Генерация тестовых данных для тестирования кэш-памяти и буфера трансляции страниц микропроцессоров

Е.В. Корныхин
Московский государственный университет им. М.В. Ломоносова
факультет Вычислительной математики и кибернетики
Email: kornevgen@gmail.com

I. Введение

Микропроцессоры, как и любые сложные системы, подвержены появлению ошибок. Для обнаружения и исправления ошибок применяют различные методы, в том числе и тестирование. В данной статье речь пойдет о системном функциональном тестировании. Это значит, что микропроцессор рассматривается целиком, на вход его подаются ассемблерные программы (*тестовые программы*), они исполняются, процесс исполнения протоколируется и затем анализируется на соответствие ожидаемому (правильному) поведению. Однако из-за сложности современных микропроцессоров таких тестовых программ требуется много, что делает актуальной задачу их автоматического построения.

В статье [1] была предложена технологическая цепочка построения тестовых программ на основе модели микропроцессора. В рамках этой цепочки сначала тестовые программы систематически строятся в абстрактном виде (в виде *тестового шаблона* – без конкретных параметров инструкций. А затем уточняются до тестовых программ, что было описано лишь схематически, хотя и сказано, что для этого можно применять разрешение ограничений. Для сложных тестовых шаблонов этот этап становится отдельной проблемой, автоматизации построения тестовых данных по тестовым шаблонам посвящена данная статья.

Сложность построения тестовых данных увеличивается при использовании инструкций работы с памятью в тестовых шаблонах. Каждая такая инструкция задействует такие подсистемы, как кэш-память, буферы трансляции адресов и др. Предполагалось выделять из тестового шаблона такие инструкции и обрабатывать их отдельно. По результатам генерации тестовых данных для них требовалось дополнительно построить инструкции инициализации состояния микропроцессора (заполнение кэш-памяти и других подсистем нужными данными в нужном порядке) – т.н. *инициализирующую программу*.

В [2] описан метод решения задачи генерации тестовых данных, в которой качестве ответа выдавалось состояние микропроцессора. Генерировать инструкции по его достижению не входило в его компетенцию. Несложно придумать инициализирующую программу очень большого размера, а генерация компактной инициализирующей программы – непростая задача. Кроме того, инициализирующая программа может дать эффект «четной ошибки», т.е. возникновение ошибки в микропроцессоре при исполнении инициализирующей программы не даст возможности обнаружить ошибки собственно на тестовом шаблоне. Поэтому поставлена задача генерации таких тестовых данных, которые максимально учитывали бы имеющееся начальное состояние микропроцессора (состояние кэш-памяти, TLB).

В данной статье описывается алгоритм решения этой задачи при запрете изменения кэша и TLB перед исполнением тестового шаблона. Единственное, что можно менять, это значения регистров (и ячейки оперативной памяти напрямую, не затрагивая кэш-памяти и других подсистем). Алгоритм может выдать ответ «нет», что означает отсутствие инициализации регистров для удовлетворения шаблона из данного начального состояния кэш-памяти и TLB. При этом алгоритм обеспечивает полное решение задачи, т.е. если алгоритм выдал ответ «нет», то искомые значения регистров действительно не существуют.

II. Обзор похожих работ

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 [**?**];
- *случайная генерация тестовых программ на основе тестовых шаблонов* предполагает разделение процесса генерации тестовой программы на два этапа (см. рис. **??**): на первом подготавливаются тестовые шаблоны – абстрактные представления тестовых программ (в тестовых шаблонах для параметров инструкций вместо значений указываются ограничения на значения) – а на втором этапе по тестовым шаблонам генерируются тестовые программы. Второй этап включает в себя *генерацию тестовых данных*, т.е. генерацию параметров инструкций (параметров-констант) и начальных значений регистров, ячеек кэш-памяти, строк TLB и т.д. Иногда выбор регистров для инструкций задается в тестовом шаблоне, а иногда выбор регистров может сделать генератор тестовых данных.

Обратимся к задаче генерации тестовых данных. Среди известных работ можно выделить следующие методы ее решения:

- комбинаторные техники;
- решение задачи ATPG;
- разрешение ограничений.

Комбинаторные техники применимы в случае простых тестовых шаблонов. Такие тестовые шаблоны включают лишь простые ограничения, а именно указание области значений переменной. Причем все значения этой области в тестовой программе равноправны.

Предлагается генеровать тестовые данные и с использованием *техник решения задачи ATPG* (Automatic Test Pattern Generation). ATPG – задача поиска значений входных сигналов («векторов») схемы с целью поиска ее некорректного поведения. ATPG чаще применяется для модульного тестирования, если известна RTL-модель микропроцессора.

Наиболее впечатляющих результатов достигают инструменты, использующие для генерации тестовых данных *разрешение ограничений*. Ограничение с логической точки зрения то же, что и предикат, а задача разрешения ограничений – то же, что и задача выполнимости системы предикатов, но для решения этой задачи применяются специальные алгоритмы. Genesys-Pro [4] позиционируется компанией IBM как разработка, впитавшая лучшее из разработок последних 20 лет. Тестовые шаблоны позволяют задавать тестовые программы переменной длины. Для любой инструкции в тестовом шаблоне может быть указана эвристика для выбора значений параметров. Среди возможных эвристик есть и эвристики на события в кэш-памяти и при трансляции адресов. Однако в известных работах не раскрывается содержание таких эвристик, что не дает возможности понять эффективность генерации программ, нацеленных на тестирование памяти. Система команд микропроцессора должна быть описана в виде ограничений (constraint net) на операнды, код операции, что не является естественным описанием поведения инструкции, особенно если в рамках нее выполняется несколько последовательных вычислений на основе параметров инструкции. Для генерации параметров очередной инструкции Genesys-Pro использует уже построенную тестовую программу и состояние микропроцессора, которое известно полностью. Этот подход обеспечил масштабируемость на большие тестовые шаблоны, но и привел к необходимости использования механизма возврата (backtracking), если выбрать параметры для очередной инструкции.

По сравнению с Genesys-Pro в данной статье делается попытка транслировать тестовый шаблон в ограничения целиком. При этом отпадает необходимость в механизме возврата. Особенностью тестовых шаблонов, получаемых в рамках [1], является фиксация для каждой инструкции регистров-параметров. Для таких шаблонов Genesys-Pro будет работать крайне неэффективно, поскольку теряется возможность с помощью выбора параметров «подогнать» исполнение очередной инструкции под заданные в тестовом шаблоне для нее события. На тестовых шаблонах из [1] Genesys-Pro будет работать следующим образом: выберет некоторое начальное состояние микропроцессора, начнет исполнять тестовый шаблон (поскольку начальное состояние ему известно), но как только дойдет до инструкции, которая будет исполнена не так, как требуется в шаблоне, Genesys-Pro сделает возврат в самое начало, а именно ему придется выбрать другое начальное состояние микропроцессора и весь процесс запустить заново. Такой процесс генерации тестовых данных слишком неэффективен.

Для задания схемы трансляции адресов в Genesys-Pro предлагается использовать подход DeepTrans [3]. Однако по имеющимся работам невозможно сделать вывод о том, как такая схема трансляции адресов

отображается в ограничения [1].

III. Предварительные сведения

A. Трансляция адресов

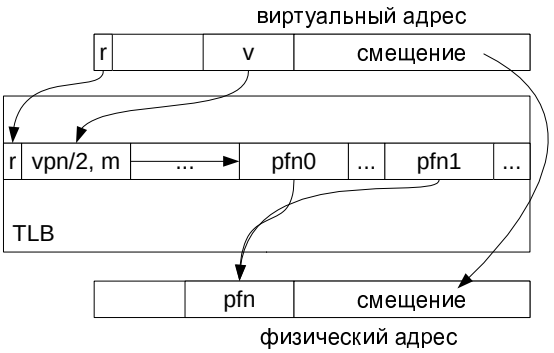

Рис. 1. Схема трансляции виртуального адреса в физический

Инструкции работы с памятью оперируют *виртуальными адресами*, однако для исполнения этих инструкций нужно построить *физический адрес*. При этом используется такая структура как *TLB* (Translation Lookaside Buffer). TLB состоит из строк, поля которых можно поделить на три части. Первая часть (теговая) служит для определения подходящей для данного виртуального адреса строки TLB, вторая и третья служат для преобразования виртуального адреса в физический. При этом преобразовании старшая битовая часть виртуального адреса заменяется на одно из полей строки TLB, а оставшаяся битовая часть виртуального адреса переносится в физический без изменений (смещение). Для замены в строке TLB есть два поля (pfn), выбор между ними осуществляется на основе значения определенного бита виртуального адреса (см. рис. 1).

Для ускорения поиска подходящей строки к структуре TLB присоединена небольшая ассоциативная кэш-память, т.н. *micro-TLB*. Если поиск строки в micro-TLB оказался удачным, происходит трансляция адреса. Если поиск строки в micro-TLB оказался неудачным, происходит поиск в TLB. Если этот поиск оказался удачным, найденная строка переносится в micro-TLB на место некоторой строки согласно *стратегии вытеснения* (например, при стратегии вытеснения *LRU*, Least Recently Used, на место той, которая не использовалась дольше остальных).

B. Механизм работы кэш-памяти

В полученном по результатам трансляции физическом адресе выделяются поля *тег*, *сет*а и поле смещения.

[1]Авторы статьи используют при описании способа трансляции адреса элементы массива Memory с неизвестными индексами. Известно, что попытки построения ограничений, описывающих работу с элементами массива при неизвестных индексах, приводит к очень сложным ограничениям, разрешимость которых за приемлимое время можно поставить под сомнение.

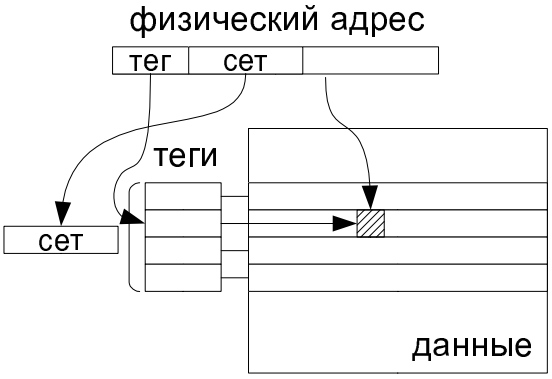

Рис. 2. Схема работы ассоциативной кэш-памяти

Кэш-память условно можно поделить на блоки, каждому блоку соответствует свое значение поля сет. Размер блока есть *ассоциативность* кэш-памяти. Каждой строке блока соответствует свое значение поля тег. Если по полученному из физического адреса сету в кэш-памяти есть строка с тегом, равным полю тег физического адреса, то обращение в оперативную память не делается (такая ситуация называется *кэш-попаданием*), из строки по полю смещения физического адреса выбираются нужные данные. Если же нужного тега в сете нет, то происходит обращение к другим уровням кэш-памяти (в них может быть другое выделение полей из физического адреса) и даже возможно обращение в оперативную память, если нужных данных нет в кэш-памяти вообще. В этом случае нужная строка (с заменой тега) подгружается в кэш-память на место той строки, которая определяется согласно *стратегии вытеснения* (например, при стратегии вытеснения LRU может быть вытеснена строка, которая не использовалась дольше остальных).

C. Генерация тестовых данных без учета начального состояния микропроцессора

В статье [2] описан алгоритм построения начального состояния кэш-памяти и регистров для удовлетворения последовательности тестовых ситуаций (кэш-попаданий и кэш-промахов). Идея алгоритма состояла в составлении и разрешении системы уравнений на конечные множества (переменными являлись теги начального состояния кэш-памяти и значения регистров). Уравнения, составляющие систему, можно поделить на 3 группы (конкретный вид уравнений зависит от стратегии вытеснения; приведенные здесь уравнения относятся к стратегии вытеснения LRU):

1) для тегов, при обращении к которым происходит кэш-попадание, и для вытесняемых тегов составляется «уравнение принадлежности»: $t \in (A \cup B) \setminus C$, где t – переменная-тег, A – множество тегов начального состояния кэш-памяти, B – множество «вытесняющих» тегов, относящихся к инструкциям до инструкции,

где встретился t, C – множество «вытесняемых» тегов, относящихся к инструкциям до инструкции, где встретился t;

2) для тегов, при обращении к которым происходит кэш-промах («вытесняющих» тегов) составляется «уравнение непринадлежности»: $t \notin (A \cup B) \setminus C$, символы t, A, B, C имеют тот же смысл, что и для тегов кэш-попадания; кроме того, «вытесняемый» и «вытесняющий» теги имеют равные *регионы* (регион тега t – это все теги тестового шаблона, попадающие в область того же сета, что и t; регион тега t будем обозначать функциональным символом $R(t)$);
3) для «вытесняемых» тегов перебором тегов, к которым производятся обращения перед этим, составляется «уравнение вытеснения»: $\bigvee_y t = y \wedge R(t) \cap ((A \cup B) \setminus (C \cup D)) = \{t\}$, где символы t, A, B, C имеют тот же смысл, D – множество тегов, к которым происходят обращения между вытеснением тега t и обращением к тегу y.

Этот же метод применим для поиска тестовых данных для тестовых ситуаций в micro-TLB, поскольку по своей архитектуре это тоже кэш-память. В качестве тега может использоваться простой (скалярный) идентификатор строки TLB, например, индекс строки TLB.

Однако упомянутый метод построения тестовых данных для кэш-памяти обладает рядом недостатков при применении его для заданного начального состояния микропроцессора:

1) после упрощения системы переменным надо выбрать значения; обычно это делается указанием области значений переменной, внутри которой специальный алгоритм, отсекая заведомо неподходящие, выбирает значения для переменных, при отсутствии возможности сделать очередное отсечение генератор значений производит перебор внутри области значений; размеры кэш-памяти современных микропроцессоров исчисляются мегабайтами (особенно если речь идет о кэш-памяти второго и последующих уровней), для них перебор становится практически невозможным, учитывая количество переменных и количество значений;
2) даже если искомые значения нашлись, приходится строить инициализирующую программу (заполнить кэш-память на всех уровнях и TLB нужными значениями и в нужном порядке согласно стратегии вытеснения, что уже непросто); опять же учитывая размеры кэш-памяти современных микропроцессоров, размер такой инициализирующей программы становится очень большим, что очень сильно понижает эффективность тестирования (ведь ошибка с очень большой вероятностью может произойти уже на инициализирующей программе – она не даст проверить поведение микропроцессоры собственно на тестовом шаблоне).

IV. Совместная генерация тестовых данных с учетом начального состояния микропроцессора

Цель – построение единой системы ограничений на тестовые ситуации в кэш-памяти и в TLB. В качестве переменных, на которые будут формулироваться ограничения, выбраны *тегсеты*. Тегсет – это битовая конкатенация тега и сета. Поэтому тегсет является старшей битовой частью физического адреса, а также поле PFN строки TLB является старшей битовой частью тегсета (см. рис. 3).

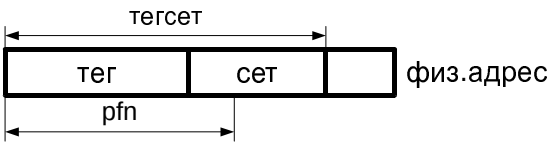

Рис. 3. Соотношение физического адреса, тегсета и поля pfn

Для каждой тестовой ситуации на кэш-память теперь будут генерироваться ограничения на тегсеты. Основные составные части таких ограничений:

- «ограничение кэш-попадания» тегсета ts выглядит как $ts \in (A \cup B) \setminus C$, где A – множество тегсетов начального состояния кэш-памяти, B – множество «вытесняющих» тегсетов, расположенных до данной ситуации кэш-попадания, C – множество «вытесняемых» тегсетов, расположенных до данной ситуации кэш-попадания; множество A задано; если множество $(A \cup B) \setminus C$ пусто, то искомые тестовые данные не существуют;
- «ограничение кэш-промаха» тегсета ts выглядит как $ts \notin (A \cup B) \setminus C$, где A – множество тегсетов начального состояния кэш-памяти, B – множество «вытесняющих» тегсетов, расположенных до данной ситуации кэш-промаха, C – множество «вытесняемых» тегсетов, расположенных до данной ситуации кэш-промаха; ts будет являться «вытесняющим» тегсетом, тут же определяется и «вытесняемый» тегсет ts': $ts' \in (A \cup B) \setminus C$; кроме того, «вытесняющий» и «вытесняемый» теги попадают в один регион: $R(ts) = R(ts')$ и верно «уравнение вытеснения»; множество A задано.

Тестовые ситуации на micro-TLB также подвергаются изменениям. Во-первых, в качестве «теговой» части теперь будет рассматриваться не индекс строки TLB, а поле PFN строки TLB. Отношение PFN и строк TLB в общем случае не является взаимнооднозначным (однако оно является таковым, если PFN во всех строках TLB имеют разные значения), что дает дополнительную свободу при разрешении ограничений на виртуальные адреса (по значению поля PFN будет делаться предположение о содержимом строки TLB, которая

соответствует очередному виртуальному адресу). По тестовой ситуации на micro-TLB формулируются ограничения того же вида, что и представленные выше для кэш-памяти. Далее будет использована операция «$[M]$», которая определяется следующим образом: $[M] = \{ts|ts = pfn||of, pfn \in M\}$. По сути эта операция сопоставляет множеству полей pfn множество тегсетов, соответствующих ему. Еще одну операцию – «$[PFN \setminus M]$» определим следующим образом: $[PFN \setminus M] = [\{pfn|pfn \in PFN, \exists TLB, pfn, M\}]$. Она позволяет описать дополнение множества с учетом возможных дубликатов полей PFN в TLB. С использованием этих операций ограничения, описывающие тестовые ситуации в micro-TLB, буду выглядеть следующим образом:

- «ограничение попадания» тегсета ts выглядит как $ts \in [(M \cup B) \setminus C]$, где M – множество pfn начального состояния micro-TLB, B – множество «вытесняющих» pfn (старших битовых частей pfn), расположенных до данной ситуации попадания, C – множество «вытесняемых» pfn, расположенных до данной ситуации попадания; множество M задано; если множество $[(M \cup B) \setminus C]$ пусто, то для данного начального состояния micro-TLB не существует искомых тестовых данных;
- «ограничение промаха» тегсета ts выглядит как $ts \in [PFN \setminus ((M \cup B) \setminus C)]$, где M – множество pfn начального состояния micro-TLB, B – множество «вытесняющих» pfn (старших битовых частей pfn), расположенных до данной ситуации промаха, C – множество «вытесняемых» pfn, расположенных до данной ситуации промаха; ts будет являться «вытесняющим» тегсетом, тут же определяется и «вытесняемый» pfn p': $p' \in (M \cup B) \setminus C$ и верно «уравнение вытеснения»; множество M задано.

Запись ограничения для «промаха» в виде принадлежности стала возможна, потому что TLB не меняется в процессе работы тестового шаблона. Таким образом, для переменной-тегсета каждой инструкции будет определено как минимум одно ограничение принадлежности множеству и другие ограничения принадлежности и непринадлежности. Из этих ограничений составляется конъюнкция, представляющая собой ограничение принадлежности (пересечению или разности множеств).

Произведем оценку размера пространства перебора значений тегсетов. В каждое множество, которому принадлежит значение тегсета, будет входить один из следующих множеств, составленных из начальных состояний кэш-памяти (разных уровней) и micro-TLB: $[M] \cap L, [M] \setminus L, [PFN \setminus M] \cap L, [PFN \setminus M] \setminus L$. Таким образом, размер области значений тегсетов ограничен размером PFN, что обычно намного меньше количества тегсетов, хранящихся в кэш-памяти. Кроме того, за счет анализа множества значений других тегсетов, входящих в ограничение принадлежности, можно дополнительно уменьшить размер области значений. Если область значений стала равна пустому множеству, то искомые тестовые данные не существуют.

Полученные в результате ограничения после подсчета пересечений и вычитаний множеств начальных тегсетов можно решать с использованием SAT-инструментов или с помощью constraint solver'ов.

V. Заключение

В статье представлен алгоритм построения начальных значений регистров для исполнения тестового шаблона с заданным начальным состоянием кэш-памяти и TLB. Показано, что этот алгоритм позволяет достаточно быстро по сравнению с размерами кэш-памяти (обычно это мегабайты!) находить искомые значения регистров.

Список литературы

[1] Камкин А.С. Генерация тестовых программ для микропроцессоров // Труды ИСП РАН, 2008. Том 14, выпуск 2. с.23-64.

[2] Корныхин Е.В. Генерация тестовых данных для системного функционального тестирования FIFO-кэш-памяти микропроцессоров // Вычислительные методы и программирование, 2009, том 10, с.107-116.

[3] Adir A., Emek R., Katz Y., Koyfman A. DeepTrans – a model-based approach to functional verification of address translation mechanisms // Proceedings of the 4th International Workshop on Microprocessor Test and Verification: Common Challenges and Solutions, 2003. 3-6pp.

[4] Adir A., Almog E., Fournier L., Marcus E., Rimon M., Vinov M., Ziv A. Genesys-Pro: innovations in test program generation for functional processor verification // Design & Test of Computers, Volume 21, Issue 2, Mar-Apr 2004, Page(s): 84 - 93