ... \\
i_W = \neg B_{k_W} & k_W = (1\ \neg B_{k_1}\ \neg B_{k_2} \ldots \neg B_{k_{W-1}})
\end{array}
$$

Кроме того при кэш-промахе после определения позиции i делается преобразование бит $B_1, B_2, ..., B_{W-1}$ так, как в случае кэш-попадания на π_i.

Определение Pseudo-LRU на ветвях бинарного дерева

Здесь будет показано, как из канонического определения Pseudo-LRU получить формулировку Pseudo-LRU , в которой рассматривается не вся последовательность бит $B_1, B_2, ..., B_{W-1}$, а всего один ......

Сначала этот переход покажем на примере $w = 4$. Первый шаг – это смена «состояния»: вместо последовательности $B_1, B_2, ..., B_{w-1}$ будем рассматривать последовательность $\beta_0, \beta_1, ..., \beta_{w-1}$. Каждый β_i соответствует i'й листовой вершине бинарного дерева. Кэш-попадание меняет теперь не внутренние вершины дерева, а листовые вершины. Каждый β_i будет представляться списком длины W – путь от корня дерева к i'й листовой вершине: β_0 и β_1 соответствует $(B_1\ B_2)$, β_2 и β_3 соответствует $(B_1\ B_3)$.



$$
\left[\begin{array}{c|ccc}
 & B_1 & B_2 & B_3 \\
\hline
\pi_0 & 0 & 0 & X \\
\pi_1 & 0 & 1 & X \\
\pi_2 & 1 & X & 0 \\
\pi_3 & 1 & X & 1
\end{array}\right]
\xrightarrow{1}
\left[\begin{array}{c|cccc}
 & \beta_0 & \beta_1 & \beta_2 & \beta_3 \\
\hline
\pi_0 & (0\ 0) & (0\ 1) & (1\ X) & (1\ X) \\
\pi_1 & (0\ 1) & (0\ 0) & (1\ X) & (1\ X) \\
\pi_2 & (1\ X) & (1\ X) & (0\ 0) & (0\ 1) \\
\pi_3 & (1\ X) & (1\ X) & (0\ 1) & (0\ 0)
\end{array}\right]
$$

Заметим, что получилась симметричная матрица. Теперь рассмотрим каждый столбец отдельно. Затем переставим элементы столбца в порядке увеличения относительных позиций.

$$\begin{bmatrix} & \beta_0 \\ \hline \pi_0 & (0\ 0) \\ \pi_1 & (0\ 1) \\ \pi_2 & (1\ \mathsf{X}) \\ \pi_3 & (1\ \mathsf{X}) \end{bmatrix} \begin{bmatrix} & \beta_1 \\ \hline \pi_0 & (0\ 1) \\ \pi_1 & (0\ 0) \\ \pi_2 & (1\ \mathsf{X}) \\ \pi_3 & (1\ \mathsf{X}) \end{bmatrix} \begin{bmatrix} & \beta_2 \\ \hline \pi_0 & (1\ \mathsf{X}) \\ \pi_1 & (1\ \mathsf{X}) \\ \pi_2 & (0\ 0) \\ \pi_3 & (0\ 1) \end{bmatrix} \begin{bmatrix} & \beta_3 \\ \hline \pi_0 & (1\ \mathsf{X}) \\ \pi_1 & (1\ \mathsf{X}) \\ \pi_2 & (0\ 1) \\ \pi_3 & (0\ 0) \end{bmatrix} \quad \stackrel{\xrightarrow{2}}{\pi_j^i \equiv \pi_{i \oplus j}}$$

$$\begin{bmatrix} & \beta_0 \\ \hline \pi_0^0 & (0\ 0) \\ \pi_1^0 & (0\ 1) \\ \pi_2^0 & (1\ \mathsf{X}) \\ \pi_3^0 & (1\ \mathsf{X}) \end{bmatrix} \begin{bmatrix} & \beta_1 \\ \hline \pi_0^1 & (0\ 0) \\ \pi_1^1 & (0\ 1) \\ \pi_2^1 & (1\ \mathsf{X}) \\ \pi_3^1 & (1\ \mathsf{X}) \end{bmatrix} \begin{bmatrix} & \beta_2 \\ \hline \pi_0^2 & (0\ 0) \\ \pi_1^2 & (0\ 1) \\ \pi_2^2 & (1\ \mathsf{X}) \\ \pi_3^2 & (1\ \mathsf{X}) \end{bmatrix} \begin{bmatrix} & \beta_3 \\ \hline \pi_0^3 & (0\ 0) \\ \pi_1^3 & (0\ 1) \\ \pi_2^3 & (1\ \mathsf{X}) \\ \pi_3^3 & (1\ \mathsf{X}) \end{bmatrix}$$

После перехода к относительным позициям (π_j^i – это позиция π_j относительно π_i) все столбцы получились одинаковыми. Иными словами, алгоритм изменения набора согласно стратегии вытеснения Pseudo-LRU на относительных позициях инвариантен относительно абсолютной позиции вытесняемого тега. Тег вытесняется в том случае, когда его список равен (1 1). Следующая теорема формально доказывает этот факт.

Теорема 1. *инвариантность .....формулировка через перекрашивание....*

Доказательство. //TODO □

Это позволяет сформулировать определение стратегии вытеснения Pseudo-LRU , сфокусированная не на изменении всего набора, а на изменении свойства одного тега набора. На этом определении будут базироваться применения предлагаемых методов генерации ограничений для стратегии вытеснения Pseudo-LRU .

Утверждение 2 (формулировка Pseudo-LRU на ветвях бинарного дерева)**.** *Сопоставим тегу вектор длины W. Каждая инструкция с этим тегом изменяет этот вектор на (0 0 … 0). Тег является вытесняемым в том и только в том случае, если этот вектор равен (1 1 … 1). Влияние других инструкций определяется относительной позицией их тега* *относиельно позиции данного тега. Если относительная позиция принадлежит множеству $[\frac{w}{2^k},\ \frac{w}{2^{k-1}}), k = 1, 2, ..., W$, то первые $k-1$ элементов вектора становятся равными 0, k'й элемент вектора становится равным 1, остальные элементы вектора не меняются.*

Вектор длины W будет соответствовать пути из корня бинарного дерева в листовую вершину дерева, соответствующую данному тегу. Будем называть процесс изменения элемента вектора *перекрашиванием вершины ветви*. Элементы вектора, равные 0, будем называть *белыми*, элементы вектора, равные 1, будем называть *черными*.

Говоря в терминах бинарного дерева, нелистовая вершина в ветви к данной листовой вершине будет «белой», если дуга от нее идет налево и она помечена цифрой 1 или дуга от нее идет направо и она помечена цифрой 0 (т.е. в том случае, когда направление дуги из нее соответствует пометке этой дуги). Нелистовая вершина будет называться «черной», если направление дуги из нее не соответствует пометке этой дуги. Вытесняется тот тег набора, путь к которому полностью состоит из несоответствующих дуг.

Таблица вытеснения для Pseudo-LRU

$$\left[\begin{array}{c|cccccccc}
\pi_0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_1 & 1 & 0 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_2 & 2 & 3 & 0 & 1 & 4 & 5 & 6 & 7 \\
\pi_3 & 3 & 2 & 0 & 1 & 4 & 5 & 6 & 7 \\
\pi_4 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\
\pi_5 & 5 & 4 & 6 & 7 & 0 & 1 & 2 & 3 \\
\pi_6 & 6 & 7 & 4 & 5 & 0 & 1 & 2 & 3 \\
\pi_7 & 7 & 6 & 4 & 5 & 0 & 1 & 2 & 3 \\
\pi_m & m & 6 & 4 & 5 & 0 & 1 & 2 & 3
\end{array}\right]$$

Это принципиально другой подход в отличие от канонического определения, поскольку элементы переставляются, а в каноническом – они остаются на месте............

Определение через ветви является связующим звеном между каноническим определением и определением с помощью таблицы вытеснения, поскольку β_i – это и есть позиция, которая меняется точно так же, как и переставляется тег в наборе согласно таблице вытеснения.........

Глава 2

Методы генерации ограничений для описания поведения тестовых программ

2.1 Совместная генерация ограничений

В этом разделе формально ставится задача генерации тестовых данных для последовательности тестовых ситуаций в кэширующем буфере, выделяется подзадача описания механизма вытеснения и описывается метод построения ограничений обозримого размера для генерации тестовых программ по тестовым шаблонам с использованием ограничений. Идея совместного метода заключается в использовании содержимого нескольких кэширующих буферов и таблиц одновременно.

2.1.1 Представление тестовых ситуаций в кэширующих буферах в виде ограничений

Тестовые шаблоны для последовательности инструкций описывают ограничение на изменение состояния микропроцессора. Это изменение достигается специальным выбором аргументов инструкций тестового шаблона.

Результат этого выбора фиксируется в виде тестовой программы.

Методика генерации тестовых программ по тестовым шаблонам *с использованием ограничений* предполагает составление системы ограничений, ее разрешение (результатом разрешения является модель ограничений, т.е. значения переменных, на которых сформулированы ограничения) и построение тестовой программы на основе модели ограничений. Переменными могут являться значения регистров, непосредственные значения в тестовой программе, значения в ячейках оперативной памяти, значения в ячейках кэш-памяти и другие подсистемы.

В данной работе особо важным классом инструкций будет являться класс *инструкций обращения к памяти*. Такие инструкций присутствуют во всех микропроцессорах, входящих в состав вычислительных систем с оперативной памятью. Инструкции обращения к памяти делятся на два класса: *инструкции загрузки данных из памяти* и *инструкции сохранения данных в памяти*. При исполнении такой инструкции кроме оперативной памяти могут быть задействованы некоторые специальные структуры данных-подсистемы микропроцессора- а именно *кэширующие буфера* и *таблицы*.

Таблицы содержат последовательный набор данных, снабженный индексами. Изменение содержимого таблиц осуществляется программно. Пример таблицы – таблица страниц виртуальной памяти. Изменение содержимого этих таблиц осуществляется операционной системой.

Кэширующие буфера содержат множество пар (ячеек) «(тег, значение)» заданного количества. Содержимое кэширующего буфера может меняться в процессе работы микропроцессора: какие-то ячейки добавляются, какие-то *вытесняются*. Управление кэширующими буферами осуществляется микропроцессором. Исполнение инструкции обращения к памяти может включать в себя обращения к кэширующим буферам для получения данных по некоторому тегу. Обращение к кэширующему буферу может быть успешным (эта ситуация называется *кэш-попаданием*), если нужные данные есть в буфере, и неуспешным (эта ситуация называется *кэш-промахом*), если нужных данных нет в буфере.

Кэширующий буфер может быть *подчинен* таблице, если при неуспешном обращении к кэширующему буферу поиск данных продолжается в таблице, при успешном – обращение к таблице не производится. Если поиск в таблице оказался успешным, то найденные данные добавляются в кэширующий буфер (некоторые данные из кэширующего буфера при этом вытесняются для поддержания постоянного размера буфера). Например, в микропроцессоре MIPS RM7000 [36] кэширующий буфер DTLB подчинен таблице JoinTLB. Можно представить, что кэш-память подчинена основной памяти, если рассматривать основную память как таблицу, правда, основная память не является подсистемой микропроцессора.

Тестовая ситуация инструкции обращения к памяти будет включать указание на то, какие обращения к кэширующим буферам в данной инструкции успешные, а какие – нет.

Будем считать тестовую ситуацию на инструкцию обращения к памяти *полной*, если она содержит тестовые ситуации на все кэширующие буфера, которые задействованы при исполнении этой инструкции. Например, если микропроцессор содержит двухуровневую кэш-память и при исполнении задействованы оба уровня кэш-памяти (например, в кэш-памяти первого уровня происходит промах, а в кэш-памяти второго уровня – попадание), то тестовая ситуация на эту инструкцию содержит тестовую ситуацию на кэш-память первого уровня и на кэш-память второго уровня.

Данная работа не рассматривает тестовые шаблоны, в которых есть инструкции обращения к памяти с неполными тестовыми ситуациями.

Обратимся более детально к построению ограничений по тестовым шаблонам. Каждая инструкция при исполнении может поменять состояние микропроцессора (значение регистров, содержимое кэширующих буферов и таблиц). Значения регистров будут представляться «скалярными» переменными. Содержимое кэширующих буферов будет представляться *множеством ячеек*. Содержимое кэширующих буферов можно разделить на две структуры – структура для хранения кэшированных данных и структура для хранения тегов адресов кэшированных данных. Для моделирования тестовых ситуаций в кэширующих буферах будет использоваться только структура для хранения тегов, поскольку тестовые ситуации на кэшируемые данные рассматриваться не будут.

Ограничения для тестовых ситуаций в кэширующих буферах могут включать адреса, с которыми работает инструкция.

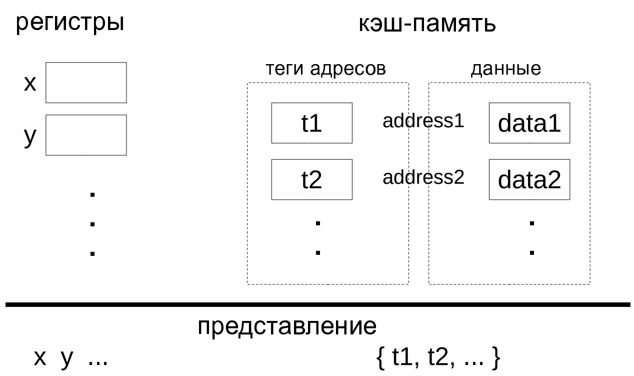


Рис. 2.1. Представление состояния микропроцессора

Утверждение 3. *Тестовые ситуации в кэширующих буферах имеют следующую простую форму с использованием переменных L – текущее состояние (содержимое) кэширующего буфера (множество тегов данных), x – тег адреса данных в инструкции):*

- кэш-попадание *выражается в виде ограничения $x \in L$;*
- кэш-промах *выражается в виде ограничения $x \notin L$.*

Для x и L надо составить дополнительные ограничения, описывающие их значения. x может быть составлен из аргументов инструкции (обычно регистров) или быть результатом обращений к другим буферам и таблицам.

Для переменной L в каждой инструкции методом индукции может быть составлено следующее выражение. База: L для первой инструкции есть начальное содержимое кэширующего буфера, это переменная величина в системе уравнений. Теперь индуктивный шаг. Пусть выражение для очередной инструкции L, а для следующей – L'. Тогда если тестовая ситуация очередной инструкции – кэш-попадание, то $L' \equiv L$ (так как содержимое не меняется), а если кэш-промах с адресом x, то $L' \equiv (L \setminus \{x'\} \cup \{x\})$ (так как в кэширующий буфер при промахе добавляются данные по нужному адресу, а некоторые данные вытесняются, x' есть адрес вытесняемых данных). Для новой переменной x' добавим в систему такие уравнения: $x' \in L \wedge displaced(x') \wedge R(x) = R(x')$, предикат $displaced(x')$ истинен, если x' является адресов вытесняемых данных в данной инструкции. Предикат $displaced$ описывает *стратегию вытеснения*, т.е. правило, по которому в кэширующем буфере выбираются данные, которые следует удалить (вместе с тегом), а на их место поместить данные, вызвавшие промах.

Для кэширующего буфера прямого отображения общезначимо утверждение $(R(x) = R(x')) \to displaced(x')$, поэтому для такого типа кэширующих буферов уравнение $displaced(x')$ можно исключить из ограничений. Функциональный символ R используется для задания набора, которому относится адрес, в кэширующих буферах прямого отображения и наборно-ассоциативных кэширующих буферах. Возможна такая семантика этого символа – $R(x)$ это множество адресов, которые потенциально могут находиться в том же наборе, что и набор адреса x (верно утверждение, что адрес не может соответствовать более чем одному набору и не соответствовать никакому набору вообще, одному набору могут соответствовать разные адреса). Или такая семантика – $R(x)$ это номер набора адреса x. Для составления уравнений может быть выбрана любая семантика. Для полностью-ассоциативных кэширующих буферов уравнение $R(x) = R(x')$ является тождественной истиной, поскольку в нем все адреса соответствуют одному набору.

Следующая теорема описывает выражение для L без использования индукции и способ составления ограничений для тестовых ситуаций в кэширующих буферах:

Лемма 1. *Пусть L – выражение для текущего состояния (содержимого) кэширующего буфера, L_0 – множество адресов данных, расположенных в кэширующем буфере перед исполнением инструкций тестового шаблона, $\{x_i\}$ – множество адресов данных в инструкциях с кэш-промахами, расположенными до текущей инструкции в том же порядке, что и в тестовом шаблоне, $\{x'_i\}$ – множество адресов вытесняемых данных в инструкциях с кэш-промахами, расположенными до текущей инструкции в том же порядке, что и в тестовом шаблоне. Тогда*

$$L \equiv L_0 \setminus \bigcup_{i=1}^{n} \{x'_i\} \cup \bigcup_{i=1}^{n} (\{x_i\} \setminus \cup_{j\,=\,i+1}^{n} \{x'_j\}).$$

Доказательство. По сути надо показать, что $((L_0 \setminus \{x'_1\} \cup \{x_1\}) \setminus \{x'_2\} \cup \{x_2\})... \setminus \{x'_n\} \cup \{x_n\} \equiv L_0 \setminus \bigcup_{i=1}^{n} \{x'_i\} \cup \bigcup_{i=1}^{n} (\{x_i\} \setminus \cup_{j\,=\,i+1}^{n} \{x'_j\})$. Покажем это по индукции. База: при $n = 0$ обе формулы имеют вид L_0, очевидно, что они эквивалентны. Пусть эквивалентность установлена для некоторого

n, т.е. установлено, что $A \equiv ((L_0 \backslash \{x'_1\} \cup \{x_1\}) \backslash \{x'_2\} \cup \{x_2\}) ... \backslash \{x'_n\} \cup \{x_n\} \equiv L_0 \backslash \bigcup_{i=1}^n \{x'_i\} \cup \bigcup_{i=1}^n (\{x_i\} \backslash \cup_{j\ =\ i+1}^n \{x'_j\})$. Покажем, что $A \backslash \{x'_{n+1}\} \cup \{x_{n+1}\} \equiv L_0 \backslash \bigcup_{i=1}^n \{x'_i\} \backslash \{x'_{n+1}\} \cup \bigcup_{i=1}^n (\{x_i\} \backslash (\cup_{j=i+1}^n \{x'_j\} \cup \{x'_{n+1}\})) \bigcup \{x_{n+1}\}$. Для этого достаточно применить правила дистрибутивности над множественными операциями: $A \backslash \{x'_{n+1}\} \cup \{x_{n+1}\} \equiv (L_0 \backslash \bigcup_{i=1}^n \{x'_i\} \cup \bigcup_{i=1}^n (\{x_i\} \backslash \cup_{j\ =\ i+1}^n \{x'_j\})) \backslash \{x'_{n+1}\} \cup \{x_{n+1}\} \equiv L_0 \backslash \bigcup_{i=1}^n \{x'_i\} \backslash \{x'_{n+1}\} \cup \bigcup_{i=1}^n (\{x_i\} \backslash \cup_{j\ =\ i+1}^n \{x'_j\} \backslash \{x'_{n+1}\}) \cup \{x_{n+1}\}$. □

Например, если перед данной инструкцией располагается 3 инструкции с кэш-промахом, то $L \equiv L_0 \backslash \{x'_1, x'_2, x'_3\} \cup (\{x_1\} \backslash \{x'_2, x'_3\}) \cup (\{x_2\} \backslash \{x'_3\}) \cup \{x_3\}$.

Теорема 2 (Дизъюнктивная форма уравнений для тестовых ситуаций в кэширующих буферах)**.** *Пусть L_0 – множество адресов данных, расположенных в кэширующем буфере перед исполнением инструкций тестового шаблона, $\{x_i\}$ – множество адресов данных в инструкциях с кэш-промахами, расположенными до текущей инструкции в том же порядке, что и в тестовом шаблоне, $\{x'_i\}$ – множество адресов вытесняемых данных в инструкциях с кэш-промахами, расположенными до текущей инструкции в том же порядке, что и в тестовом шаблоне. Тогда*

- *для инструкции с кэш-попаданием адреса x следует добавить следующую совокупность уравнений:*

$$\left[\begin{array}{l} x \in L_0 \wedge x \notin \{x'_1, x'_2, ..., x'_n\} \\ x = x_1 \wedge x \notin \{x'_2, ..., x'_n\} \\ x = x_2 \wedge x \notin \{x'_3, ..., x'_n\} \\ ... \\ x = x_{n-1} \wedge x \notin \{x'_n\} \\ x = x_n \end{array}\right.$$

- *для инструкции с кэш-промахом адреса x (и адресом вытесненных* *данных x') следует добавить следующую систему уравнений:*

$$\begin{cases} \left[\begin{array}{l} x \notin L_0 \wedge x \notin \{x_1, x_2, ..., x_n\} \\ x = x'_1 \wedge x \notin \{x_2, ..., x_n\} \\ x = x'_2 \wedge x \notin \{x_3, ..., x_n\} \\ ... \\ x = x'_{n-1} \wedge x \notin \{x_n\} \\ x = x'_n \end{array}\right. \\ \\ \left[\begin{array}{l} x' \in L_0 \wedge x \notin \{x'_1, x'_2, ..., x'_n\} \\ x' = x_1 \wedge x \notin \{x'_2, ..., x'_n\} \\ x' = x_2 \wedge x \notin \{x'_3, ..., x'_n\} \\ ... \\ x' = x_{n-1} \wedge x \notin \{x'_n\} \\ x' = x_n \end{array}\right. \\ \\ displaced(x') \\ \\ R(x) = R(x') \end{cases}$$

Доказательство. Применим утверждение 3 для представления тестовой ситуации и лемму 1 для записи текущего состояния кэширующего буфера. Для вытесняемого тега записываем те же ограничения, что и для кэш-попадания, поскольку вытесняемый тег принадлежит текущему состоянию кэширующего буфера. Кроме того для вытесняемого тега формулируются дополнительные ограничения – $displaced(x')$ (стратегия вытеснения) и $R(x) = R(x')$ (из определения вытеснения: вытесняемый тег обязательно относится к тому же региону, что и вытесняющий). □

Заметьте, что получившиеся ограничения для кэш-попадания и кэш-промаха получились очень похожими, хотя изначально у них было два совершенно противоположных представления.

Теорему 2 можно переформулировать без использования вытеснямых тегов:

Утверждение 4. *Пусть L_0 – множество адресов данных, расположенных в кэширующем буфере перед исполнением первой инструкции тестового шаблона. Тогда*

- *для инструкции с кэш-попаданием адреса x следует добавить следующую совокупность уравнений:*

$$\left[\begin{array}{l} x \in L_0 \wedge x \text{ все еще не вытеснен} \\ x \text{ внесен одним из кэш-промахов} \wedge \text{с тех пор не вытеснен} \end{array}\right.$$

- *для инструкции с кэш-промахом адреса x следует добавить следующую систему уравнений ($\{x_i\}$ – множество адресов данных в инструкциях с кэш-промахами, расположенными до текущей инструкции):*

$$\left[\begin{array}{l} x \notin L_0 \wedge x \notin \{x_1, x_2, ..., x_n\} \\ x \text{ был вытеснен} \wedge \text{не был больше внесен в буфер} \end{array}\right.$$

Формально показано, что утверждение 4 описывает все возможные сценарии появления и вытеснения данных в кэширующих буферах. Однако применение этих ограничений в данном виде для реальных микропроцессоров может быть ограничено из-за большого размера L_0 (что влечет к большому размеру ограничений и к невозможности разрешения таких больших ограничений доступными инструментами). Далее будет показано, как *совместное рассмотрение* тестовых ситуаций разных буферов и таблиц позволят существенно сократить размер этой формулы и обратиться к генерации ограничений для описания вытеснения.

2.1.2 Особенности исполнения инструкций обращения к памяти на современных микропроцессорах

В инструкции обращения к памяти в современных микропроцессорах задействована не одна подсистема. Исполнение инструкции обращения к памяти можно разбить на два этапа – подготовка физического адреса и собственно обращение с памятью (см.рис. 2.2).

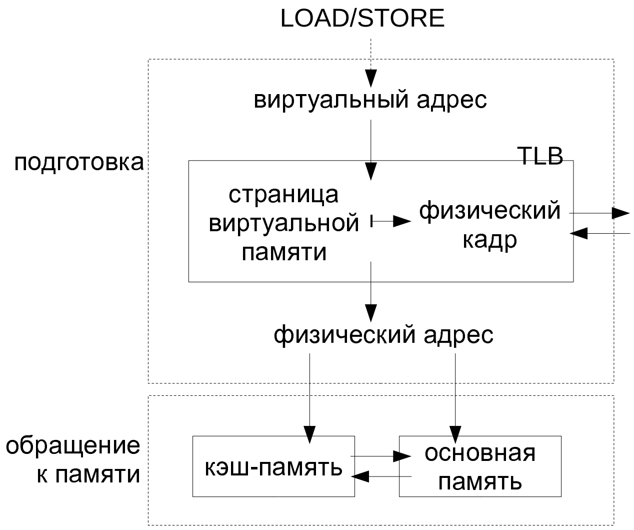


Рис. 2.2. Модель исполнения инструкции обращения к памяти

Подготовка физического адреса включает в себя формирование *виртуального адреса данных*, с которыми необходимо выполнить операцию (некоторые архитектуры сначала вычисляется *эффективный адрес*, затем на его основе виртуальный, а на его основе физический). Виртуальный адрес формируется на основе аргументов инструкции. Формирование физического адреса на основе виртуального адреса производится с использованием TLB. По сути в виртуальном адресе выделяется номер страницы виртуальной памяти и смещение внутри этой страницы, для страницы виртуальной памяти выбирается соответствующий физический кадр, используя TLB, и, наконец, физический адрес составляется из полученного номера физического кадра и смещения внутри страницы (оно берется из виртуального адреса). TLB содержит некоторое количество пар, задающих соответствие номера страницы виртуальной памяти и номера физического кадра. Размер самой страницы в виртуальной памяти и физической памяти совпадает, поэтому смещение внутри страницы используется в физическом адресе без изменений по сравнению с виртуальным адресом.

Когда физический адрес готов, осуществляется обращение к памяти: загрузка данных из памяти или сохранение данных в памяти. При этом если данные по физическому адресу имеются в кэш-памяти, основная память может остаться неизменной. Это сделано для повышения эффективности работы с основной памятью.

2.1.3 Уровни генерации тестовых данных

Тестовая программа некоторым специальным образом меняет состояние микропроцессора. Однако для того, чтобы это исполнение было согласовано с тестовым шаблоном, необходимо перед исполнением инструкций тестового шаблона перевести микропроцессор в некоторое специальное состояние (изменить значения в регистрах, в ячейках кэш-памяти и TLB, возможно в ячейках оперативной памяти). Практика показала, что изменение значения в одном регистре делается одной инструкцией, которая не затрагивает остальной части микропроцессора, кроме данного регистра. Таким образом, изменение значений в регистрах можно проводить последовательностью инструкций, каждая из которых меняет один регистр. Значения, которые надо поместить в регистры, входят в модель ограничений, генерируемых по тестовому шаблону. По-иному ведут себя такие подсистемы, как кэш-память или TLB, поскольку инструкции, которые изменяют их состояние независимо от остальной части микропроцессора, могут отсутствовать. Иными словами, одна инструкция может изменить состояние и кэш-памяти, и TLB.

С точки зрения генерации ограничений эта особенность выражается в возможности выделения частных случаев задачи, генерирования некоторых специальных ограничений, которые кроме описания инструкций будут соответствовать некоторому заданному способу подготовки состояния микропроцессора. Переменными, на которые формулируются такие ограничения, называются *тестовыми данными*, а задача их вычисления – задачей *генерации тестовых данных*. Эта задача может быть представлена в следующих формах:

- *простая форма:* найти начальное состояние микропроцессора (тестовыми данными являются содержимое кэш-памяти, TLB и других подсистем и значения регистров); генерирование инструкций, приводящих кэш-память и TLB в это состояние, не входит в ограничения и должно выполняться после разрешения ограничений (т.е. получения тестовых данных); при выполнении тестовой программы сгенерированные инструкции инициализации микропроцессора могут быть исполнены некорректно и тогда на инструкциях тестового шаблона могут не проявиться действительные ошибки или появиться ложные ошибки;

- *минимальная форма:* найти лишь значения регистров, используя данное начальное состояние (содержимое) кэш-памяти, TLB и других подсистем (тестовыми данными являются только значения регистров); инструкции, подготавливающие состояние микропроцессора, не меняют кэш-память и TLB, но в такой форме задача генерации тестовых данных может быть неразрешима (при разрешимой другой форме);

- *смешанная форма:* требуется построить значения регистров и последовательность инструкций инициализации состояния микропроцессора (тестовыми данными являются значения регистров и аргументы инструкций инициализации); эта форма является компромиссом между простой и минимальной формой, правда в такой форме увеличивается сложность задачи, потому что невозможно заранее предугадать, сколько необходимо и достаточно дополнительных инструкций.

Задачу поиска тестовых данных в минимальной форме будем называть задачей генерации тестовых данных *нулевого уровня*. Дальнейшие уровни определяются возможностью изменять кэш-память и другие подсистемы разными инструкциями. Например, для архитектуры MIPS [35] были выделены следующие уровни генерации тестовых данных помимо нулевого уровня:

- на *первом уровне* разрешается менять те строки TLB, которые не кэшированы в буфере TLB; изменение одной строки можно делать независимо от остальных строк и буфера одной инструкцией (TLBWI);

- на *втором уровне* разрешается менять любую строку TLB; при этом кроме смены строк, не входящих в буфер TLB, нужно переинициализировать содержимое буфера (на каждую строку отдельная инструкция);

- на *третьем уровне* разрешается менять и TLB, и кэш-память.

Чем больше уровень, тем длиннее будет инициализирующая программа и тем сложнее ее построить.

2.1.4 Модульный алгоритм генерации тестовых данных

На основе представленной модели инструкции обращения к памяти можно составить ограничения для каждого шага, получив тем самым *модульный алгоритм* генерации тестовых данных. Более формально, пусть $\{(I_i, R_i, \{As\}_i$ – тестовый шаблон, $\{I_i\}_{i=1,2,...,n}$ – последовательность инструкций, R_i – регистр с данными, $\{As\}_i$ – параметры инструкции, задающие адрес в памяти, $\{C_i\}_{i=1,2,...,n}$ – последовательность тестовых ситуаций в кэш-памяти, $\{T_i\}_{i=1,2,...,n}$ – последовательность тестовых ситуаций в TLB. Поскольку TLB может содержать дополнительные буфера, ведущие себя как кэш-память, то в TLB также возможны кэш-попадания и кэш-промахи. Тогда инструкция может быть представлена в виде следующих уравнений для каждого i:

$$\begin{cases} v_i = CalculateVirtualAddress(\{As\}_i) \\ AddressTranslation(T_i,\ p_i,\ v_i,\ TLB_0,\ \{v_1, ..., v_{i-1}\}) \\ CacheAccess(C_i,\ p_i,\ L_0,\ \{p_1, ..., p_{i-1}\}) \\ MemoryAccess(I_i,\ R_i,\ p_i,\ \{p_1, ..., p_{i-1}\},\ \{R_1, ..., R_{i-1}\}) \end{cases}$$

где v_i и p_i – новые переменные, TLB_0 – начальное состояние (содержимое) TLB, L_0 – начальное состояние (содержимое) кэш-памяти, $CalculateVirtualAddress$ – функция, вычисляющая виртуальный адрес на основе аргументов инструкции. $AddressTranslation$ – предикат, описывающий трансляцию виртуального адреса в физический (здесь может быть задействован TLB). $CacheAccess, MemoryAccess$ – предикаты, описывающие обращение в память (в первом может быть задействована кэш-память, во втором – основная память).

Поскольку система уравнений для тестового шаблона составляется как конъюнкция систем для каждой инструкции, то система из предикатов может быть выделена в отдельные подзадачи (в этом проявляется модульность). Таким образом выделяются следующие подзадачи:

- *задача на TLB*

$$\begin{cases} AddressTranslation(T_1,\ p_1,\ v_1,\ TLB_0,\ \varnothing) \\ AddressTranslation(T_2,\ p_2,\ v_2,\ TLB_0,\ \{v_1\}) \\ ... \\ AddressTranslation(T_n,\ p_n,\ v_n,\ TLB_0,\ \{v_1, ..., v_{n-1}\}) \end{cases}$$

- *задача на кэш-память*

$$\begin{cases} CacheAccess(C_1, p_1, L_0, \varnothing) \\ CacheAccess(C_2, p_2, L_0, \{p_1\}) \\ ... \\ CacheAccess(C_n, p_n, L_0, \{p_1, ..., p_{n-1}\}) \end{cases}$$

- *задача на основную память*

$$\begin{cases} MemoryAccess(I_1,\ R_1,\ p_1,\ \varnothing, \varnothing) \\ MemoryAccess(I_2,\ R_2,\ p_2,\ \{p_1\}, \{R_1\}) \\ ... \\ MemoryAccess(I_n,\ R_n,\ p_n,\ \{p_1, ..., p_{n-1}\}, \{R_1, ..., R_{n-1}\}) \end{cases}$$

Задача на основную память задает соответствие между значениями регистров, физическими адресами и значениями ячеек оперативной памяти. Если представить основную память в виде одномерного массива $memory$, индексация в котором идет по физическим адресам, то

- для инструкции, осуществляющей загрузку из памяти, $MemoryAccess$ можно представлять как $R_i := memory[physicalAddress_i]$;

- для инструкции, осуществляющей сохранение в памяти, $MemoryAccess$ можно представлять как $memory[physicalAddress_i] := R_i$.

Таким образом, получается последовательность присваиваний, которая может быть преобразована в систему уравнений с помощью *редукции Аккермана* (или *аккерманизации*) [40]. А именно,

- для каждой упорядоченной пары инструкций (не обязательно находящиеся подряд в тестовом шаблоне, но в том же порядке)

$STORE(R_1, p_1)$ и $LOAD(R_2, p_2)$ создается ограничение

$$(p_1 = p_2 \wedge p_2 \notin \{p_{(1)}, p_{(2)}, ..., p_{(k)}\}) \rightarrow R_1 = R_2$$

где $p_{(1)}, p_{(2)}, ..., p_{(k)}$ – физические адреса инструкций $STORE$, расположенных между двумя инструкциями этой пары;

- для каждой упорядоченной пары инструкций (не обязательно находящиеся подряд в тестовом шаблоне, но в том же порядке) $LOAD(R_1, p_1)$ и $LOAD(R_2, p_2)$ создается ограничение

$$(p_1 = p_2 \wedge p_2 \notin \{p_{(1)}, p_{(2)}, ..., p_{(k)}\}) \rightarrow R_1 = R_2$$

где $p_{(1)}, p_{(2)}, ..., p_{(k)}$ – физические адреса инструкций $STORE$, расположенных между двумя инструкциями этой пары.

Задача на TLB должна задавать в виде ограничений соответствие между начальным состоянием (содержимым) TLB, виртуальными адресами, физическими адресами и вносимыми в TLB соответствиями в случае промаха. Поскольку содержимое TLB также может быть рассмотрено в виде массива записей, то для задачи на TLB тоже применима аккерманизация. Кроме того, здесь также могут быть использованы методы построения уравнений на множества тегов для описания тестовых ситуаций на буферы, которые ведут себя как кэш-память, если таковые присутствуют в TLB.

Для разрешения полученных ограничений применяется решатель CSP (Constraint Satisfaction Problem) [38]. Сначала надо дать определение, что такое CSP. Пусть $X = \{x_1, x_2, ..., x_n\}$ – множество переменных (для каждой переменной известна область определения – обычно это конечное множество или ограниченный интервал). CSP состоит из предикатов (*ограничений*) $C(y_1, y_2, ..., y_N)$, где $y_i \in X$ (см. рис. 2.3).

Решение задачи заключается в поиске значений переменных из областей определения, на которых выполнены все предикаты. Основной методикой решения CSP является *constraint propagation*, а именно итеративное построение новых ограничений на основе данного в задаче множества ограничений (логических следствий). Если в процессе constraint propagation будет построено тождественно ложное ограничение, то CSP считается *несов-*

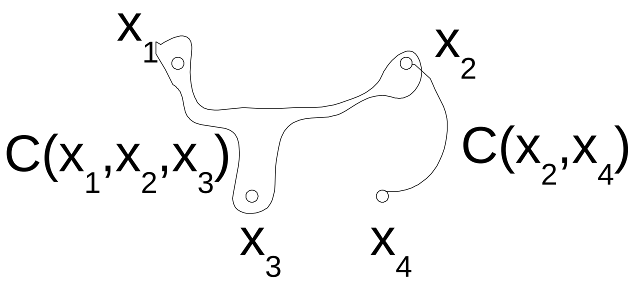


Рис. 2.3. Constraint Satisfaction Problem

местной. Иными словами, для ее переменных не существует значений, при которых выполнены все ограничения. Особо обращается внимание на одноместные ограничения, поскольку с помощью них уменьшается область определения переменной. Если constraint propagation не привел к тождественно ложному ограничению, то если области определения уменьшены до единственного значения, то это значение и будет ответом. Если же в области определения всё ещё много значений, то для выбора из области определения используются различные техники перебора (последовательный перебор, перебор в случайном порядке, метод ветвей и границ). Эвристические алгоритмы решения CSP обычно чередуют этапы перебора значений и constraint propagation. Одними из таких алгоритмов являются алгоритмы семейства MAC (Maintaining Arc Consistency) [38]. Одним из важных направлений развития CSP стала интеграция с парадигмой логического программирования – результат этого слияния именуют CLP (Constraint Logic Programming) [13]. Примеры систем CLP – SICStus Prolog [26], ILOG [37], ECLiPSe [13].

Достоинством модульного алгоритма является простота построения ограничений. Другим достоинством является его гибкость по отношению к механизмам работы подсистем микропроцессора. Эти свойства успешно использованы в инструменте Genesys-Pro [11] от компании IBM. Цель инструмента – генерация тестовых программ по данным тестовым шаблонам. Тестовые шаблоны позволяют задать инструкции тестовой программы, ограничения на их аргументы и некоторые параметры генерации аргументов. Тестовые программы строятся итеративно по одной инструкции. А именно цель одного шага – сгенерировать аргументы для очередной инструкции. Для этого на основе аргументов инструкции и модели состояния микропроцессора перед инструкцией составляется CSP, описывающая тестовую ситуацию.

Если эта CSP совместна, она разрешается с получением аргументов инструкции, инструкция с построенными аргументами исполняется, фиксируется состояние микропроцессора после этого и генерация продолжается со следующей инструкции. Если эта CSP несовместна, происходит возврат к предыдущей инструкции с целью сгенерировать для нее другие аргументы. Тестовый шаблон может содержать указание эвристики для выбора значения для переменной в ее области определения. Кроме того тестовые шаблоны могут содержать указания повторить некоторую последовательности инструкции некоторое количество раз. Для тестирования механизмов трансляции этот инструмент содержит специальный генератор ограничений DeepTrans [12]. Эффективность генерации тестовых программ падает с усложнением тестовых шаблонов. В крайнем случае вместо эффективного constraint propagation инструмент будет перебирать всевозможные начальные состояния микропроцессора, пока не подберется допустимый тестовым шаблоном.

Модульный алгоритм требует продвинутый решатель CSP, заточенный под особенности генерации тестовых данных для тестовых шаблонов (как минимум такие ограничения могут включать битовые операции). Подобный решатель был разработан в IBM для инструмента Genesys-Pro [15]. Создание такого решателя – отдельное сложное исследование, которое не входило в цели данного исследования. В данной работе было принято решение использовать доступные существующие решатели (не обязательно CSP), а сосредоточиться на упрощении генерируемых ограничений для некоторых частных случаев архитектур. Кроме наличия битовых операций, ограничения усложняются за счет огромных областей определения и размерности переменных. Например, кэш-память может содержать порядка $10^4 - 10^5$ тегов – такие размерности могут вылиться в невозможность даже просто хранить в памяти ограничения на такое большое количество переменных.

2.1.5 Метод совместной генерации ограничений

Введем понятие «тегсет» и с помощью него выразим тестовые ситуации в кэширующих буферах. Обращение к кэширующему буферу по данному адресу осуществляется на основе тега и индекса, которые вычисляются на основе адреса. По индексу из всех секций кэширующего буфера выбираются пары «(тег, значение)». Далее осуществляется поиск тега адреса среди тегов выбранных пар. Зачастую тег адреса и индекс адреса вычисляются как битовые поля адреса. Битовую конкатенацию тега и индекса будем называть *тегсетом* адреса. Если кэширующий буфер является полностью ассоциативным, то тегсет совпадает с тегом адреса.

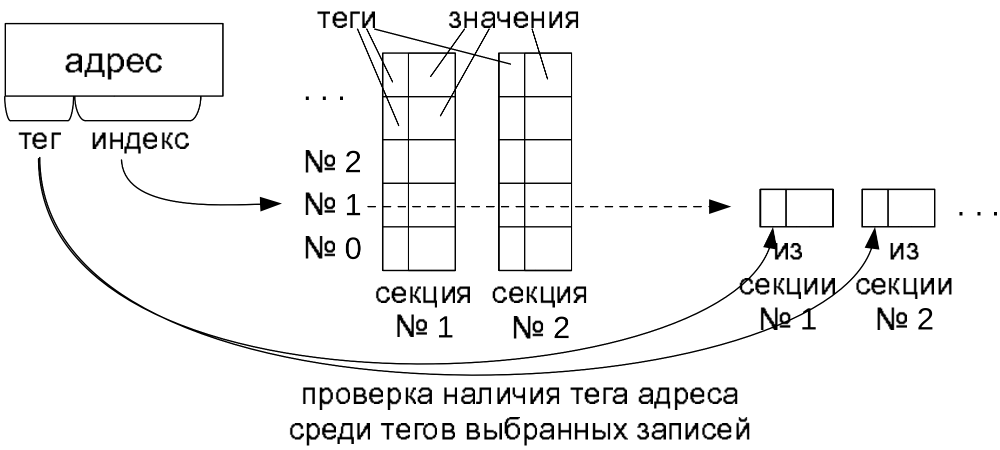


Рис. 2.4. Тег и индекс адреса

В микропроцессорах зачастую тегсет является инвариантом при обращениях в разные уровни кэш-памяти (обычно это делается для того, чтобы не менялись оставшиеся биты физического адреса, они задают смещение в строке кэш-памяти, и постоянство этих бит позволяет легко перемещать строки кэш-памяти между разными уровнями).

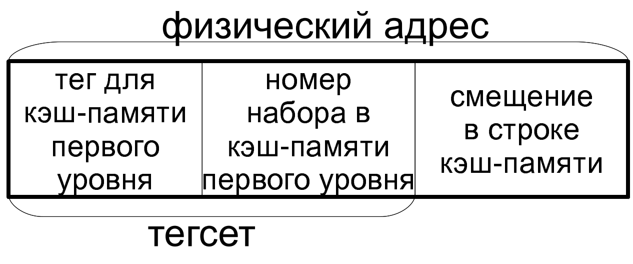


Рис. 2.5. Тегсет физического адреса

Тегсеты могут быть использованы для представления тестовых ситуаций в кэширующих буферах с использованием ограничений таким же образом, как это делалось для тегов: $x \in L$ для кэш-попадания и $x \notin L$ для кэш-промаха, где x – тегсет адреса, а L – множество тегсетов данных, хранящихся в кэширующем буфере перед исполнением инструкции. Множество тегсетов составляется битовой конкатенацией тега и индекса в каждом элементе начального содержимого кэширующего буфера. Для тегсетов аналогичным образом формулируются и доказываются лемма 1 и теорема 2 об ограничениях для тестовых ситуаций в кэширующих буферах, сформулированных уже на тегсетах.

Рассмотрим следующее представление тестового шаблона, которое назовем *схемой последовательностей тестовых ситуаций*. По одной оси будут располагаться инструкции тестового шаблона, по другой оси – кэширующие буферы микропроцессора (см.рис. 2.6). На пересечении инструкции и буфера будет помещаться переменная – тег или тегсет, если для этой инструкции в тестовом шаблоне есть тестовая ситуация на обращение в этот кэширующий буфер. Будем считать, что в исполнении инструкции задействованы те кэширующие буферы, тестовые ситуации на которые указаны в тестовом шаблоне для этой инструкции. Схема последовательностей тестовых ситуаций позволяет увидеть имеющиеся в тестовом шаблоне *совместные* обращения в кэширующие буферы, увидеть последовательности обращений к отдельным буферам, таким образом оценить сложность будущих ограничений.

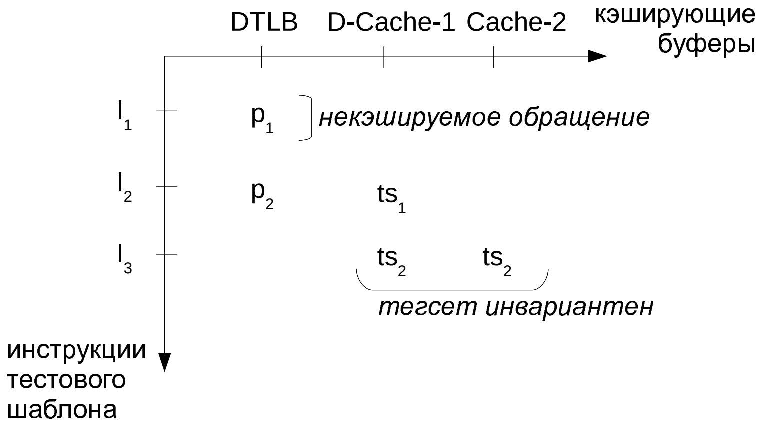


Рис. 2.6. Пример схемы последовательностей тестовых ситуаций

Для каждой последовательности тестовых ситуаций вводятся переменные-теги или тегсеты, а генерируемые для тестового шаблона ограничения состоят из ограничений для каждой такой переменной и ограничения, описывающие отношения введенных переменных. Этот процесс можно выразить следующей последовательностью шагов:

1. составить модель поведения MMU (выделить кэширующие буферы и таблицы);

2. выделить последовательности тестовых ситуаций в тестовом шаблоне (составить *схему последовательностей тестовых ситуаций*);

3. ввести переменные-теги тестовых ситуаций; если возможно, уменьшить количество переменных, заменив некоторые теги на тегсеты;

4. выделить последовательности тестовых ситуаций, для записи которых потребуются большие массивы данных;

5. построить ограничения для каждого тегсета – при обращении к большим массивам данных строить *совместные ограничения.*

Последний шаг требует пояснения. Пусть имеются две тестовые ситуации на кэширующие буферы. Упорядочим их в таком порядке, что данные, полученные из первого буфера, связаны с тегом обращения ко второму кэширующему буферу (см.рис. 2.7). Причем первый кэширующий буфер подчинен некоторой таблице.

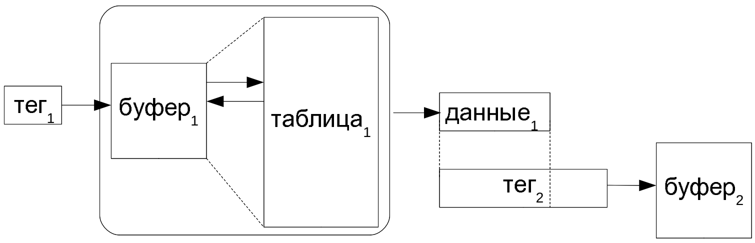


Рис. 2.7. Совместные обращения в буферы

В ограничениях появляется содержимое буферов целиком (иначе можно просто выписать ограничения на первый и на второй буфер без изменения). Согласно теореме 2 для кэш-попадания в первом буфере можно записать ограничения:

$$\left[\begin{array}{l} t_1 \in T_1 \wedge ... \\ t_1 = t_2 \wedge ... \end{array}\right.$$

Значит, при этом тег принадлежит либо множеству тегов первого буфера, либо множеству тегов из таблицы, которой подчинен первый буфер. Из этого следует, что данные – результат обращения в первый буфер – либо принадлежат данным из буфера (TD_1), либо принадлежат данные из таблицы (FD_1). Для кэш-промаха ограничения:

$$\left[\begin{array}{l} t_1 \notin T_1 \wedge ... \\ t_1 = t'_2 \wedge ... \end{array}\right.$$

Соответственно, либо тег не принадлежит множеству тегов первого буфера, но принадлежит множеству тегов в таблице, которой подчинен первый буфер, либо тег принадлежит тегам той же таблицы (т.к. этим тегам принадлежит t'_2). Значит, при кэш-промахе данные, получаемые после обращения в первый кэширующий буфер, либо принадлежат $FD_1 \setminus TD_1$, либо принадлежат FD_1.

Далее, учтем, что биты данных, полученных из первого буфера, связаны с битами тега для обращения во второй буфер. Для обращения во второй кэширующий буфер снова выпишем ограничения, согласно теореме 2. Для кэш-попадания в одну из конъюнкций входит большой массив тегов второго буфера:

$$\left[\begin{array}{l} x_1 \in X_1 \wedge ... \\ x_1 = x_2 \wedge ... \end{array}\right.$$

Но поскольку тег связан с данными, полученными из первого буфера, конъюнкция ограничений позволяет сократить множество тегов L, оставив только те, которые подходят под множество констант, участвующих в ограничении для обращения в первый буфер. Например, вместо ограничения $x \in L$ если данные из первого буфера d являются битовым полем x (например, номер физического кадра является битовым полем тегсета при обращении в кэш-память) и $d \in DD$ можно записать ограничение $x \in L \cap [DD]$, т.е. $x \in \{\lambda | \lambda \in L \wedge \lambda_{\text{биты данных}} \in DD\}$. Соответствующее упрощение можно сделать и для d: $d \in L \widehat{\cap [DD]}$, т.е. $d \in \{\delta | \delta \in DD \wedge \exists x \in L : \delta = x_{\text{биты данных}}\}$. Множества констант $L \cap [DD]$ и $L \widehat{\cap [DD]}$ могут быть вычислены до генерации ограничений. Обычно множество $L \cap [DD]$ имеет значительно меньший размер, чем L, что позволяет существенно сократить размер ограничений. В этом и заключается основной эффект применения совместной генерации ограничений. Для кэш-промаха получаются похожие ограничения, только вместо $x \notin L$ будет $x \notin L \cap [DD]$.

Надо быть аккуратным, ведь не всегда таблицы и буферы имеют действительно большой размер и иногда множество тегов, которое входит в ограничения, может быть выписано целиком. Например, при рассмотрении конъюнкции следующих подформул (первая получена от кэш-попадания в кэш-памяти по тегу x, вторая получена от тестовой ситуации в TLB, $\hat{x}$ – номер физического кадра, входящий в x):

$$\begin{cases} x = x_i \wedge ... \\ \hat{x} \in PFN \wedge ... \end{cases}$$

искать множество L и пересекать его с PFN не нужно, потому как PFN небольшого размера выписывается целиком (а большого размера и не участвует в ограничения – см.теорему 2).

2.2 Зеркальная генерация тестовых данных

Метод совместной генерации ограничений позволяет эффективно построить тестовую программу, если для каждой инструкции имеется более одного задействовано кэширующего буфера. Однако возможны случаи, например, неотображаемого кэшируемого обращения в микропроцессоре с TLB и кэш-памятью большого размера, когда в ограничениях, генерируемых согласно теореме 2, невозможно уменьшить размер L_0.

Другой случай – это так называемая *VIVT кэш-память* (virtually indexed virtually tagged) [25]. В этой кэш-памяти данные снабжены тегами виртуального адреса (в virtually indexed physically tagged кэш-памяти данные снабжены тегами физического адреса). Такая кэш-память в основном применяется для кэширования инструкций. Эта кэш-память характеризуется тем, что обращение к кэш-памяти первого уровня не требует предварительной трансляции виртуального адреса в физический. Однако ограничения, генерируемые согласно теореме 2, для кэш-попадания нет возможности выделить совместное обращение (для кэш-промаха совместное обращение возможно).

Противоположный случай – когда составить систему ограничений методом совместной генерации можно, но эта система оказывается несовместной.

Однако если архитектура микропроцессора позволяет изменять кэширующие буферы с помощью отдельных инструкций (возможно, при особом значении некоторых регистров микропроцессора или области виртуальной памяти), то можно воспользоваться этими инструкциями для добавления в кэширующий буфер данных по тем тегам, которые будут использованы в тестовой программе (а их уже можно выбирать произвольно, что сильно упростит систему ограничений). При этом можно не задумываться над тем, были ли эти теги в L_0 или нет. Иными словами, если обращение к кэширующему буферу по некоторому тегу должно быть успешным, то перед этим обращением должно быть другое обращение по этому же тегу, после которого данные по этому тегу не вытесняются до нужного обращения. Дополнительных ограничений, кроме уже упомянутых, на тег не накладывается. Если обращение к кэширующему буферу по некоторому тегу должно быть неуспешным, то перед этим обращением всё равно должно быть другое обращение по этому же тегу, после которого однако данные по этому тегу должны быть вытеснены и не положены вновь до нужного обращения. Таким образом, у каждого тега в тестового шаблона есть свой «зеркальный» тег среди предыдущих тегов тестового шаблона или дополнительных тегов инициализирующей программы.

Более формально, для данной последовательности тестовых ситуаций для кэширующего буфера (S_i, x_i), где $i = 1, 2, ..., n$, S_i – hit или miss, x_i – тег данных, требуется построить последовательность тегов t_j (*инициализирующая последовательность тегов*), $j = 1, 2, ..., m$, которые обеспечивают данную последовательность тестовых ситуаций. Согласно зеркальному методу для каждого данного тега x_i при $S_i = \text{hit}$ надо составить систему уравнений

$$\begin{cases} x_i \in \{t_1, ..., t_m, x_1, ..., x_{i-1}\} \\ x \text{ не вытеснен с момента последнего к нему} \\ \quad \text{обращения в } t_1, ..., t_m, x_1, ..., x_{i-1} \end{cases}$$

а при $S_i = \text{miss}$ надо составить систему уравнений

$$\begin{cases} x_i \in \{t_1, ..., t_m, x_1, ..., x_{i-1}\} \\ x \text{ вытеснен и не добавлен с момента последнего} \\ \quad \text{к нему обращения в } \ t_1, ..., t_m, x_1, ..., x_{i-1} \end{cases}$$

Лемма 2. *Если существует решение для некоторого m, то существует решение и для $m + 1$.*

Доказательство. Достаточно взять $t_{m+1} = t_m$. □

Ниже будет показано, что достаточно рассматривать m, ограниченные линейной функцией от n и w (ассоциативности кэширующего буфера). Поэтому может быть поставлена задача минимизации параметра m (длины инициализирующей программы). Это увеличит качество тестирования, поскольку уменьшит влияние дополнительных, инициализирующих, инструкций на исполнение инструкций тестового шаблона. Минимизация может быть эффективно выполнена с использованием двоичного поиска оптимального значения m (лемма 2 показывает корректность применения двоичного поиска) – границу сверху для значения m дает теорема 4.

Утверждение 5 (Применимость зеркального метода)**.** *Зеркальный метод генерации ограничений применим к данному тестовому шаблону для данной архитектуры микропроцессоров, если система команд микропроцессора содержит инструкции, позволяющие (при определенных условиях) изменение задействованных в тестовом шаблоне кэширующих буферов по отдельности от остальных кэширующих буферов.*

Например, в архитектуре MIPS [35] инструкции обращения к памяти могут быть исполнены:

- в некэширующем отображаемом режиме – это позволяет изменять буфер данных TLB отдельно от кэш-памяти;
- в кэшируемом неотображаемом режиме – это позволяет изменять кэш-память данных отдельно от буфера данных TLB.

При этом инструкции могут быть исполнены в кэшируемом или отображаемом режиме по отношению к кэш-памяти инструкций или буферу инструкций TLB. Для возможности применения зеркального метода в случае, когда тестовый шаблон содержит тестовые ситуации на кэш-память данных и на кэш-память инструкций, надо выбирать расположение инициализирующих инструкций в памяти так, чтобы при исполнении каждой такой инструкции был задействован всего один кэширующий буфер.

Как будет показано далее, этих условий хватает для того, чтобы зеркальный метод генерации ограничений был корректным, но не хватает для его полноты. Однако для наиболее часто используемых в микропроцессорах стратегий вытеснений зеркальный метод всё же является полным.

2.2.1 Корректность зеркального метода

Далее формулируется и доказывается теорема о корректности зеркального метода генерации ограничений. Она формально обоснует применение зеркального метода для генерации тестовых программ по тестовым шаблонам.

Лемма 3 (Существование последнего вытеснения)**.** *Пусть* $(S_i, x_i), i = 1, 2, \ldots, n$ *– последовательность тегов с тестовыми ситуациями (если* $S_i = miss$*, символом* x'_i *будет обозначаться вытесняемый тег). Тогда если* $S_n = miss$ *и* $x_n = x'_j$ *для некоторого* $j \in [1, n-1]$*, то существует* $k \in [j, n-1]$ *такой, что* $x_n = x'_k$ *и* $x'_k \notin [x_{k+1}, \ldots, x_{n-1}]_{miss}$ *(*$[x_{k+1}, \ldots, x_{n-1}]_{miss} \equiv \{x_p | p \in \{k+1, \ldots, n-1\} \wedge S_p = miss\}$*).*

Доказательство. Докажем от противного. Допустим, что такого k не существует. Тогда получается, что для любого $l \in [j, n-1]$ справедлива дизъюнкция $x_n \neq x'_l$ или $x'_l \in [x_{l+1}, \ldots, x_{n-1}]_{\mathrm{miss}}$. В частности для $l = j$ получаем $x_n \neq x'_j \vee x'_j \in [x_{j+1}, \ldots, x_{n-1}]_{\mathrm{miss}}$. Так как по условию $x_n = x'_j$, то получаем, что существует такой $j_1 \in [j+1, \ldots, n-1]$, что $x'_j = x_{j_1}$. Так как $x_n = x'_j$, то $x_n = x_{j_1}$. Тогда существует $j_2 \in [j_1 + 1, \ldots, n-1]$ такой, что $x'_{j_2} = x_{j_1}$ (в противном случае $x_n \notin \{x'_{j_1+1}, \ldots, x'_{n-1}\}$, что означает ситуацию, когда x_n не будет вытеснен и останется в буфере, а это противоречит тому, что по условию $S_n = \mathrm{miss}$). Таким образом, существует j_2 такое, что $j_2 < n$ и $j_2 > j_1$ и $x_n = x'_{j_2}$. Получено то же условие, что и для j. Рассуждая аналогично, получим целую последовательность $j_3, j_4, \ldots$. Поскольку $j_3 < j_4 < \cdots < n$ (возрастающая последовательность, ограниченная сверху), то существует j^* такой, что $x_n = x_{j^*}$ и $x_{j^*} \notin \{x_{j^*+1}, \ldots, x_n\}$ (в противном случае будет нарушено $S_n = \mathrm{miss}$). Однако по предположению $x_n \neq x_{j^*}$ или $x_{j^*} \in \{x_{j^*+1}, \ldots, x_n\}$. Противоречие. □

Теорема 3 (Корректность зеркального метода)**.** *Если зеркальный метод генерации ограничений применим, то тестовая программа, построенная по зеркальному методу удовлетворяет требованиям тестового шаблона (тестовым ситуациям инструкций).*

Доказательство. Пусть $t_1, t_2, \ldots, t_m$ – последовательность инициализирующих тегов, $(S_1, x_1), (S_2, x_2), \ldots, (S_n, x_n)$ – тестовый шаблон, $x_1, x_2, \ldots, x_n$ – значения тегов, построенная в результате разрешения ограничений, сгенерированных согласно зеркальному методу.

Без потери общности будем доказывать теорему для произвольного x_i из $\{x_1, x_2, \ldots, x_n\}$. Наша задача – показать, что значение x_i, сгенерированное с помощью зеркального метода, соответствует тестовой ситуации S_i.

Далее будут использованы следующие обозначения для произвольной последовательности тегов $\alpha_1, \alpha_2, ..., \alpha_N$: $[\alpha_1, \alpha_2, ..., \alpha_N]$ – это подпоследовательность последовательности $\alpha_1, \alpha_2, ..., \alpha_N$, состоящая только из тех тегов, обращения к которым дают кэш-промах (*подпоследовательность вытесняющих тегов*); $[\alpha_1, \alpha_2, ..., \alpha_N]'$ – это последовательность всех тегов, вытесняемых тегами из $[\alpha_1, \alpha_2, ..., \alpha_N]$ с сохранением порядка (*подпоследовательность вытесняемых тегов*).

Зафиксируем некоторое начальное состояние кэширующего буфера перед последовательностью инициализирующих тегов L_0. Тогда каждому элементу этой последовательности t_k можно приписать «тестовую ситуацию» R_k ($R_k = \text{hit}$ или $R_k = \text{miss}$) в зависимости от того, успешным или неуспешным было обращение к t_k в буфере.

Рассмотрим сначала случай $S_i =$ hit. По условию x_i сгенерирован с помощью зеркального метода, т.е. $x_i \in \{t_1, t_2, \ldots, t_m, x_1, x_2, \ldots, x_{i-1}\}$ и x_i не равен ни одному вытесняемому тегу после последнего обращения к x_i. Запишем это условие с использованием операций над множествами (от последовательности используется множество элементов):

$$\left[\begin{array}{l}
x_i = t_1 \wedge x_i \notin [t_2, t_3, \ldots, t_m]' \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\
x_i = t_2 \wedge x_i \notin [t_3, t_4, \ldots, t_m]' \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\
\ldots \\
x_i = t_m \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\
x_i = x_1 \wedge x_i \notin [x_2, x_3, \ldots, x_{i-1}]' \\
x_i = x_2 \wedge x_i \notin [x_3, x_4, \ldots, x_{i-1}]' \\
\ldots \\
x_i = x_{i-1}
\end{array}\right.$$

где последовательность $y_1, y_2, ..., y_p \equiv [x_1, x_2, ..., x_{i-1}]$, последовательность $y'_1, y'_2, ..., y'_p \equiv [x_1, x_2, ..., x_{i-1}]'$

Надо показать, что из этого условия следует следующее условие:

$$\left[\begin{array}{l} x_i \in L_1 \wedge x_i \notin \{y'_1, y'_2, ..., y'_p\} \\ x_i = y_1 \wedge x_i \notin \{y'_2, y'_3, ..., y'_p\} \\ x_i = y_2 \wedge x_i \notin \{y'_3, y'_4, ..., y'_p\} \\ ... \\ x_i = y_p \end{array}\right.$$

где L_1 – содержимое кэширующего буфера перед инструкциями тестового шаблона (после t_m).

По теореме 2 выражение $x_i \in L_1$ может быть переписано в следующем эквивалентном виде:

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_q\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_q\} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_q\} \\ ... \\ x_i = s_q \end{array}\right.$$

где последовательность $s_1, s_2, ..., s_q \equiv [t_1, t_2, ..., t_m]$, а последовательность $s'_1, s'_2, ..., s'_q \equiv [t_1, t_2, ..., t_m]'$.

Далее будет показано, что каждый элемент дизъюнкции 2.2.1 будет присутствовать среди элементов дизъюнкции 2.2.1, что и даст нужное обоснование.

Рассмотрим k'й элемент дизъюнкции 2.2.1 ($k = 1, 2, ..., m$). Если R_k = miss, то элемент дизъюнкции $x_i = t_k \wedge x_i \notin \{s'_{r+1}, s'_{r+2}, \ldots, s'_q\} \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\}$ входит целиком в дизъюнкцию 2.2.1 для r такого, что $t_k \equiv s_r$, так как $x_i = s_r \wedge x_i \notin \{s'_{r+1}, s'_{r+2}, \ldots, s'_q\}$ является частью $x_i \in L_1$.

Если R_k = hit, то по теореме 2 $x_i = t_k$ можно записать в следующем эквивалентном виде:

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_r\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_r\} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_r\} \\ ... \\ x_i = s_r \end{array}\right.$$

где s_r – ближайший предыдущих элемент к t_k (или более формально $t_k \in \{s_1, s_2, ..., s_{r+1}\} \setminus \{s_1, s_2, ..., s_r\}$). Таким образом, элемент дизъюнкции $x_i = t_k \wedge x_i \notin \{s'_{r+1}, s'_{r+2}, \ldots, s'_q\} \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\}$ эквивалентен дизъюнкции

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_q\} \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_q\} \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_q\} \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \\ ... \\ x_i = s_q \wedge x_i \notin \{y'_1, y'_2, \ldots, y'_p\} \end{array}\right.$$

Получен нужный вид части дизъюнкции 2.2.1.

Рассмотрим $k+m$'й элемент дизъюнкции 2.2.1 ($k = 1, 2, ..., i-1$). Если $S_k = \text{miss}$, то $k+m$'й элемент дизъюнкции 2.2.1 без изменений переходит в дизъюнкцию 2.2.1. Если $S_k = \text{hit}$, то по теореме 2 $x_i = x_k$ можно записать в следующем виде:

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_{p_i}\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_{p_i} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_{p_i} \\ ... \\ x_i = s_q \wedge \notin \{y'_1, y'_2, ..., y'_{p_i} \\ x_i = y_1 \wedge \notin \{y'_2, y'_3, ..., y'_{p_i} \\ x_i = y_2 \wedge \notin \{y'_3, y'_4, ..., y'_{p_i} \\ ... \\ x_i = y_{p_i} \end{array}\right.$$

где y_{p_i} – ближайший предыдущих элемент к x_k (или более формально $x_k \in \{y_1, y_2, ..., y_{p_i+1}\} \setminus \{y_1, y_2, ..., y_{p_i}\}$). Таким образом, элемент дизъюнкции $x_i =$

$x_k \wedge x_i \notin \{y'_{p_i+1}, y'_{p_i+2}, \ldots, y'_p\}$ эквивалентен дизъюнкции

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_p\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_p\} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_q\} \wedge \notin \{y'_1, y'_2, ..., y'_p\} \\ ... \\ x_i = s_q \wedge x_i \notin \{y'_1, y'_2, ..., y'_p\} \\ x_i = y_1 \wedge x_i \notin \{y'_2, y'_3, ..., y'_p\} \\ x_i = y_2 \wedge x_i \notin \{y'_3, y'_4, ..., y'_p\} \\ ... \\ x_i = y_{p_i} \wedge x_i \notin \{y'_{p_i+1}, y'_{p_i+2}, ..., y'_p\} \end{array}\right.$$

которая является частью дизъюнкции 2.2.1.

Теперь рассмотрим случай $S_i =$ miss. По условию x_i сгенерирован с помощью зеркального метода, т.е. $x_i \in \{t_1, t_2, \ldots, t_m, x_1, x_2, \ldots, x_{i-1}\}$ и x_i равен некоторому вытесняемому тегу после последнего обращения к x_i. Запишем это условие с использованием операций над множествами (от последовательности используется множество элементов):

$$\left[\begin{array}{l} x_i = t_1 \wedge (x_i \in [t_2, t_3, \ldots, t_m]' \vee x_i \in \{y'_1, y'_2, \ldots, y'_p\}) \\ x_i = t_2 \wedge (x_i \in [t_3, t_4, \ldots, t_m]' \vee x_i \in \{y'_1, y'_2, \ldots, y'_p\}) \\ ... \\ x_i = t_m \wedge (x_i \in \{y'_1, y'_2, \ldots, y'_p\}) \\ x_i = x_1 \wedge (x_i \in [x_2, x_3, \ldots, x_{i-1}]') \\ x_i = x_2 \wedge (x_i \in [x_3, x_4, \ldots, x_{i-1}]') \\ ... \\ x_i = x_{i-2} \wedge (x_i \in [x_{i-1}]') \end{array}\right.$$

где последовательность $y_1, y_2, ..., y_p \equiv [x_1, x_2, ..., x_{i-1}]$, а последовательность $y'_1, y'_2, ..., y'_p \equiv [x_1, x_2, ..., x_{i-1}]'$.

Надо показать, что из этого условия следует следующее условие:

$$\left[\begin{array}{l} x_i \notin L_1 \wedge x_i \notin \{y_1, y_2, ..., y_p\} \\ x_i = y'_1 \wedge x_i \notin \{y_2, y_3, ..., y_p\} \\ x_i = y'_2 \wedge x_i \notin \{y_3, y_4, ..., y_p\} \\ ... \\ x_i = y'_p \end{array}\right.$$

где L_1 – содержимое кэширующего буфера перед инструкциями тестового шаблона (т.е. после t_m).

По теореме 2 выражение $x_i \notin L_1$ может быть переписано в следующем эквивалентном виде:

$$\left[\begin{array}{l} x_i \notin L_0 \wedge x_i \notin \{s_1, s_2, ..., s_q\} \\ x_i = s'_1 \wedge x_i \notin \{s_2, s_3, ..., s_q\} \\ x_i = s'_2 \wedge x_i \notin \{s_3, s_4, ..., s_q\} \\ ... \\ x_i = s'_q \end{array}\right.$$

где последовательность $s_1, s_2, ..., s_q \equiv [t_1, t_2, ..., t_m]$, а последовательность $s'_1, s'_2, ..., s'_q \equiv [t_1, t_2, ..., t_m]'$. С учетом этого дизъюнкция 2.2.1 переписывается в виде:

$$\left[\begin{array}{l} x_i \notin L_0 \wedge x_i \notin \{s_1, s_2, ..., s_q\} \wedge x_i \notin \{y_1, y_2, ..., y_p\} \\ x_i = s'_1 \wedge x_i \notin \{s_2, s_3, ..., s_q\} \wedge x_i \notin \{y_1, y_2, ..., y_p\} \\ x_i = s'_2 \wedge x_i \notin \{s_3, s_4, ..., s_q\} \wedge x_i \notin \{y_1, y_2, ..., y_p\} \\ ... \\ x_i = s'_q \wedge x_i \notin \{y_1, y_2, ..., y_p\} \\ x_i = y'_1 \wedge x_i \notin \{y_2, y_3, ..., y_p\} \\ x_i = y'_2 \wedge x_i \notin \{y_3, y_4, ..., y_p\} \\ ... \\ x_i = y'_p \end{array}\right.$$

Далее будет показано, что каждый элемент дизъюнкции 2.2.1 будет присутствовать среди элементов дизъюнкции 2.2.1, что и даст нужное обоснование.

Сначала избавимся от тех элементов дизъюнкции 2.2.1 среди первых m элементов $x_i = t_k \wedge ...$, в которых $R_k = \text{hit}$. По теореме 2:

$$\left[\begin{array}{l} t_k \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_r\} \\ t_k = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_r\} \\ t_k = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_r\} \\ ... \\ t_k = s_r \end{array}\right.$$

где s_r – ближайший предыдущих элемент к t_k (или более формально $t_k \in \{s_1, s_2, ..., s_{r+1}\} \setminus \{s_1, s_2, ..., s_r\}$). С учетом этого равенство $x_i = t_k$ преобразуется эквивалентным образом в дизъюнкцию:

$$\left[\begin{array}{l} x_i \in L_0 \wedge x_i \notin \{s'_1, s'_2, ..., s'_r\} \\ x_i = s_1 \wedge x_i \notin \{s'_2, s'_3, ..., s'_r\} \\ x_i = s_2 \wedge x_i \notin \{s'_3, s'_4, ..., s'_r\} \\ ... \\ x_i = s_r \end{array}\right.$$

что дает возможность переписать эквивалентным образом дизъюнкцию 2.2.1 с использованием правила поглощения в следующем виде:

$$\left[\begin{array}{l} x_i \in L_0 \wedge (x_i \in \{y'_1, y'_2, ..., y'_p\}) \\ x_i = s_1 \wedge (x_i \in \{s'_2, s'_3, ..., s'_q\} \vee x_i \in \{y'_1, y'_2, ..., y'_p\}) \\ x_i = s_2 \wedge (x_i \in \{s'_3, s'_4, ..., s'_q\} \vee x_i \in \{y'_1, y'_2, ..., y'_p\}) \\ ... \\ x_i = s_m \wedge x_i \in \{y'_1, y'_2, ..., y'_p\} \\ x_i = x_1 \wedge (x_i \in [x_2, x_3, \ldots, x_{i-1}]') \\ x_i = x_2 \wedge (x_i \in [x_3, x_4, \ldots, x_{i-1}]') \\ ... \\ x_i = x_{i-2} \wedge (x_i \in [x_{i-1}]') \end{array}\right.$$

Сгруппируем элементы этой ДНФ следующим образом:

$$\left[\begin{array}{l} x_i = s'_2 \wedge (x_i \in \{s_1\} \vee x_i \in L_0) \\ x_i = s'_3 \wedge (x_i \in \{s_1, s_2\} \vee x_i \in L_0) \\ ... \\ x_i = s'_q \wedge (x_i \in \{s_1, s_2, ..., s_{q-1}\} \vee x_i \in L_0) \\ x_i \in \{y'_1, y'_2, ..., y'_p\} \wedge (x_i \in \{s_1, s_2, ..., s_q\} \vee x_i \in L_0) \\ x_i = x_1 \wedge (x_i \in [x_2, x_3, \ldots, x_{i-1}]') \\ x_i = x_2 \wedge (x_i \in [x_3, x_4, \ldots, x_{i-1}]') \\ ... \\ x_i = x_{i-2} \wedge (x_i \in [x_{i-1}]') \end{array}\right.$$

Рассмотрим каждый элемент этой дизъюнкции $x_i = s'_k \wedge (x_i \in \{s_1, s_2, ..., s_{k-1}\} \vee x_i \in L_0)$. По лемме 3 следует, что существует такой («последний») $s_{k'}$ ($k' \in [k+1, ..., i-1]$), что $s'_{k'} = s'_k$ и $s'_{k'} \notin \{s_{k'+1}, ..., s_q, y_1, ..., y_p\}$, или существует такой $y_{i'}$ ($i' \in [1, p]$), что $s'_k = y_{i'}$ и $s_k \notin \{y_{i'+1}, ..., y_p\}$. Это условие можно записать в виде дизъюнкции по всем k' из $[k+1, ..., i-1]$ и i' из $[1, p]$ следующим образом:

$$\left[\begin{array}{l} x_i = s'_2 \wedge x_i = s'_3 \wedge (x_i \notin \{s_4, s_5, ..., s_q, y_1, ..., y_p\} \\ x_i = s'_2 \wedge x_i = s'_4 \wedge (x_i \notin \{s_5, s_6, ..., s_q, y_1, ..., y_p\} \\ ... \\ x_i = s'_3 \wedge x_i = s'_4 \wedge (x_i \notin \{s_5, s_6, ..., s_q, y_1, ..., y_p\} \\ x_i = s'_3 \wedge x_i = s'_5 \wedge (x_i \notin \{s_6, s_7, ..., s_q, y_1, ..., y_p\} \\ ... \\ x_i \in \{y'_1, y'_2, ..., y'_p\} \wedge (x_i \in \{s_1, s_2, ..., s_q\} \vee x_i \in L_0) \\ x_i = x_1 \wedge (x_i \in [x_2, x_3, \ldots, x_{i-1}]') \\ x_i = x_2 \wedge (x_i \in [x_3, x_4, \ldots, x_{i-1}]') \\ ... \\ x_i = x_{i-2} \wedge (x_i \in [x_{i-1}]') \end{array}\right.$$

Преобразуем дизъюнкцию, используя тождество $s_k \neq s'_k$ для всех k:

$$\left[\begin{array}{l} x_i = s'_2 \wedge x_i \notin \{s_3, s_4, ..., s_q, y_1, ..., y_p\} \\ x_i = s'_2 \wedge x_i \notin \{s_4, s_5, ..., s_q, y_1, ..., y_p\} \\ ... \\ x_i = s'_3 \wedge x_i \notin \{s_4, s_5, ..., s_q, y_1, ..., y_p\} \\ x_i = s'_3 \wedge x_i \notin \{s_5, s_6, ..., s_q, y_1, ..., y_p\} \\ ... \\ x_i \in \{y'_1, y'_2, ..., y'_p\} \wedge (x_i \in \{s_1, s_2, ..., s_q\} \vee x_i \in L_0) \\ x_i = x_1 \wedge (x_i \in [x_2, x_3, \ldots, x_{i-1}]') \\ x_i = x_2 \wedge (x_i \in [x_3, x_4, \ldots, x_{i-1}]') \\ ... \\ x_i = x_{i-2} \wedge (x_i \in [x_{i-1}]') \end{array}\right.$$

Из этой системы следует искомая дизъюнкция (над $[x_k, x_{k+1}, \ldots, x_{i-1}]'$ надо выполнить те же преобразования, что проводились для $[t_k, t_{k+1}, \ldots, t_m]'$).

□

2.2.2 Полнота зеркального метода. Верхняя оценка длины инициализирующей программы

Далее сформулируем и докажем теорему о полноте зеркального метода генерации ограничений. Из нее будет следовать в частности то, что метод можно использовать для определения возможности построения хотя бы одной тестовой программы для данного тестового шаблона.

Будем называть тестовый шаблон *совместным*, если для него существует удовлетворяющая ему тестовая программа.

Теорема 4 (Полнота зеркального метода)**.** *Если данный тестовый шаблон является совместным, т.е. для последовательности тестовых ситуаций* $(S_1, x_1), (S_2, x_2), ..., (S_n, x_n)$ *и дополнительного ограничения* $P(x_1, x_2, ..., x_n)$ *при некотором начальном состоянии* L_1 *существует удовлетворяющая им последовательность тегов* $x_1, x_2, ..., x_n$*, применим зеркальный метод генерации ограничений и стратегия вытеснения позволяет рано или поздно вытеснить любой тег в буфере, то с помощью зеркального метода мо-*

жет быть построена система ограничений, имеющая решение для той же последовательности тегов $x_1, x_2, ..., x_n$.

Доказательство. Доказательство проведем указанием способа построения последовательности инициализирующих тегов $t_1, t_2, ..., t_m$ по известной последовательности $x_1, x_2, ..., x_n$. Выделим из последовательности $x_1, x_2, ..., x_n$ подпоследовательности, соответствующие одинаковому значению региона. Части тестового шаблона, соответствующие разным регионам, ведут себя независимо, поэтому и последовательность инициализирующих тегов будет составляться из подпоследовательностей инициализирующих тегов для подпоследовательностей последовательности $x_1, x_2, ..., x_n$. Далее без ограничения общности можно считать, что все $x_1, x_2, ..., x_n$ относятся к одному региону.

Сначала поместим в последовательность инициализирующих тегов **различные** теги в таком количестве, чтобы вытеснить все теги из кэширующего буфера, которые там были до первого инициализирующего тега (назовем эту последовательность «вытесняющей последовательностью»). В отличие от последовательности тегов тестового шаблона теги в вытесняющей последовательности не снабжаются указанием тестовой ситуации. Плюс к этому последовательность должна обеспечивать вытеснение при любом начальном состоянии кэширующего буфера L_0 (только в тот момент, когда L_0 станет известен, можно будет точно сказать, успешным или неуспешным будет обращение по тегу вытесняющей последовательности). Однако по условию теоремы стратегия вытеснения обеспечивает вытеснение любого тега L_0 за конечное количество обращений. Соответственно конечное же количество обращений **различных** тегов обеспечит вытеснение всех тегов L_0. Начиная с некоторого элемента вытесняющей последовательности, каждое обращение будет приводить к кэш-промаху (это можно сказать точно, даже не зная содержимого L_0).

Обязательно в эту последовательность вставим те теги из последовательности $x_1, x_2, ..., x_n$, которые должны давать кэш-промахи. Тем самым для этих тегов будет выполнено требование «зеркальности» (к тегу должно быть обращение и после этого тег должен быть вытеснен).

С помощью выбора подходящей вытесняющей последовательности можно добиться любого наперед заданного состояния буфера. Осталось добиться того, чтобы для еще не упомянутых тегов из $x_1, x_2, ..., x_n$ был выполнен зеркальный принцип. Для этого добавим после вытесняющей последовательности нужные теги, возможно, перемешивая их с тегами, чьи значения не используются среди $x_1, x_2, ..., x_n$, для помещения первых на нужные позиции в буфере. Эти позиции задаются определением стратегии вытеснения в зеркальном методе для кэш-попадания (т.е. правило того, что тег не должен быть вытеснен). Ограничения для таких правил будут разрешимы, поскольку существуют $x_1, x_2, ..., x_n$. □

Как определять, позволяет ли стратегия вытеснения вытеснить любой тег в наборе? (тем самым понять, полным ли будет зеркальный метод генерации ограничений для данной стратегии вытеснения) Для ответа на этот вопрос дадим более точное определение стратегии вытеснения, нежели просто «правило определения вытесняемого тега». Для этого воспользуемся *таблицами вытеснения* (*policy table*). Они были предложены в 2008 году исследователями из немецкого университета Саарланда [23]. Таблица вытеснения однозначно описывает изменение порядка и вытеснение тегов в наборе. Пример таблицы вытеснения (для стратегии вытеснения Pseudo-LRU) смотрите на рисунке 2.8.

$$\begin{bmatrix} \pi_0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \pi_1 & 1 & 0 & 2 & 3 & 4 & 5 & 6 & 7 \\ \pi_2 & 2 & 3 & 0 & 1 & 4 & 5 & 6 & 7 \\ \pi_3 & 3 & 2 & 0 & 1 & 4 & 5 & 6 & 7 \\ \pi_4 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ \pi_5 & 5 & 4 & 6 & 7 & 0 & 1 & 2 & 3 \\ \pi_6 & 6 & 7 & 4 & 5 & 0 & 1 & 2 & 3 \\ \pi_7 & 7 & 6 & 4 & 5 & 0 & 1 & 2 & 3 \\ \pi_m & m & 6 & 4 & 5 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Рис. 2.8. Таблица вытеснения для стратегии вытеснения Pseudo-LRU

Первые строки таблицы вытеснения описывают изменение порядка элементов набора при тестовых ситуациях кэш-попадания. Каждой такой строке соответствует свой случай кэш-попадания, при этом первый столбец показывает, на какой тег происходит кэш-попадание, а части строк, не включающие первый столбец, показывают, каким образом осуществляется перестановка тегов набор из последовательности индексов (0 1 2 3 4 5 6 7) (в примере). Например, для стратегии вытеснения Pseudo-LRU , представленной на рисунке 2.8, при кэш-попадании тега 5 набор (4 6 5 7 1 0 2 3) изменится на (смотрим строку с π_2, потому что тег 5 находится на втором месте) (5 7 4 6 1 0 2 3).

Последняя строка таблицы вытеснения соответствует ситуации кэш-промаха. Вытесняющий элемент набора помечается буквой m. Вытесняемый элемент – элемент набора (0 1 2 3 4 5 6 7) (в примере), который отсутствует в последней строке таблицы вытеснения (в примере – это 7, т.е. вытесняется последний элемент, а вытесняющий помещается на нулевое место – в этом стратегия вытеснения Pseudo-LRU проявляет схожесть со стратегией вытеснения LRU).

Таблица вытеснения позволит ответить на вопрос о возможности вытеснения любого тега в наборе. Для этого предлагается построить орграф, вершинами которого будут всевозможные состояния буфера (включая m), а дуги снабжены пометками – числом от 0 до $w-1$ или символом m. Две вершины соединены дугой с пометкой-числом, если из одной вершины в другую осуществляется переход в результате кэш-попадания с тегом – этим числом. Две вершины соединены дугой с пометкой m, если из одной вершины в другую осуществляется переход в результате кэш-промаха.

Утверждение 6. *Если в построенном графе есть цикл из дуг с пометками m, в который ведет путь из вершины $(0\ 1\ \dots\ w-1)$, на дугах которого не встречаются одинаковые пометки-числа, то если цикл не включает вершину $(m\ m\ \dots m)$, то в стратегии вытеснения не всегда возможно вытеснение любого тега набора.*

Назовем этот путь с циклом – *путем невытеснения*. Наличие пути – признак неполноты зеркального метода для этой стратегии вытеснения.

Приведем пример стратегии вытеснения, для которой путь невытеснения есть:

$$\left[\begin{array}{c|ccc} \pi_0 & 0 & 1 & 2 \\ \pi_1 & 1 & 0 & 2 \\ \pi_2 & 0 & 1 & 2 \\ \pi_m & 0 & 1 & m \end{array}\right]$$

Соответствующий граф изображен на рисунке 2.9. В нем отсутствует какой-либо путь из вершины (0 1 2) в вершину (m m m). Пример пути вытеснения в этом графе: 1 m $m\ldots$. Что и означает невозможность вытеснить некоторые теги (например, тег 0) из набора любой последовательностью различных тегов.

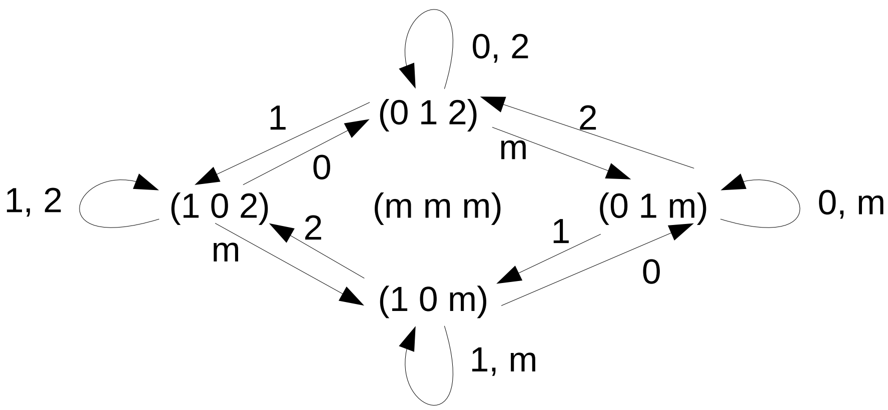


Рис. 2.9. Граф для модельной стратегии вытеснения

Для определения существования пути невытеснения может применяться следующий алгоритм: сначала перебираются порядки на множестве чисел $\{0, 1, ..., w-1\}$; обозначим очередной порядок как $i_1, i_2, ..., i_w$; строим множество вершин графа V_1, достижимых из $(01...w-1)$ только по дугам с пометками m; если обнаружился цикл, алгоритм завершается с ответом «путь вытеснения есть»; иначе строим множество вершин V_1', достижимых из V_1 по дугам с пометкой i_1; строим множество вершин графа V_2, достижимых из вершин V_1' по дугам с пометками m; если обнаружился цикл, алгоритм завершается с ответом «путь вытеснения есть»; иначе строим множество вершин V_2', достижимых из V_2 по дугам с пометкой i_2; и так далее. Если цикл нигде не встретился и все возможные порядки просмотрены, алгоритм завершается с ответом «пути вытеснения нет».

Граф для стратегий вытеснения LRU и Pseudo-LRU в случае двух - ассоциативного буфера (для сокращения пометки заменены штриховкой: дуга с пометка m обозначена сплошной линией, дуга с пометкой 0 обозначена линией из точек, дуга с пометкой 1 обозначена линией из пунктиров) изображен на рисунке 2.10. В этом графе есть всего один цикл, состоящий из сплошных дуг – петля на вершине (m m). Он включает в себя вершину (m m), поэтому в этом графе нет пути вытеснения. В случае буферов с большим количеством секций ситуация будет аналогичной.

Аналогичная ситуация будет и со стратегией вытеснения FIFO (граф для двух - ассоциативного буфера изображен на рисунке 2.11). В этом графе тоже всего один цикл, состоящий из сплошных дуг – петля на вершине (m m). Поскольку этот цикл включает в себя вершину (m m), то в этом графе нет пути вытеснения. В случае буферов с большим количеством секций ситуация будет аналогичной.

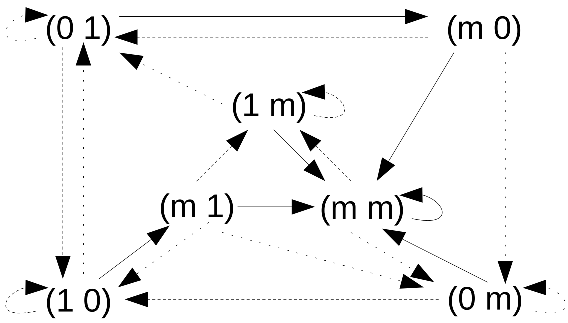


Рис. 2.10. Граф для стратегий вытеснения LRU и Pseudo-LRU с ассоциативностью 2

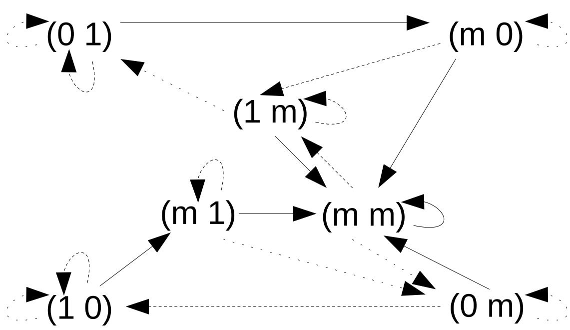


Рис. 2.11. Граф для стратегии вытеснения FIFO с ассоциативностью 2

Таким образом, справедливо

Утверждение 7. *Зеркальный метод генерации ограничений является полным для стратегий вытеснения* LRU , FIFO *и* Pseudo-LRU .

Это утверждение является важным, поскольку именно эти стратегии вытеснения наиболее часто используются в микропроцессорах.

Доказательство теоремы 4 дает способ построения последовательности инициализирующих тегов. Однако для некоторых стратегий вытеснения такая последовательность будет избыточна и существуют способы построения более коротких последовательностей инициализирующих тегов. Следующая теорема дает линейное ограничение для длины последовательности инициализирующих тегов от длины тестового шаблона для стратегии вытеснения LRU .

Теорема 5 (Верхняя оценка для длины инициализирующей последовательности тегов для стратегии вытеснения LRU)**.** *Если данный тестовый шаблон является совместным, т.е. для последовательности тестовых ситуаций* $(S_1, x_1), (S_2, x_2), ..., (S_n, x_n)$ *и дополнительного ограничения*

$P(x_1, x_2, ..., x_n)$ при некотором начальном состоянии L_1 существует удовлетворяющая им последовательность тегов $x_1, x_2, ..., x_n$, применим зеркальный метод генерации ограничений и стратегией вытеснения является LRU *, то с помощью зеркального метода может быть построена система ограничений, имеющая решение для той же последовательности тегов $x_1, x_2, ..., x_n$, причем длина последовательности инициализирующих тегов m:*

$$0 \leqslant m \leqslant n \cdot w + M$$

где M – количество инструкций тестового шаблона с кэш-промахами.

Доказательство. Так же, как это было сделано при доказательстве теоремы 4, разделим все $x_1, x_2, ..., x_n$ по регионам. Для каждого задействованного региона составим свою последовательность инициализирующих тегов (обозначим ее длину m_i для i'го задействованного региона) и объединим эти последовательности для получения искомой последовательности инициализирующих тегов для всего тестового шаблона. Подпоследовательность последовательности $x_1, x_2, ..., x_n$, соответствующая одному региону, обозначим $y_1, y_2, ..., y_{n_i}$.

Докажем, что

$$m_i = M_i + w$$

где M_i – количество тегов последовательности $y_1, y_2, ..., y_{n_i}$, которые дают кэш-промахи при первым обращениям к ним. Тогда для всего тестового шаблона $m = \sum_i m_i = \sum_i M_i + \sum_i w = M + w \cdot r$, где r – количество регионов, задействованных в $x_1, x_2, ..., x_n$. Очевидно, что $r \leqslant n$, тем самым это приводит к искомой оценке $m \leqslant M + n \cdot w$. Осталось доказать формулу для m_i.

Укажем способ построения последовательности инициализирующих тегов. Выберем из $y_1, y_2, ..., y_{n_i}$ подпоследовательность, состоящую из тех тегов, которые дают кэш-промах и встречаются впервые. Обозначим их как $\mu_1, \mu_2, ..., \mu_{MM_i}$. Они будут первыми тегами в последовательности инициализирующих тегов. Далее выберем из $y_1, y_2, ..., y_{n_i}$ все теги, при обращении к которым происходят кэш-попадания. Обозначим их как $\eta_1, \eta_2, ..., \eta_{HH_i}$. Если $MM_i > 0$ и $MM_i + HH_i < w + 1$, выберем произвольные различные теги $\nu_1, \nu_2, ..., \nu_{NN_i}$, которые не встречаются в $y_1, y_2, ..., y_{n_i}$.

Покажем, что такая последовательность инициализирующих тегов удовлетворяет зеркальному методу построения ограчений. Все теги, при обращении к которым происходят кэш-попадания, встречаются в этой последовательности. Это следует из того, что первые такие теги мы поместили явно (в конец последовательности), а дальнейшие теги не могут встречаться впервые в тестовом шаблоне, в противном случае они были бы вытеснены до того, как должно быть кэш-попадание. Очевидно, что эти первые теги не вытеснены, поскольку они помещены в конец последовательности инициализирующих тегов. Все теги, при обращении к которыми происходят кэш-промахи, тоже встречаются в этой последовательности (мы их туда поместили явно). При этом поскольку от своего кэш-промаха они отделены не менее $w+1$ инструкцией, то к моменту кэш-промаха они будут вытеснены (для первой инструкции это очевидно по построению, а для остальных следует из леммы 5 о невложенных диапазонах вытеснения).

Длина такого тестового шаблона при $MM_i = 0$ равна количеству встречающихся тегов $y_1, y_2, ..., y_{n_i}$, при обращении к которым происходят кэш-попадания. Таких тегов не более чем w, т.к. последовательность из кэш-попаданий может задать лишь часть или целиком весь набор. При $MM_i > 0$ длина последовательности есть сумма из M_i (поскольку туда включаются все теги $y_1, y_2, ..., y_{n_i}$, при обращении к которым происходят кэш-промахи) и w (поскольку в последовательность инициализирующих тегов добавляются фиктивные теги и теги, при обращении к которым происходят кэш-попадания). В обоих случаях $m_i = M_i + w$. □

Следствие. *Для длины последовательности инициализирующих тегов m в случае стратегии вытеснения* LRU *справедливо равенство:*

$$m = O(n)$$

Следствие показывает, что зеркальный метод может быть эффективно использован при стратегии вытеснения LRU для поиска минимального m методом дихотомии.

2.2.3 Совместно-зеркальная генерация

Можно заметить, что при $m = 0$ формулировка ограничений для зеркальной генерации становится частным случаем теоремы 2. Это позволяет сформулировать расширенный вариант этой теоремы, добавив туда «зеркальную» инициализирующую последовательность тегов, и тем самым по сути этим показывается соединение совместной и зеркальной генерации, ведь даже, уменьшив множество констант L_0, система ограничений, составленная на основе совместной генерации, может оказаться несовместной – в этом случае методом зеркальной генерации можно будет добиться выполнения последовательности тестовых ситуаций, указанных в тестовом шаблоне.

Утверждение 8. *Пусть L_0 – множество адресов данных, расположенных в кэширующем буфере перед исполнением первой инструкции тестового шаблона, $t_1, ..., t_m$ – инициализирующая последовательность тегов. Тогда*

- *для инструкции с кэш-попаданием адреса x следует добавить следующую совокупность уравнений:*

$$\left[\begin{array}{l} x \in L_0 \wedge x \text{ все еще не вытеснен} \\ x \in \{t_1, ..., t_m\} \wedge x \text{ не вытеснен} \\ x \text{ внесен одним из кэш-промахов} \wedge \text{с тех пор не вытеснен} \end{array}\right.$$

- *для инструкции с кэш-промахом адреса x следует добавить следующую систему уравнений ($\{x_i\}$ – множество адресов данных в инструкциях с кэш-промахами, расположенными до текущей инструкции):*

$$\left[\begin{array}{l} x \notin L_0 \wedge x \notin \{x_1, x_2, ..., x_n\} \\ x \in \{t_1, ..., t_m\} \wedge x \text{ вытеснен и не внесен} \\ x \text{ был вытеснен} \wedge \text{не был больше внесен в буфер} \end{array}\right.$$

Некоторые решатели ограничений ([18]) позволяют указывать веса конъюнктов - ограничений в ДНФ. Эти веса могут использоваться для построения решений, удовлетворяющих конъюнктам с минимальным или максимальным суммарным весом. Таким образом, для дальнейшей минимизации длины инициализирующей программы можно задавать конъюнктам с t_i больший вес, чем конъюнктам с L_0, и искать решения с минимальным суммарным весом.

2.2.4 Построение инициализирующей программы

Если кэширующий буфер, для которого применяется зеркальный метод генерации ограничений, подчинен некоторой таблице, то перед инициализирующей последовательностью тегов $(t_1, t_2, ..., t_m)$ следует поместить инструкции, заполняющие нужные для этих тегов строки таблицы. Например, перед последовательностью инициализирующих обращений в TLB (допустим, что TLB является кэширующим буфером, подчиненным таблице страниц виртуальной памяти) в таблицу страниц надо поместить (если их там не было) страницы, соответствующие инициализирующей последовательности тегов.

Если зеркальная генерация применяется к последовательностям тестовых ситуаций для нескольких кэширующих буферов, то для каждого буфера будет своя инициализирующая последовательность тегов. Может ставиться задача построения более компактной инициализирующей последовательности, объединяя некоторые обращения к буферам. Однако эта задача не рассматривалась в работе.

2.3 Единый взгляд на все предлагаемые методы

На рисунке 2.12 дан общий взгляд на предлагаемые методы генерации ограничений для тестовых шаблонов. Надо записать в виде ограничений последовательности кэш-попаданий и кэш-промахов (это центральный столбец рисунка) – это можно сделать с использованием совместной генерации, зеркальной генерации, совместно-зеркальной генерации или просто воспользовавшись теоремой 2, в зависимости от свойств тестового шаблона и MMU. Эти методы записи тестовых ситуаций в кэширующих буферах

Рис. 2.12. Построение ограничений для тестовых шаблонов

позволяют использовать различные методы записи стратегий вытеснения в виде ограничений (правый столбец рисунка). Самих по себе методов записи тестовых ситуаций еще недостаточно для генерации ограничений, поскольку они содержат в себе параметрическую часть – запись стратегии вытеснения. Для записи стратегий вытеснения можно воспользоваться диапазонами вытеснения или функциями полезности, к рассмотрению которых мы и переходим. Выбор в пользу того или иного метода записи стратегии вытеснения производится на основе возможностей решателя ограничений и требований к эффективности желаемого генератора тестовых программ.

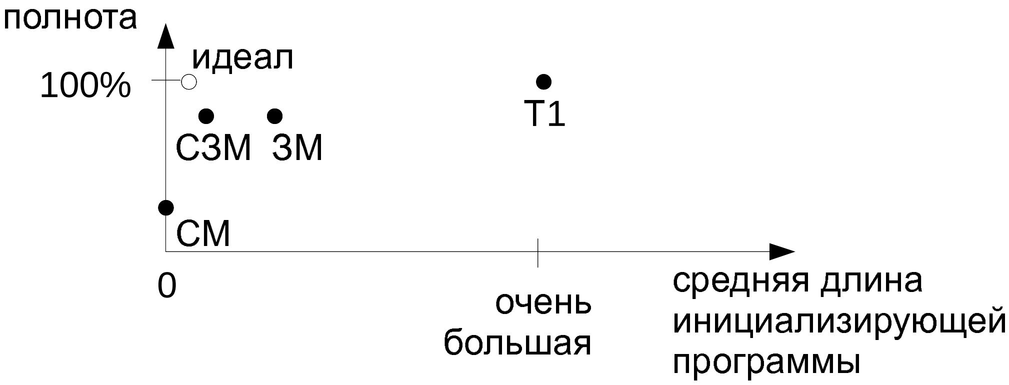


Рис. 2.13. Сравнение полноты и средней длины инициализирующей программы, которую дают предлагаемые методы

Рисунок 2.13 показывает сравнение средней длины инициализирующих программ (без учета инструкций, не меняющих кэширующие буферы и таблицы) и полноту предлагаемых методов. Совместный метод генерации ограничений (СМ) не дает вообще никакой инициализирующей программы, зато и применим он далеко не ко всем тестовым шаблонам (в том числе и к тем, для которых возможно построение тестовой программы) – поэтому этот метод не является полным. Совместно-зеркальный метод (СЗМ) и зеркальный метод (ЗМ) дают неплохие показатели полноты, поскольку применимость этих методов не так сильно зависит от тестового шаблона. Применение теоремы 2 (Т1) без учета существующего начального состояния микропроцессора (в противном случае эта теорема уже не будет давать полного метода из-за большого размера генерируемых ею ограничений – L_0 должен быть выписан полностью) дает полный метод всегда: если возможна хотя бы какая-нибудь инициализация микропроцессора, она будет найдена. Однако ценою этого является очень большая длина инициализирующей программы, поскольку необходимо переинициализировать полностью весь микропроцессор, даже если делать это не всегда обязательно.

Глава 3

Методы генерации ограничений для описания стратегий вытеснения

3.1 Метод перебора диапазонов вытеснения записи стратегии вытеснения в виде ограничений

В разделе рассматривается метод составления ограничений, описывающих стратегию вытеснения. Метод применяется к стратегиям вытеснения, для которых можно определить *метрику вытеснения* и *диапазон вытеснения.* Составляемые ограничения представляют собой дизъюнкции по всем возможным диапазонам вытеснения для данного вытесняемого тега. В разделе приведены метрики вытеснения и ограничения для трех наиболее часто использующихся в микропроцессорах стратегий вытеснения — LRU , FIFO и Pseudo-LRU .

Неформально говоря, *диапазон вытеснения* – это непрерывная часть тестового шаблона, заканчивающаяся в данной инструкции (это т.н. *конец диапазона вытеснения*), которая непосредственно влияет на вытеснение некоторого элемента кэширующего буфера. Зачастую *началом диапазона вытеснения* является инструкция, в которой осуществляется последнее обращение к вытесняемому элементу.

Метрикой вытеснения будем называть функцию от текущего состояния кэширующего буфера и части тестового шаблона. Она максимальна в конце диапазона вытеснения и минимальна в начале диапазона вытеснения. Определение диапазона вытеснения будет производиться на основе такой метрики.

3.1.1 Метод перебора диапазонов вытеснения для стратегии вытеснения LRU

LRU (Least Recently Used) — это стратегия вытеснения, определяющая вытесняемые данные как наименее используемые. Она эффективна для алгоритмов, обладающих свойством локальности данных, т.е. чаще использующих те данные, к которым недавно происходило обращение. Эта стратегия используется, например, в микропроцессорах архитектуры MIPS [35].

Стратегия вытеснения LRU обычно определяется с использованием счетчиков обращений. Для каждого элемента кэширующего буфера вводится счетчик обращений к нему. Каждое обращение увеличивает счетчик. Вытесняемым будет элемент с минимальным счетчиком. Поскольку границы значений счетчика неизвестны, формулирование метрики вытеснения на основе счетчика провести сложно.

Другой способ описания LRU основан на введении порядка на элементах набора (т.е. набор представляется списком элементов). После каждой инструкции элементы переупорядочиваются согласно следующим правилам (см.рис. 3.1):

- при кэш-попадании элемент, соответствующий адресу инструкции, перемещается в начало, остальные элементы от первого до данного сдвигаются на одну позицию;
- при кэш-промахе вытесняется последний элемент, в начало вставляется элемент, вызвавший промах.

Это описание подходит для определения метрики вытеснения: ею будет *индекс элемента в этом списке*. Эта метрика максимальна в момент вытеснения (индекс равен длине списка). Минимальное значение она принимает

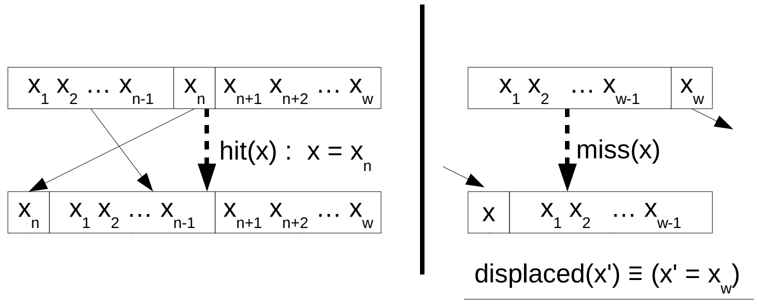


Рис. 3.1. Стратегия вытеснения LRU (w — ассоциативность кэш-памяти) — реализация на списках

в момент кэш-попадания на этот элемент (т.к. он переносится в самое начало, индекс становится равным 1). Значит, применение перебора диапазонов вытеснения возможно (выделена метрика вытеснения), началом диапазонов вытеснения будет последнее обращение к вытесняемому элементу.

Утверждение 9 (метрика вытеснения для стратегии вытеснения LRU). *Метрикой вытеснения элемента для стратегии вытеснения* LRU *является индекс элемента в наборе согласно порядку последних обращений. Диапазон вытеснения начинается в инструкции, последний раз обращающейся к элементу (или в начальном состоянии, если инструкции тестового шаблона к этому элементу не обращаются).*

Другое объяснение таким диапазонам вытеснения можно дать, исходя из самого определения LRU . А именно, если элемент должен стать LRU , т.е. наиболее неиспользуемым, все остальные элементы, наоборот, должны быть хотя бы раз использованы (т.е. к ним должны быть обращения до вытесняющей инструкции). Иными словами, чтобы элемент был вытеснен, необходимо и достаточно, чтобы между последним обращением к нему и вытеснением были обращения ко всем элементам текущего состояния кэш-памяти, кроме него (см. рис. 3.2).

Запишем в виде уравнений на множества эту логику [31]. Предикат $displaced(x')$ будет представлен дизъюнкцией уравнений — каждый элемент дизъюнкции соответствует некоторому диапазону вытеснения. Тогда для диапазона вытеснения к инструкции, обращающейся к адресу y надо составить такую систему уравнений ($x_1, x_2, ..., x_n$ – множество адресов, к которым происходят обращения внутри диапазона вытеснения (как с кэш-попаданиями, так и с кэш-промахами, а также элементы начального со-

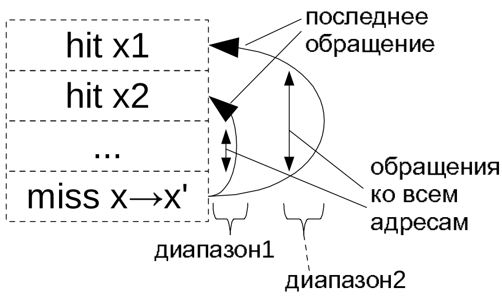


Рис. 3.2. Диапазоны вытеснения для стратегии вытеснения LRU

стояния, если диапазон начинается там), L – выражение для состояния кэш-памяти перед инструкцией, в которой вытесняется x'):

$$\begin{cases} x' = y \\ \{x_1, x_2, ..., x_n\} \cap R(y) = (L \setminus \{y\}) \cap R(y) \end{cases}$$

Функциональный символ R используется в смысле множества адресов того же региона. С использованием следующей леммы упростим эту систему:

Лемма 4. *Для любых конечных множеств X, Y и Z таких, что $X \cap Y \subseteq Z$, если существует y такой, что $y \in (Y \cap Z) \setminus X$, то $X \cap Y = (Z \setminus \{y\}) \cap Y \Leftrightarrow Y \cap (Z \setminus X) = \{y\}$.*

Доказательство. Необходимость. По определению вычитания множеств и коммутативности операции пересечения множеств $X \cap Y = (Z \setminus \{y\}) \cap Y \Leftrightarrow X \cap Y = Z \cap Y \cap \overline{\{y\}}$. Обозначим $A = Z \cap Y$, $B = X \cap Y$. Следовательно, $B = A \setminus \{y\}$. По условию $y \notin B$ и $y \in A$. Значит, $A = B \sqcup \{y\}$. Отсюда $A \setminus B = \{y\}$. Осталось показать, что $A \setminus B = (Z \setminus X) \cap Y : A \setminus B = A \cap \overline{B} = Z \cap Y \cap \overline{X \cap Y} = Z \cap Y \cap (\overline{X} \cup \overline{Y}) = (Z \cap Y \cap \overline{X}) \cup (Z \cap Y \cap \overline{Y}) = Z \cap \overline{X} \cap Y = (Z \setminus X) \cap Y$.

Достаточность. Обозначим $A = Z \cap Y$, $B = X \cap Y$. С использованием определений операций над множествами и их свойств получаем $X \cap Y \subseteq Z \Leftrightarrow (X \cap Y) \setminus Z = \varnothing \Leftrightarrow X \cap Y \cap \overline{Z} = \varnothing \Leftrightarrow X \cap Y \cap (\overline{Z} \cup \overline{Y}) = \varnothing \Leftrightarrow B \setminus A = \varnothing$. Кроме того, по условию $A \setminus B = \{y\}$. Следовательно, $A = (A \setminus B) \cup (A \cup B) = \{y\} \cup (B \setminus (B \setminus A)) = \{y\} \cup (B \setminus \varnothing) = \{y\} \cup B$. Таким образом, $A = B \cup \{y\}$. Кроме того, $y \notin B$, значит, $A = B \sqcup \{y\}$, следовательно, $B = A \setminus \{y\}$. Подставляя определения множеств A и B, получаем: $X \cap Y = (Z \cap Y) \setminus \{y\} = Z \cap Y \cap \overline{\{y\}} = (Z \setminus \{y\}) \cap Y$. □

Лемма 5 (Отсутствие вложенных диапазонов)**.** *Пусть* $x_1, x_2, ..., x_n$ *и* $y_1, y_2, ..., y_m$ *— два разных диапазона вытеснения. Тогда невозможно, чтобы* $y_1, y_2, ..., y_m$ *была бы подпоследовательностью последовательности* $x_1, x_2, ..., x_n$.

Доказательство. Докажем от противного. Предположим, что $\{y_1, y_2, ..., y_m\} \subset \{x_2, x_3, ..., x_n\}$. По определению диапазона вытеснения $x_1 \notin \{x_2, x_3, ..., x_n\}$. Следовательно, $x_1 \notin \{y_1, y_2, ..., y_m\}$. По определению диапазона вытеснения в нем должны встретиться все содержимое кэширующего буфера. Обозначим это содержимое перед началом внутреннего диапазона вытеснения L. Тогда $\{y_1, y_2, ..., y_m\} = L$ и, следовательно, $x_1 \notin L$. Однако к моменту начала внутреннего диапазона x_1 еще не вытеснен, т.е. $x_1 \in L$. Противоречие. □

Лемма 6 (О выполнимости условий леммы для диапазонов вытеснения)**.**

$$L \supseteq \{x_1, x_2, ..., x_n\} \cap R(y)$$

Доказательство (от противного). Пусть среди $x_1, x_2, ..., x_n$ есть x_i такой, что $x_i \notin L \wedge x_i \in R(y)$. Пусть L_{i+1} – состояние кэш-памяти после обращения к x_i. Верно, что $x_i \in L_{i+1}$, но $x_i \notin L$, следовательно, x_i был вытеснен между x_{i+1} и x_n. Иными словами, среди $x_1, x_2, ..., x_n$ есть элемент, чей диапазон вытеснения вложен в диапазон вытеснения y. Но согласно лемме 5 это невозможно. Противоречие. □

Таким образом, можно применить лемму 6 для упрощения уравнения для LRU . Далее идет теорема, которая формально обосновывает приведенную логику для определения вытесняемого тега в случае стратегии вытеснения LRU :

Теорема 6 (Уравнение для LRU)**.** *Решение системы (тег* x'*)*

$$\begin{cases} x' = y \\ R(y) \cap (L \setminus \{x_1, x_2, ..., x_n\}) = \{y\} \end{cases}$$

где последовательность тегов $y, x_1, x_2, ..., x_n$ *– диапазон вытеснения, а* L *– состояние кэширующего буфера перед концом диапазона, является вы-*

тесняемым тегом для стратегии вытеснения LRU *согласно определению на списках.*

Доказательство. В доказательстве будет активно использоваться таблица вытеснения для стратегии вытеснения LRU [23]. Поэтому приведем ее здесь еще раз:

$$\left[\begin{array}{c|cccccc} \pi_0 & 0 & 1 & 2 & 3 & \dots & w-1 \\ \pi_1 & 1 & 0 & 2 & 3 & \dots & w-1 \\ \pi_2 & 2 & 0 & 1 & 3 & \dots & w-1 \\ \vdots & & & & & & \\ \pi_{w-1} & w-1 & 0 & 1 & 2 & \dots & w-2 \\ \pi_m & m & 0 & 1 & 2 & \dots & w-2 \end{array}\right]$$

Сначала докажем, что если уравнение имеет решение, то это решение является вытесняемым тегом. Из уравнения следует, что $(L \setminus \{y\}) \cap R(y) \subseteq \{x_1, x_2, ..., x_n\} \cap R(y)$, из чего следует, что $L \cap R(y) \subseteq \{y, x_1, x_2, ..., x_n\} \cap R(y)$. Иными словами, все теги набора (которые находятся в наборе перед концом диапазона вытеснения), относящиеся к региону $R(y)$ присутствуют среди тегов в инструкциях диапазона вытеснения.

Из таблицы вытеснения следует, что каждый тег x набора после выполнения инструкции с тегом x_i может либо сдвинуться на одну позицию к концу списка, либо остаться на месте, либо быть вытеснен. Сдвиг происходит после обращения к x_i в том случае, если $S_i = \text{miss}$ (это следует из последней строки таблицы вытеснения) или $S_i = \text{hit}$ и тег x_i встречается впервые в диапазоне вытеснения (согласно таблице вытеснения сдвиг элемента x при кэш-попадании будет в том случае, когда он находится перед x_i; остается показать, что это условие эквивалентно тому, что к x_i не было обращения после последнего обращения к x – и в самом деле, в противном случае была бы ситуация, когда к x_i уже обращение было, но x так и остался перед x_i, однако это противоречит следующим свойствам (они следуют из таблицы вытеснения): при кэш-попадании x_i этот тег x_i перемещается перед всеми тегами набора, т.е. становится и перед x, а затем, без обращений к ним, якобы этот порядок меняется – это противоречит другому свойству о том, что порядок двух тегов в наборе не меняется, если к ним не осуществляются обращения).

Поскольку среди тегов диапазона вытеснения, относящихся к региону $R(y)$, есть все теги из соответствующего сета L и нет лишних (это следует из уравнения), то количество различных тегов (т.е. количество раз, когда тег встречается впервые) вместе с количеством кэш-промахов в диапазоне вытеснения равно $w-1$. Из стратегии вытеснения следует, что после первой инструкции диапазона вытеснения тег y будет помещен в начало списка. Значит, за $w-1$ сдвигов на 1 позицию он будет перемещен на последнее место в списке. На этом месте (согласно таблице вытеснения) и располагается вытесняемый тег.

Доказательство остается справедливым и в том случае, когда к y не было обращений в самом тестовом шаблоне (однако в диапазон вытеснения он входит, такой диапазон включает в себя и часть начального состояния кэширующего буфера L_0). В этом случае можно считать, что набор, соответствующий региону $R(y)$ в L_0, представляется последовательностью из w кэш-промахов с тегами – элементами этого набора.

Теперь докажем обратное утверждение, а именно если y — вытесняемый тег в некоторой инструкции (т.е. находится в конце списка), то существует такой диапазон вытеснения $x_1, x_2, ..., x_n$ (возможно, задействующий часть начального состояния L_0), для которого справедливы вложения $\{x_1, x_2, ..., x_n\} \cap R(y) \subseteq (L \setminus \{y\}) \cap R(y)$ и $\{x_1, x_2, ..., x_n\} \cap R(y) \supseteq (L \setminus \{y\}) \cap R(y)$.

Очевидно, что поскольку перед инструкцией, вытесняющей y, лишь конечное количество инструкций (и начальное состояние L_0), то среди них существует такая, которая обращается к y в последний раз перед вытеснением. Тогда в качестве $x_1, x_2, ..., x_n$ рассмотрим последовательность инструкций от последнего обращения к y до данной инструкции. Из этого следует, что $y \notin \{x_1, x_2, ..., x_n\}$. Кроме того по лемме 6 $L \supseteq \{x_1, x_2, ..., x_n\} \cap R(y)$. Значит, $(L \setminus \{y\}) \supseteq \{x_1, x_2, ..., x_n\} \cap R(y)$. И поскольку для любых множеств X, Y, Z справедливо следствие: если $X \supseteq Y \cap Z$, то $X \cap Z \supseteq Y \cap Z$ (формула $(y \wedge z \rightarrow x) \rightarrow (y \wedge z \rightarrow x \wedge z)$ тождественно истинна) – то $(L \setminus \{y\}) \cap R(y) \supseteq \{x_1, x_2, ..., x_n\} \cap R(y)$. Первое вложение доказано.

Для доказательства второго вложения покажем, что верно вложение $\{y, x_1, x_2, ..., x_n\} \cap R(y) \supseteq L \cap R(y)$, из которого будет следовать $\{y, x_1, x_2, ..., x_n\} \cap R(y) \supseteq L \cap R(y)$, а из него — требуемое вложение (поскольку для любых множеств X, Y, Z верно следствие: если $X \cup Y \supseteq Z$, то $X \supseteq Z \setminus Y$ – оно справедливо потому, что формула $(z \to (x \vee y)) \to (\overline{y}z \to x)$ тождественно истинна).

Иными словами, надо показать, что нет такого тега в $L \cap R(y)$, к которому в диапазоне не было бы обращения. Докажем это от противного. Допустим, что такой тег имеется. Т.е. количество различных тегов набора, относящихся к региону $R(y)$, меньше $w-1$ (один отняли за счет y в начале диапазона). Однако по условию y является вытесняемым тегом, т.е. к концу диапазона вытеснения перемещен из начала списка в конец. Для этого он должен был сдвинут $w-1$ раз. Сдвиг происходит при кэш-промахе или кэш-попадании тега, встречавшегося впервые (доказано выше). Следовательно, таких инструкций ровно $w-1$ и все они расположены в $x_1, x_2, ..., x_n$. Но поскольку $|L \cap R(y)| = w$, то среди $x_1, x_2, ..., x_n$ встречается ровно $w-1$ различных тегов, относящихся к региону $R(y)$, что противоречит предположению. □

3.1.2 Метод перебора диапазонов вытеснения для стратегии вытеснения FIFO

FIFO (First-In First-Out) – это стратегия вытеснения, определяющая вытесняемые данные согласно принципу очереди FIFO. Например, в микропроцессоре PowerPC 970FX вытеснение из небольшого буфера, хранящего последние преобразованные эффективные адреса в физические, D-ERAT происходит согласно FIFO [9].

Стратегия FIFO может быть описана на основе порядка на элементах набора (т.е. набор представляется списком элементов). После каждой инструкции элементы переупорядочиваются согласно следующим правилам (см.рис. 3.3):

- при кэш-попадании порядок элементов не меняется;
- при кэш-промахе вытесняется последний элемент, в начало вставляется элемент, вызвавший промах.

Отличие от LRU лишь в том, что при FIFO не происходит перестановки элементов набора при возникновении кэш-попадания. Поэтому таблица

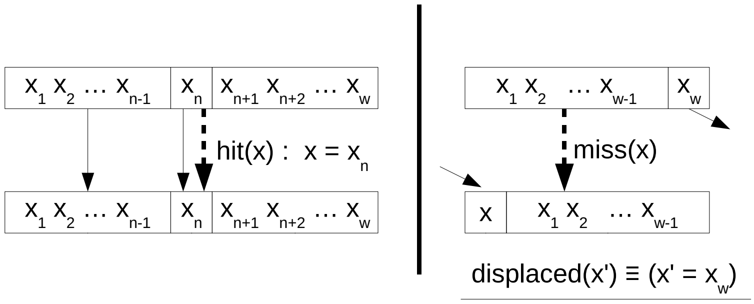


Рис. 3.3. Стратегия вытеснения FIFO (w - ассоциативность кэш-памяти)

$$\left[\begin{array}{c|cccccc} \pi_0 & 0 & 1 & 2 & 3 & \dots & w-1 \\ \pi_1 & 0 & 1 & 2 & 3 & \dots & w-1 \\ \pi_2 & 0 & 1 & 2 & 3 & \dots & w-1 \\ \vdots & & & & & & \\ \pi_{w-1} & 0 & 1 & 2 & 3 & \dots & w-1 \\ \pi_m & m & 0 & 1 & 2 & \dots & w-2 \end{array}\right]$$

Рис. 3.4. Таблица вытеснения для FIFO

вытеснения [23] для стратегии вытеснения FIFO будет выглядеть так, как изображено на рисунке 3.4.

Аналогично LRU в качестве метрики вытеснения можно взять индекс элемента в списке, что дает возможность использовать перебор диапазонов вытеснения для описания стратегии вытеснения FIFO . Началом диапазона вытеснения будет внесение элемента в кэширующий буфер, концом диапазона вытеснения – его вытеснение. При составлении ограничений все инструкции с кэш-попаданиями внутри диапазона будем игнорировать (они не влияют на вытеснение с точки зрения FIFO). Тогда FIFO *будет выполнено в том случае, когда в диапазоне встречаются все теги кэширующего буфера, хранящиеся в нем перед вытеснением, без самого вытесняемого тега.*

Утверждение 10 (метрика вытеснения для стратегии вытеснения FIFO)**.** *Метрикой вытеснения тега для стратегии вытеснения* FIFO *является его индекс в наборе согласно порядку последних обращений. Диапазон вытеснения начинается в инструкции с кэш-промахом, последний раз обращающейся к вытесняемому тегу (или в начальном состоянии, если инструкции тестового шаблона к этому тегу не обращаются).*

Запишем в виде уравнений на множества эту логику [7]. Предикат

$displaced(y')$ будет представлен дизъюнкцией уравнений – каждый элемент дизъюнкции соответствует некоторому диапазону вытеснения. Тогда для диапазона вытеснения к инструкции, обращающейся к адресу y, надо составить такую систему уравнений ($y_1, y_2, ..., y_n$ – множество адресов, к которым происходят обращения внутри диапазона вытеснения **с кэш-промахами**, а также элементы начального состояния, если диапазон начинается там, L – выражение для состояния кэш-памяти для инструкции, вытесняющей y'):

$$\begin{cases} y' = y \\ \{y_1, y_2, ..., y_n\} \cap R(y) = (L \setminus \{y\}) \cap R(y) \end{cases}$$

Функциональный символ R используется в смысле множества адресов того же региона.

Для FIFO справедливы все леммы о диапазонах вытеснения, сформулированные для LRU . В частности, с их использованием теорема об уравнении, описывающем вытесняемый тег, может быть переписана следующим образом:

Теорема 7 (Уравнение для FIFO). *Решение системы (тег y')*

$$\begin{cases} y' = y \\ R(y) \cap (L \setminus \{y_1, y_2, ..., y_n\}) = \{y\} \end{cases}$$

где последовательность тегов $y, y_1, y_2, ..., y_n$ – диапазон вытеснения, является вытесняемым тегом для стратегии вытеснения FIFO *согласно определению на списках.*

Доказательство. Сначала докажем, что если уравнение имеет решение, то это решение является вытесняемым тегом. Из уравнения следует, что $(L \setminus \{y\}) \cap R(y) \subseteq \{y_1, y_2, ..., y_n\} \cap R(y)$, из чего следует, что $L \cap R(y) \subseteq \{y, y_1, y_2, ..., y_n\} \cap R(y)$. Иными словами, все теги набора (которые находятся в наборе перед концом диапазона вытеснения), относящиеся к региону $R(y)$ присутствуют среди тегов в инструкциях диапазона вытеснения (тестовыми ситуациями в этих инструкциях являются кэш-промахи).

Из таблицы вытеснения следует, что каждый тег x набора после выполнения инструкции с тегом x_i может либо сдвинуться на одну позицию к концу списка, либо остаться на месте, либо быть вытеснен. Сдвиг происходит после обращения к x_i в том случае, если $S_i = \text{miss}$ (это следует из последней строки таблицы вытеснения).

Поскольку среди тегов диапазона вытеснения, относящихся к региону $R(y)$, есть все теги из соответствующего сета L и нет лишних (это следует из уравнения), то количество различных тегов в диапазоне вытеснения равно $w-1$. Из стратегии вытеснения следует, что после первой инструкции диапазона вытеснения тег y будет помещен в начало списка. Значит, за $w-1$ сдвигов на 1 позицию он будет перемещен на последнее место в списке. На этом месте (согласно таблице вытеснения) и располагается вытесняемый тег.

Доказательство остается справедливым и в том случае, когда к y не было обращений в самом тестовом шаблоне (однако в диапазон вытеснения он входит, такой диапазон включает в себя и часть начального состояния кэширующего буфера L_0). В этом случае можно считать, что набор, соответствующий региону $R(y)$ в L_0, представляется последовательностью из w кэш-промахов с тегами – элементами этого набора.

Теперь докажем обратное утверждение, а именно если y — вытесняемый тег в некоторой инструкции (т.е. находится в конце списка), то существует такой диапазон вытеснения $y_1, y_2, ..., y_n$ (возможно, задействующий часть начального состояния L_0), для которого справедливы вложения $\{y_1, y_2, ..., y_n\} \cap R(y) \subseteq (L \setminus \{y\}) \cap R(y)$ и $\{y_1, y_2, ..., y_n\} \cap R(y) \supseteq (L \setminus \{y\}) \cap R(y)$.

Очевидно, что поскольку перед инструкцией, вытесняющей y, лишь конечное количество инструкций (и начальное состояние L_0), то среди них существует такая, которая обращается к y в последний раз перед вытеснением. Тогда в качестве $y_1, y_2, ..., y_n$ рассмотрим последовательность инструкций от последнего обращения к y до данной инструкции. Из этого следует, что $y \notin \{y_1, y_2, ..., y_n\}$. Аналогично лемме 6 можно сформулировать и доказать утверждение для стратегии вытеснения FIFO : $L \supseteq \{y_1, y_2, ..., y_n\} \cap R(y)$. Значит, $(L \setminus \{y\}) \supseteq \{y_1, y_2, ..., y_n\} \cap R(y)$. И поскольку для любых множеств X, Y, Z справедливо следствие: если $X \supseteq Y \cap Z$, то $X \cap Z \supseteq Y \cap Z$ (формула $(y \wedge z \to x) \to (y \wedge z \to x \wedge z)$ тождественно истинна) – то $(L \setminus \{y\}) \cap R(y) \supseteq \{y_1, y_2, ..., y_n\} \cap R(y)$. Первое вложение доказано.

Для доказательства второго вложения покажем, что верно вложение $\{y, y_1, y_2, ..., y_n\} \cap R(y) \supseteq L \cap R(y)$, из которого будет следовать $\{y, y_1, y_2, ..., y_n\} \cap R(y) \supseteq L \cap R(y)$, а из него — требуемое вложение (поскольку для любых множеств X, Y, Z верно следствие: если $X \cup Y \supseteq Z$, то $X \supseteq Z \setminus Y$ – оно справедливо потому, что формула $(z \to (x \vee y)) \to (\overline{y}z \to x)$ тождественно истинна).

Иными словами, надо показать, что нет такого тега в $L \cap R(y)$, к которому в диапазоне не было бы обращения. Докажем это от противного. Допустим, что такой тег имеется. Т.е. количество различных тегов набора, относящихся к региону $R(y)$, меньше $w-1$ (один отняли за счет y в начале диапазона). Однако по условию y является вытесняемым тегом, т.е. к концу диапазона вытеснения перемещен из начала списка в конец. Для этого он должен был сдвинут $w-1$ раз. Сдвиг происходит при кэш-промахе или кэш-попадании тега, встречавшегося впервые (доказано выше). Следовательно, таких инструкций ровно $w-1$ и все они расположены в $y_1, y_2, ..., y_n$. Но поскольку $|L \cap R(y)| = w$, то среди $y_1, y_2, ..., y_n$ встречается ровно $w-1$ различных тегов, относящихся к региону $R(y)$, что противоречит предположению. □

3.1.3 Метод перебора диапазонов вытеснения для стратегии вытеснения Pseudo-LRU

Стратегия вытеснения LRU хоть и хорошо приближает поведение кэширующего буфера к идеальному случаю (когда данные находятся в буфере в тот момент, когда они нужны), однако все известные на сегодняшний момент реализации для микропроцессоров требуют большого количества дополниительной логики. Поэтому производятся поиски стратегии вытеснения, близкой по качеству к LRU , но имеющей реализацию с меньшими накладными расходами. Эти поиски привели к стратегии вытеснения Pseudo-LRU . Она определяется только для кэширующих буферов с ассоциативностью, являющейся степенью двойки. Стратегия вытеснения Pseudo-LRU используется во многих микропроцессорах архитектур PowerPC и Pentium [17].

Для каждого набора хранится битовая строка длины $w-1$, где w – ассоциативность кэширующего буфера. Каждая инструкция, обращающаяся к набору, меняет эту битовую строку. Определение вытесняемого элемента производится тоже на основании этой битовой строки. Для наглядности алгоритм изменения битовой строки описывают на упорядоченном бинарном дереве высоты $\log_2 w$, в листьях которого подряд расположены элементы набора. Вытесняющий элемент помещается в дереве на место вытесняемого. Между элементами битовой строки и нелистовыми элементами установлено взаимнооднозначное соответствие. Каждая дуга в дереве помечена числом 0 или 1, из каждого нелистового узла выходят дуги, помеченные разными числами.

При кэш-попадании по некоторому адресу меняются элементы битовой строки, соответствующие вершинам дерева, которые входят в путь от корня до этого адреса (см. рис. 3.5). А именно, элемент битовой строки становится равным пометке дуги, исходящей из соответствующего ему узла. Остальные элементы битовой строки не меняются.

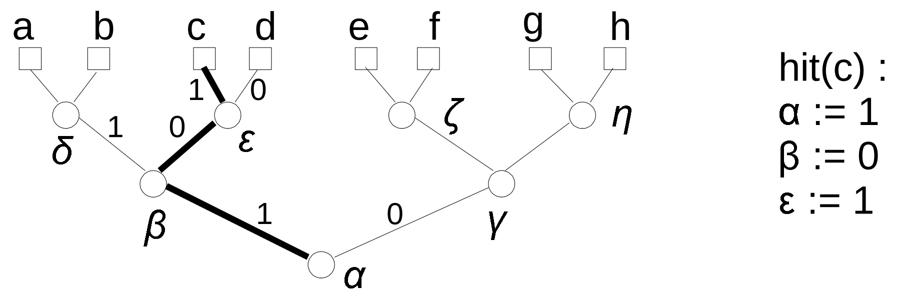


Рис. 3.5. Кэш-попадание для стратегия вытеснения Pseudo-LRU (16-ассоциативная кэш-память)

Поиск вытесняемого элемента производится следующим образом: на основе значений элементов битовой строки (т.е. нелистовых узлов дерева) определяется единственный путь. Лист, к которому ведет этот путь, и является вытесняемым элементом. Путь определяется итеративно: первая вершина – всегда корень, из него выбирается дуга, помеченная значением, противоположным значению элемента битовой строки, соответствующей корню. Затем эта же операция повторяется для узла – конца этой дуги, а именно, выбирается исходящая из него дуга, пометка которого имеет значение, противоположное тому, какому этот узел соотнесен в битовой строке. На место вытесняемого элемента помещается вытесняющий, битовая строка меняется так, будто к вытесняющему элементу было обращение с кэш-попаданием. Пример того, как определяется вытесняемый элемент, показан на рис. 3.6. Цветом нелистовых узлов показано значение соответствующего им элементов битовой строки: черный узел соответствует значению 1, белый – 0. В изображенном дереве будет выбран путь $\alpha - \gamma - \zeta$, согласно которому будет вытеснен элемент e.

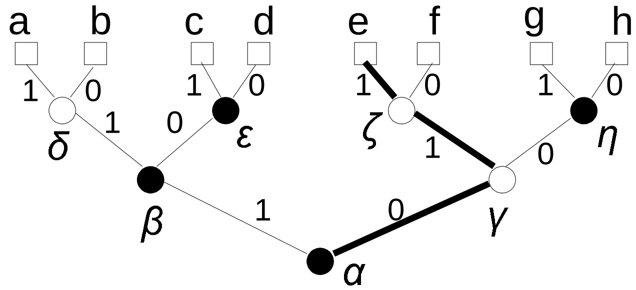


Рис. 3.6. Определение вытесняемого элемента для стратегия вытеснения Pseudo-LRU (16-ассоциативная кэш-память)

Иными словами, определение вытесняемого элемента можно проводить, последовательно рассматривая тестовые ситуации от данной инструкции с кэш-промахом назад к первой инструкции. Каждая очередная инструкция отсекает то поддерево, которому принадлежит адрес в этой инструкции (если адрес принадлежит уже отсеченной части дерева, инструкция игнорируется). В результате, на некотором шаге останется дерево из одного элемента, вытесняемый элемент и будет тем самым элементом.

Взглянем на эту схему со стороны одного элемента и попробуем вывести логику, согласно которой именно он был бы вытеснен. Для простоты рассмотрим сначала самый левый элемент. Он будет вытеснен в том и только в том случае, когда вся ветвь дерева к нему состоит из черных вершин. После обращения к этому элементу ветвь дерева к нему состоит целиком из белых вершин. Таким образом, вытеснение можно понимать как процесс перекрашивания вершин ветви от белых к черным. Каждое обращение к какому-либо элементу дерева перекрашивает часть ветви к данному элементу, и когда будет закрашена в черный цвет вся ветвь целиком будет вытеснен сам данный (самый левый) элемент. Иными словами, метрикой вытеснения может стать количество черных вершин в ветви, если рассматривается самая левая ветвь. В момент вытеснения это количество максимально (равно $\log_2 w - 1$). Минимальное значение этой метрики равно 0, оно соответствует моменту обращения к соответствующей листовой вершине. Таким образом, возможно применение перебора диапазонов вытеснения для описания Pseudo-LRU : началом диапазона будет последнее обращение к элементу (листовой вершине дерева), концом диапазона будет вытесняющая этот элемент инструкция.

Аналогичные рассуждения проводятся для всех остальных листовых вершин, только конкретные цвета, белый и черный, надо заменить на те цвета, которые ведут в вершину и противоположные к ним.

Утверждение 11 (метрика вытеснения для стратегии вытеснения Pseudo–LRU). *Метрикой вытеснения элемента для стратегии вытеснения* Pseudo-LRU *является количество вершин в ветви к вытесняемой листовой вершине с пометками, противоположными пометкам при прохождении по ветви при кэш-попадании. Диапазон вытеснения начинается в инструкции, последний раз обращающейся к листовой вершине (или в начальном состоянии, если инструкции тестового шаблона к этой листовой вершине не обращаются).*

Осталось записать уравнения, описывающие предложенные диапазоны вытеснения. Каждый элемент кэш-памяти снабдим *позицией* – номером этого элемента среди листовых вершин дерева. Будем обозначать позицию буквой π. Предикат $displaced(x')$ будет представлен дизъюнкцией уравнений – каждый элемент дизъюнкции соответствует некоторому диапазону вытеснения. Тогда для диапазона вытеснения к инструкции, обращающейся к адресу x_1 с позицией π_1 надо составить такую систему уравнений ($x_2, x_3, ..., x_n$ – множество адресов, к которым происходят обращения внутри диапазона вытеснения, $\pi_2, \pi_3, ..., \pi_n$ – соответствующие им позиции, $\delta_i = \pi_i \oplus \pi', i \in 2..n$, где $\oplus$ – операция сложения по модулю 2 двоичных разложений операндов):

$$\begin{cases} x' = x_1 \\ \pi' = \pi_1 \\ \pi = \pi' \\ ((0\ op_x\ \delta_2)\ op_x\ \delta_3)...\ op_x\ \delta_n = w - 1 \end{cases}$$

Операция op_x выполняет очередной шаг по «перекрашиванию» ветви,

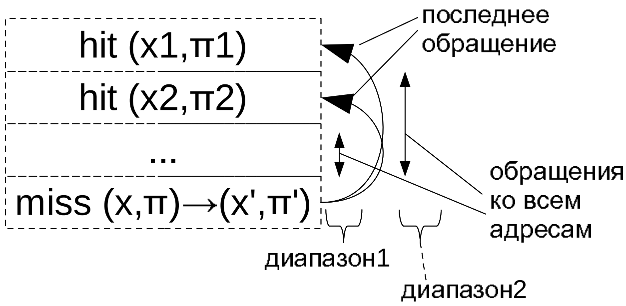


Рис. 3.7. Диапазоны вытеснения для стратегии вытеснения Pseudo-LRU

ведущей в элемент x. Она может быть определена следующей формулой:

$$X\ op_x\ \delta \equiv \text{ if } R(X) \neq R(x) \text{ then } X \text{ else } (X \& \delta_{<1>})|\delta_{<0>} \text{ end}$$

где & – побитовая конъюнкция, | – побитовая дизъюнкция, $\delta_{<1>} = 2*\delta_{<0>}-1$, а $\delta_{<0>} = 2^{[\log_2 \delta]}$ может быть определено следующим переборным способом: $\delta_{<0>} = \text{ if } 1 \leqslant \delta < 2 \text{ then } 1 \text{ elsif } 2 \leqslant \delta < 4 \text{ then } 2 \text{ elsif } ... \text{ else } w \text{ end}$. Другой способ получения $\delta_{<0>}$ и $\delta_{<1>}$ удобно применять при побитовом рассмотрении δ: $\delta_{<0>} = (\delta_{<1>} + 1) \gg 1$, $\delta_{<1>}[i] = \delta[1] \vee \delta[2] \vee ... \vee \delta[i]$, где символом $d[i]$ обозначен i'й бит числа d, биты нумеруются со старших к младшим.

3.2 Метод функций полезности записи стратегии вытеснения в виде ограничений

В разделе рассматривается метод составления ограничений, описывающих стратегию вытеснения, для которых можно определить метрик вытеснения. Стратегия вытеснения описывается ограничением сверху на количество *полезных* инструкций (т.е. помогающих вытеснению). В разделе приведены метрики полезности и ограничения для трех наиболее часто использующихся в микропроцессорах стратегий вытеснения – LRU , FIFO и PSEUDO-LRU . Освещается понятие *монотонной метрики вытеснения*, которая является залогом более компактной системы ограничений.

Пусть для стратегии вытеснения сформулирована метрика вытеснения (ее значение максимально в вытесняющей инструкции). Будем называть инструкцию *полезной*, если она увеличивает метрику на этапе монотонного увеличения метрики до максимального значения (см. рис. 3.8).

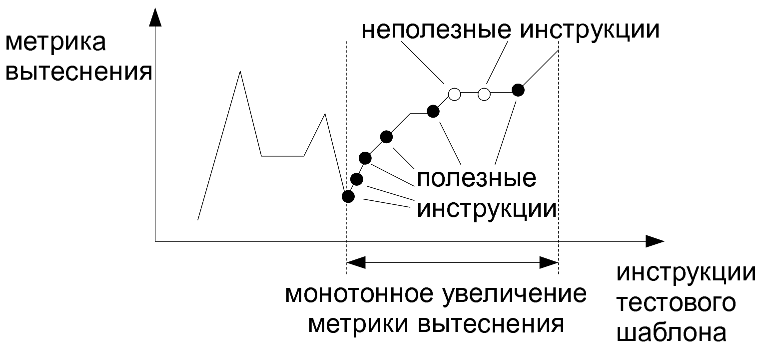


Рис. 3.8. К определению полезных инструкций

Тогда вытеснение будет происходить в том случае, когда количество полезных инструкций превысит некоторое константное количество. Вытеснение не будет происходить, если количество полезных инструкций не превысит некоторой константной верхней границы. Количество полезных инструкций можно записать в виде суммы переменных-полезностей, каждая такая переменная соответствует своей инструкции и равна 1 тогда и только тогда, когда инструкция является полезной, и 0 тогда и только тогда, когда инструкция не является полезной. Иными словами, ограничение будет иметь вид $\sum_{i=1}^{n} u(x_i) < N$ или $\sum_{i=1}^{n} u(x_i) = N$, где $u(x_i)$ – *функция полезности* (равна 1, если x_i – полезная инструкция, и равна 0, если x_i не является полезной инструкцией).

3.2.1 Метод функций полезности для стратегии вытеснения LRU

Функцией полезности является номер вытесняемого элемента согласно порядку счетчика LRU (см. рис. 3.1). Значит, полезной будет инструкция, переставляющая вытесняемый элемент в этом порядке к концу. Такими инструкциями являются все кэш-промахи (поскольку они вытесняют последний элемент с передвижением всех остальных на одну позицию к концу, в том числе будет передвинут и данный вытесняемый элемент) и кэш-попадания к элементам, находившимся ближе к концу, чем данный вытесняемый (потому как при кэш-попадании они передвинутся в самое начало, а все элементы от начала и до них сдвинутся на одну позицию к концу, в том числе и данный вытесняемый).

Осталось выразить эту идею в виде ограничений [30]. Для этого удобно использовать формулировку тестовых ситуаций в кэширующем буфере из утверждения 4. Символом λ_δ будет обозначаться элемент домена – начального состояния буфера – с индексом δ по порядку LRU , $1 \leqslant \delta \leqslant w$. Индекс 1 обозначает самый молодой элемент, индекс w обозначает самый старый элемент.

Применение полезностей эффективно в том случае, когда домен имеет небольшой размер (такие домены как раз обеспечивает совместная генерация ограничений). В этом случае можно перебрать все элементы домена (это и будут λ_δ) и составить для них свои полезности, причем для каждого элемента будет известен индекс по порядку LRU (δ). Ограничение, описывающее стратегию вытеснения, будет при этом иметь вид дизъюнкции по элементам домена.

Если вытесняемый элемент был в начальном состоянии (пусть это λ_δ) и к нему не было обращений, то для его вытеснения необходимо $w - \delta + 1$ полезных инструкций, потому что столько раз надо подвинуть элемент с индексом δ в LRU -списке в сторону к концу (к элементам с индексом w), чтобы он вышел за границу списка (иными словами, чтобы он был вытеснен).

Если вытесняемый элемент был в начальном состоянии и к нему было обращение, то для его вытеснения необходимо w инструкций, так как во время обращения элемент был поставлен в самое начало LRU -списка. То же справедливо для внесенных в кэширующий буфер новых тегсетов – чтобы их вытеснить, надо так же w полезных инструкций, чтобы переместить их к концу LRU -списка.

В таблице 3.1 приведены все функции полезности для кэш-попаданий и кэш-промахов. Далее идет ряд формулировок и теорем, доказывающих корректность применения полезностей для записи стратегии вытеснения LRU в виде ограничений.

	случай	переменная перебора	система	функция полезности для кэш-попадания	функция полезности для кэш-промаха
кэш-попадание	тегсет находится в начальном состоянии кэш-памяти, к нему нет обращений до данной инструкции и он всё ещё не вытеснен	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) \leqslant w - \delta \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$ $\wedge\ x_i \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$
	тегсет находится в начальном состоянии кэш-памяти, к нему есть обращение до данной инструкции и он всё ещё не вытеснен	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \in \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) < w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$ $\wedge\ \sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
	тегсет был внесен одним из кэш-промахов и с того момента не вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \sum\limits_{i=1}^{n} u(x_i) < w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)\ \wedge$ $\sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
кэш-промах	тегсет встречается впервые	–	$\begin{cases} x \notin D \\ x \notin [x_1, ..., x_n]_{miss} \end{cases}$	–	–
	тегсет ранее был внесен одной из инструкций шаблона, затем вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \sum\limits_{i=1}^{n} u(x_i) \geqslant w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)\ \wedge$ $\sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему не было обращений в шаблоне	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \sum_{i=1}^{n} u(x_i) \geqslant w - \delta + 1 \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$ $\wedge\ x_i \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему было обращение в шаблоне после последнего внесения в кэш-память	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \in \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) \geqslant w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$ $\wedge\ \sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$

Таблица 3.1. Таблица систем уравнений для тестовых ситуаций в LRU кэш-памяти с использованием функций полезности

Кэш-попадание – первый случай Это случай обозначен в формулировке утверждения 4 фразой «$x \in D \wedge x$ все еще не вытеснен». Возможны два подслучая в зависимости от того, было ли обращение к x до вытесняющей его инструкции. Подслучай отсутствия такого обращения будем называть кэш-попаданием-I', а наличия – кэш-попаданием-I''. Ограничения для этих подслучаев объединяются в дизъюнкцию.

Лемма 7 (представление кэш-попадания-I' с помощью функций полезности для LRU). *Пусть x – тегсет текущей инструкции при стратегии вытеснения* LRU . *Тогда если $x = \lambda \in L_0$ и $x \notin \{x_1, x_2, ..., x_n\}$, где $x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, то x не вытеснен к моменту текущей инструкции согласно определению* LRU *на списках тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) \leqslant w - \delta$, где $\delta \in \{1, 2, ..., w\}$ – индекс λ в своем наборе L_0 согласно метрике вытеснения* LRU , *а $u(x_i)$ (*функция полезности*) определена следующим образом:*

$$u(x_i) \equiv \begin{cases} x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge x_i \notin \{x_1, ..., x_{i-1}\}, \textit{если } S_i = hit \\ R(x_i) = R(x), \textit{если } S_i = miss \end{cases}$$

Доказательство. Напомню таблицу вытеснения для стратегии вытеснения LRU [23]:

$$\left[\begin{array}{c|cccccc} \pi_0 & 0 & 1 & 2 & 3 & \dots & w-1 \\ \pi_1 & 1 & 0 & 2 & 3 & \dots & w-1 \\ \pi_2 & 2 & 0 & 1 & 3 & \dots & w-1 \\ \vdots & & & & & & \\ \pi_{w-1} & w-1 & 0 & 1 & 2 & \dots & w-2 \\ \pi_m & m & 0 & 1 & 2 & \dots & w-2 \end{array}\right]$$

Сначала докажем, что если x не вытеснен, то справедливо ограничение сверху на сумму функций полезности. Из таблицы вытеснения следует, что x не вытеснен, если он не был сдвинут к концу списка с последующим кэш-промахом, который бы его вытеснил. По условию x находился в начальном состоянии на месте с индексом δ. Кроме того, поскольку $x \notin \{x_1, x_2, ..., x_n\}$, т.е. к x не было обращений, то x мог сдвигаться только в одном направлении – к концу списка. Причем таких сдвигов не должно быть больше $w - \delta$, поскольку именно столько сдвигов требуется для перемещения из позиции

δ в конец списка.

Покажем, что полученное требование на количество сдвигов (оно не должно превышать $w-\delta$) как раз и выражается ограничением суммы функций полезности. Покажем, что функция полезности $u(x_i) = 1$ в том случае и только в том случае, когда при обращении к x_i происходит сдвиг тега x на 1 позицию (другие сдвиги запрещает таблица вытеснения). Итак, сдвиг тега x происходит в результате обращения к x_i в том случае, если $S_i = \text{miss}$ (это следует из последней строки таблицы вытеснения) или $S_i = \text{hit}$ и тег x_i встречается впервые в $x_1, x_2, ..., x_n$, и (согласно таблице вытеснения) x находился перед x_i (из этого условия следует, что $x_i \in L_0$, поскольку в противном случае согласно таблице вытеснения x не мог бы находиться x_i, ибо последний был бы помещен в самое начало списка и без обращения к x взаимное отношение x и x_i не могло быть изменено). Не надо забывать, что таблица вытеснения формулируется только на наборе, таким образом, к сформулированным требованиям осуществления сдвига надо добавить совпадение регионов тегов x и x_i.

Перефразируя выше сказанное, получим, если $S_i = \text{miss}$, то сдвиг тега x будет осуществлен при выполнении условия $R(x) = R(x_i)$. Если $S_i = \text{hit}$, то сдвиг тега x будет осуществлен при выполнении условия $R(x) = R(x_i) \wedge x_i$ встречается впервые $\wedge$ x находится перед x_i $\wedge$ $x_i \in L_0$. Ровно в этих же случаях предложенная в формулировке теоремы функция полезности $u(x_i)$ будет равна 1. А именно, при $S_i = \text{miss}$ $u(x_i) \equiv R(x) = R(x_i)$, что и утверждает теорема. Напомню, что $R(x) \cap L_0 = \{\lambda_1, \lambda_2, ..., \lambda_\delta, ..., \lambda_w\}$, что дает возможность записать при $S_i = \text{hit}$ свойство сдвига как «x_i встречается впервые $\wedge$ x находится перед x_i $\wedge$ $x_i \in \{\lambda_1, \lambda_2, ..., \lambda_\delta, ..., \lambda_w\}$», что можно еще упростить, учитывая равенство $x = \lambda_\delta$: «x_i встречается впервые $\wedge$ $x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$». Ровно при выполнении этого же свойства функция полезности $u(x_i) = 1$ при $S_i = \text{hit}$.

В обратную сторону лемма доказывается с помощью тех же шагов, что и в прямую сторону. □

Лемма 8 (представление кэш-попадания-I" с помощью функций полезности для LRU). *Пусть x – тегсет текущей инструкции при стратегии вытеснения* LRU . *Тогда если $x = \lambda \in L_0$ и $x \in \{x_1, x_2, ..., x_n\}$, где $x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, то x не вытеснен к мо-*

менту текущей инструкции согласно определению LRU *на списках тогда и только тогда, когда* $\sum_{i=1}^{n} u(x_i) < w$, *где* $u(x_i)$ *(*функция полезности*) определена следующим образом:*

$$u(x_i) \equiv \begin{cases} x \notin \{x_i, ..., x_n\} \wedge x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge \sum_{j=1}^{i-1} c_i(x_j) = 0, \\ \quad \text{\textit{если} } S_i = hit \\ x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x), \text{\textit{если} } S_i = miss \end{cases}$$

$\delta \in \{1, 2, ..., w\}$ *– индекс* λ *в своем наборе* L_0 *согласно метрике вытеснения LRU (*$\lambda = \lambda_\delta$*),*

$$c_i(x_j) \equiv (x \notin \{x_j, x_{j+1}, ..., x_i\} \wedge x_i = x_j)$$

Доказательство. Сначала докажем, что если x не вытеснен, то справедливо ограничение сверху на сумму функций полезности. Рассмотрим подпоследовательность последовательности $x_1, x_2, ..., x_n$ из тегов, равных x. Поскольку такая последовательность по условию непустая и количество тегов в ней ограничено (не более n), то среди них существует элемент подпоследовательности с максимальным индексом. Обозначим его x_s. Из таблицы вытеснения следует, что x не вытеснен, если он не был сдвинут к концу списка с последующим кэш-промахом, который бы его вытеснил. Каждая инструкция с тегов x сдвигает его в самое начало списка. Кроме того, поскольку $x = x_s \notin \{x_{s+1}, x_{s+2}, ..., x_n\}$, т.е. к x не было обращений после x_s, то x мог сдвигаться после x_s только в одном направлении – к концу списка. Причем таких сдвигов не должно быть больше $w - 1$, поскольку именно столько сдвигов требуется для перемещения из начала списка в конец списка.

Покажем, что полученное требование на количество сдвигов (оно не должно превышать $w-1$) как раз и выражается ограничением суммы функций полезности. Покажем, что функция полезности $u(x_i) = 1$ в том случае и только в том случае, когда при обращении к x_i после обращения к x_s происходит сдвиг тега x на 1 позицию (другие сдвиги запрещает таблица вытеснения). Итак, сдвиг тега x происходит в результате обращения к x_i в том случае, если $S_i =$ miss (это следует из последней строки таблицы вытеснения) или S_i = hit и тег x_i встречается впервые после x_s, и (согласно таблице вытеснения) x находился перед x_i (из этого условия следует, что $x_i \in L_0$, поскольку в противном случае согласно таблице вытеснения x не мог бы находиться x_i, ибо последний был бы помещен в самое начало списка и без обращения к x взаимное отношение x и x_i не могло быть изменено). Не надо забывать, что таблица вытеснения формулируется только на наборе, таким образом, к сформулированным требованиям осуществления сдвига надо добавить совпадение регионов тегов x и x_i.

Далее будет использовано следующий способ выразить условие «обращении к x_i происходит после обращения к x_s», т.е. $i > s$. А именно, $i > s \Leftrightarrow x \notin \{x_i, ..., x_n\}$. Докажем от противного. Пусть $i > s$ и $x \in \{x_i, ..., x_n\}$. Так как $i > s$, то последнее обращение к x уже произошло и среди $x_i, x_{i+1}, ..., x_n$ тег x больше не встречается. Иными словами, $x \notin \{x_i, x_{i+1}, ..., x_n\}$, что противоречит предположению. В обратную сторону, предположим, что $i \leqslant s$ и $x \notin \{x_i, ..., x_n\}$. Поскольку $i \leqslant s$, то $x_s \in \{x_i, x_{i+1}, ..., x_n\}$. Но так как $x = x_s$, получаем $x \in \{x_i, ..., x_n\}$, что противоречит предположению.

Перефразируя выше сказанное, получим, если S_i = miss, то сдвиг тега x будет осуществлен при выполнении условия $R(x) = R(x_i) \ \wedge \ i > s$. Если S_i = hit, то сдвиг тега x будет осуществлен при выполнении условия $R(x) = R(x_i) \wedge x_i$ встречается впервые $\wedge$ x находится перед x_i после $x_s \ \wedge \ x_i \in L_0$. Ровно в этих же случаях предложенная в формулировке теоремы функция полезности $u(x_i)$ будет равна 1. А именно, при S_i = miss $u(x_i) \equiv R(x) = R(x_i) \ \wedge \ x \notin \{x_i, ..., x_n\}$, что и утверждает теорема. Напомню, что $R(x) \cap L_0 = \{\lambda_1, \lambda_2, ..., \lambda_\delta, ..., \lambda_w\}$, что дает возможность записать при S_i = hit свойство «x находится перед x_i после $x_s \ \wedge \ x_i \in \{\lambda_1, \lambda_2, ..., \lambda_\delta, ..., \lambda_w\}$», что можно еще упростить, учитывая равенство $x = \lambda_\delta$: «$x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\}$».

Осталось при S_i = hit записать свойство «x_i встречается впервые после обращения к x_s». Докажем, что это свойство эквивалентно условию $\forall j < i \ (x \notin \{x_j, x_{j+1}, ..., x_i\} \vee x_i \neq x_j)$ (что эквивалентно $\sum_{j=1}^{i-1} c_i(x_j) = 0$). В очередной раз доказываем от противного. Пусть x_i встречается после x_s впервые, однако существует такой $j < i$, что $x = x_s \notin \{x_j, ..., x_i\} \wedge x_i = x_j$. $x_s \notin \{x_j, ..., x_i\}$ тогда и только тогда, когда $s < j$ (т.к. x_s – это последнее обращение к x, до x_i он точно не встречается). Значит, хотя по предположению в i'й инструкции должно было быть первое обращение к x_i после s'й инструкции, нашлась между ними другая инструкция (j'я), тег в которой тоже равен x_i. Обнаружено противоречие, которое завершает основную цепочку преобразований. Остается только собрать все полученные следствия и получить требуемый вид $u(x_i)$ при $S_i = \text{hit}$.

В обратную сторону лемма доказывается с помощью тех же шагов, что и в прямую сторону. □

Неформально говоря, все инструкции до последнего обращения к вытесняемому элементу считаются бесполезными, а после этого обращения полезным считается лишь первое обращение к элементу, находящемуся между λ_δ и концом списка (т.е. между $\lambda_{\delta+1}$ и λ_w). Функциональный символ c как раз призван считать количество предшествующих обращений к таким элементам с момента последнего обращения к вытесняемому элементу. Если $c = 0$, значит, это первое обращение (предшествующих нет).

Кэш-попадание – второй случай Этот случай представлен фразой «x был внесен $\wedge$ с тех пор не вытеснен». Функции полезности совпадают со случаем, когда x был в начальном состоянии, к нему до вытеснения было обращение и он все еще не вытеснен, потому что с момента последнего обращения поведение списка LRU не зависит от инструкций, предшествовавших последнему обращению. Разница только в том, что в данном случае не известен индекс элемента δ, потому как нет равенства $x = \lambda_\delta$. Но, как оказалось, ограничение можно записать в этом случае и без знания δ – ограничения $R(x_i) = R(x)$ достаточно при условии, что это первое обращение.

Лемма 9 (представление кэш-попадания-II с помощью функций полезности для LRU). *Пусть x – тегсет текущей инструкции при стратегии вытеснения* LRU . *Тогда если $x \in [x_1, x_2, ..., x_n]_{miss}$, где $x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, а $[x_1, x_2, ..., x_n]_{miss}$ – тегсеты, дающие кэш-промах, то x не вытеснен к моменту текущей инструкции согласно определению* LRU *на списках тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) < w$, где $u(x_i)$ (*функция полезности*) определена следующим образом:*

$$u(x_i) = \begin{cases} x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x) \wedge \sum\limits_{j=1}^{i-1} c_i(x_j) = 0, \textit{если } S_i = hit \\ x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x), \textit{если } S_i = miss \end{cases}$$

$$c_i(x_j) \equiv (x \notin \{x_j, x_{j+1}, ..., x_i\} \wedge x_i = x_j)$$

Доказательство. В точности повторяет доказательство леммы 8, за исключением отсутствия условия $x \in L_0$, что не дает возможности упростить выражение $R(x) = R(x_i)$, поэтому оно остается в функции полезности в таком виде. □

Кэш-промах – первый случай Этот случай описывает тегсет, который еще не встречался ни среди предыдущих инструкций тестового шаблона, ни среди начального состояния кэширующего буфера. Он может быть описан вообще без привлечения функций полезности, что и сделаем:

$$\begin{cases} x \notin D \\ x \notin [x_1, x_2, ..., x_n]_{miss} \end{cases} \tag{3.1}$$

$x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, $[x_1, x_2, ..., x_n]_{\text{miss}}$ – тегсеты, дающие кэш-промах.

Кэш-промах – второй случай описывает ситуацию, когда тегсет был внесен в кэширующий буфер одним из предыдущих кэш-промахов, затем некоторым последующим кэш-промахом он был вытеснен и с того момента не был внесен в буфер вновь. Обращение к такому тегсету в данной инструкции вызовет кэш-промах. С помощью функций полезности запишем тот факт, что, начиная с последнего обращения к элементу, было не менее w полезных инструкций. Именно столько раз надо сдвинуть элемент в списке LRU от начала до самого конца, чтобы его вытеснить.

Лемма 10 (представление кэш-промаха-II с помощью функций полезности для LRU)**.** *Пусть x – тегсет текущей инструкции при стратегии вытеснения* LRU *. Тогда если $x \in [x_1, x_2, ..., x_n]_{miss}$, где $x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, а $[x_1, x_2, ..., x_n]_{miss}$ – тегсеты, дающие кэш-*

промах, то x вытеснен к моменту текущей инструкции согласно определению LRU *на списках тогда и только тогда, когда* $\sum_{i=1}^{n} u(x_i) \geqslant w$, *где* $u(x_i)$ *(*функция полезности*) определена следующим образом:*

$$u(x_i) = \begin{cases} x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x) \wedge \sum\limits_{j=1}^{i-1} c_i(x_j) = 0, \text{\it если } S_i = hit \\ x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x), \text{\it если } S_i = miss \end{cases}$$

$$c_i(x_j) \equiv (x \notin \{x_j, x_{j+1}, ..., x_i\} \wedge x_i = x_j)$$

Доказательство. По лемме 9 x не вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) < w$. Значит, x вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) \geqslant w$. □

Кэш-промах – третий случай Это случай обозначен в формулировке утверждения 4 фразой «$x \in D \wedge x$ был вытеснен $\wedge$ не внесен вновь». Возможны два подслучая в зависимости от того, было ли обращение к x до вытесняющей его инструкции (кэш-промах-III' будет соответствовать отсутствию обращений до вытесняющей инструкции, кэш-промах-III", наоборот, наличию такого обращения). Ограничения для этих подслучаев объединены в дизъюнкцию. Для кэш-промаха-III' нужно более $w - \delta$ полезных инструкций ($x = \lambda_\delta$), для кэш-промаха-III" нужно не менее w полезных инструкций.

Лемма 11 (представление кэш-промаха-III' с помощью функций полезности для LRU). *Пусть x – тегсет текущей инструкции при стратегии вытеснения* LRU . *Тогда если* $x = \lambda \in L_0$ *и* $x \notin \{x_1, x_2, ..., x_n\}$, *где* $x_1, x_2, ..., x_n$ *– тегсеты предыдущих инструкций, то x вытеснен к моменту текущей инструкции согласно определению* LRU *на списках тогда и только тогда, когда* $\sum_{i=1}^{n} u(x_i) > w - \delta$, *где* $\delta \in \{1, 2, ..., w\}$ *– индекс λ в своем наборе L_0 согласно метрике вытеснения* LRU , *а* $u(x_i)$ *(*функция полезности*) определена следующим образом:*

$$u(x_i) \equiv \begin{cases} x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge x_i \notin \{x_1, ..., x_{i-1}\}, \textit{если } S_i = hit \\ R(x_i) = R(x), \textit{если } S_i = miss \end{cases}$$

Доказательство. По лемме 7 x не вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) \leqslant w - \delta$. Значит, x вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) > w - \delta$. □

Лемма 12 (представление кэш-промаха-III" с помощью функций полезности для LRU)**.** *Пусть x – тегсет текущей инструкции при стратегии вытеснения LRU. Тогда если $x = \lambda \in L_0$ и $x \in \{x_1, x_2, ..., x_n\}$, где $x_1, x_2, ..., x_n$ – тегсеты предыдущих инструкций, то x вытеснен к моменту текущей инструкции согласно определению LRU на списках тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) \geqslant w$, где $u(x_i)$ (*функция полезности*) определена следующим образом:*

$$u(x_i) = \begin{cases} x \notin \{x_i, ..., x_n\} \wedge x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge \sum_{j=1}^{i-1} c_i(x_j) = 0, \\ \qquad \textit{если } S_i = hit \\ x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x), \textit{если } S_i = miss \end{cases}$$

$\delta \in \{1, 2, ..., w\}$ – индекс λ в своем наборе L_0 согласно метрике вытеснения LRU ($\lambda = \lambda_\delta$),

$$c_i(x_j) \equiv (x \notin \{x_j, x_{j+1}, ..., x_i\} \wedge x_i = x_j$$

Доказательство. По лемме 8 x не вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) < w$. Значит, x вытеснен тогда и только тогда, когда $\sum_{i=1}^{n} u(x_i) \geqslant w$. □

Неформально говоря, все инструкции до последнего обращения к вытесняемому элементу считаются бесполезными, а после этого обращения полезным считается лишь первое обращение к элементу, находящемуся между λ_δ и концом списка (т.е. между $\lambda_{\delta+1}$ и λ_w). Функциональный символ c как раз призван считать количество предшествующих обращений к таким элементам с момента последнего обращения к вытесняемому элементу. Если $c = 0$, значит, это первое обращение (предшествующих нет).

Теорема 8 (корректность использования функций полезности для записи LRU). *Тестовая программа, построенная по ограничениям, которые сгенерированы с использованием предъявленных выше функций полезности, удовлетворяет своему тестовому шаблону.*

Доказательство. Каждая тестовая ситуация представима в виде схемы ограничений, указанных в утверждении 4, а эта схема уточняется до конкретных ограничений с использованием лемм 7 – 12. □

Несколько слов об уменьшении ограничений для всех случаев. Представленные ограничения достаточны для полного описания кэш-попаданий и кэш-промахов. В некоторых случаях однако их количество можно сократить, используя следующие эвристики:

- *тождественные ограничения мощности*: ограничения вида $\sum_{i=1}^{n} a_i \leqslant C$ можно не включать в конъюнкцию, если $C > n$; если $C < 0$, то вся конъюнкция несовместна; если $C = 0$ или $C = n$, то ограничение мощности можно сразу расписать в конъюнкцию вида $\bigwedge_i (a_i = \alpha)$, где $\alpha = 0$, если $C = 0$, и $\alpha = 1$, если $C = n$; аналогично с ограничениями вида $\sum_{i=1}^{n} a_i \geqslant C$;

- *ограничения на δ*: если $\delta + 1 < w$, то функция полезности, в которую входит множество $\{\lambda_{\delta+1}, ..., \lambda_w\}$, равна 0;

- *пересечение тегсетов*: при построении ограничений для нескольких кэширующих буферов, чьи тегсеты могут быть битовыми полями (как, например, в случае кэш-памяти и буфера TLB в микропроцессоре MIPS), возникают конъюнкции ограничений вида $x \in \{x_1, ..., x_n\} \wedge \widehat{x} \notin \{\widehat{y_1}, ..., \widehat{y_m}\}$, где x – тег в одном буфере, а $\widehat{x}$ – тег в другом буфере; поскольку неравенство битовых полей чисел влечет неравенство самих чисел, то общие тегсеты среди $x_1, ..., x_n$ и $y_1, ..., y_m$ можно исключить из ограничения на x.

3.2.2 Метод функций полезности для стратегии вытеснения FIFO

Из сравнения таблиц вытеснения для FIFO и LRU следует, что стратегию вытеснения FIFO можно воспринимать, как частный случай LRU , в котором кэш-попадание не меняет состояния списка LRU . Поэтому и ограничения с функциями полезности для FIFO будем строить на основе уже сформулированных и обоснованных ограничений с функциями полезности для LRU . Кроме того все инструкции с кэш-попаданиями, поскольку они не влияют на вытеснение, можно вообще исключить из ограничений. Получившиеся ограничения показаны в таблице 3.2. Доказательства корректности и полноты этих ограничений идентичны доказательствами для LRU . Символом $[\sum\limits_{i=1}^{n}]_{miss}u(x_i)$ обозначена сумма $u(x_i)$, где $i = 1..n$ и тегсет x_i дает в своей инструкции кэш-промах.

	случай	переменная перебора	система	функция полезности для кэш-попадания	функция полезности для кэш-промаха
кэш-попадание	–	–	–	–	–
кэш-промах	тегсет встречается впервые	–	$\begin{cases} x \notin D \\ x \notin [x_1, ..., x_n]_{miss} \end{cases}$	–	–
	тегсет ранее был внесен одной из инструкций шаблона, затем вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \left[\sum\limits_{i=1}^{n}\right]_{miss} u(x_i) \geqslant w - \delta + 1 \end{cases}$	–	$x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему не было обращений в шаблоне	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \left[\sum\limits_{i=1}^{n}\right]_{miss} u(x_i) \geqslant w - \delta + 1 \end{cases}$	–	$R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему было обращение в шаблоне после последнего внесения в кэш-память	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \in \{x_1, ..., x_n\} \\ \left[\sum\limits_{i=1}^{n}\right]_{miss} u(x_i) \geqslant w \end{cases}$	–	$x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x)$

Таблица 3.2. Таблица систем ограничений в случае стратегии вытеснения FIFO с использованием функций полезности

3.2.3 Метод функций полезности для стратегии вытеснения Pseudo-LRU

При использовании функций полезности не происходит выделение участка тестового шаблона, непосредственно влияющего на вытеснение данного тегсета. Считается, что это влияние начинается с момента появления тегсета в кэширующем буфере. Другое дело, что одни инструкции влияют на его вытеснение (это и есть «полезные» инструкции), а другие – нет.

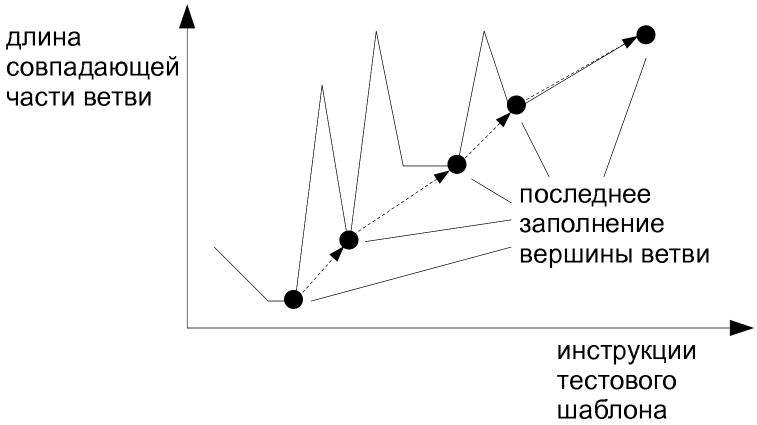


Рис. 3.9. Заполнение ветви черными вершинами в стратегии вытеснения Pseudo-LRU

Рассмотрим в качестве метрики вытеснения для стратегии вытеснения Pseudo-LRU длину черной части ветви, начиная от листовых вершин к корню дерева. Причем вершина будет учитываться в метрике как черная не в тот момент, когда ее перекрашивают, а в тот момент, когда это ее последнее покрашивание в черный цвет. Если таким образом будет закрашена вся ветвь целиком перед кэш-промахом, то листовая вершина будет вытеснена. Представленный на рисунке 3.9 шаблон успевает покрасить 5 вершин ветви в черный цвет.

Утверждение 12. *Инструкция считается полезной в случае стратегии вытеснения* Pseudo-LRU *, если все последующие обращения не затрагивают элементов вершины не выше той, до которой данное обращение совпадает в ветви вытесняемого элемента, после последнего обращения к вытесняемому элементу.*

Количество полезных инструкций, необходимых для вытеснения, зависит от цвета ветви, с которого начинается отсчет полезных инструкций:

если обращение к тегсету было, то нужно не менее $\log_2 w$ инструкций (длина ветви), если обращения не было и тегсет был в кэширующем буфере изначально, то не менее $\log_2 w - n_0$, где n_0 – это количество черных вершин от листовой вершины, изначально покрашенных в ветви, ниже которых нет обращений в тестовом шаблоне. Далее для сокращения записи символ W будет обозначать $\log_2 w$.

Отличие этой метрики вытеснения от метрики вытеснения для LRU является *немонотонность*. Это означает, что полезные инструкции надо считать для каждого кэш-промаха заново — инструкции между двумя соседними кэш-промахами могут забелить несколько вершин ветви, что уменьшит метрику вытеснения (см.рис. 3.10). Метрика для LRU является монотонной, потому что инструкции между кэш-промахами не могут уменьшить метрику вытеснения – либо не меняют, либо увеличивают ее, сдвигая вытесняемый тегсет к концу списка LRU (см. рис. 3.11). Таким образом, ограничение, описывающее стратегию вытеснения PSEUDO-LRU , будет представлено дизъюнкцией ограничений по всем предыдущим кэш-промахам.

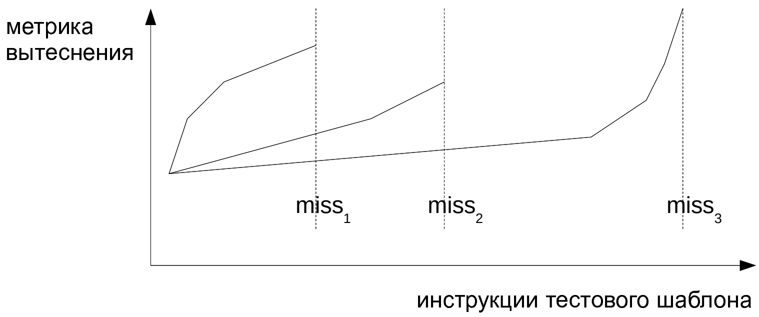


Рис. 3.10. Немонотонная метрика вытеснения

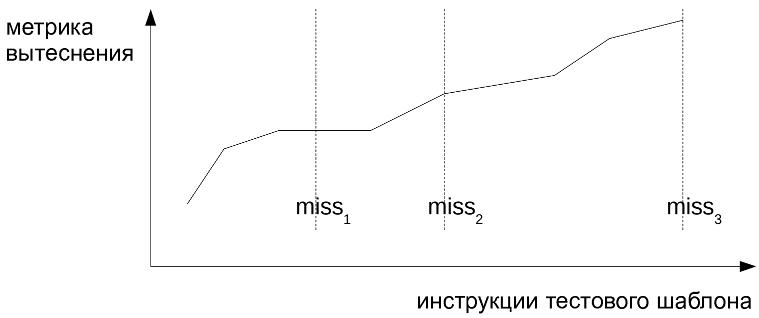


Рис. 3.11. Монотонная метрика вытеснения

Осталось записать понятие полезной инструкции для PSEUDO-LRU в виде ограничений. Напомню, что каждый тегсет кроме своего значения x_i

снабжен позицией π_i ($\pi_i \in \{0..w-1\}$). Пусть считается функция полезности тегсета (x_i, π_i) относительно тегсета (x, π). Пусть выбрана некоторая инструкция с кэш-промахом между i'й и вытесняющей. Пусть (x_{i+1}, π_{i+1}), (x_{i+2}, π_{i+2}), $\ldots$, (x_m, π_m) – тегсеты с позициями инструкций, расположенными между (x_i, π_i) и выбранным кэш-промахом, а $(x_{i+1}, \pi_{i+1}), \ldots, (x_n, \pi_n)$ – тегсеты с позициями инструкций, расположенными между (x_i, π_i) и (x, π). Тогда (x_i, π_i) будет полезным, если выполнены одновременно три условия:

- $x \notin \{x_i, x_{i+1}, \ldots, x_n\}$ – инструкция расположена после последнего обращения к вытесняемому тегсету;
- $R(x) = R(x_i)$ – инструкция принадлежит тому же региону;
- $P(\pi_i \oplus \pi,\ \pi_{i+1} \oplus \pi) \wedge \ldots \wedge P(\pi_i \oplus \pi,\ \pi_m \oplus \pi)$ – все последующие обращения должны пересекаться только в более верхних частях ветви (это выражает предикат P для пары векторов); предикат $P(\delta_i, \delta_j)$ истинен тогда и только тогда, когда количество старших нулевых бит у δ_i больше количества старших нулевых бит у δ_j, иными словами, только и только тогда, когда существует k такое, что $\delta_i < 2^k \leqslant \delta_j$; с использованием битовых операций этот предикат можно записать в следующем виде: $P(\delta_i, \delta_j) \equiv (\delta_j \geqslant 2\delta_i\ \vee\ \delta_j > \delta_i \wedge \delta_j \oplus \delta_i > \delta_i)$, сравнения беззнаковые.

Таблица 3.3 содержит ограничения для разных случаев кэш-попаданий и кэш-промахов (см. утверждение 4). В каждое из них включается ограничение на количество полезных инструкций согласно предлагаемой методике использования функций полезности. Полезности считаются относительно некоторого кэш-промаха (для их перебора используется сокращение x_m : miss).

Лемма 13. *Пусть x – тегсет текущей инструкции при стратегии вытеснения* Pseudo-LRU *. Тогда если $x = \lambda \in L_0$ и $x \notin \{x_1, x_2, \ldots, x_n\}$, где $x_1, x_2, \ldots, x_n$ – тегсеты предыдущих инструкций, то x не вытеснен к моменту текущей инструкции согласно определению* Pseudo-LRU *на списках тогда и только тогда, когда для каждого x_m : miss ($1 \leqslant m \leqslant n$) выполнено $\sum_{i=1}^{m-1} u_m(x_i) \leqslant W - \pi$, где $W = \log_2 w$, $\pi \in \{0, 1, \ldots, w-1\}$ – позиция λ в своем наборе L_0 согласно метрике вытеснения* Pseudo-LRU *,*

	случай	переменная перебора	система
кэш-попадание	тегсет находится в начальном состоянии кэш-памяти, к нему нет обращений до данной инструкции и он всё ещё не вытеснен	$\lambda_p \in L_0$	$\begin{cases} x = \lambda_p \\ x \notin \{x_1, ..., x_n\} \\ \bigwedge\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) \leqslant W - p \end{cases}$
	тегсет находится в начальном состоянии кэш-памяти, к нему есть обращение до данной инструкции и он всё ещё не вытеснен	$\lambda_p \in L_0$	$\begin{cases} x = \lambda_p \\ x \in \{x_1, ..., x_n\} \\ \bigwedge\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) < W \end{cases}$
	тегсет был внесен одним из кэш-промахов и с того момента не вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \bigwedge\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) < W \end{cases}$
кэш-промах	тегсет встречается впервые	–	$\begin{cases} x \notin L_0 \\ x \notin [x_1, ..., x_n]_{miss} \end{cases}$
	тегсет ранее был внесен одной из инструкций шаблона, затем вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \bigvee\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) \geqslant W \end{cases}$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему не было обращений в шаблоне	$\lambda_p \in L_0$	$\begin{cases} x = \lambda_p \\ x \notin \{x_1, ..., x_n\} \\ \bigvee\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) \geqslant W - p + 1 \end{cases}$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему было обращение в шаблоне после последнего внесения в кэш-память	$\lambda_p \in L_0$	$\begin{cases} x = \lambda_p \\ x \in \{x_1, ..., x_n\} \\ \bigvee\limits_{x_m:\text{miss}} \sum\limits_{i=1}^{m-1} u_m(x_i) \geqslant W \end{cases}$

Таблица 3.3. Таблица систем уравнений в случае стратегии вытеснения Pseudo-LRU с использованием функций полезности

а $u_m(x_i)$ *(*функция полезности*) определена следующим образом:*

$$u_m(x_i) \equiv \begin{cases} R(x_i) = R(x) \wedge \bigwedge_{j=i+1}^{m} P(\pi_i \oplus \pi, \pi_j \oplus \pi), \text{\textit{если} } S_i = hit \\ R(x_i) = R(x), \text{\textit{если} } S_i = miss \end{cases}$$

$$P(\delta_i, \delta_j) \equiv (\delta_j \geqslant 2\delta_i \ \vee \ \delta_j > \delta_i \wedge \delta_j \oplus \delta_i > \delta_i)$$

Доказательство. Согласно определению на списках тег x будет вытеснен, если к какому-либо кэш-промаху он будет находиться на позиции $w-1$ (то же самое: если в бинарном дереве к нему будет вести вытесняющий путь, т.е. в каждой нелистовой вершине этого пути направление последнего попадания будет противоположно направлению дуги в следующую вершину пути). Значит, для того, чтобы он не был вытеснен, к каждому кэш-промаху его позиция не должна равняться $w-1$.

На рисунке **??** показана интерпретация этого требования в условиях теоремы. Введем понятие *относительной позиции.* Относительной позицией позиции π_i относительно позиции π будем называть $\pi_i \oplus \pi$. Введение относительных позиций позволяет переставить все (абсолютные) позиции тегов $x_1, x_2, ..., x_n$ так, чтобы (относительная) позиция тега x стала равна нулю (см. рисунок **??**). По горизонтальной оси отложены инструкции тестового шаблона (теги), по вертикальной – позиции тегов относительно π. Далее рассмотрение будет вестись именно на относительных позициях.

[рисунок-точки от оси] [рисунок-точки от нуля] [рисунок-тот же, что и второй, только с выделенными лентами F_1, ...,]

Каждая инструкция влияет на вытеснение тега x, «перекрашивая» вершины пути бинарного дерева, которые ведут к π. Вытеснение происходит в том случае, когда на одном из кэш-промахов путь в π полностью «перекрашен» в черный цвет. Относительная позиция тега позволяет понять, какую часть ветви перекрашивает инструкция с этим тегом. Если относительная позиция принадлежит интервалу $[\frac{w}{2},\ w-1]$, то перекрашивается только верхняя вершина (наиболее удаленная от листовой) в черный цвет, остальные в вершины остаются без изменений. Если относительная позиция принадлежит интервалу $[\frac{w}{4},\ \frac{w}{2}-1]$, то верхняя вершина перекрашивается в белый цвет, а вершина, следующая за верхней, — в черный цвет, остальные вершины остаются без изменений. И так далее.

Эта идея позволяет провести доказательство следующим образом. В последовательности $x_1, x_2, ..., x_n$ выделим подпоследовательность ....

Вводим полосы........... если $\pi_i \in F_k$, то π_i перекрашивает k'ю вершину в черный, $k+1, ..., w$ – в белый.....

Выделяем "спец.инструкции— после которых в ленте и ниже ничего нет......

Надо показать, что количество таких "спец.инструкций"равно сумме функций полезности и что вытеснение $<=>$ количество "спец.инструкций-W ....

$\pi_i \in F_k <=> 2^k <= \pi_i \oplus \pi < 2^{k+1}$

$\square$

Теорема 9 (корректность использования функций полезности для записи Pseudo-LRU). *Тестовая программа, построенная по ограничениям, которые сгенерированы с использованием предъявленных выше функций полезности, удовлетворяет своему тестовому шаблону.*

Доказательство. //TODO $\square$

3.2.4 Разрешение уравнений, описывающих стратегии вытеснения

Ограничения, которые предлагается генерировать для описания тестовых ситуаций в кэш-памяти, можно разделить на две группы: ограничения на конечные множества тегсетов и *ограничения мощности*.

Ограничения вида $C_1 \leqslant \sum_{i=1}^{n} a_i \leqslant C_2$, где C_1, C_2 – неотрицательные целые числа, а a_i принимают значения 0 или 1, называются *ограничениями мощности* (cardinality constraints). Речь идет об ограничении размера некоторого множества элементов, возможно, заданного с помощью характеристической функции. Такие ограничения можно рассматривать, как компактную форму записи уравнения вида $\bigvee_{C_1 \leqslant C \leqslant C_2} \sum_{i=1}^{n} a_i = C$, где равенство есть

- тождественная ложь, если $C < 0$ или $C > n$;
- конъюнкция $\bigwedge_{1 \leqslant i \leqslant n} (a_i = 0)$, если $C = 0$;

- дизъюнкция по всевозможным выборкам индексов $i_1, ..., i_C$, где для каждого индекса i_k справедливы свойства $1 \leqslant i_k \leqslant n$ и $i_k < i_{k+1}$, конъюнкций $\bigwedge_{i_k}(a_{i_k} = 1)$, если $1 \leqslant C \leqslant n$.

Задача организации особой процедуры разрешения ограничений не входила в проводимое исследование, поэтому были использованы имеющиеся инструменты решения систем уравнений и неравенств.

После устранения ограничений мощности в формуле остаются только ограничения на конечные множества тегсетов: принадлежности и непринадлежности тега конечному множеству тегсетов и равенства и неравенства битовых полей тегсетов. Поскольку конечные множества тегсетов известны (заданы перечислением тегсетов, которые в входят в это множество), то ограничения принадлежности и непринадлежности могут быть переписаны без использования этих отношений. Отношение принадлежности $x \in \{x_1, x_2, ..., x_n\}$ может быть переписано в виде дизъюнкции $(x = x_1) \vee (x = x_2) \vee ... \vee (x = x_n)$, а отношение непринадлежности $x \notin \{x_1, x_2, ..., x_n\}$ – в виде конъюнкции $(x \neq x_1) \wedge (x \neq x_2) \wedge ... \wedge (x \neq x_n)$.

В результате получается предикат, в котором переменными величинами являются неотрицательные целые числа с конечной областью значений (тегсеты), над переменными возможны операции получения битового поля, в предикате используется отношение равенства и неравенства над битовыми полями. Кроме того, этот предикат задается с использованием ограничений мощности.

Для разрешения такого рода предикатов можно было бы разрабатывать собственные процедуры распространения ограничений, но это свело бы на нет все усилия по выработке собственного представления стратегии вытеснения. В последний десяток лет разрабатываются инструменты, поддерживающие идею SMT (SAT Modulo Theories) [18, 19]. Задачей для SMT является разрешение предиката, т.е. выяснение наличия модели у этого предиката. Однако язык предикатов для SMT намного богаче языка предикатов для SAT (только пропозициональная логика). Язык предикатов для SMT включает целые числа с линейными операциями и отношениями сравнения целых чисел, термы (неинтерпретируемые функции), битовые строки и др. Этого языка вполне хватает, чтобы выразить в нем генерируемые предикаты для тестовых ситуаций в кэш-памяти.

3.3 Ограничения, описывающие тестовые ситуации в некоторых частных случаях, для стратегии вытеснения LRU

3.3.1 Тестовые шаблоны без кэш-промахов

В случае тестовых шаблонов, в которых нет кэш-промахов, нет ни вытесняющих, ни вытесняемых тегсетов. Поэтому в таких шаблонов уравнения для кэш-попаданий имеют очень простой вид:

$$\begin{cases} x \in D \\ \text{...(тестовая ситуация на буфер TLB)} \end{cases}$$

3.3.2 Тестовые шаблоны без кэш-попаданий

В случае тестовых шаблонов, в которых нет кэш-попаданий, надо генерировать ограничения для вытесняющих и лишь иногда для вытесняемых тегсетов. А именно, вытесняемый тегсет требуется лишь в том случае, когда кэш-промах вносит в кэш-память ранее вытесненный тегсет. В этом случае для вытесняемого тегсета известен домен, что позволяет построить уравнения обозримого размера. Кроме того, поскольку отсутствуют кэш-попадания, повторные обращения к вытесняемым тегсетам (кроме кэш-промаха, который их может внести в кэш-память) невозможны, что также упрощает генерируемые уравнения.

В результате получается, что вытесняющий тегсет описывается в тестовом шаблоне без кэш-попаданий следующей системой уравнений:

$$F'(x) \vee F''(x) \vee \bigvee_{\lambda_\delta \in D} F'''(x, \lambda_\delta)$$

где

$$F'(x) \equiv (x \notin D \wedge x \notin \{x_1, ..., x_n\})$$

$$F''(x) \equiv (x \in \{x_1, ..., x_n\} \wedge \sum_{i=1}^{n} u''(x_i) \geqslant w)$$

$$u''(x_i) \equiv (x \notin \{x_i, ..., x_n\} \wedge R(x_i) = R(x))$$

$$F'''(x, \lambda_\delta) \equiv (x = \lambda_\delta \wedge x \notin \{x_1, ..., x_n\} \wedge \sum_{i=1}^{n} (R(x_i) = R(x)) \geqslant w - \delta + 1)$$

3.3.3 Простые тестовые шаблоны

Рассмотрим еще один класс тестовых шаблонов – т.н. *простые тестовые шаблоны*. Структура этих тестовых шаблонов такова, что все диапазоны вытеснения будут начинаться в начальном состоянии кэш-памяти. Это позволит строить более простые уравнения по сравнению с общим случаем.

Тестовый шаблон называется *простым*, если в нем не более w кэш-промахов.

Теорема 10. *Случай, когда вытесняемый тегсет не находился в начальном состоянии кэш-памяти, а был внесен одной из инструкций тестового шаблона, невозможен для простых тестовых шаблонов.*

Доказательство. //TODO □

3.3.4 Короткие тестовые шаблоны

Будем называть тестовый шаблон *коротким*, если в нем не более w инструкций обращения к памяти. Очевидно, что любой короткий тестовый шаблон является простым. Из 7 случаев для коротких тестовых шаблонов остается всего 5 (первые два можно еще объединить в более компактную систему уравнений).

Теорема 11 (корректность использования функций полезности для записи LRU в коротких тестовых шаблонах)**.** *Тестовая программа, построенная по ограничениям, которые сгенерированы с использованием предъявленных в таблице 3.4 функций полезности, удовлетворяет своему короткому тестовому шаблону.*

Доказательство. //TODO ∎

	случай	переменная перебора	система	функция полезности для кэш-попадания	функция полезности для кэш-промаха
кэш-попадание	тегсет находится в начальном состоянии кэш-памяти и он всё ещё не вытеснен	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ \sum\limits_{i=1}^{n} u(x_i) \leqslant w - \delta \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge x_i \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$
	тегсет уже встречался в шаблоне	–	$x \in \{x_1, ..., x_n\}$	–	–
кэш-промах	тегсет встречается впервые	–	$\begin{cases} x \notin D \\ x \notin \{x_1, ..., x_n\} \end{cases}$	–	–
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен	$\lambda_\delta \in D$, $\delta \geqslant w - n + 1$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) > w - \delta \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_w\} \wedge x \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$

Таблица 3.4. Таблица систем уравнений для тестовых ситуаций в кэш-памяти для коротких тестовых шаблонов в случае стратегии вытеснения LRU

3.3.5 Генерация тестовых данных для кэш-памяти, содержащей «грязные» ячейки

Любая ячейка в кэш-памяти может быть помечена *грязной* (*invalid*). Это означает, что данные, находящиеся в кэш-памяти по этому адресу, не могут использоваться в качестве данных, хранящихся в памяти по этому адресу.

Рассмотренные ранее в этой работе случаи не учитывали грязные ячейки кэш-памяти, хотя они зачастую присутствуют в микропроцессоре после его запуска – с таким состоянием кэш-памяти работают первые после запуска микропроцессора инструкции.

Кэш-попадание возникает в том случае, когда требуемые данные присутствуют среди «чистых» ячеек кэш-памяти. Кэш-промах возникает в том случае, когда требуемых данных нет среди «чистых» ячеек. Причем при наличии «грязных» ячеек вытеснения может и не произойти. А именно, если все ячейки набора, с которым работает инструкция, являются «чистыми», то происходит вытеснение согласно стратегии вытеснения, остальные наборы не меняются. Если же среди ячеек набор есть «грязные» ячейки, то вытеснение не происходит, а на место одной из «грязных» ячеек помещаются данные из основной памяти по заданному адресу и ячейка объявляется «чистой». Остальные ячейки не меняются. В стратегии вытеснения LRU эта бывшая «грязная» ячейка становится самой новой.

Для генерации тестовых данных для кэш-памяти с грязными ячейками предлагается применять ограничения с функциями полезности. Примечательно, что наличие грязных ячеек не меняет качественно систему уравнений.

В данном разделе рассматривается случай, когда начальное состояние микропроцессора известно. Кроме того, рассматриваемый случай учитывает отсутствие инструкций в тестовом шаблоне, которые превращали бы «чистые» ячейки в «грязные» (т.е. все такие изменения должны делаться явно вне тестовых шаблонов).

случай полностью-ассоциативной кэш-памяти

В случае полностью-ассоциативной кэш-памяти очевидно, что первые кэш-промахи будут заполнять «грязные» ячейки. Пусть N – количество «грязных» ячеек в начальном состоянии кэш-памяти, а L_0 – начальное состояние (выражение) кэш-памяти (только «чистые» ячейки). Тогда для тестовых ситуаций надо генерировать такие ограничения (L – выражение для состояния кэш-памяти перед исполнением инструкции, L' – выражение для состояния кэш-памяти после исполнения инструкции):

- для *кэш-попадания* $\mathrm{hit}(x)$ генерируются ограничения

$$\begin{cases} x \in L \\ L' \equiv L \end{cases}$$

- для *кэш-промаха* $\mathrm{miss}(x)$, если это один из первых N кэш-промахов, генерируются ограничения:

$$\begin{cases} x \notin L \\ L' \equiv L \cup \{x\} \end{cases}$$

- для *кэш-промаха* $\mathrm{miss}(x)$, являющегося по счету более чем N'м кэш-промахом тестового шаблона, генерируются ограничения:

$$\begin{cases} x \notin L \\ x' \in L \\ L' \equiv L \setminus \{x'\} \cup \{x\} \\ displaced(x', L) \end{cases}$$

Предикат $displaced(x', L)$ истинен, если x' является вытесняемым тегом в текущем состоянии кэш-памяти L. Для стратегии вытеснения LRU этот предикат может быть записан с использованием тех же диапазонов вытеснения, что и для кэш-памяти без «грязных» ячеек (см.п. 3.1.1). А именно, диапазон вытеснения начинается на инструкции, которая последний раз перед вытеснением тега обращается к нему. Тогда между этой инструкцией и инструкцией, вытесняющей x, должны быть обращения ко всем остальным тегам текущего состояния кэш-памяти. Эта логика может быть записана в виде тех же уравнений, что и в пункте 3.1.1. Нетрудно проверить, что для кэш-памяти с «грязными» ячейками остается справедливой лемма о невложенных диапазонах вытеснения, что доказывает корректность использования ограничений из пункта 3.1.1 для кэш-памяти с «грязными» ячейками.

случай наборно-ассоциативной кэш-памяти

Рассмотрим совместную генерацию тестовых данных для кэш-памяти и TLB. Как и прежде, заметим, что в зависимости от значения виртуального адреса, обращения в память можно разделить по двум критериями: Cached-unCached и Mapped-unMapped. Для полностью ассоциативной кэш-памяти и TLB генерация ограничений рассмотрена в предыдущем пункте. Генерация ограничений для наборно-ассоциативной кэш-памяти будет рассмотрена здесь. Для примера рассмотрим случай Cached-Mapped. В этом пункте будет показано, что ограничения для кэш-памяти, начальное состояние которой содержит «грязные» ячейки, качественно не отличаются от ограничений для кэш-памяти без «грязных» ячеек.

Аналогично тому, как это делалось для кэш-памяти без «грязных» ячеек, для тестовых ситуаций на кэш-память с «грязными» ячейками тоже возможно следующее исчерпывающее выделение случаев:

- кэш-попадание тега:
    1. данный тег находился в начальном состоянии кэш-памяти и не был вытеснен к моменту кэш-попадания;
    2. данный тег был внесен в кэш-память одной из инструкций кэш-промаха и с тех пор не был вытеснен.
- кэш-промах тега:
    1. данный тег не встречался ранее (не находился в начальном состоянии кэш-памяти и не был внесен какими-либо кэш-промахами);
    2. данный тег был ранее вытеснен из кэш-памяти и с тех пор не был внесен в кэш-память вновь.

Соответствующие ограничения приведены в таблице 3.5.

	случай	переменная перебора	система	функция полезности для кэш-попадания	функция полезности для кэш-промаха
кэш-попадание	тегсет находится в начальном состоянии кэш-памяти, к нему нет обращений до данной инструкции и он всё ещё не вытеснен	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) \leqslant w - \delta \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_\Delta\}$ $\wedge\ x_i \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$
	тегсет находится в начальном состоянии кэш-памяти, к нему есть обращение до данной инструкции и он всё ещё не вытеснен	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \in \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) < w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ x_i \in \{\lambda_{\delta+1}, ..., \lambda_\Delta\}$ $\wedge\ \sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
	тегсет был внесен одним из кэш-промахов и с того момента не вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \sum\limits_{i=1}^{n} u(x_i) < w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)\ \wedge$ $\sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
кэш-промах	тегсет встречается впервые	–	$\begin{cases} x \notin D \\ x \notin [x_1, ..., x_n]_{miss} \end{cases}$	–	–
	тегсет ранее был внесен одной из инструкций шаблона, затем вытеснен	–	$\begin{cases} x \in [x_1, ..., x_n]_{miss} \\ \sum\limits_{i=1}^{n} u(x_i) \geqslant w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)\ \wedge$ $\sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему не было обращений в шаблоне	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \notin \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) \geqslant w - \delta + 1 \end{cases}$	$x_i \in \{\lambda_{\delta+1}, ..., \lambda_\Delta\}$ $\wedge\ x_i \notin \{x_1, ..., x_{i-1}\}$	$R(x_i) = R(x)$
	тегсет находился в начальном состоянии кэш-памяти и был вытеснен, к нему было обращение в шаблоне после последнего внесения в кэш-память	$\lambda_\delta \in D$	$\begin{cases} x = \lambda_\delta \\ x \in \{x_1, ..., x_n\} \\ \sum\limits_{i=1}^{n} u(x_i) \geqslant w \end{cases}$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ x_i \in \{\lambda_{\delta+1}, ..., \lambda_\Delta\}$ $\wedge\ \sum_{j=1}^{i-1} c_i(x_j) = 0,$ $c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\}$ $\wedge\ x_i = x_j)$	$x \notin \{x_i, ..., x_n\}$ $\wedge\ R(x_i) = R(x)$

Таблица 3.5. Таблица систем уравнений для тестовых ситуаций в кэш-памяти с «грязными» ячейками в начальном состоянии, использующих функции полезности

В таблице 3.5 символ Δ означает количество «чистых» ячеек в начальном состоянии того региона, про который идет речь в уравнении. На самом деле Δ есть функция региона ($\Delta = \Delta(\lambda_\delta)$, но для сокращения записи оставлен только функциональный символ. Кроме того, в приведенных уравнениях домен переменной включает только «чистые» ячейки.

Сходства уравнений (со случаем кэш-памяти без «грязных» ячеек) удалось добиться за счет рассмотрения «грязных» ячеек, как ячеек с наименьшим счетчиком LRU, которые не участвуют в определении нахождения тега в кэш-памяти. Поэтому в функциях полезности участвуют множества не $\{\lambda_{\delta+1}, ..., \lambda_w\}$, а множества $\{\lambda_{\delta+1}, ..., \lambda_\Delta\}$. Все «чистые» ячейки получили первые индексы, т.е. индексы всех от 1 до Δ.

Теорема 12 (корректность использования функций полезности для записи LRU в случае наличия «грязных» ячеек в начальном состоянии кэширующего буфера)**.** *Тестовая программа, построенная по ограничениям, которые сгенерированы с использованием предъявленных в таблице 3.5 функций полезности, в случае наличия «грязных» ячеек в начальном состоянии кэширующего буфера удовлетворяет своему тестовому шаблону.*

Доказательство. //TODO □

Для приведенных ограничений также могут быть применены эвристики, сокращающие их количество, которые были упомянуты для кэш-памяти без «грязных» ячеек. Кроме того, в данном случае возможна дополнительная эвристика *ограничение на* δ: если $\delta + 1 < \Delta$, то функция полезности, в которую входит множество $\{\lambda_{\delta+1}, ..., \lambda_\Delta\}$, равна 0.

3.3.6 Функции полезности для зеркальной генерации тестовых данных

Рассмотрим ограничения, генерируемые для тестовых шаблонов зеркальным методом с использованием функций полезности. По сравнению с представленными ограничениями (см. табл. 3.1) зеркальная генерация имеет свои особенности:

1. множества констант (как, например, L, D) не используются, поэтому в ограничениях будут отсутствовать соответствующие им случаи;

2. так как теги инструкций тестового шаблона должны появиться среди инициализирующей последовательности, то для вытеснения требуется $w-1$ инструкций, где w – ассоциативность кэширующего буфера;

3. учет полезных инструкций начинается уже в инициализирующей последовательности, тем самым необходимо сформулировать функцию полезности для инициализирующих инструкций.

Следующая теорема описывает функцию полезности для инициализирующих инструкций и описывает ограничения, генерируемые для тестовых шаблонов зеркальным методом с использованием функций полезности (количество инициализирующих инструкций зафиксировано, оно будет обозначено параметром m):

Теорема 13 (Корректность ограничений, генерируемые зеркальным методом с использованием функций полезности для LRU)**.** *Пусть $t_1, t_2, ..., t_m$ – теги инициализирующей последовательности, x – текущий тег тестового шаблона, $x_1, x_2, ..., x_n$ – теги предыдущих инструкций тестового шаблона. Тогда x удовлетворяет тестовой ситуации согласно определению на списках тогда и только тогда, когда:*

- *если текущая инструкция дает кэш-попадание, то*

$$\begin{cases} x \in \{t_1, ..., t_m, x_1, ..., x_n\} \\ \sum\limits_{i=1}^{m} u_x(t_i) + \sum\limits_{i=1}^{n} u_x(x_i) < w \\ \{t_1, ..., t_m\} - \textit{все разные} \end{cases}$$

- *если текущая инструкция дает кэш-промах, то*

$$\begin{cases} x \in \{t_1, ..., t_m, x_1, ..., x_n\} \\ \sum\limits_{i=1}^{m} u_x(t_i) + \sum\limits_{i=1}^{n} u_x(x_i) \geqslant w \\ \{t_1, ..., t_m\} - \textit{все разные} \end{cases}$$

где функции полезности определены следующим образом:

$$u_x(t_i) \equiv (x \notin \{t_i, ..., t_m, x_1, ..., x_n\} \wedge R(x) = R(t_i))$$

$$u_x(x_i) \equiv (x \notin \{x_i, ..., x_n\} \wedge R(x) = R(x_i)),$$
$$\textit{если инструкция с } x_i \textit{ содержит кэш-промах}$$

$$u_x(x_i) \equiv (x \notin \{x_i, ..., x_n\} \wedge R(x) = R(x_i)$$
$$\wedge \sum_{j=1}^{n} \tilde{c}_{x_i}(t_j) = 0 \wedge \sum_{j=1}^{i-1} c_i(x_j) = 0),$$
$$\textit{если инструкция с } x_i \textit{ содержит кэш-попадание}$$

$$c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\} \wedge x_i = x_j)$$

$$\tilde{c}_{x_i}(t_j) \equiv (x \notin \{t_j, ..., t_m, x_1, ..., x_{i-1}\} \wedge x_i = t_j)$$

$$c_i(x_j) \equiv (x \notin \{x_j, ..., x_i\} \wedge x_i = x_j)$$

Доказательство. //TODO □

Стоит заметить, что функции полезности добавили новое дополнительное условие на теги инициализирующих инструкций: они должны быть различными. В этом выражается свойство «простоты» инициализирующей последовательности, эта последовательность не должна содержать сложной внутренней последовательности изменений состояния кэширующего буфера.

Рассмотрим один часто встречающийся случай кэширующих буферов, инициализация которого может вызывать трудности. Речь идет о кэш-памяти второго уровня. Зачастую кэш-память второго уровня не может быть инициализирована отдельно от остальных подсистем микропроцессора, обычно оно связано с изменением кэш-памяти первого уровня. Это создает дополнительные сложности при формулировании ограничений методом зеркальной генерации, поскольку инициализирующая последовательность должна подготавливать сразу два кэширующих буфера одновременно – кэш-память первого уровня и кэш-память второго уровня. Кроме того, зачастую кэш-память второго уровня является совместной для хранения в ней данных и инструкций. Поэтому на инициализацию кэш-памяти второго уровня влияют и сами инициализирующие инструкции, и даже адрес расположения тестовой программы в памяти (от него зависит виртуальный адрес инструкций, а значит теги и индексы при обращении к кэш-памяти инструкций).

Если принять дополнительное требование (и оно даст решение), что в кэш-памяти второго уровня наборы, используемые для доступа к инструкциям, не пересекаются с наборами, используемыми для доступа к данным, то генерируемые ограничения упрощаются (кэширование инструкций можно вообще не учитывать). С точки зрения зеркальной генерации это означает, что надо сформулировать требования на инициализирующую последовательность. Напомню, что одним из ключевых требований является произвольность начального состояния (содержимого) кэш-памяти.

Предположим, что обращение к кэш-памяти второго уровня осуществляется при кэш-промахе в кэш-памяти первого уровня и кэш-память не является virtually indexed virtually tagged [?]. Для составления ограничений с использованием функций полезности необходимо знать, которые инструкции среди инициализирующей последовательности действительно обращаются в кэш-память второго уровня (иными словами, в каких инструкциях среди инициализирующей последовательности происходит кэш-промах при обращении к кэш-памяти первого уровня). Возможным решением было бы перебирать всевозможные распределения тестовых ситуаций в кэш-памяти первого уровня на элементах инициализирующей последовательности (с предварительной подготовкой этих тестовых ситуаций). Однако следующая лемма 14 показывает, что для любого такого произвольного распределения тестовых ситуаций в кэш-памяти первого уровня существует решение со специальным распределением тестовых ситуаций. Это позволяет перебирать только такие специальные распределения тестовых ситуаций в кэш-памяти первого уровня. При этом вычислительная сложность процедуры поиска инициализирующей последовательности, дающей решение, изменяется от экспоненциальной от длины тестового шаблона к полиномиальной, что показывает лемма 15

Лемма 14 (О существовании специальной инициализации кэш-памяти). *Если для данного тестового шаблона (S_i, x_i), где $i = 1, 2, ..., n$, $S_i \in \{l1Hit, l1Miss$ x_i – тегсет, существует некоторое решение, полученное зеркальным методом, а именно, $t_1, ..., t_m$ – инициализирующая последовательность и*

$v_1, ..., v_n$ – значения тегсетов тестового шаблона, то для этого же тестового шаблона существует и решение следующего вида: инициализирующая последовательность состоит из трех подпоследовательностей $s_1, ..., s_k$, $p_1, ..., p_l$, $q_1, ..., q_r$, где при обращении к тегсетам $p_1, ..., p_l$ происходят кэш-промахи в кэш-памяти первого уровня, при обращении к тегсетам $q_1, ..., q_r$ происходят кэш-попадания в кэш-памяти первого уровня, последовательность тегсетов $s_1, ..., s_k$ обеспечивают выполнение тестовых ситуаций в кэш-памяти первого уровня для последующих элементов инициализирующей последовательности, значения тегсетов тестового шаблона те же, $v_1, ..., v_n$.

Доказательство. //TODO □

Лемма 15 (Верхняя оценка длины специальной инициализирующей последовательности)**.**

$$0 \leqslant k \leqslant 3|\mathit{l2Hit}| * w_1 + |\mathit{l2Miss}| * (w_2 + 2) * w_1 + |\mathit{l1Hit}|$$

$$0 \leqslant l \leqslant |\mathit{l2Hit}| + |\mathit{l2Miss}| * w_2$$

$$0 \leqslant r \leqslant |\mathit{l1Hit}| + 2|\mathit{l2Hit}| * w_1 + 2|\mathit{l2Miss}| * w_1$$

где w_1 – ассоциативность кэш-памяти первого уровня, w_2 – ассоциативность кэш-памяти второго уровня, $|\mathit{l1Hit}|$ – количество инструкций в тестовом шаблоне с кэш-попаданием в кэш-памяти первого уровня, $|\mathit{l2Hit}|$ – количество инструкций в тестовом шаблоне с кэш-попаданием в кэш-памяти второго уровня, $|\mathit{l2Miss}|$ – количество инструкций в тестовом шаблоне с кэш-промахом в кэш-памяти второго уровня.

Доказательство. //TODO □

Следствие.

$$0 \leqslant m \leqslant n * (w_1 * w_2 + 5w_1 + w_2 + 3)$$

где m – длина специальной инициализирующей последовательности, n – длина тестового шаблона, w_1 – ассоциативность кэш-памяти первого уровня, w_2 – ассоциативность кэш-памяти второго уровня.

Для получения инициализирующей программы минимальной длины, можно применять сначала двоичный поиск суммы $k+l+r$ с применением дальнейшего поиска допустимых значений k, l и r.

Глава 4

Программная реализация

4.1 Структура генератора тестовых программ

В главе описывается система генерации тестовых программ на основе тестовых шаблонов. Входными данными системы являются:

- тестовый шаблон;
- описания тестовых ситуаций;
- начальное состояние микропроцессора;
- дополнительные параметры конфигурации.

Выходом системы является тестовая программа, удовлетворяющая тестовому шаблону с учетом данных описаний тестовых ситуаций и начального значения микропроцессора. Ядром системы является *генератор ограничений* (см. рис. 4.1). Ограничения разрешаются другим компонентом системы – *решателем ограничений*. Модель, построенную решателем ограничений, анализирует *генератор инструкций тестовой программы*, с целью построить готовую тестовую программу.

На рис. 4.2 показано сравнение предлагаемого генератора тестовых программ с известным генератором Genesys-Pro [11]. Оба генератора получают на вход тестовый шаблон, а на выходе у них тестовые программы. Однако Genesys-Pro на вход требует *architectural model* и *testing knowledge* – первая дает по сути эталонный симулятор микропроцессора, а второй описывает

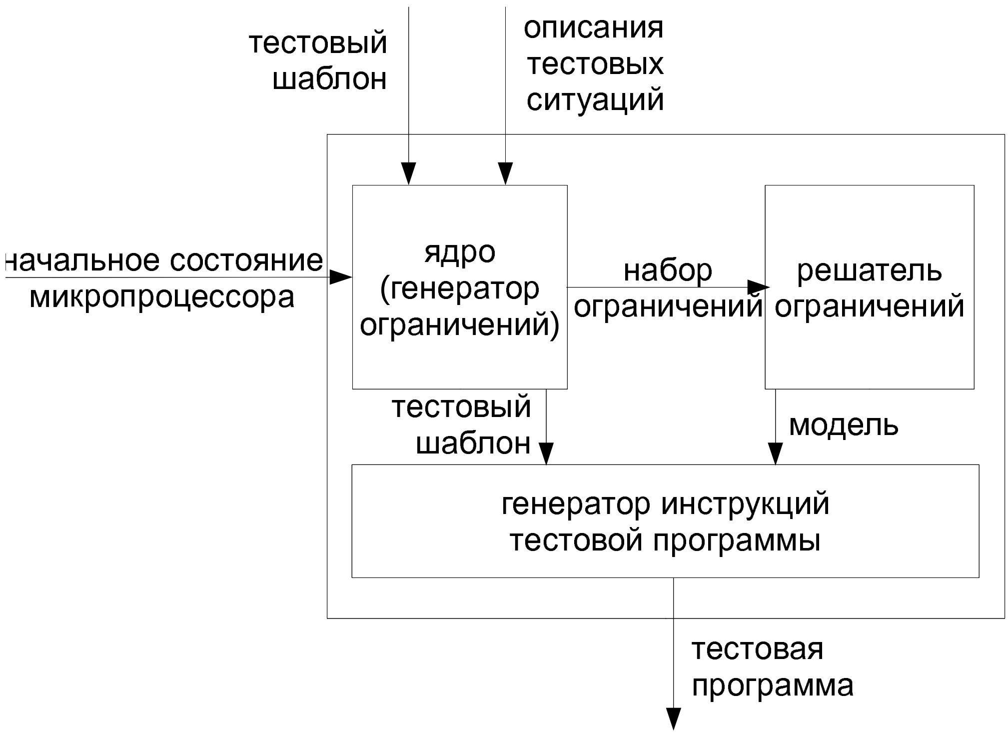


Рис. 4.1. Структура системы генерации тестовых программ

эвристики выбора аргументов для инструкции. Genesys-Pro не предполагает систематического описания семантики инструкций, выделения ветвей их функциональности, описания программных контрактов инструкций. Симулятор нужен для построения модельного состояния микропроцессора после исполнения очередной сгенерированной инструкции, а эвристики выбора аргументов составляют основу тех ограничений, которые описывают значения аргументов очередной инструкции. Идея заключается в разделении описания функции, которую реализует инструкция, и ограничения на аргументы инструкции. Другой особенностью Genesys-Pro является то, что поддерживаемые им тестовые шаблоны зачастую не фиксируют последовательность инструкций (это позволяет строить более простые ограничения, потому как генерируемая последовательность инструкций может по ходу генерации подстраиваться под уже сгенерированные инструкции со сгенерированными значениями аргументов, под состояние микропроцессора, в которое привели сгенерированные инструкции).

В отличие от Genesys-Pro в предлагаемом инструменте описание семантики инструкций задается в едином виде – в виде описаний тестовых ситуаций [29, 3]. Каждая тестовая ситуация описывает не только ограничение на свои аргументы, но и результат исполнения инструкции *при данном ограни-*

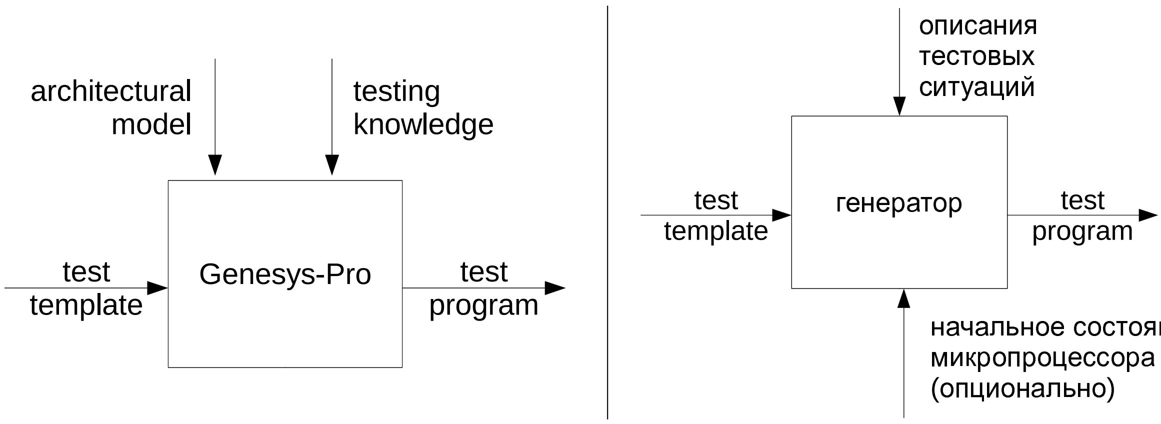


Рис. 4.2. Сравнение с Genesys-Pro

чении на аргументы инструкции. В функции, которую реализует инструкция, выделяются отдельные *ветви функциональности*, ситуации различного поведения инструкций, каждая ветвь функциональности становится отдельной тестовой ситуацией. Например, инструкция целочисленного сложения ADD может быть исполнена либо точно, либо с возникновением переполнения. Поэтому у этой инструкции можно выделить 2 ветви функциональности (точное исполнение и исполнение с переполнением), каждая ветвь дает свою тестовую ситуацию. Кроме того, предлагаемый генератор дает возможность указать начальное состояние микропроцессора, которое будет эффективно использовано при построении тестовой программы. Последовательность инструкций фиксирована и задается в тестовом шаблоне.

Описания тестовых ситуаций можно составлять по следующей схеме:

1. выделить тестируемые инструкции;

2. найти описание семантики выбранных инструкций (обычно оно входит в документацию по тестируемой архитектуре);

3. для каждой инструкции выделить:

   - аргументы: имена и битовые длины;
   - предусловие (ограничение на значения аргументов инструкции, при которых она может быть результативно исполнена);
   - ветви функциональности инструкции (ситуации различного поведения инструкции);

4. для каждой ветви функциональности составить описание тестовой ситуации, поместив туда объявления аргументов, предусловие и операторы, описывающие поведение инструкции на данной ветви функциональности, т.е. вычисление выходного значения инструкции или создание условий возникновения исключительной ситуации.

Раздел 4.2 содержит описание предлагаемого языка описания тестовых шаблонов и тестовых ситуаций, пригодный для описания инструкций арифметической, логической подсистем, подсистемы обращения к памяти, инструкции переходов.

4.2 Описание тестовых шаблонов

Описание тестового шаблона состоит из следующих секций:

1. *заголовок шаблона*: объявление регистров и констант;

2. *тело шаблона*: инструкции и ограничения тестового шаблона.

Заголовок шаблона должен содержать объявления всех задействованных в тестовом шаблоне регистров (для каждого регистра указывается его имя и битовая длина) и констант (или по-другому, «непосредственных значений» – для каждой константы так же указывается ее имя и битовая длина). Регистр может использоваться в качестве аргумента-результата инструкции, константа не может использоваться в качестве аргумента-результата инструкции. Регистры и константы сохраняют свою битовую длину на протяжении всего тестового шаблона. Генератор ограничений трактует регистр как переменную со значением и генерирует начальное значение такой переменной, при которой выполнены все заявленные в тестовом шаблоне тестовые ситуации. Обычно для инициализации регистра достаточно одной-двух инструкций (это зависит от количества бит, которое может изменить одна инструкция). Константа трактуется как значение, которое не меняется. Для нее тоже генерируется значение и при составлении тестовой программы оно вставляется непосредственно на место аргумента инструкций.

Пример объявления регистра и константы:

```
<register id="z" length="64" />
<constant id="c" length="16" />
```

Тело тестового шаблона состоит из описаний инструкций и ограничений. Для инструкции необходимо указать:

- имя инструкции;
- аргументы инструкции (объявленные ранее регистры или константы);
- внешние переменные инструкции;
- тестовую ситуацию.

Тестовые ситуации на косвенные обращения не рассматриваются, поэтому в описания тестовых шаблонов не включены механизмы описания косвенных обращений.

Механизм внешних переменных инструкции позволяет формулировать ограничения на локальные переменные, определенные в разных тестовых ситуациях. При объявлении новой локальной переменной кроме имени можно указать идентификатор (имя надо указывать обязательно, а идентификатор – необязательно). Все идентификаторы внутри тестовой ситуации должны быть уникальными. Это может быть виртуальный адрес, физический адрес, некое внутреннее выражение. Если тестовая ситуация является составной, то все тестовые ситуации должны определять выносимые на уровень тестового шаблона идентификаторы. На уровне тестового шаблона идентификатору ставится в соответствие некоторое новое уникальное внутри шаблона имя, которое можно использовать наравне с другими переменными (регистрами или константами).

Тестовая ситуация описывает некоторое поведение инструкции. Обычно можно выделить два типа поведений инструкции: существенное исполнение (вычисление значения, осуществление взаимодействия) и генерация исключения. При существенном исполнении тестовая ситуация описывает предусловие инструкции и набор условий и вычислений, определяющих данное поведение инструкции. При исполнении с генерацией исключения описывается предусловие инструкции и набор условий и вычислений, приводящих к возникновению исключения. Само возникновение исключения, как оператор, не описывается.

Тестовая ситуация состоит из набора *ветвей* – простейших поведений инструкции. Ветви могут *комбинироваться* в дизъюнкции и конъюнкции.

Дизъюнкция ветвей (или их комбинаций) означает, что в данный момент инструкция может себя вести согласно хотя бы одной из ветвей. Конъюнкция ветвей (или их комбинаций) означает, что в данный момент инструкция может себя вести согласно всем ветвям одновременно. Объединенные в конъюнкцию ветви не могут иметь существенное исполнение.

Пример тестовой ситуации ветви:

```
<situation>
    <branch name="overflow" />
</situation>
```

Для ветви указывается имя. Оно используется при поиске соответствующего файла с описанием этой тестовой ситуации. Поиск производится на основе имени тестовой ситуации и имени инструкции, для которой она указана.

Пример тестовой ситуации, включающей комбинацию ветвей:

```
<situation>
    <or>
        <and>
            <branch name="overflow" />
            <branch name="normal" />
        </and>
        <branch name="zero" />
    </or>
</situation>
```

Эту комбинацию ветвей можно прочесть следующим образом: *данная инструкция должна себя вести как overflow с normal или как zero*.

В описаниях тестовых ситуациях могут быть фиксированы обращения к различным подсистемам микропроцессора. Указание тестовой ситуации в шаблоне может фиксировать и тестовую ситуацию на эти обращения (а для полных тестовых шаблонов оно *должно* фиксировать тестовую ситуацию на эти обращения). Среди подсистем можно выделить различные уровни кэш-памяти и TLB. Содержимое секции тестовых ситуаций обращений к подсистемам определяется архитектурой тестируемого микропроцессора. Например, оно может быть следующим:

```
<situation>
        ...
        <access>
```

```
            <cache level="1" type="DATA" id="miss" />
            <cache level="2" type="DATA" id="miss" />
            <cache level="3" type="DATA" id="hit" />
            <tlb  id="invalid">
                <microtlb type="DATA" id="miss" />
            </tlb>
        </access>
</situation>
```

Это описание говорит о том, что при исполнении данной инструкции в кэш-памяти данных первого и второго уровней должен быть кэш-промах, а в кэш-памяти данных третьего уровня – кэш-попадание; в TLB должна произойти тестовая ситуация invalid, а в MicroTLB данных – промах. Интерпретация терминов `cache, tlb, microtlb, miss, hit, invalid` заложена в части генератора ограничений, ответственного за тестируемую архитектуру.

Кроме инструкций тестовый шаблон может содержать ограничения (assert) на текущее состояние микропроцессора и введенные внешние переменные. Можно выделить следующие основные применения этих ограничений:

1. задание зависимостей на адреса разных инструкций (например, у двух инструкций одинаковые физические адреса при разных виртуальных адресах – одна инструкция определяет свои виртуальный и физический адрес, другая инструкция делает то же, а ограничение фиксирует связь значений этих четырех переменных);

2. задание ветви в графе потока управления: при тестировании инструкций перехода с некоторым сравнением вместо их непосредственного описания предлагается описывать конкретные результаты сравнений (истинно это сравнение в данный момент или ложно); тестовый шаблон не позволяет описывать разветвленные потоки управления, разрешается описывать лишь последовательности инструкций, поэтому такие дополнительные ограничения позволяют описать в тестовом шаблоне один из путей в графе потока управления и сгенерировать для этого пути свою тестовую программу.

Описание ограничения, как и описание тестовой ситуации, может состоять из указания ветви или их комбинаций. Пример:

```
<assert>
    <or>
        <and>
            <branch name="ff1" />
            <branch name="ff2" />
            <branch name="ff3" />
        </and>
        <branch name="ff4" />
    </or>
</assert>
```

Пример описания тестового шаблона целиком:

```
<template>
  <register id="x" length="64" />
  <register id="y" length="64" />
  <register id="z" length="64" />
  <constant id="c" length="16" />

  <instruction name="ADD">
      <argument name="x" />
      <argument name="y" />
      <argument name="z" />
      <external name="v1" id="virtual" />
      <external name="p1" id="phys" />
      <situation>
          <or>
              <and>
                  <branch name="overflow" />
                  <branch name="normal" />
              </and>
              <branch name="zero" />
          </or>

          <access>
              <cache level="1" type="DATA" id="miss" />
              <cache level="2" type="DATA" id="miss" />
              <cache level="3" type="DATA" id="hit" />
              <tlb  id="invalid">
                  <microtlb type="DATA" id="miss"></microtlb>
              </tlb>
          </access>
```

```
        </situation>
    </instruction>

    <assert>
        <or>
            <and>
                <branch name="ff1" />
                <branch name="ff2" />
                <branch name="ff3" />
            </and>
            <branch name="ff4" />
        </or>
    </assert>
</template>
```

4.3 Описание тестовых ситуаций

Описание тестовой ситуации состоит из следующих секций:

1. *заголовок тестовой ситуации*: объявление аргументов инструкции;

2. *тело тестовой ситуации*: последовательность операторов.

Заголовок тестовой ситуации должен содержать объявления всех аргументов инструкции. Для каждого аргумента указывается его имя (локальное внутри данной тестовой ситуации), статус «только для чтения/результат» и битовая длина. Последовательность аргументов тестовой ситуации должна совпадать с последовательностью аргументов инструкции с точностью до переименования. Иными словами, первый аргумент тестовой ситуации должен обозначать первый аргумент инструкции (их битовые длины должны совпадать), второй аргумент тестовой ситуации – второй аргумент инструкции и так далее. Аргументы, помеченные статусом «только для чтения» не могут менять свое значение во время всей тестовой ситуации. Аргументы, помеченные статусом «результат» обязаны получить значение в данной инструкции. Тестовая ситуация может иметь произвольное количество аргументов обоих статусов в произвольном порядке. Статус «только для чтения» позволяет передать в качестве аргументов инструкции одинаковые переменные (одинаковые регистры или константы). Однако все аргументы инструкции, соответствующие аргументам тестовой ситуации со статусом «результат», должны иметь разные имена (они могут быть среди аргументов со статусом «только для чтения»).

Пример:

```
<argument name="rt" state="result" length="64"/>
<argument name="base" state="readonly" length="64"/>
<argument name="offset" state="readonly" length="16"/>
```

Тело тестовой ситуации состоит из последовательности операторов трех видов:

- оператор `let` – объявление новой локальной переменной вместе с ее инициализацией;

- оператор `assert` – фиксация некоторого ограничения на значения переменных;

- оператор `procedure` – вызов процедуры (ее семантика не задается в описании тестовой ситуации).

Тестовая ситуация по своей сути является ветвью функциональности инструкции. Поэтому ее описание содержит лишь последовательность операторов, условные операторы и операторы цикла отсутствуют.

Оператор `let` объявляет новую переменную и инициализирует ее результатом вычисления некоторого выражения. Оператор может содержать указание имени переменной (оно должно быть новым) *или* указание идентификатора новой переменной (все идентификаторы внутри тестовой ситуации должны быть разными; идентификатор может совпадать с именем этой или любой другой переменной). Безымянные операторы `let` позволяют задать идентификатор существующей переменной.

Тело оператора содержит выражение, результат вычисления которого станет значением объявляемой переменной. Выражение может содержать следующие операции:

- переменные (`var`) и константы (`constant`); допустимы только неотрицательные константы, у каждой константы должна быть указана битовая длина;
- битовые операции: выделение бита с заданными номером (`bit`), выделение непрерывной последовательности бит с заданными границами (`bits`), битовая конкатенация (`concat`), битовая степень (`power`);
- арифметические операции: сложение (`sum`), вычитание (`sub`); операции проводятся по модулю экспоненты битовой длины аргументов.

Производится строгая «проверка типов», т.е. битовых длин аргументов операций. Например, сложению подвергаются только выражения с одинаковыми битовыми длинами. В частности это позволяет автоматически вычислить битовую длину объявляемой переменной.

Пример:

```
<let name="vAddr" id="virtual">
    <sum>
        <sign_extend size="64"><var>offset</var></sign_extend>
        <var>base</var>
    </sum>
</let>
```

Оператор `assert` позволяет указать ограничение, справедливое в некоторый момент на значениях аргументов тестовой ситуации и локальных переменных. Выражения объединяются с помощью отношений сравнения. Пример:

```
<assert>
    <eq>
        <bits end="1" start="0"><var>vAddr</var></bits>
        <constant length="2">0</constant>
    </eq>
</assert>
```

Оператор `procedure` позволяет указать более сложное действие, в котором могут участвовать различные подсистемы микропроцессора. Семантика процедур не фиксируется при описании тестовой ситуации. Аргументы, наоборот, фиксируются. Каждый аргумент может иметь идентификатор (это позволяет располагать аргументы в произвольном порядке, указывая семантику каждого аргумента с помощью идентификатора). Тело аргумента может быть следующих типов:

- выражение на определенные к моменту вызова процедуры переменные; выражение задается с использованием того же синтаксиса, что и в операторе `let`;
- новая переменная (`new`);
- символьная константа (`symbol`).

Набор допустимых процедур и идентификаторов их аргументов задается генератором ограничений.

Пример:

```
<procedure name="AddressTranslation">
        <argument id="physical"><new name="pAddr"/></argument>
        <argument id="virtual"><var>vAddr</var></argument>
        <argument id="points_to"><symbol name="DATA"/></argument>
        <argument id="points_for"><symbol name="LOAD"/></argument>
</procedure>
```

Пример описания тестовой ситуации целиком:

```
<situation>
    <argument name="rt" state="result" length="64" />
    <argument name="base" state="readonly" length="64" />
    <argument name="offset" state="readonly" length="16" />

    <let name="vAddr" id="virtual">
        <sum>
            <sign_extend size="64"><var>offset</var></sign_extend>
            <var>base</var>
        </sum>
    </let>

    <assert>
        <eq>
            <bits end="1" start="0"><var>vAddr</var></bits>
            <constant length="2">0</constant>
        </eq>
```

```
    </assert>

    <procedure name="AddressTranslation">
        <argument id="physical"><new name="pAddr"/></argument>
        <argument id="virtual"><var>vAddr</var></argument>
        <argument id="points_to"><symbol name="DATA"/></argument>
        <argument id="points_for"><symbol name="LOAD"/></argument>
    </procedure>

    <let id="physical"><var>pAddr</var></let>

    <let name="dwByteOffset">
        <bits end="2" start="0"><var>vAddr</var></bits>
    </let>

    <!-- dwByteOffset can be changed
          according to BigEndian/LittleEndian -->

    <procedure name="LoadMemory">
        <argument id="data"><new name="memdoubleword"/></argument>
        <argument id="size"><symbol name="WORD"/></argument>
        <argument id="physical"><var>pAddr</var></argument>
        <argument id="virtual"><var>vAddr</var></argument>
        <argument id="points_to"><symbol name="DATA"/></argument>
    </procedure>

    <procedure name="BytesSelect">
        <argument id="type"><symbol name="WORD"/></argument>
        <argument id="from"><new name="data"/></argument>
        <argument id="content"><var>memdoubleword</var></arument>
        <argument id="index"><var>dwByteOffset</var></argument>
    </procedure>

    <assert>
        <eq>
            <var>rt</var>
            <sign_extend size="64"><var>data</var></sign_extend>
        </eq>
    </assert>
</situation>
```

4.4 Генератор ограничений (ядро)

Генератор ограничений имеет модульную структуру (см. рис. 4.3) [4, 5]. В его составе есть модуль, ответственный за генерацию ограничений для общих механизмов описания архитектур микропроцессоров, такие как работа с регистрами, работа с последовательностью инструкций, работа с локальными переменными в тестовых ситуаций, и модуль, специфичный для тестируемой архитектуры (он содержит алгоритмы генерации ограничений для таких механизмов, как трансляция адресов, как кэш-память).

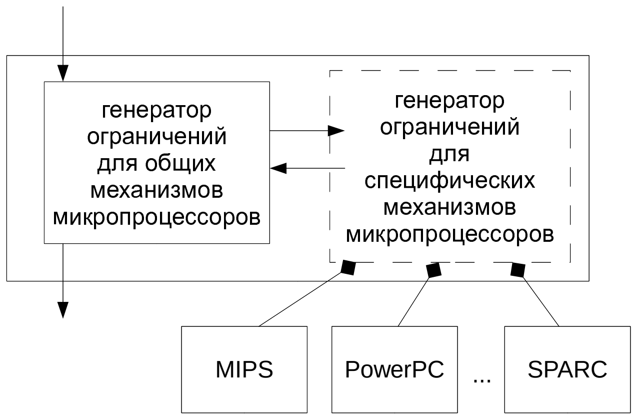


Рис. 4.3. Структура генератора ограничений

Генерация ограничений производится последовательно для каждой инструкции тестового шаблона. Однако разрешение ограничений может происходить быстрее при некотором порядке ограничений, это можно учесть при генерации ограничений.

Глава 5

Апробация

Целью является построение генератора тестовых программ по тестовым шаблонам для некоторого микропроцессора. Это может быть выполнено следующей последовательностью шагов:

1. построение схемы MMU микропроцессора (выделение кэширующих буферов и таблиц, определение подчиненных буферов) – для этого надо ознакомиться с документацией по MMU микропроцессора;

2. выбор процедур для языка описания тестовых ситуаций – для этого надо ознакомиться с документацией по системе команд микропроцессора;

3. написание генератора ограничений, анализирующего последовательности тестовых ситуаций в кэширующих буферах – для этого можно применить методы совместной и зеркальной генерации ограничений;

4. написание генератора ограничений для процедур, выбранных на шаге 2 – может потребоваться ознакомление с документацией по микропроцессору;

5. подготовка описаний тестовых ситуаций для инструкций микропроцессора – для этого надо ознакомиться с документацией по системе команд микропроцессора;

6. написание анализатора модели решателя и генератора тестовой программы;

7. объединение написанных модулей генератора в единое целое (с использованием готового компонента построения ограничений, независимого от конкретного микропроцессора);

8. запуск генератора ограничений с решателем ограничений.

5.1 Генерация ограничений для архитектуры MIPS

Утверждение 13. *Для архитектуры MIPS возможно применение применение методов генерации ограничений, описываемых в диссертации, для генерации тестовых программ по тестовым шаблонам; причем методов достаточно для полного описания поведения MMU микропроцессоров архитектуры MIPS.*

Рассмотрим исполнение инструкции обращения к памяти в микропроцессоре архитектуры MIPS [36]. MMU в микропроцессорах MIPS включает в себя (количественные характеристики приведены для микропроцессора MIPS R10000 – см. рис. 5.1):

- *кэш-память данных первого уровня (D-Cache-1)*: virtually indexed physically tagged, размер 32 килобайта, размер строки кэш-памяти 32 байта, наборно-ассоциативная кэш-память, ассоциативность равна 2, стратегия вытеснения LRU;

- *кэш-память второго уровня (Cache-2)*: размер от 512 килобайт до 16 мегабайт [1], стратегия вытеснения LRU;

- *кэширующий буфер TLB (D-TLB)*: полностью ассоциативный, размер 4 строки;

- *объединенный TLB (Joint-TLB)*: размер 48 строк, размер виртуального адреса 64 бита.

Таким образом, основная проблема при записи ограничений – большой размер содержимого кэш-памяти.

Инструкция может содержать тестовые ситуации в:

- кэш-памяти данных первого уровня, первого и второго уровней;
- кэш-буфере данных TLB (D-TLB).

Совместная генерация возможна на границе TLB–кэш-память, так как получаемый из TLB номер физического кадра (pfn) становится битовым полем тегсета физического адреса (см.рис. 5.1 – стрелками обозначены места применения совместной генерации, пунктиром обозначено подчинение кэширующего буфера таблице).

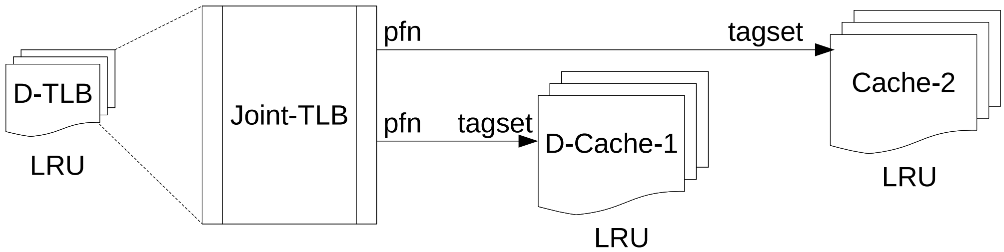


Рис. 5.1. Схема MMU микропроцессора MIPS

По шагам подготовки генератора тестовых программ:

1. структура MMU построена, для этого пришлось ознакомиться с документацией по архитектуре микропроцессора [36], это заняло 1 человеко-день;

2. на основе анализа системы команд архитектуры MIPS [35] были выделены следующие процедуры для описания тестовых ситуаций: AddressTransla LoadMemory, StoreMemory, BytesSelect, BytesExpand – на это ушел 1 человеко-день;

3. в генераторе ограничений был использован зеркальный метод генерации ограничений для кэшируемых неотображаемых обращений и совместно-зеркальный метод генерации ограничений для остальных обращений – на это ушло с учетом отладки 5 человеко-дней;

4. для процедуры AddressTranslation в виде ограничений была записана модель виртуальной памяти (виды обращений в разных областях виртуальной памяти – кэшируемое или некэшируемое, отображаемое или неотображаемое), для процедуры LoadMemory в виде ограничений были записаны взаимосвязи физических адресов и считанных из

основной памяти данных (см. п. 2.1.4 диссертации), для процедур BytesSelect и BytesExpand был выписан перебор значений младших бит физического адреса и границ части нужной длины двойного слова, являющего результатом чтения из памяти или записи в память – с учетом отладки на это ушло 3 человеко-дня;

5. по результатам знакомства с документацией по системе команд архитектуры MIPS [35] было выделено 8 инструкций (load/store byte/halfword/wo в каждой инструкции по 2 тестовые ситуации (AddressError(невыровненный виртуальный адрес) / полное выполнение инструкции), на подготовку описаний тестовых ситуаций ушло 1 человеко-день;

6. использовался решатель ограничений Z3 [18], который печатал модель в виде пар «(имя, значение)», среди имен выбирались начальные значения регистров и инициализирующие тегсеты кэшируемых неотображаемых обращений, по которым генерировались инициализирующие инструкции – с учетом отладки на это ушло 1 человеко-день;

7. были объединены имеющиеся компоненты чтения описаний тестовых ситуаций и построения ограничений для операторов `let` и `assert` и новые компоненты, описывающие последовательности тестовых ситуаций в кэш-памяти и TLB, описывающие процедуры описаний тестовых ситуаций и генерирующие искомую тестовую программу – на это ушло 1 человеко-день.

Итого на построение генератора тестовых программ для MIPS ушло около 2,5 человеко-недель. При этом при построении генератора тестовых программ для другого микропроцессора архитектуры MIPS (он может отличаться количественными параметрами кэширующих буферов, размерами виртуальных и физических адресов) повторно можно использовать результаты шагов 1, 2, 5, 6 и 7, остальные шаги выполняются аналогичным образом с заменой количественных параметров . Это позволяет достичь уровень переиспользования в 40% ($\frac{5*100\%}{13}$). Ручное создание генератора тестовой программы для микропроцессора MIPS RM7000 [2] заняло _______ человеко-дней. Данные о трудоемкости показывают, что при сходной полноте тестового набора представленные в диссертации методы позволяют сократить время построения генератора тестовых программ в _______ раз.

5.2 Генерация ограничений для архитектуры PowerPC

Утверждение 14. *Для архитектуры PowerPC возможно применение применение методов генерации ограничений, описыванных в диссертации, для генерации тестовых программ по тестовым шаблонам; причем методов достаточно для полного описания поведения MMU микропроцессоров архитектуры PowerPC.*

Рассмотрим исполнение инструкции обращения к памяти в микропроцессоре архитектуры PowerPC [1]. MMU в микропроцессорах PowerPC включает в себя (количественные характеристики приведены для микропроцессора PowerPC 970FX [9] – см. рис. 5.2):

- *кэш-память данных первого уровня (D-Cache-1)*: размер 32 килобайта, наборно-ассоциативная кэш-память, количество секций равно 2, размер строки кэш-памяти 128 байт, effective index real tag, стратегия вытеснения LRU;
- *кэш-память второго уровня (Cache-2)*: размер 512 килобайт, наборно-ассоциативная кэш-память, количество секций равно 8, стратегия вытеснения LRU, размер строки кэш-памяти 128 байт, real index real tag;
- *кэш-буфер TLB (D-TLB)*: наборно-ассоциативный буфер, количество секций 4, количество наборов в каждой секции 256; стратегия вытеснения LRU;
- *таблица страниц виртуальной памяти (PageTable)*: размер виртуального адреса 65 бит, размер физического адреса 42 бита;
- *сегментные регистры (SLB)*: полностью ассоциативный буфер, размер 64 строки;

- *буфер непосредственной трансляции адресов (D-ERAT)*: наборно-ассоциати буфер, количество секций равно 2, в каждой секции по 64 строки; стратегия вытеснения FIFO.

Таким образом, проблема возникнет при записи ограничений на кэш-память и на TLB.

Инструкция может содержать тестовые ситуации в:

- кэш-памяти данных первого уровня, первого и второго уровней;
- кэш-буфере данных TLB (D-TLB);
- кэш-буфере непосредственной трансляции адресов (D-ERAT).

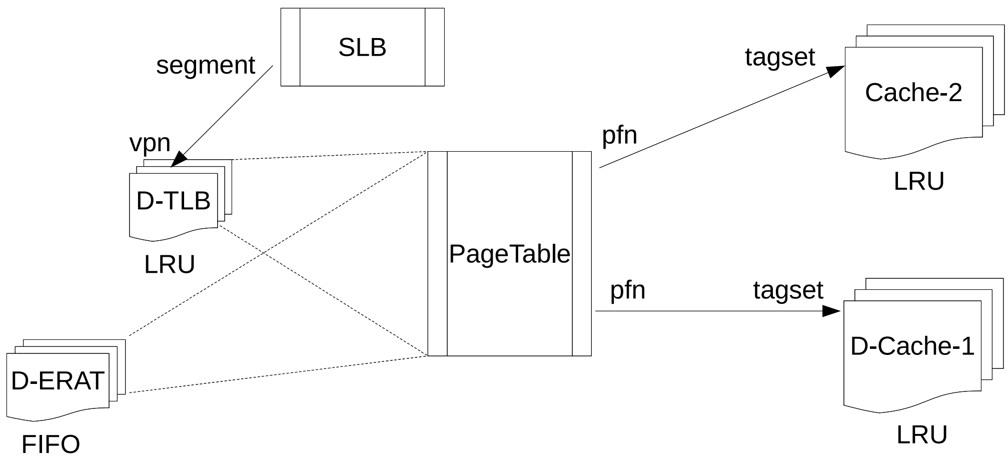


Рис. 5.2. Схема MMU микропроцессора PowerPC 970FX

Совместная генерация возможна (см. рис. 5.2):

- на границе D-ERAT–кэш-память, так как получаемый из D-ERAT номер физического кадра (pfn) становится битовым полем тегсета физического адреса;
- на границе SLB–D-TLB, так как значение сегментного регистра становится битовым полем номера страницы виртуальной памяти (vpn);
- на границе D-TLB–кэш-память, так как получаемый из D-TLB номер физического кадра (pfn) становится битовым полем тегсета физического адреса.

5.3 Генерация ограничений для архитектуры Alpha

Утверждение 15. *Для архитектуры Alpha возможно применение применение методов генерации ограничений, описываемых в диссертации, для генерации тестовых программ по тестовым шаблонам.*

Рассмотрим исполнение инструкции обращения к памяти в микропроцессоре архитектуры Alpha. MMU в микропроцессорах Alpha включает в себя (количественные характеристики приведены для микропроцессора Alpha 21264 [25] – см. рис. 5.3):

- *кэш-память данных первого уровня (D-Cache-1)*: virtually indexed physically tagged, размер 64 килобайта, наборно-ассоциативный, количество секций равно 2, стратегия вытеснения LRU, размер строки кэш-памяти 64 байта;

- *кэш-память второго уровня (Cache-2)*: physically indexed physically tagged, прямого отображения, размер от 1 до 16 мегабайт

- *таблица TLB (D-TLB)*: 128 строк, полностью ассоциативная, размер виртуального адреса 48/43 бит, размер физического адреса 44/41 бит, размер страницы виртуальной памяти от 8 килобайт до 4 мегабайт;

- *буфер вытесненных данных (VictimBuffer)*: полностью ассоциативный, количество строк равно 8, стратегия вытеснения FIFO.

Таким образом, основная проблема при записи ограничений – большой размер содержимого кэш-памяти.

Инструкция может содержать тестовые ситуации в:

- кэш-памяти данных первого уровня;

- кэш-памяти первого и второго уровней.

Совместная генерация возможна на границе TLB–кэш-память, так как получаемый из TLB номер физического кадра (pfn) становится битовым полем тегсета физического адреса (см.рис. 5.3).

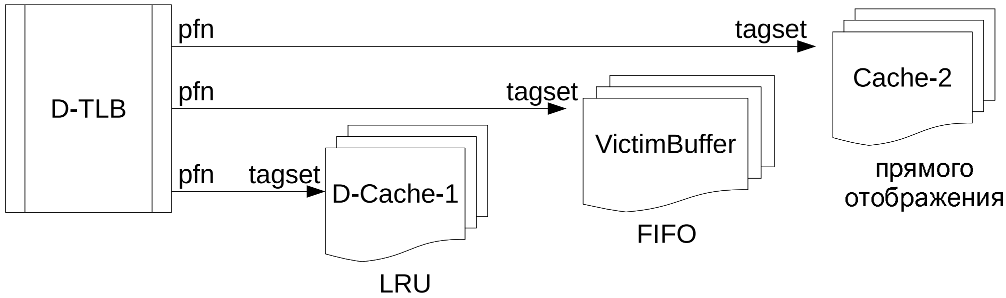


Рис. 5.3. Схема MMU микропроцессора Alpha

5.4 Генерация ограничений для архитектуры Pentium

Утверждение 16. *Для архитектуры Pentium возможно применение применение методов генерации ограничений, описываннных в диссертации, для генерации тестовых программ по тестовым шаблонам.*

Рассмотрим исполнение инструкции обращения к памяти в микропроцессоре архитектуры Pentium P6. MMU в микропроцессорах Pentium включает в себя (количественные характеристики приведены для микропроцессора Intel Pentium III [25] – см. рис. 5.4):

- *кэш-память данных первого уровня (D-Cache-1)*: размер 16 килобайт, наборно-ассоциативная, количество секций равно 2, стратегия вытеснения LRU, размер строки 32 байта;
- *кэш-память второго уровня (Cache-2)*: размер от 256 килобайт до 2 мегабайт, наборно-ассоциативная, количество секций равно 8, стратегия вытеснения LRU, размер строки 32 байта;
- *кэш-буфер TLB (D-TLB)*: наборно-ассоциативный, количество секций равно 4, в каждой секции 16 строк, стратегия вытеснения Pseudo-LRU;
- *таблица страниц виртуальной памяти (PageTable)*: размер страницы от 8 килобайт, длина логического адреса 48 бит, длина линейного адреса 32 бита, длина физического адреса 32 бита;
- *таблица дескрипторов сегментов (SDT)*: размер от 8 байт до 64 килобайт [17].

Таким образом, проблема возникнет при записи ограничений на кэш-память и на TLB.

Инструкция может содержать тестовые ситуации в:

- кэш-памяти данных первого уровня, первого и второго уровней;
- кэш-буфере данных TLB (D-TLB).

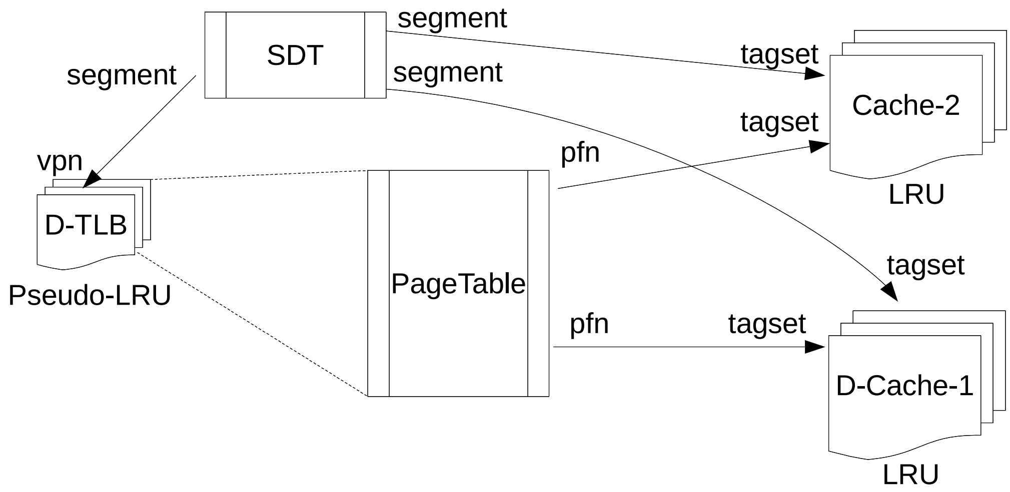


Рис. 5.4. Схема MMU микропроцессора Pentium

Совместная генерация возможна (см. рис. 5.4):

- на границе SDT–D-TLB, так как значение сегментного регистра становится битовым полем номера страницы виртуальной памяти (vpn);
- на границе D-TLB–кэш-память, так как получаемый из D-TLB номер физического кадра (pfn) становится битовым полем тегсета физического адреса;
- на границе SDT–кэш-память при неотображаемом обращении, так как получаемый из SDT сегментный регистр становится битовым полем тегсета физического адреса.

Заключение

Литература

[1] Виктор З. Шнитман. Современные высокопроизводительные компьютеры. *Центр информационных технологий*, 1996.

[2] Александр С. Камкин. Комбинаторная генерация тестовых программ для микропроцессоров на основе моделей. *Препринт Института Системного Программирования РАН*, 21, 2008.

[3] Евгений Корныхин. Генерация тестовых данных для тестирования арифметических операций центральных процессоров. *Труды Института Системного Программирования*, 15:107–117, 2008.

[4] Евгений Корныхин. Система генерации тестовых программ с использованием ограничений ТЕСЛА. *Сборник тезисов конференции Ломоносов*, pages XX–XX, 2009.

[5] Евгений Корныхин. Система генерации тестовых данных для системного функционального тестирования микропроцессоров ТЕСЛА. *Сборник тезисом конференции «Микроэлектроника и информатика»*, pages XX–XX, 2009.

[6] Евгений Корныхин. Генерация тестовых данных для системного функционального тестирования микропроцессоров с учетом кэширования и трансляции адресов. *Труды Института Системного Программирования*, XX:XX–XX, 2009.

[7] Евгений Корныхин. Генерация тестовых данных для системного функционального тестирования fifo-кэш-памяти микропроцессоров. *Вычислительные методы и программирование*, 10:XX–XX, 2009.

[8] Евгений Корныхин. Генерация тестовых данных для тестирования механизмов кэширования и трансляции адресов микропроцессоров. *Программирование*, (1):XX–XX, 2010.

[9] Powerpc g5 user's manual. Technical report, IBM, 2008.

[10] A.Adir, E.Almog, L.Fournier, E.Marcus, M.Rimon, M.Vinov, and A.Ziv. Genesys-pro: Innovations in test program generation for functional processor verification. *IEEE Design and Test of Computers*, 21(2):84–93, Mar/Apr 2004.

[11] Allon Adir, Eli Almog, Laurent Fournier, Eitan Marcus, Michal Rimon, Michael Vinov, and Avi Ziv. Industrial experience with test generation languages for processor verification. *IEEE Design and Test of Computers*, 21(2):84–93, Mar./Apr. 2004.

[12] Allon Adir, Roy Emek, Yoav Katz, and Anatoly Koyfman. Deeptrans - a model-based approach to functional verification of address translation mechanisms. In *MTV*, pages 3–6, 2003.

[13] Krzysztof R. Apt and Mark Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007.

[14] Peter J. Ashenden. *The Designer's Guide to VHDL*. Elsevier, 2008.

[15] Eyal Bin, Roy Emek, Gil Shurek, and Avi Ziv. Using a constraint satisfaction formulation and solution techniques for random test program generation. *IBM Systems Journal*, 41(3):386–402, 2002.

[16] F. Corno, E. Sanchez, M.S. Reorda, and G. Squillero. Automatic test program generation – a case study. *IEEE Design & Test, Special issue on Functional Verification and Testbench Generation*, 21(2):102–109, March-April 2001.

[17] Sivarama P. Dandamudi. *Fundamentals of computer organization and design*. Springer, 2003.

[18] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[19] Leonardo de Moura and Bruno Dutertre. Yices 1.0: An efficient smt solver. *The Satisfiability Modulo Theories Competition (SMT–COMP)*, 2006.

[20] F.Fallah and K.Takayama. A new functional test program generation methodology. *Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 76–81, 2001.

[21] Laurent Fournier, Yaron Arbetman, and Moshe Levinger. Functional verification methodology for microprocessors using the genesys test-program generator-application to the x86 microprocessors family. In *DATE*, pages 434–441, 1999.

[22] S. Hanono G. Hadjiyiannis and S. Devadas. Isdl: An instruction set description language for retargetability. *Proceedings of the 34th Design Auiomation Conference*, pages 299–302, June 1997.

[23] Daniel Grund and Jan Reineke. Estimating the performance of cache replacement policies. *6th ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2008), June 5-7, 2008, Anaheim, CA, USA*, pages 101–112, 2008.

[24] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau. Expression: A language for architecture exploration through compiler/simulator retargetability. In *In Proceedings of the European Conference on Design, Automation and Test*, pages 485–490, 1999.

[25] John L. Hennessy and David A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 3 edition, 2003.

[26] Intelligent Systems Laboratory, Swedish Institute of Computer Science. *SICStus Prolog User's manual, release 4*, 2009.

[27] Andrea C. Arpaci-Dusseau John L. Hennessy, David A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 4 edition, 2007.

[28] K.Kohno and N.Matsumoto. A new verification methodology for complex pipeline behavior. *Proceedings of the 38st Design Automation Conference (DAC'01)*, 2001.

[29] Evgeni Kornikhin. Test data generation for arithmetic subsystem of cpus mips64. *Proceedings of Spring Young Researchers Colloqium on Software Engineering*, 2:XX–XX, 2008.

[30] Evgeni Kornikhin. Smt-based test program generation for cache-memory testing. *Proceedings of East-West D T S*, pages XX–XX, 2009.

[31] Evgeni Kornikhin. Test data generation for lru cache-memory testing. *Proceedings of Spring Young Researchers Colloqium on Software Engineering*, pages XX–XX, 2009.

[32] M.Beardo, F.Bruschi, F.Ferrandi, and D.Sciuto. An approach to functional testing of vliw architectures. *Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT'00)*, pages 29–33, 2000.

[33] M.Behm, J.Ludden, Y.Lichtenstein, M.Rimon, and M.Vinov. Industrial experience with test generation languages for processor verification. *Proceedings of the 41st Design Automation Conference (DAC'04)*, 2004.

[34] Alexander Miczo. *Digital logic testing and simulation.* Wiley-Interscience; 2 edition, 2003.

[35] MIPS Technologies. *MIPS64^{TM} Architecture For Programmers Volume II: The MIPS64^{TM} Instruction Set*, 2001.

[36] MIPS Technologies. *MIPS64^{TM} Architecture For Programmers Volume III: The MIPS64^{TM} Privileged Resource Architecture*, 2001.

[37] Jean-Francois Puget. A c++ implementation of clp. *Proceedings of the 2nd Singapore International Conference on Intelligent Systems*, 1994.

[38] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[39] T.Li, D.Zhu, Y.Guo, G.Liu, and S.Li. Maatg: A functional test program generator for microprocessor verification. *Proceedings of the 2005 8th Euromicro conference on Digital System Design (DSD'05)*, 2005.

[40] W.Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundations of Mathematics*, 1954.