Генерация тестовых данных для системного функционального тестирования кэш-памяти микропроцессоров

Е.В. Корныхин
Московский государственный университет им. М.В. Ломоносова
Email: kornevgen@gmail.com

***Аннотация*—В статье рассматривается задача генерации начального состояния кэш-памяти для тестовых шаблонов, использующихся в системном функциональном тестировании процессоров. Хотя задача решена для кэш-памяти общего вида, в статье детально рассматривается генерация тестовых данных для базисных способов организации кэш-памяти: полностью ассоциативный кэш и кэш прямого отображения. Генерация начального состояния кэш-памяти осуществляется путем решения ограничений, составленных для тестового шаблона.**

I. Введение

Системное функциональное тестирование микропроцессоров является хорошо зарекомендовавшей себя техникой тестирования микропроцессоров. Оно выполняется с использованием большого числа программ на языке ассемблера (тестовых программ). Такие программы загружаются в память машины, запускаются, процесс их исполнения протоколируется и затем анализируется. В результате выносится вердикт, в каких случаях микропроцессор себя ведет некорректно (некорректным считается поведение, отличное от того, что заявлено в спецификации). Однако для проведения системного тестирования для современных микропроцессоров требуется множество тестовых программ, что делает актуальной задачу автоматической генерации тестовых программ.

В статье **??** 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. Тестовые шаблоны описывают последовательность инструкций, параметры инструкций с указанием происходящих событий (например, переполнение, промахи или попадания в кэш-памяти). В качестве параметров инструкции могут быть указаны регистры и константы. Последовательность инструкций чаще всего задана явно. Необходимость в тестовом шаблоне обычно возникает тогда, когда тестирование проводится нацеленным образом и эта цель выражена последовательностью инструкций, каждая из которых должна быть исполнена заданным образом.

Чтобы получить тестовую программу по заданному тестовому шаблону, достаточно найти начальные значения регистров и той части кэш-памяти и других подсистем, с которыми работают инструкции шаблона. Добавляемые к шаблону данные называются тестовыми данными, а задача их построения – задачей генерации тестовых данных. По тестовым данным строится набор инструкций инициализации микропроцессора (загрузка значений в регистры, кэш и т.д.), который добавляется в начало тестового шаблона. Полученную таким образом тестовую программу можно исполнить и проверить, совпадает ли поведение каждой инструкции с тем, что было заявлено в тестовом шаблоне.

Среди известных работ по генерации тестовых данных для микропроцессоров можно выделить следующие методы ее решения:

1) комбинаторные техники;
2) решение задачи ATPG;
3) разрешение ограничений.

Однако комбинаторные техники применимы только в случае простых шаблонов, где для тестовых данных явно указаны диапазоны значений, и внутри это диапазона все значения равноправны. Генерация тестовых данных через решение задачи ATPG (Automatic Test Pattern Generation) скорее подходит не для функционального, а для структурного тестирования микропроцессоров. Использование разрешения ограничений представляется наиболее перспективной техникой генерации тестовых данных.

Почти 20-летние разработки компании IBM в этой области привели к созданию инструмента Genesys-Pro для генерации тестовых программ по тестовых шаблонам с использованием разрешения ограничений. Однако для шаблонов, генерируемых методом из **??**, в ряде ситуаций, где есть зависимости, Genesys-Pro будет работать неэффективно. А авторы других инструментов зачастую останавливаются на тестировании лишь регистровой части микропроцессоров. Все эти факторы делают актуальной задачу генерации тестовых данных иным методом, но с использованием разрешения ограничений.

II. Описание тестовых шаблонов

Тестовый шаблон задает свойства будущей тестовой программы. Как и тестовая программа, в тестовом шаблоне указывается последовательность инструкций тестовой программы. Каждый элемент такой последовательности содержит указание имени инструкции, параметров и тестовой ситуации — связи значений операндов и состояния микропроцессора (ячейки кэш-памяти, регистры, другие подсистемы) перед началом выполнения инструкции. В качестве параметров инструкции в тестовом шаблоне могут выступать переменные величины — регистры, непосредственные значения, адреса. Отдельно могут быть зафиксированы в тестовой программе зависимости (предикаты) между переменными величинами.

Пример описания тестового шаблона для модельной системы команд:

REGISTER reg1 : 32;
REGISTER reg2 : 32;
ADD reg1, reg2, reg2
LOAD reg1, reg2 @ l1Miss, l2Hit
XOR reg2, reg1, reg2

В этом тестовом шаблоне три команды - ADD, LOAD и XOR. Шаблон начинается с объявления переменных с указанием их битовых длин. Тестовая ситуация указывается после знака «@»: тестовая ситуация второй команды — «l1Miss, l2Hit»: l1Miss – указание на то, что при загрузке из памяти должен произойти промах в кэш-памяти первого уровня, l2Hit – указание на то, что пр загрузке из памяти задействована кэш-память второго уровня и во время выполнения инструкции должно произойти кэш-попадание.

Сейчас нас будут интересовать лишь инструкции работы с памятью, коих две:

- «LOAD reg, address» осуществляет загрузку значения в переменную reg из памяти по адресу в переменной address;
- «STORE reg, address» осуществляет сохранение значения из переменной reg в памяти по адресу в переменной address.

Задача поиска тестовых значений для шаблонов с такими инструкциями состоит в поиске не только начальных значений регистров, но и начального состояния кэш-памяти (т.е. значения ячеек кэш-памяти перед началом исполнения тестового шаблона). Эта задача решена для кэш-памяти произвольной организации. Но здесь будут разобраны лишь 2 более простых случая организации кэш-памяти: полностью ассоциативный кэш и кэш прямого отображения, а общий случай будет дан лишь иллюстративно. Произвольный кэш включает в себя характеристики как полностью ассоциативного, так и кэша прямого отображения. Поэтому в рамках общего случая рассматриваемые способы организации кэш-памяти могут считаться базисными. Также для простоты далее будет рассматриваться одноуровневый кэш.

III. Генерация тестовых данных для полностью ассоциативной кэш-памяти

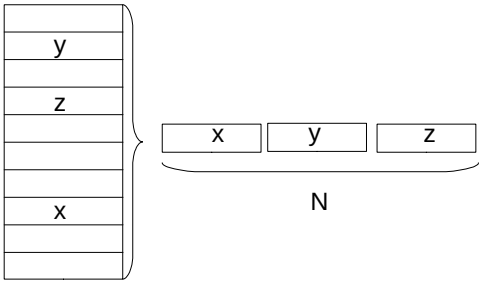

Рис. 1. Полностью N-ассоциативный кэш

Полностью ассоциативный кэш состоит из заданного количества ячеек (их количество называется *ассоциативностью кэша*). В каждой ячейке кэша могут храниться данные любых ячеек памяти. Все ячейки кэша соответствуют разным ячейкам памяти. Все ячейки кэш-памяти равноправны. При обращении к памяти первым делом данные ищутся в кэше во всех ячейках кэш-памяти *параллельно*. Ситуация *кэш-попадания* означает, что одна из ячеек кэш-памяти соответствует требуемому адресу. Ситуация *кэш-промаха* означает, что ни одна из ячеек кэш-памяти не соответствует требуемому адресу. В случае кэш-промаха одна из ячеек кэш-памяти *вытесняется* и заменяется на данные по требуемому адресу. Согласно стратегии вытеснения LRU (Least Recently Used) будут вытеснены данные, к которым дольше всего не было обращений. Везде далее под фразой «вытесняемый адрес x» будут пониматься вытесняемые данные по адресу x.

Предлагаемый алгоритм построения ограничений для тестового шаблона в случае полностью ассоциативной кэш-памяти основывается на следующих свойствах вытесняемых адресов:

1) вытесняемый адрес был добавлен ранее одной из инструкций тестового шаблона (или находился среди адресов начального состояния кэш-памяти);
2) между вытеснением адреса и последним кэш-попаданием к нему (кэш-промах с замещением на этот адрес тоже считается кэш-попаданием) происходят кэш-попадания ко всем остальным адресам кэша.

Алгоритм строит ограничения на следующие переменные:

1) $\alpha_1, \alpha_2, \alpha_3, \ldots$ – адреса начального состояния кэш-памяти (их количество равно ассоциативности кэш-памяти);
2) адреса, при обращении к которым происходят кэш-попадания (их количество равно количеству инструкций, при обращении к которым происходят кэш-попадания);
3) адреса, при обращении к которым происходят кэш-промахи (их количество равно количеству ин-

струкций, при обращении к которым происходят кэш-промахи);

4) вытесняемые адреса (их количество равно количеству инструкций, при обращении к которым происходят кэш-промахи);
5) $L_0, L_1, ...$ – переменные-состояния кэш-памяти (их количество – на единицу больше количества инструкций, при обращении к которым происходят кэш-промахи).

Итак, каждая инструкция, при обращении к которой происходит кэш-попадание, порождает 1 новую переменную; каждая инструкция, при обращении к которой происходит кэш-промах порождает 1 переменную-состояние кэш-памяти, 1 переменную-вытесняющий адрес и 1 переменную-вытесняемый адрес. Алгоритм формирует ограничения для каждой очередной инструкции следующим образом (N – ассоциативность кэш-памяти):

1) «начальные ограничения» генерируются для любого шаблона один раз: $L_0 = \{\alpha_1, \alpha_2, ..., \alpha_N\}$, $|L_0| = N$ (или, по-другому, все числа $\alpha_1, \alpha_2, ..., \alpha_N$ разные);
2) «ограничения кэш-попадания» генерируются для каждой инструкции, при обращении к которой происходит кэш-попадание: $x \in L$, где x – адрес в инструкции, L – текущая переменная-состояние кэш-памяти;
3) «ограничения кэш-промаха» генерируются для каждой инструкции, при обращении к которой происходит кэш-промах (x – вытесняющий адрес, y – вытесняемый адрес, L – текущая переменная-состояние кэш-памяти): $y \in L, x \notin L, L' = L \cup \{x\} \setminus \{y\}, lru(y)$, L' становится текущей переменной-состояния кэш-памяти для следующей инструкции.

Ограничение $lru(y)$ описывает свойство, что y является вытесненным адресом.

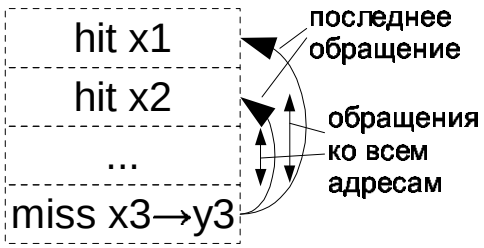

Рис. 2. LRU

Ограничение $lru(y)$ представляется дизъюнкцией ограничений, соответствующих всевозможным местам последнего обращения к y. Каждый дизъюнкт к кэш-попаданию адреса x фиксирует конъюнкцию двух свойств:

1) $x = y$
2) $L \setminus \{y\} = \{x_1, x_2, ..., x_n\}$, где $x_1, x_2, ..., x_n$ – все адреса, к которым происходят обращения между обращением к x и вытеснением y.

В качестве «кэш-попаданий адреса x» рассматриваются также «кэш-промахи адреса x», т.к. в результате кэш-промаха данные по этому адресу подгружаются в кэш, и адреса данных, лежащих перед исполнением тестового шаблона в кэше (в порядке «последнего к ним обращения»).

Рассмотрим пример тестового шаблона и построение для него тестовых данных. Будем рассматривать его для 3-х ассоциативного кэша.

LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Первым делом определимся с именами переменных так, чтобы они не меняли свое значение (введем «версии» переменных). Учтем, что LOAD дает новую версию переменной, идущей первым аргументом (т.к. в него загружается значение из памяти). Для описания вытесняемого адреса введем фиктивную переменную z'_0 (в ответ ее значение не попадет):

LOAD x_1, y_0 @ Hit
STORE u_0, z_0 @ Miss $\rightarrow z'_0$
LOAD z_1, y_0 @ Hit

Введем переменные начального состояния кэш-памяти: $\{\alpha, \beta, \gamma\}$ (их количество равно ассоциативности кэш-памяти).

Наша задача состоит в поиске значений $x_0, y_0, z_0, u_0, \alpha, \beta, \gamma$, на которых инструкции тестового шаблона будут исполнены в заданных тестовых ситуациях. Логично предположить, что решение не будет единственным. Нам в данном случае достаточно найти какое-либо одно решение.

Первые ограничения связаны с описанием попаданий и промахов, как принадлежности текущей переменной-состоянию кэш-памяти:

$y_0 \in \{\alpha, \beta, \gamma\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\} \cup \{z_0\}$,
α, β, γ – все разные

Осталось записать свойство $lru(z'_0)$. Кандидаты на последнее обращение к этому адресу – $y_0, \gamma, \beta, \alpha$. Первые два кандидата не подходят, поскольку для них ограничение $L \setminus \{z'_0\} = X$ не выполняется из-за несовпадения мощностей сравниваемых множеств. Оставшиеся кандидаты дают следующую дизъюнкцию:

$z'_0 = \beta \wedge \{\alpha, \beta, \gamma\} \setminus \{z'_0\} = \{\gamma, y_0\}$
$\vee$
$z'_0 = \alpha \wedge \{\alpha, \beta, \gamma\} \setminus \{z'_0\} = \{\beta, \gamma, y_0\}$

Упростим ее:

$z'_0 = \beta \wedge \{\alpha, \gamma\} = \{\gamma, y_0\}$
$\vee$
$z'_0 = \alpha \wedge \{\beta, \gamma\} = \{\beta, \gamma, y_0\}$

Что упрощается далее:

$z'_0 = \beta \wedge y_0 = \alpha$
$\vee$
$z'_0 = \alpha \wedge y_0 \in \{\beta, \gamma\}$

Рассмотрим первую альтернативу дизъюнкции вместе с остальными ограничениями (фиктивная переменную z'_0 в ответ не попадает):

$y_0 = \alpha$

$z_0 \notin \{\alpha, \beta, \gamma\}$,

α, β, γ – все разные

Заметьте, что x_0 и u_0 нигде не принимают участие – их значение может быть произвольным.

Пусть адреса занимают 8 бит. Тогда все переменные принимают значения из области от 0 до 255. Разрешая полученные ограничения, можно получить такие значения для переменных (замечу снова, что набор значений не является единственным):

$\alpha = y_0 = x_0 = u_0 = 0$

$\beta = 1$

$\gamma = 2$

$z_0 = 3$

Проверим исполнение тестового шаблона с такими значениями переменных:

начальное значения кэша: [2, 1, 0]

LOAD x, 0 - Hit, т.к. $0 \in \{2, 1, 0\}$; согласно lru новое состояние кэш-памяти равно [0, 2, 1]

STORE 0, 3 - Miss, т.к. $3 \notin \{0, 2, 1\}$; согласно lru 3 попадает в кэш-память, 1 из кэша вытесняется, новое ее состояние равно [3, 0, 2]

LOAD z, 0 - Hit, т.к. $0 \in \{3, 0, 2\}$

Все инструкции тестового шаблона были исполнены согласно указанным тестовым ситуациям.

IV. Генерация тестовых данных для кэш-памяти прямого отображения

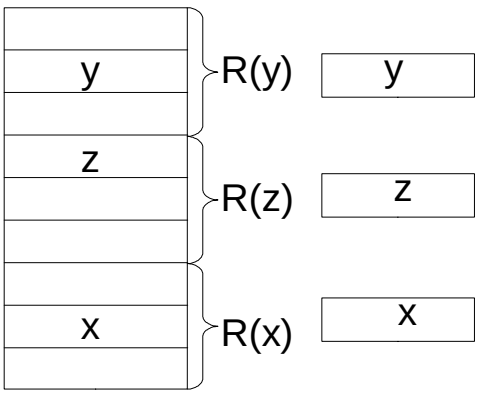

Рис. 3. Кэш прямого отображения

Вся память поделена на непересекающиеся части (будем называть их *регионами*). Каждому региону соответствуют 1 ячейка кэша. Других ячеек в кэше нет. В каждой ячейке кэша могут храниться данные только своего региона. При обращении к памяти по адресу первым делом данные ищутся в (единственной) ячейке кэша своего региона. Ситуация *кэш-попадания* означает, что в этой ячейке кэш-памяти находятся данные именно по требуемому адресу. Ситуация *кэш-промаха* означает, что в ячейке кэш-памяти требуемого региона содержатся данные по адресу отличному от требуемого. В случае кэш-промаха данные из ячейки кэш-памяти *вытесняются* и заменяются на данные по требуемому адресу. Поскольку вытесняемые данные определены однозначно, о стратегиях вытеснения для кэш-памяти прямого отображения не говорят.

Предлагаемый алгоритм построения ограничений для тестового шаблона в случае кэш-памяти прямого отображения основывается на следующих свойствах вытесняемых адресов:

1) любой (в том числе и вытесняемый) адрес был добавлен ранее одной из инструкций тестового шаблона (или находился среди адресов начального состояния кэш-памяти);
2) между вытеснением адреса и последним кэш-попаданием к нему (кэш-промах с замещением на этот адрес тоже считается кэш-попаданием) не происходит обращения к региону вытесняемого адреса.

Алгоритм строит ограничения на следующие переменные:

1) $\alpha_1, \alpha_2, \alpha_3, ...$ – адреса начального состояния кэш-памяти (их количество равно количеству регионов);
2) адреса, при обращении к которым происходят кэш-попадания (их количество равно количеству инструкций, при обращении к которым происходят кэш-попадания);
3) адреса, при обращении к которым происходят кэш-промахи (их количество равно количеству инструкций, при обращении к которым происходят кэш-промахи);
4) вытесняемые адреса (их количество равно количеству инструкций, при обращении к которым происходят кэш-промахи);
5) $L_0, L_1, ...$ – переменные-состояния кэш-памяти (их количество – на единицу больше количества инструкций, при обращении к которым происходят кэш-промахи).

Введем функциональный символ $R(y)$, который для адреса y возвращает множество всех ячеек того же региона, что и y. Для этого функционального символа справедливы следующие свойства:

$\forall x \ (x \in R(x))$

$\forall x \ \forall y \ (x = y \rightarrow R(x) = R(y))$

$\forall x \ \forall y \ (R(x) = R(y) \leftrightarrow x \in R(y))$

$\forall x \ \forall y \ (R(x) = R(y) \leftrightarrow y \in R(x))$

$\forall x \ \forall y \ (x \notin R(y) \rightarrow x \neq y)$

Алгоритм формирует ограничения для каждой очередной инструкции следующим образом (N – количество регионов):

1) «начальные ограничения» генерируются для любого шаблона один раз: $|\{\alpha_1, \alpha_2, ..., \alpha_N\}| = N$ (или, по-другому, все числа $\alpha_1, \alpha_2, ..., \alpha_N$ разные),

$|\{R(\alpha_1), R(\alpha_2), ..., R(\alpha_N)\}| = N$ (или, по-другому, все множества $R(\alpha_1), R(\alpha_2), ..., R(\alpha_N)$ разные);

2) «ограничения кэш-попадания» генерируются для каждой инструкции, при обращении к которой происходит кэш-попадание: $x \in L$, где x – адрес в инструкции, L – текущая переменная-состояние кэш-памяти;
3) «ограничения кэш-промаха» генерируются для каждой инструкции, при обращении к которой происходит кэш-промах (x – вытесняющий адрес, y – вытесняемый адрес, L – текущая переменная-состояние кэш-памяти): $y \in L, x \notin L, L' = L \cup \{x\} \setminus \{y\}, R(y) = R(x)$, L' становится текущей переменной-состояния кэш-памяти для следующей инструкции.

Ограничения для кэша прямого отображения отличаются от ограничений для полностью ассоциативного кэша лишь описанием вытесняемого адреса.

Рассмотрим тот же пример тестового шаблона и построение для него тестовых данных. Пусть память поделена на 3 региона в зависимости от остатка от деления на 3 адреса (т.е. $R(x) = R(y) \Leftrightarrow 3|(x - y)$).

LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Первым делом определимся с именами переменных так, чтобы они не меняли свое значение (введем «версии» переменных). Учтем, что LOAD дает новую версию переменной, идущей первым аргументом (т.к. в него загружается значение из памяти). Для описания вытесняемого адреса введем фиктивную переменную z'_0 (в ответ ее значение не попадет):

LOAD x_1, y_0 @ Hit
STORE u_0, z_0 @ Miss $\rightarrow z'_0$
LOAD z_1, y_0 @ Hit

Введем переменные начального состояния кэш-памяти: $\{\alpha, \beta, \gamma\}$ (по одной для каждого региона).

Наша задача состоит в поиске значений $x_0, y_0, z_0, u_0, \alpha, \beta, \gamma$, на которых инструкции тестового шаблона будут исполнены в заданных тестовых ситуациях. Логично предположить, что решение не будет единственным. Нам в данном случае достаточно найти какое-либо одно решение.

Согласно алгоритму для кэш-попаданий и кэш-промахов будут выделены следующие ограничения:

$y_0 \in \{\alpha, \beta, \gamma\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\} \cup \{z_0\}$,
$R(z_0) = R(z'_0)$,
α, β, γ – все разные
$R(\alpha), R(\beta), R(\gamma)$ – все разные

Упростим полученную систему ограничений:

$z'_0 \in \{\alpha, \beta, \gamma\}$,
$y_0 \in \{\alpha, \beta, \gamma\} \setminus \{z'_0\}$,
$z_0 \notin \{\alpha, \beta, \gamma\}$,
$3|(z_0 - z'_0)$,
α, β, γ – все разные
$R(\alpha), R(\beta), R(\gamma)$ – все разные

Заметьте, что x_0 и u_0 нигде не принимают участие – их значение может быть произвольным.

Пусть адреса занимают 8 бит. Тогда все переменные принимают значения из области от 0 до 255. Разрешая полученные ограничения, можно получить такие значения для переменных (замечу снова, что набор значений не является единственным):

$\alpha = x_0 = u_0 = 0$
$\beta = y_0 = 1$
$\gamma = 2$
$z_0 = 3$

Проверим исполнение тестового шаблона с такими значениями переменных:

начальное значения кэша: $L = [(R = 0) \mapsto 0, (R = 1) \mapsto 1, (R = 2) \mapsto 2]$

LOAD x, 1 - Hit, т.к. $R(1) = L[R = (1\ mod\ 3)]$

STORE 0, 3 - Miss, т.к. $R(3) \neq L[R = (3\ mod\ 3)]$, 1 из кэша вытесняется, новое ее состояние равно $L = [(R = 0) \mapsto 0, (R = 1) \mapsto 3, (R = 2) \mapsto 2]$

LOAD z, 0 - Hit, т.к. $R(0) = L[R = (0\ mod\ 3)]$

Все инструкции тестового шаблона были исполнены согласно указанным тестовым ситуациям.

V. Общий случай

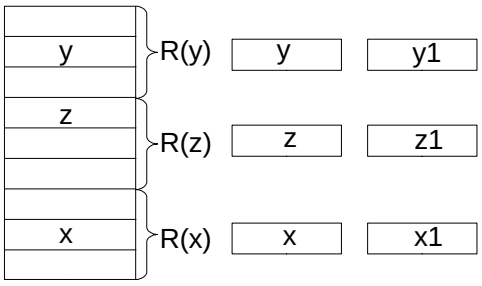

Рис. 4. Кэш общего вида

Без объяснений лишь в иллюстративных целях приведу ограничения и их решение для кэш-памяти общего вида (см. рис 4).

Функциональный символ $R(x)$ будет иметь ровно тот же смысл, что и при кэш-памяти прямого отображения.

Рассмотрим тот же тестовый шаблон для памяти из 3-х регионов ($R(x) = R(y) \leftrightarrow 3|(x - y)$) и 2-х ассоциативной кэш-памяти:

LOAD x, y @ Hit
STORE u, z @ Miss
LOAD z, y @ Hit

Вводим аналогично двум предыдущим случаям версии переменных и фиктивную переменную z'_0:

LOAD x_1, y_0 @ Hit
STORE u_0, z_0 @ Miss $\rightarrow z'_0$
LOAD z_1, y_0 @ Hit

Введем переменные для начального состояния кэша: α_1, α_2 для первого региона, β_1, β_2 для второго региона, γ_1, γ_2 для третьего. Сама система ограничений будет иметь следующий вид:

$y_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\}$,
$z'_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\}$,
$z_0 \notin \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \cap R(z'_0)$,
$y_0 \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \cup \{z_0\} \setminus \{z'_0\}$,
$R(z_0) = R(z'_0)$,
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – все разные,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – все разные

И дизъюнкция, описывающая $lru(z'_0)$ (достаточно одного дизъюнкта для получения решения):

$z'_0 = \gamma_2 \wedge (\{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2\} \setminus \{z'_0\}) \cap R(z'_0) = \{y_0\} \cap R(z'_0)$
$\vee$
...

Упрощаем полученную систему ограничений:

$y_0 \in \{\alpha_1, ..., \gamma_2\}$,
$z'_0 \in \{\alpha_1, ..., \gamma_2\}$,
$z_0 \notin \{\alpha_1, ..., \gamma_2\} \cap R(z'_0)$,
$y_0 \in \{\alpha_1, ..., \gamma_2, z_0\} \setminus \{z'_0\}$,
$R(z_0) = R(z'_0)$,
$z'_0 = \gamma_2$,
$\{\gamma_1\} = \{y_0\} \cap R(\gamma_2)$
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – все разные,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – все разные

И окончательное упрощение:

$z'_0 = \gamma_2$,
$y_0 = \gamma_1$,
$z_0 \notin \{\gamma_1, \gamma_2\}$,
$R(z_0) = R(\gamma_2)$,
$\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2$ – все разные,
$R(\alpha_1) = R(\alpha_2)$,
$R(\beta_1) = R(\beta_2)$,
$R(\gamma_1) = R(\gamma_2)$,
$R(\alpha_1), R(\beta_1), R(\gamma_1)$ – все разные

Разрешая эти ограничения для 8-битных адресов, можно получить такое решение:

$\alpha_1 = 0, \alpha_2 = 3$,
$\beta_1 = 1, \beta_2 = 4$,
$\gamma_1 = 2, \gamma_2 = 5$,
$x_0 = 0, y_0 = 2, z_0 = 7, u_0 = 0$.

В общем случае для символьного решения подобного рода систем ограничений (с операциями над множествами) могут применяться специальные алгоритмы разрешения на основе того, что все множества конечные (достаточно рассматривать $R(x)$ как множество лишь тех адресов, к которым обращаются инструкции тестового шаблона).

VI. Заключение

В статье рассматривалась задача генерации начального состояния кэш-памяти для тестовых шаблонов. В статье предложен алгоритм, строящий систему ограничений на конечные множества адресов. Результатом решения такой системы являются начальные значения регистров и ячеек кэш-памяти. В статье детально рассмотрено построение ограничений для полностью ассоциативной кэш-памяти и кэш-памяти прямого отображения (однако задача решена в общем случае).

Список литературы

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