

Кэш процессора

Материал из Википедии — свободной энциклопедии

Кэш микропроцессора — кэш (сверхоперативная память), используемый микропроцессором компьютера для уменьшения среднего времени доступа к компьютерной памяти. Является одним из верхних уровней иерархии памяти^[1]. Кэш использует небольшую, очень быструю память (обычно типа SRAM), которая хранит копии часто используемых данных из основной памяти. Если большая часть запросов в память будет обрабатываться кэшем, средняя задержка обращения к памяти будет приближаться к задержкам работы кэша.

Когда процессору нужно обратиться в память для чтения или записи данных, он сначала проверяет, доступна ли их копия в кэше. В случае успеха проверки процессор производит операцию, используя кэш, что значительно быстрее использования более медленной основной памяти. Подробнее о задержках памяти см. Латентность SDRAM: tCAS, tRCD, tRP, tRAS.

Данные между кэшем и памятью передаются блоками фиксированного размера, также называемыми **линиями кэша** (англ. *cache line*) или блоками кэша.

Большинство современных микропроцессоров для компьютеров и серверов имеет как минимум три независимых кэша: **кэш инструкций** для ускорения загрузки машинного кода, **кэш данных** для ускорения чтения и записи данных, и буфер ассоциативной трансляции (TLB) для ускорения трансляции виртуальных (логических) адресов в физические, как для инструкций, так и для данных. Кэш данных часто реализуется в виде многоуровневого кэша (L1, L2, L3, L4).

Увеличение размера кэш-памяти может положительно влиять на производительность почти всех приложений^[2], хотя в некоторых случаях эффект незначителен^[3]. Работа кэш-памяти обычно прозрачна для программиста, однако для её эффективного использования в некоторых случаях применяются специальные алгоритмические приёмы, изменяющие порядок обхода данных в ОЗУ или повышающие их локальность (например, при блочном умножении матриц)^[4].

Содержание

Принцип работы

Структура записи в кэше

Ассоциативность

Псевдоассоциативный кэш

Виды промахов

Категории промахов (*Three Cs*)

Трансляция адресов

Виртуальное тегирование и механизм vhint

Виртуальное индексирование и пересечения виртуальных адресов

Проблема гомонимов и синонимов

Расцвечивание страниц

Иерархия кэшей в современных микропроцессорах

[Специализированные кэши](#)
[Многоуровневые кэши](#)
[Эксклюзивность \(исключительность\) и инклюзивность](#)
[Victim cache](#)
[Кэш трасс](#)

Реализации

[История](#)
[В X86](#)
[Пример кэша \(процессорное ядро K8\)](#)
[DEC Alpha](#)
[PA-RISC](#)
[PowerPC](#)
[MIPS](#)
[Текущие разработки](#)

Примечания

[См. также](#)

[Ссылки](#)

Принцип работы

Данный раздел описывает типичный кэш данных и некоторые виды кэшей инструкций; буфер ассоциативной трансляции (TLB) может быть устроен сложнее, а кэш инструкций — проще. На диаграмме справа изображены основная и кэш-память. Каждая строка — группа ячеек памяти содержит данные, организованные в *кэш-линии*. Размер каждой кэш-линии может

различаться в разных процессорах, но для большинства x86-процессоров он составляет 64 байта. Размер кэш-линии обычно больше размера данных, к которому возможен доступ из одной машинной команды (типичные размеры от 1 до 16 байт). Каждая группа данных в памяти размером в 1 кэш-линию имеет порядковый номер. Для основной памяти этот номер является адресом памяти с отброшенными младшими битами. В кэше каждой кэш-линии дополнительно ставится в соответствие **тег**, который является адресом продублированных в этой кэш-линии данных в основной памяти.

При доступе процессора в память сначала производится проверка, хранит ли кэш запрашиваемые из памяти данные. Для этого производится сравнение адреса запроса со значениями всех тегов кэша, в которых эти данные могут храниться. Случай совпадения с тегом какой-либо кэш-линии называется *попаданием в кэш* (англ. *cache hit*), обратный же случай называется *кэш-промахом* (англ. *cache miss*). Попадание в кэш позволяет процессору немедленно произвести чтение или запись данных в кэш-линии с совпавшим тегом. Отношение количества попаданий в кэш к общему количеству запросов к памяти называют рейтингом попаданий (англ. *hit rate*), оно является мерой эффективности кэша для выбранного алгоритма или программы.

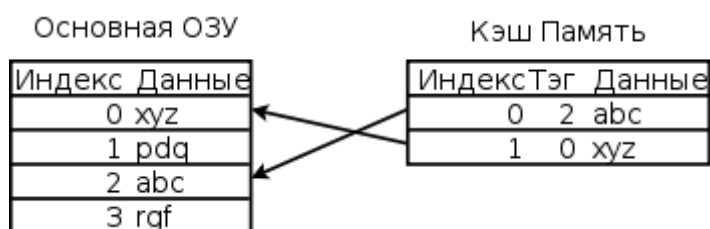


Диаграмма кэшей ЦПУ

В случае промаха в кэше выделяется новая запись, в тег которой записывается адрес текущего запроса, а в саму кэш-линию — данные из памяти после их прочтения либо данные для записи в память. Промехи по чтению задерживают исполнение, поскольку они требуют запроса данных в более медленной основной памяти. Промехи по записи могут не давать задержку, поскольку записываемые данные сразу могут быть сохранены в кэше, а запись их в основную память можно произвести в фоновом режиме. Работа кэшей инструкций во многом похожа на вышеприведенный алгоритм работы кэша данных, но для инструкций выполняются только запросы на чтение. Кэши инструкций и данных могут быть разделены для увеличения производительности (принцип, используемый в Гарвардской архитектуре) или объединены для упрощения аппаратной реализации.

Для добавления данных в кэш после кэш-промаха может потребоваться вытеснение (англ. *evict*) ранее записанных данных. Для выбора замещаемой строки кэша используется эвристика, называемая политика замещения (англ. *replacement policy*). Основной проблемой алгоритма является предсказание, какая строка вероятнее всего не потребуется для последующих операций. Качественные предсказания сложны, и аппаратные кэши используют простые правила, такие, как LRU. Пометка некоторых областей памяти как некэшируемых (англ. *non cacheable*) улучшает производительность за счёт запрета кэширования редко используемых данных. Промехи для такой памяти не создают копии данных в кэше.

При записи данных в кэш должен существовать определённый момент времени, когда они будут записаны в основную память. Это время контролируется *политикой записи* (англ. *write policy*). Для кэшей со *сквозной записью* (англ. *write-through*) любая запись в кэш приводит к немедленной записи в память. Другой тип кэшей, *обратная запись* англ. *write-back* (иногда также называемый *copy-back*), откладывает запись на более позднее время. В таких кэшах отслеживается состояние кэш-линеек ещё не сброшенных в память (пометка битом «грязный» англ. *dirty*). Запись в память производится при вытеснении подобной строки из кэша. Таким образом, промах в кэше, использующем политику обратной записи, может потребовать двух операций доступа в память, один для сброса состояния старой строки и другой — для чтения новых данных.

Существуют также смешанные политики. Кэш может быть со сквозной записью (англ. *write-through*), но для уменьшения количества транзакций на шине записи могут временно помещаться в очередь и объединяться друг с другом.

Данные в основной памяти могут изменяться не только процессором, но и периферией, использующей прямой доступ к памяти, или другими процессорами в многопроцессорной системе. Изменение данных приводит к устареванию их копии в кэше (состояние *stale*). В другой реализации, когда один процессор изменяет данные в кэше, копии этих данных в кэшах других процессоров будут помечены как *stale*. Для поддержания содержимого нескольких кэшей в актуальном состоянии используется специальный протокол поддержки когерентности.

Структура записи в кэше

Типичная структура записи в кэше

Блок данных	тег	бит актуальности
----------------	-----	---------------------

Блок данных (кэш-линия) содержит непосредственную копию данных из основной памяти. Бит актуальности означает, что данная запись содержит актуальную (самую свежую) копию.

Структура адреса

--	--	--

Адрес памяти разделяется (от старших бит к младшим) на *Тег*, *индекс* и *смещение*. Длина поля индекса равна $\lceil \log_2(cache_rows) \rceil$ бит и соответствует ряду (строке) кэша, используемой для записи. Длина смещения равна $\lceil \log_2(data_blocks) \rceil$.

Ассоциативность

Ассоциативность является компромиссом. Проверка большего числа записей требует больше затрат энергии, площади чипа, и, потенциально, времени. Если бы существовало 10 мест, в которые алгоритм вытеснения мог бы отобразить место в памяти, тогда проверка наличия этого места в кэше потребовала бы просмотра 10 записей в кэше. С другой стороны, кэши с высокой ассоциативностью подвержены меньшему количеству промахов (см. ниже «конфликтующие промахи») и процессор тратит меньше времени на чтения из медленной основной памяти. Существует эмпирическое наблюдение, что удвоение ассоциативности (от прямого отображения — к 2-канальной или от 2- — к 4-канальной) имеет примерно такое же влияние на интенсивность попаданий (*hit rate*), что и удвоение размера кэша. Увеличение ассоциативности свыше 4 каналов приносит меньший эффект для уменьшения количества промахов (*miss rate*) и обычно производится по другим причинам, например, из-за пересечения виртуальных адресов.

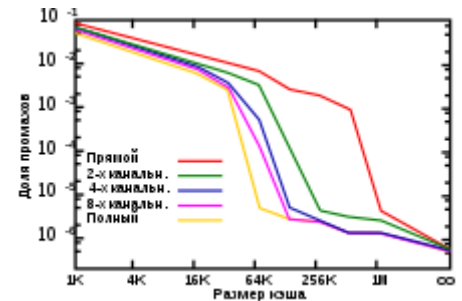
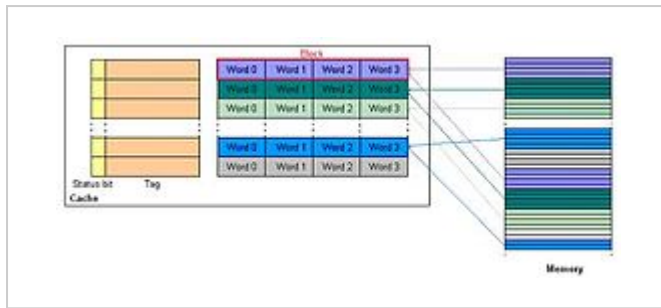


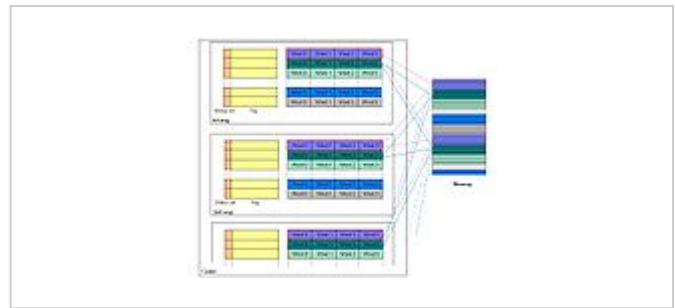
График эффективности кэша (количество промахов по оси ординат), в зависимости от степени ассоциативности и количества каналов. По оси абсцисс — размер кэша.

В порядке ухудшения (увеличения длительности проверки на попадание) и улучшения (уменьшения количества промахов):

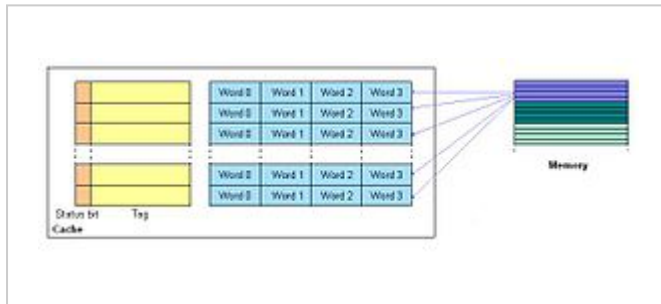
1. кэш прямого отображения ([англ. direct mapped cache](#)) — наилучшее время попадания и, соответственно, лучший вариант для больших кэшей;
2. 2-канальный множественно-ассоциативный кэш [англ. 2-way set associative cache](#);
3. 2-канальный skewed ассоциативный кэш ([англ. «the best tradeoff for caches whose sizes are in the range 4K-8K bytes» — André Seznec](#));
4. 4-канальный множественно-ассоциативный кэш ([англ. 4-way set associative cache](#));
5. полностью ассоциативный кэш, [англ. fully associative cache](#) — наилучший (самый низкий) процент промахов (*miss rate*) и лучший вариант при чрезвычайно высоких затратах при промахе (*miss penalty*).



Кэш прямого отображения



n-канальный множественно-ассоциативный
кэш



Полностью ассоциативный кэш

Псевдоассоциативный кэш

Виды промахов

Промар по чтению из кэша инструкций. Обычно дает очень большую задержку, поскольку процессор не может продолжать исполнение программы (по крайней мере, текущего потока исполнения) и вынужден простаивать в ожидании загрузки инструкции из памяти.

Промар по чтению из кэша данных. Обычно дает меньшую задержку, поскольку инструкции, не зависящие от запрошенных данных, могут продолжать исполняться, пока запрос обрабатывается в основной памяти. После получения данных из памяти можно продолжать исполнение зависимых инструкций.

Промар по записи в кэш данных. Обычно дает наименьшую задержку, поскольку запись может быть поставлена в очередь и последующие инструкции практически не ограничены в своих возможностях. Процессор может продолжать свою работу, кроме случаев промара по записи с полностью заполненной очередью.

Категории промахов (*Three Cs*)

- **Compulsory misses** — промахи, вызванные первым упоминанием запрошенного адреса. Размеры кэшей и их ассоциативность не влияют на количество данных промахов. Предвыборка (prefetching), как программная, так и аппаратная, может помочь, так же, как и увеличение размера кэш-линии (в качестве вида аппаратной предвыборки). Такие промахи иногда называются «холодными».
- **Capacity misses** — промахи, вызванные исключительно конечным размером кэша, происходящие вне зависимости от степени ассоциативности или размера кэш-линии. График таких промахов относительно размера кэша может дать некоторую меру временной локальности (temporal locality) некоторого набора запросов в память. Стоит заметить, что не существует понятия полного кэша, пустого кэша или почти полного кэша,

так как кэши процессора почти все время имеют кэш-линии в занятом состоянии, и, значит, практически каждое заведение новой линии потребует гашения уже занятой.

- **Conflict misses** — промахи, вызванные конфликтом. Их можно избежать, если бы кэш не вытеснил запись ранее. Можно дополнительно разделить на промахи, вызванные отображением (конкретным значением ассоциативности) и промахи замещения, которые вызваны конкретным алгоритмом выбора записей для замещения.

Трансляция адресов

Большая часть процессоров общего назначения реализует какой-либо вариант виртуальной памяти. Кратко говоря, каждая программа, исполняющаяся на машине, видит собственное упрощенное адресное пространство, содержащее код и данные только этой программы. Любая программа использует своё виртуальное адресное пространство вне зависимости от его местоположения в физической памяти.

Наличие виртуальной памяти требует от процессора проведения трансляции виртуальных (математических) адресов, используемых программой, в физические адреса, соответствующие реальному местоположению в ОЗУ. Часть процессора, проводящая это преобразование, называется устройством управления памятью (MMU). Для ускорения трансляций в MMU добавлен кэш недавно использованных отображений (соответствий виртуальных и физических адресов), называемый Translation Lookaside Buffer (TLB).

Для дальнейшего описания важны три особенности процесса трансляции адресов:

- **Задержка:** Физический адрес будет получен от MMU только спустя некоторое время, вплоть до нескольких тактов, после подачи на вход MMU виртуального адреса с генератора адресов.
- **Эффект наложения:** Несколько виртуальных адресов могут соответствовать одному физическому. В большинстве процессоров гарантируется, что все записи по физическому адресу будут совершены в порядке, заданном программой. Для выполнения этого свойства требуется проверка, что только один экземпляр копии данных с физического адреса находится в данный момент в кэше.
- **Единица отображения:** Виртуальное адресное пространство разбито на страницы — блоки памяти фиксированного размера, начинающиеся с адресов, кратных их размеру. Например, 4 ГБ адресного пространства можно разделить на 1048576 страниц по 4 кБ, для каждой из которых возможно независимое соответствие физическим страницам. В современных процессорах часто поддерживается использование одновременно нескольких размеров страниц, например, 4 кБ и 2 МБ для x86-64, а в некоторых современных AMD-процессорах ещё и 1 ГБ.

Важно также заметить, что первые системы виртуальной памяти были очень медленными, потому что они требовали проверки таблицы страниц (хранимой в основной ОЗУ) перед любым программным обращением в память. Без использования кэширования для отображений такие системы уменьшают скорость работы с памятью примерно в 2 раза. Поэтому использование TLB очень важно и иногда его добавление в процессоры предшествовало появлению обычных кэшей данных и инструкций.

По отношению к виртуальной адресации кэши данных и инструкций могут быть поделены на 4 типа. Адреса в кэшах используются для двух разных целей: индексирования и тегирования.

- *Physically indexed, physically tagged (PIPT)* — физически индексируемые и физически теглируемые. Такие кэши просты и избегают проблем с наложением (aliasing), но они медленны, так как перед обращением в кэш требуется запрос физического адреса в TLB.

Этот запрос может вызвать промах в TLB и дополнительное обращение в основную память перед тем, как наличие данных будет проверено в кэше.

- *Virtually indexed, virtually tagged (VIVT)* — виртуально индексируемые и виртуально тегированные. И для тегирования, и для индекса используется виртуальный адрес. Благодаря этому проверки наличия данных в кэше происходят быстрее, не требуя обращения к MMU. Однако возникает проблема наложения, когда несколько виртуальных адресов соответствуют одному и тому же физическому. В этом случае данные будут закэшированы дважды, что сильно усложняет поддержку когерентности. Другой проблемой являются омонимы, ситуации, когда один и тот же виртуальный адрес (например, в разных процессах) отображается в различные физические адреса. Становится невозможным различить такие отображения исключительно по виртуальному индексу. Возможные решения: сброс кэша при переключении между задачами (context switch), требование непересечения адресных пространств процессов, тегирование виртуальных адресов идентификатором адресного пространства (address space ID, ASID) или использование физических тегов. Также возникает проблема при изменении отображения виртуальных адресов в физические, что требует сброса кэш-линий, для которых изменилось отображение.
- *Virtually indexed, physically tagged (VIPT)* — виртуально индексируемые и физически тегированные. Для индекса используется виртуальный адрес, а для тега — физический. Преимуществом над первым типом является меньшая задержка, поскольку можно искать кэш-линию одновременно с трансляцией адресов в TLB, однако сравнение тега задерживается до получения физического адреса. Преимуществом над вторым типом является обнаружение омонимов (homonyms), так как тег содержит физический адрес. Для данного типа требуется больше бит для тега, поскольку индексные биты используют иной тип адресации.
- *Physically indexed, virtually tagged* — физически индексируемые и виртуально тегированные кэши считаются бесполезными и маргинальными и представляют исключительно академический интерес^[5].

Скорость этих действий (задержка загрузки из памяти) критически важна для производительности процессоров, и поэтому большинство современных L1-кэшей является виртуально индексируемым, что как минимум позволяет блоку MMU производить запрос в TLB одновременно с запросом данных из кэш-памяти.

Виртуальное тегирование и механизм vhints

Но виртуальная индексация не является лучшим выбором для других уровней кэша. Стоимость обнаружения пересечения виртуальных адресов (aliasing) растет с увеличением размера кэша и, в результате, большинство реализаций L2 и более дальних от процессора уровней кэша использует индексирование по физическим адресам.

Достаточно длительное время кэши использовали для тегов как физические, так и виртуальные адреса, хотя виртуальное тегирование в настоящее время встречается очень редко. Если TLB-запрос заканчивается раньше запроса в кэш-память, физический адрес будет доступен для сравнения с тегом к моменту, когда это будет необходимо, и, следовательно, виртуальное тегирование не потребуются. Большие кэши чаще тегированы физическими адресами, и только небольшие быстродействующие кэши используют для тегов виртуальные адреса. В современных процессорах общего назначения, виртуальное тегирование заменено на механизм vhints, описанный далее.

Виртуальное индексирование и пересечения виртуальных адресов

Проблема гомонимов и синонимов

Расцвечивание страниц

Иерархия кэшес в современных микропроцессорах

Большинство современных процессоров содержит в себе несколько взаимодействующих кэшес.

Специализированные кэши

Суперскалярные ЦПУ осуществляют доступ к памяти из нескольких этапов конвейера: чтение инструкции (instruction fetch), трансляция виртуальных адресов в физические, чтение данных (data fetch). Очевидным решением является использование различных физических кэшес для каждого из этих случаев, чтобы не было борьбы за доступ к одному из физических ресурсов с разных стадий конвейера. Таким образом, наличие конвейера обычно приводит к наличию, по крайней мере, трёх отдельных кэшес: кэш инструкции, кэш трансляций TLB и кэш данных, каждый из которых специализирован на своей задаче.

Конвейерные процессоры, использующие отдельные кэши для данных и для инструкций (такие процессоры сейчас повсеместны), называются процессорами с Гарвардской архитектурой. Изначально данный термин применялся для компьютеров, у которых инструкции и данные разделены полностью и хранятся в различных устройствах памяти. Однако такое полное разделение не оказалось популярным, и большинство современных компьютеров имеет одно устройство основной памяти, поэтому могут считаться машинами с архитектурой фон Неймана.

Многоуровневые кэши

Одной из проблем является фундаментальная проблема баланса между задержками кэша и интенсивностью попаданий. Большие кэши имеют более высокий процент попаданий но, вместе с тем, и большую задержку. Чтобы ослабить противоречие между этими двумя параметрами, большинство компьютеров использует несколько уровней кэша, когда после маленьких и быстрых кэшес находятся более медленные большие кэши (в настоящий момент — суммарно до 3 уровней в иерархии кэшес).

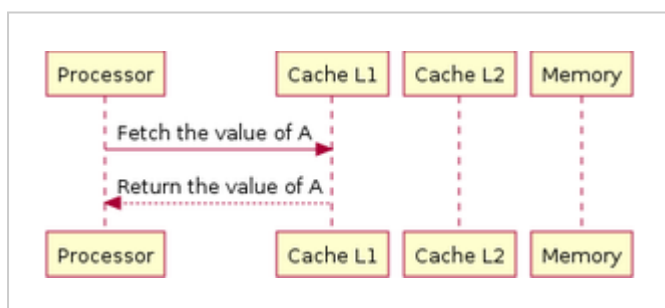
В единичных случаях реализуют 4 уровня кэш-памяти^{[6][7]}.

Многоуровневые кэши обычно работают в последовательности от меньших кэшес к большим. Сначала происходит проверка наименьшего и наибо́льшего кэша первого уровня (L1), в случае попадания процессор продолжает работу на высокой скорости. Если меньший кэш дал промах, проверяется следующий, чуть больший и более медленный кэш второго уровня (L2), и так далее, пока не будет запроса к основному ОЗУ.

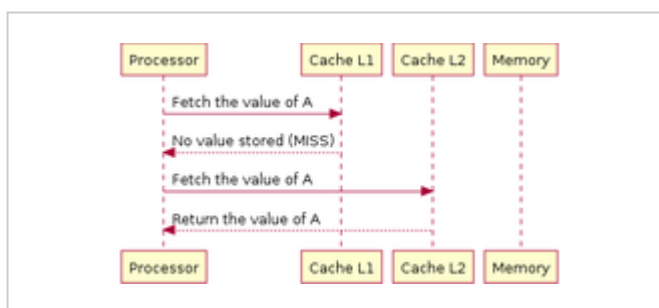
По мере того, как разница задержек между ОЗУ и быстрейшим кэшем увеличивается, в некоторых процессорах увеличивают количество уровней кэша (в некоторых — до 3х уровней на кристалле). К примеру, процессор Alpha 21164 в 1995 году имел накристалльный кэш 3го уровня в 96 кБ; IBM POWER4 в 2001 году имел до четырёх кэшес L3 по 32 МБ^[8] на отдельных кристаллах, используемых совместно несколькими ядрами; Itanium 2 в 2003 году имел 6 МБ кэш L3 на кристалле; Xeon MP под кодом «Tulsa» в 2006 году — 16 МБ кэша L3 на кристалле, общий на 2 ядра; Phenom II

в 2008 году — до 6 МБ универсального L3 кэша; Intel Core i7 в 2008 году — 8 МБ накристалльного кэша L3, являющимся инклюзивным и разделяемым между всеми ядрами. Польза от кэша L3 зависит от характера обращений программы в память.

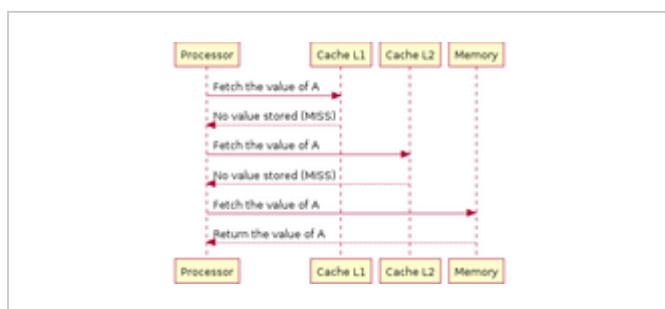
Наконец, с другой стороны иерархии памяти находится регистровый файл самого микропроцессора, который можно рассматривать как небольшой и самый быстрый кэш в системе со специальными свойствами (например, статическое планирование компилятором при распределении регистров, когда он располагает данные из ОЗУ на регистр). Подробнее см. [loop nest optimization](#). Регистровые файлы также могут иметь иерархию: Cray-1 (в 1976 году) имел 8 адресных «А»-регистров и 8 скалярных «S»-регистров общего назначения. Также машина содержала набор из 64 адресных «В» и 64 скалярных «Т» регистров, обращение к которым было дольше, но все же значительно быстрее основной ОЗУ. Эти регистры были введены по причине отсутствия в машине кэша данных (хотя кэш команд в машине имелся)



Затребованные данные считываются из кэша первого (англ. L1) уровня.



Затребованные данные считываются из кэша второго (англ. L2) уровня.



В случае отсутствия затребованных данных в кэшах, производится чтение из ОЗУ.

Эксклюзивность (исключительность) и инклюзивность

Для многоуровневых кэшей требуется делать новые архитектурные решения.

Например, в некотором процессоре могут потребовать, чтобы все данные, хранящиеся в кэше L1, хранились также и в кэше L2. Такие пары кэшей называют **строго инклюзивными** (англ. *inclusive*). Другие процессоры (например, AMD Athlon) могут не иметь подобного требования, тогда кэши называются **эксклюзивными (исключительными)** — данные могут быть либо в L1, либо в L2 кэше, но никогда не могут быть одновременно в обоих.

До сих пор другим процессорам (таким, как Pentium II, Pentium III, и Pentium 4) не требуются, чтобы данные в кэше первого уровня также размещались в кэше второго уровня, тем не менее, они продолжают так делать. Нет никакого универсального общепринятого имени для этой промежуточной политики, хотя часто используется термин **главным образом инклюзивно** (англ. *mainly inclusive*).

Преимущество исключительных кэшей в том, что они хранят больше данных. Это преимущество больше, когда исключительный кэш L1 сопоставим по размеру с кэшем L2, и меньше, если кэш L2 во много раз больше, чем кэш L1. Когда L1 пропускает и L2 получает доступ в случае попадания, строка кэша попадания в L2 обменивается со строкой в L1.

Victim cache

Victim cache или *Victim buffer*^[9] (дословно Кэш жертв) — это небольшой специализированный кэш, хранящий те кэш-линии, которые были недавно вытеснены из основного кэша микропроцессора при их замещении. Данный кэш располагается между основным кэшем и его англ. refill path. Обычно кэш жертв является полностью ассоциативным и служит для уменьшения количества конфликтных промахов (conflict miss). Многие часто используемые программы не требуют полного ассоциативного отображения для всех попыток доступа к памяти. По статистике только небольшая доля обращений к памяти потребует высокой степени ассоциативности. Именно для таких обращений служит кэш жертв, предоставляющий высокую ассоциативность для подобных редких запросов. Был предложен Norman Jouppi (DEC) в 1990^[10]. Размер такого кэша может составлять от 4 до 16 кэш-линий^[11].

Кэш трасс

Одним из наиболее экстремальных случаев специализации кэшей можно считать **кэш трасс** (англ. trace cache), используемый в процессорах Intel Pentium 4. Кэш трасс — это механизм для увеличения пропускной способности загрузки инструкций и для уменьшения тепловыделения (в случае Pentium 4) за счёт хранения декодированных трасс инструкций. Таким образом этот кэш исключал работу декодера при повторном исполнении недавно выполнявшегося кода.

Одной из ранних публикаций о кэше трасс была статья коллектива авторов (Eric Rotenberg, Steve Bennett и Jim Smith), вышедшая в 1996 году под названием «*Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching.*» (Кэш трасс: низколатентный подход для обеспечения высокой пропускной способности загрузки инструкций).

Кэш трасс сохраняет декодированные инструкции либо после их декодирования, либо после окончания их исполнения. Обобщая, инструкции добавляются в кэш трасс в группах, представляющих собой либо базовые блоки, либо динамические трассы. Динамическая трасса (путь исполнения) состоит только из инструкций, результаты которых были значимы (использовались впоследствии), и удаляет инструкции, которые находятся в неисполняющихся ветвях, кроме того, динамическая трасса может быть объединением нескольких базовых блоков. Такая особенность позволяет устройству подгрузки инструкций в процессоре загружать сразу несколько базовых блоков без необходимости заботиться о наличии ветвлений в потоке исполнения.

Линии трасс хранятся в кэше трасс по адресам, соответствующим счётчику инструкций первой машинной команды из трассы, к которым добавлен набор признаков предсказания ветвлений. Такая адресация позволяет хранить различные трассы исполнения, начинающиеся с одного адреса, но представляющие различные ситуации по результату предсказания ветвлений. На стадии подгрузки инструкции (instruction fetch) конвейера инструкций для проверки попадания в кэш трасс используется как текущий счётчик инструкций (program counter), так и состояние предсказателя ветвлений. Если попадание свершилось, линия трассы непосредственно подается на конвейер без необходимости опрашивать обычный кэш (L2) или основное ОЗУ. Кэш трасс подает машинные команды на вход конвейера, пока не кончится линия трассы, либо пока не произойдет ошибка предсказания в конвейере. В случае промаха кэш трасс начинает строить следующую линию трассы, загружая машинный код из кэша или из памяти.

Похожие кэши трасс использовались в Pentium 4 для хранения декодированных микроопераций и микрокода, реализующего сложные x86-инструкции. Smith, Rotenberg and Bennett's paper См полный текст работы (<https://web.archive.org/web/20080403043445/http://citeseer.ist.psu.edu/rotenberg96trace.html>) в Citeseer.

Реализации

История

В ранние годы микропроцессорных технологий доступ в память был лишь немного медленнее доступа к процессорным регистрам. Но с 1980-х^[12] разрыв в производительности между процессорами и памятью стал нарастать. Микропроцессоры совершенствовались быстрее, чем память, особенно в плане частоты функционирования, таким образом, память становилась узким местом при достижении полной производительности от системы. Хотя было технически возможным иметь основную память столь же быстрой, как и ЦПУ, был выбран более экономичный путь: использовать избыточное количество низкоскоростной памяти, но ввести в систему небольшую, но быструю кэш-память, для смягчения разрыва в производительности. В итоге получили на порядок большие объёмы памяти, примерно за ту же цену и с небольшими потерями общей производительности.

Чтение данных из кэша для современных процессоров обычно занимает более одного такта. Время исполнения программ является чувствительным к задержкам чтения из кэша данных первого уровня. Много усилий разработчиков, а также мощности и площади кристалла при создании процессора отводится для ускорения работы кэшей.

Простейшим кэшем является виртуально индексируемый кэш прямого отображения. Виртуальный адрес подсчитывается при помощи сумматора, соответствующая часть адреса выделяется и используется для индексирования SRAM, который вернет загружаемые данные. Данные могут быть выровнены по байтовым границам в байтовом сдвигателе и затем переданы следующей операции. При таком чтении не требуется какая-либо проверка тегов, фактически нет даже необходимости считывать тег. На более поздних стадиях конвейера, перед окончанием исполнения инструкции чтения, потребуется чтение тега и его сравнение с виртуальным адресом, чтобы удостовериться, что произошло попадание в кэш. Если же был промах, потребуется чтение из памяти либо более медленного кэша с дальнейшим обновлением рассматриваемого кэша и перезапуском конвейера.

Ассоциативный кэш более сложен, потому что некоторый вариант тега нужно считать для определения, какую часть кэша выбрать. Кэш N-way set-associative первого уровня обычно считывает одновременно все N возможных тегов и N данных параллельно, затем проводит сравнение тегов с адресом и выбор данных, ассоциированных с совпавшим тегом. Кэши 2-го уровня в целях экономии энерговыделения иногда выполняют сначала чтение тегов, и только затем чтение одного элемента данных из SRAM-данных.

Диаграмма справа должна показать, как происходит использование различных частей адреса. Бит 31 адреса является наиболее значимым битом (старшим), бит 0 — наименее значимым битом (младшим). На диаграмме показаны две SRAM, индексация и мультиплексирование для 4 КБ, 2-way set-associative, виртуально индексированного и виртуально тегированного кэша с 64байтными блоками, 32битной шириной чтения и 32битным виртуальным адресом.

Поскольку кэш имеет размер 4 КБ и линии размером 64 байта, в нём хранится 64 линии, и мы можем считать за два раза из тега SRAM, который содержит 32 столбца, каждый из которых содержит пару 21-битных тегов. Хотя может быть использована любая функция виртуальной адресации битов 31

по 6, чтобы индексировать тег и данные SRAM, проще всего воспользоваться младшими разрядами. Так же, потому что объём кэша составляет 4 кБ и имеет четырёхбайтный путь для чтения, и чтение производится по двум путями для каждого доступа, данные SRAM составляют 512 рядов шириной 8 байт.

Более современный кэш, возможно, был бы 16-килобайтным, четырёхпутным, набор-ассоциативным, виртуально индексировемым, виртуально попадаемым и физически помечаемым (тегом), с 32-битными строками, 32-битной шириной шины

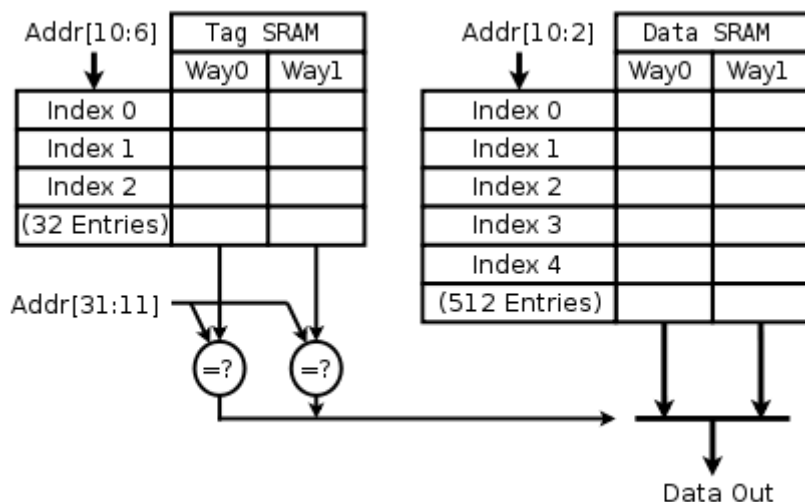
чтения и 36-битным физическим адресованием. Рекуррентное соотношение пути чтения для такого кэша выглядит очень схоже с рассмотренными выше. Вместо тегов читаются *виртуальные попадания*? (англ. *vhits*), и снова производится соответствие подмножества виртуальному адресу. Позже, в конвейере, виртуальный адрес переводится в физический адрес TLB, и производится чтение физического тега (только одно, так как виртуальное попадание поставяет путь для чтения кэша). В конце физический адрес сравнивается с физическим тегом, чтобы определить, произошло ли попадание.

Некоторые процессоры SPARC имели ускоренные на несколько задержек затвора (англ. *gate delay*) L1 кэши за счёт использования SRAM-декодеров вместо сумматора виртуальных адресов. Подробнее см. en:Sum addressed decoder.

В X86

Когда микропроцессоры x86 достигли частот в 20 и более мегагерц (начиная с Intel 80386), для увеличения производительности в них было добавлено небольшое количество быстрой кэш-памяти. Это было необходимо из-за того, что используемая как системная ОЗУ DRAM имела значительные задержки (до 120 нс), и требовала такты для обновления. Кэш был построен на базе более дорогой, но значительно более быстрой SRAM, которая в те времена имела задержки 15-20 нс. Ранние кэши были внешними по отношению к процессору и часто располагались на материнской плате как 8 или 9 микросхем в корпусах DIP, расположенные в сокетах для возможности увеличения или уменьшения размера кэша. Некоторые версии процессора I386 поддерживали от 16 до 64 кБ внешнего кэша^[13].

С выходом процессора Intel 80486 8 кБ кэша было интегрировано непосредственно на кристалл микропроцессора. Этот кэш был назван L1 (первого уровня, англ. *level 1*), чтобы отличать его от более медленного кэша на материнской плате,



4KB, 2-way set-associative 64B line cache read path

Путь чтения для 2-way associative cache



Четыре микросхемы кэша второго уровня на материнской плате для процессоров семейства i486. Располагаются в буквальном смысле между ЦП и ОЗУ.

названного L2 (второго уровня, англ. level 2). Последние были значительно больше, вплоть до 256 кБ.

В дальнейшем случаи отделения кэша производились, лишь исходя из соображений маркетинговой политики, например, в микропроцессоре Celeron, построенном на ядре Pentium II.

В микропроцессоре Pentium используется отдельный кэш команд и данных^[14]. Буфер трансляции адресов (TLB) преобразует адрес в ОЗУ в соответствующий адрес в кэше. Кэш данных Pentium использует метод обратной записи (англ. write-back), который позволяет модифицировать данные в кэше без дополнительного обращения к оперативной памяти (данные записываются в ОЗУ только при удалении из кэша) и протокол MESI (Modified, Exclusive, Shared, Invalid), который обеспечивает когерентность данных в кэшах процессоров и в ОЗУ при работе в мультипроцессорной системе.

Каждый из отдельных кэшей, данных и команд, микропроцессора Pentium MMX имеет объём 16 кБ и содержит два порта, по одному для каждого исполнительного конвейера. Кэш данных имеет буфер трансляции адресов (TLB).

Следующий вариант реализации кэшей в x86 появился в Pentium Pro, в котором кэш второго уровня (объединённый для данных и команд, размером 256—512 кБ) размещён в одном корпусе с процессором и кэшем первого уровня, размером 8 кБ, отдельным для данных и команд, и поднял его частоту до частоты ядра. Позже кэш второго уровня стал размещаться на том же кристалле, что и процессор.

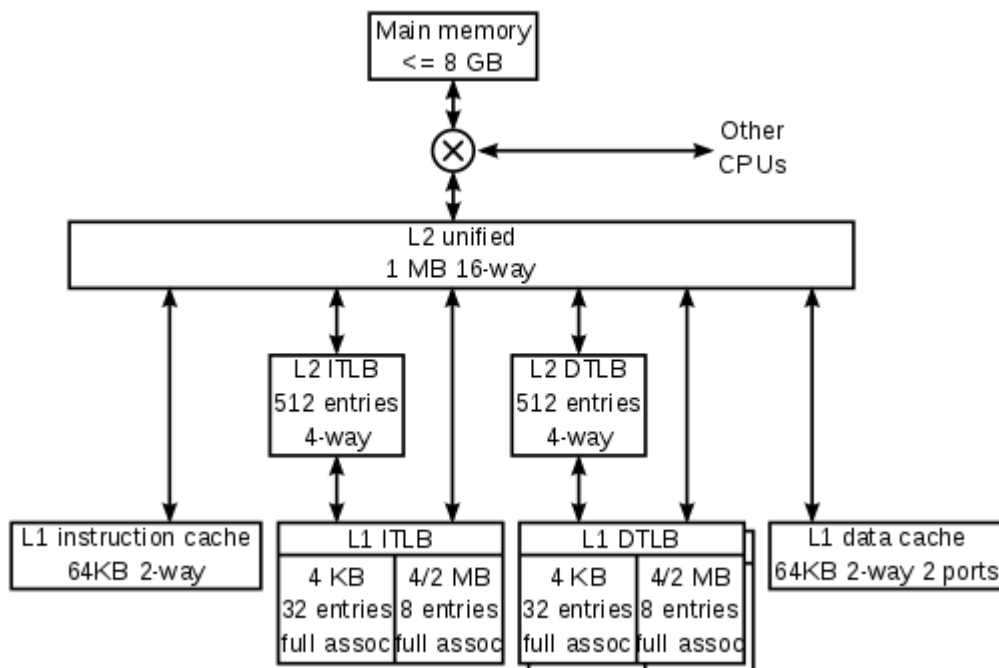
Двойная независимая шина (англ. Dual Independent Bus) — новая архитектура кэш-памяти, использует разные шины для соединения процессорного ядра с основной оперативной памятью. Кэш первого уровня двухпортовый, не блокирующий, поддерживает одну операцию загрузки и одну операцию записи за такт. Работает на тактовой частоте процессора. За такт передаётся 64 бита.

В микропроцессоре Pentium II кэш первого уровня увеличен — 16 кБ для данных и 16 кБ для команд. Для кэша второго уровня используется BSRAM, расположенная на одной с процессором плате в картридже S.E.C. для установки в Slot 1.

С ростом популярности многоядерных процессоров на кристалл стали добавлять кэши третьего уровня, названные L3. Этот уровень кэша может быть общим для нескольких ядер и реализовывать эффективное взаимодействие между ядрами. Его объём обычно больше суммарного объёма кэшей всех подключенных к нему ядер и может достигать 16 МБ.

Популярным кэш на материнской плате оставался до эры выхода Pentium MMX и вышел из употребления с введением SDRAM и ростом разницы между частотой шины процессора и частотой ядра процессора: кэш на материнской плате стал лишь немногим быстрее основной ОЗУ.

Пример кэша (процессорное ядро K8)



Приведена схема кэшей ядра микропроцессоров AMD K8, на которой видны как специализированные кэши, так и их многоуровневость.

Ядро использует четыре различных специализированных кэша: кэш инструкций, TLB инструкций, TLB данных и кэш данных:

- Кэш инструкций состоит из 64-байтных блоков, являющихся копией основной памяти, и может подгружать до 16 байтов за такт. Каждый байт в этом кэше хранится в 10 битах, а не в 8, причём в дополнительных битах отмечены границы инструкций (т. о. кэш проводит частичное преддекодирование). Для проверки целостности данных используется лишь контроль четности, а не ЕСС, так как бит четности занимает меньше места, а в случае сбоя повреждённые данные можно обновить правильной версией из памяти.
- TLB инструкций содержит копии записей из таблицы страниц. На каждый запрос чтения команд требуется трансляция математических адресов в физические. Записи о трансляции бывают 4- и 8-байтными, и TLB разбит на 2 части, соответственно одна для 4 КБ отображений и другая для 2 и 4 МБ отображений (большие страницы). Такое разбиение упрощает схемы полностью ассоциативного поиска в каждой из частей. ОС и приложения могут использовать отображения различного размера для частей виртуального адресного пространства.
- TLB данных является сдвоенным, и оба буфера содержат одинаковый набор записей. Их сдвоенность позволяет производить каждый такт трансляцию для двух запросов к данным одновременно. Так же, как и TLB инструкций, этот буфер разделен между записями двух видов.
- Кэш данных содержит 64-байтные копии фрагментов памяти. Он разделен на 8 банков (банок), каждый содержит по 8 килобайт данных. Кэш позволяет производить по два запроса к 8-байтовым данным каждый такт, при условии, что запросы будут обработаны различными банками. Теговые структуры в кэше продублированы, так как каждый 64-байтный блок распределен по всем 8 банкам. Если совершается 2 запроса в один такт, они работают с собственной копией теговой информации.

Также в этом ядре используются многоуровневые кэши: двухуровневые TLB инструкций и данных (на втором уровне хранятся лишь записи о 4-кБ отображениях), и кэш второго уровня (L2), унифицированный для работы как с кэшами данных и инструкций 1го уровня, так и для различных TLB. Кэш L2 является эксклюзивным для L1 данных и L1 инструкций, то есть каждый кэшированный 8-байтовый фрагмент может находиться либо в L1 инструкций, либо в L1 данных,

либо в L2. Исключением могут быть лишь байты, составляющие записи PTE, которые могут находиться одновременно в TLB и в кэше данных во время обработки виртуального отображения со стороны ОС. В таком случае ОС отвечает за своевременный сброс TLB после обновления записей трансляции.

DEC Alpha

В микропроцессоре DEC Alpha 21164 (выпущенном в ноябре 1995 года с тактовой частотой 333 МГц) кэш первого уровня может поддерживать некоторое количество (до 21) необработанных промахов. Имеется шестиэлементный файл адресов необработанных промахов ([англ. miss address file, MAF](#)), каждый элемент которого содержит адрес и регистр для загрузки при промахе (если адреса промаха принадлежат одной строке кэша, то в MAF они рассматриваются как один элемент).

Помимо отдельных кэшей L1 со сквозной записью, на кристалле процессора расположены частично-ассоциативный кэш L2 с обратной записью и контроллер кэша L3, работающего как в синхронном, так и в асинхронном режиме.

В выпущенном в марте 1997 года DEC Alpha 21164PC внешний кэш второго уровня; объём кэша команд увеличен до 16 кБ.

В микропроцессоре DEC Alpha 21264 нет кэша второго уровня (контроллер которого, тем не менее, размещается на кристалле), но кэш первого уровня увеличен до 128 кБ (по 64 кБ для кэша команд и кэша данных соответственно).

PA-RISC

Разработка Hewlett-Packard для научных и инженерных вычислений PA-8000 содержит буфер переупорядочивания адресов ([англ. ARB](#)), отслеживающий все команды загрузки/сохранения, который позволяет сократить задержку адресования внешней кэш-памяти данных и команд, которая в данном процессоре может иметь объём до 4 МБ. Тем не менее, даже эффективное управление внешним кэшем при помощи высокоскоростных линий управления и предвыборки данных и команд из основной памяти в кэш не компенсировало невысокую скорость и высокую стоимость.

Устранить указанные недостатки удалось в PA-8500, в котором за счёт техпроцесса 0,25 мкм удалось добавить на кристалл 512 кБ кэша команд и 1 МБ кэша данных.

PowerPC

Построенный на гарвардской архитектуре PowerPC 620 содержит два встроенных кэша, ёмкостью 32 кБ каждый, которые имеют собственные блоки управления памятью ([англ. MMU](#)) и функционируют независимо друг от друга. Команды и адреса переходов кэшируются в кэше BTAC ([англ. Branch-Target Address Cache](#)).

Шинный интерфейс процессора включает полную реализацию поддержки кэша второго уровня (объёмом до 128 МБ, работающем на частоте процессора либо вдвое/вчетверо меньшей), и для управления работой внешнего кэша не требует дополнительных тактов. Реализована комбинация сквозной и обратной записи а также поддержка протокола MESI.

MIPS

Своя специфика у кэша L1, применяемого в RA-10000, — каждая команда в кэше снабжена дополнительным четырёхбитным тегом, который используется в дальнейшем декодировании и классификации команды.

Текущие разработки


Примечания

1. Корнеев В. В., Киселев А. В. 1.2.3 Структурные методы уменьшения времени доступа к памяти // Современные микропроцессоры. — М: «Нолидж», 1998. — С. 75—76. — 240 с. — 5000 экз. — ISBN 5-98251-050-6.
2. Зависимость производительности процессора от размера кэша L2 (http://www.thg.ru/cpu/performance_vs_cache/performance_vs_cache-02.html)
3. AMD Athlon II X4 или Phenom II: влияние кэш-памяти L3 на производительность (http://www.thg.ru/cpu/amd_athlon_ii_x4_620_vs_phenom_ii/print.html)
4. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 1: Basic Architecture. Order number 253665-021.
5. Understanding Caching (<http://www.linuxjournal.com/article/7105>). Linux Journal. Дата обращения: 2 мая 2010. Архивировано (<https://www.webcitation.org/67EG0Yf0S?url=http://www.linuxjournal.com/article/7105>) 27 апреля 2012 года.
6. https://www.theregister.co.uk/2004/05/06/hp_mx2_itaniummodule/ "HP has packed mx2 with 32MB of L4 cache "
7. https://www.theregister.co.uk/2010/07/23/ibm_z196_mainframe_processor/ «L4 cache memory, which most servers do not have. (IBM added some L4 cache to its EXA chipsets for Xeon processors from Intel a few years back). This L4 cache is necessary for one key reason»
8. IBM POWER4 Processor Review. Ixبتlabs (<http://ixبتlabs.com/articles/ibmpower4/>) «An important feature of the L3 cache is a capability to combine separate caches of POWER4 chips up to 4 (128 MBytes) which allows using address interleaving to speed up the access.»
9. Детальное исследование архитектуры AMD64 (<http://www.ixبت.com/cpu/amd-hammer-family-2.shtml>) // ixبت.com, «Подсистема кэша. Поиск и анализ изменений»
10. N.P.Jouppi. «Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers.» — 17th Annual International Symposium on Computer Architecture, 1990. Proceedings., DOI:10.1109/ISCA.1990.134547
11. Victim Cache Simulator (<http://www.ecs.umass.edu/ece/koren/architecture/VCache/home.html>)
12. The Processor-Memory performance gap (<http://www.acm.org/crossroads/xrds5-3/pmgap.html>) (недоступная ссылка). acm.org. Дата обращения: 8 ноября 2007. Архивировано (<http://www.webcitation.org/67EG1BzQt?url=http://xrds.acm.org/>) 27 апреля 2012 года.
13. Гук М. 4. Кэширование памяти // Процессоры Pentium II, Pentium Pro и просто Pentium. — М: Питер, 1999. — С. 126—143. — 288 с. — 7000 экз. — ISBN 5-8046-0043-5.
14. Корнеев В. В., Киселев А. В. 2.2.1.2 Раздельные кэш-память команд и данных // Современные микропроцессоры. — М: «Нолидж», 1998. — С. 75—76. — 240 с. — 5000 экз. — ISBN 5-98251-050-6.

См. также

- [Когерентность кэша](#)
- [Кэш](#)

Ссылки

-
- [Memory part 2: CPU caches \(http://lwn.net/Articles/252125/\)](http://lwn.net/Articles/252125/) Статья на lwn.net (автор Ulrich Drepper) с детальным описанием кэшей
 - [8-way set-associative кэш \(https://web.archive.org/web/20110718154522/http://www.zipcores.com/skin1/zipdocs/datasheets/cache_8way_set.pdf\)](https://web.archive.org/web/20110718154522/http://www.zipcores.com/skin1/zipdocs/datasheets/cache_8way_set.pdf)  написанный на [VHDL](#)
-

Источник — https://ru.wikipedia.org/w/index.php?title=Кэш_процессора&oldid=114009127

Эта страница в последний раз была отредактирована 4 мая 2021 в 21:09.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike; в отдельных случаях могут действовать дополнительные условия.

Wikipedia® — зарегистрированный товарный знак некоммерческой организации Wikimedia Foundation, Inc.