**Кэш-память**

Кэш-память представляет собой быстродействующее ЗУ, размещенное на одном кристалле с ЦП или внешнее по отношению к ЦП. Кэш служит высокоскоростным буфером между ЦП и относительно медленной основной памятью. Идея кэш-памяти основана на прогнозировании наиболее вероятных обращений ЦП к оперативной памяти. В основу такого подхода положен принцип временной и пространственной локальности программы.

Если ЦП обратился к какому-либо объекту оперативной памяти, с высокой долей вероятности ЦП вскоре снова обратится к этому объекту. Примером этой ситуации может быть код или данные в циклах. Эта концепция описывается принципом временной локальности, в соответствии с которым часто используемые объекты оперативной памяти должны быть "ближе" к ЦП (в кэше).

***Для согласования содержимого кэш-памяти и оперативной памяти используют три метода записи:***

* *сквозная запись (write through) - одновременно с кэш-памятью обновляется оперативная память.*
* *буферизованная сквозная запись (buffered write through) - информация задерживается в кэш-буфере перед записью в оперативную память и переписывается в оперативную память в те циклы, когда ЦП к ней не обращается.*
* *обратная запись (write back) - используется бит изменения в поле тега, и строка переписывается в оперативную память только в том случае, если бит изменения равен 1.*

Как правило, все методы записи, кроме сквозной, позволяют для увеличения производительности откладывать и группировать операции записи в оперативную память.

***В структуре кэш-памяти выделяют два типа блоков данных:***

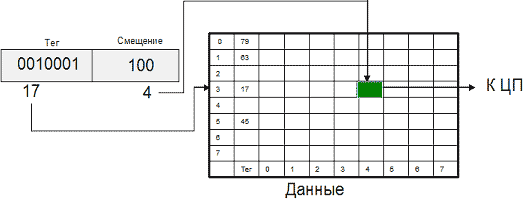
* *память отображения данных (собственно сами данные, дублированные из оперативной памяти);*
* *память тегов (признаки, указывающие на расположение кэшированных данных в оперативной памяти).*

Пространство памяти отображения данных в кэше разбивается на строки - блоки фиксированной длины (например, 32, 64 или 128 байт). Каждая строка кэша может содержать непрерывный выровненный блок байт из оперативной памяти. Какой именно блок оперативной памяти отображен на данную строку кэша, определяется тегом строки и алгоритмом отображения. По алгоритмам отображения оперативной памяти в кэш выделяют три типа кэш-памяти:

* *полностью ассоциативный кэш;*
* *кэш прямого отображения;*
* *множественный ассоциативный кэш.*

Для полностью ассоциативного кэша характерно, что кэш-контроллер может поместить любой блок оперативной памяти в любую строку кэш-памяти (рис. *Полностью ассоциативный кэш 8х8 для 10-битного адреса*). В этом случае физический адрес разбивается на две части: смещение в блоке (строке кэша) и номер блока. При помещении блока в кэш номер блока сохраняется в теге соответствующей строки. Когда ЦП обращается к кэшу за необходимым блоком, кэш-промах будет обнаружен только после сравнения тегов всех строк с номером блока.

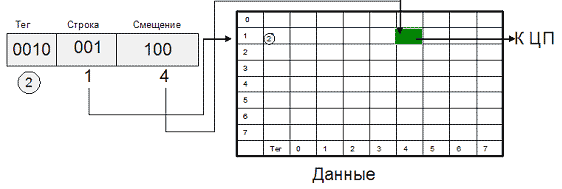
Одно из основных достоинств данного способа отображения - хорошая утилизация оперативной памяти, т.к. нет ограничений на то, какой блок может быть отображен на ту или иную строку кэш-памяти. К недостаткам следует отнести сложную аппаратную реализацию этого способа, требующую большого количества схемотехники (в основном компараторов), что приводит к увеличению времени доступа к такому кэшу и увеличению его стоимости.



*Рис. Полностью ассоциативный кэш 8х8 для 10-битного адреса*

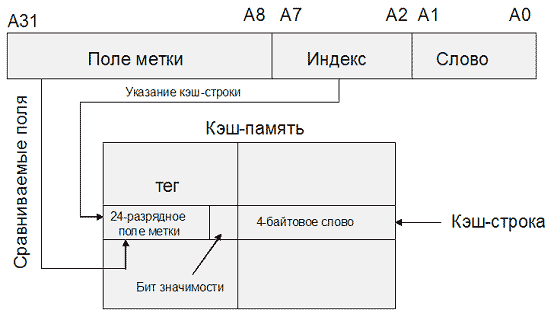
Альтернативный способ отображения оперативной памяти в кэш - это кэш прямого отображения (или одновходовый ассоциативный кэш). В этом случае адрес памяти (номер блока) однозначно определяет строку кэша, в которую будет помещен данный блок. Физический адрес разбивается на три части: смещение в блоке (строке кэша), номер строки кэша и тег. Тот или иной блок будет всегда помещаться в строго определенную строку кэша, при необходимости заменяя собой хранящийся там другой блок. Когда ЦП обращается к кэшу за необходимым блоком, для определения удачного обращения или кэш-промаха достаточно проверить тег лишь одной строки.

Очевидными преимуществами данного алгоритма являются простота и дешевизна реализации. К недостаткам следует отнести низкую эффективность такого кэша из-за вероятных частых перезагрузок строк. Например, при обращении к каждой 64-й ячейке памяти в системе на рис.*Кэш прямого отображения 8х8 для 10-битного адреса* кэш-контроллер будет вынужден постоянно перегружать одну и ту же строку кэш-памяти, совершенно не задействовав остальные.



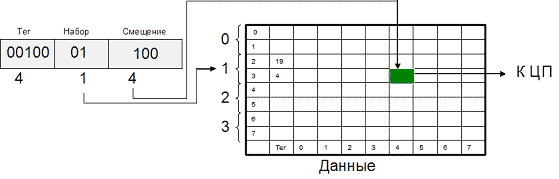
*Рис. Кэш прямого отображения 8х8 для 10-битного адреса*

Несмотря на очевидные недостатки, данная технология нашла успешное применение, например, в МП Motorola MC68020, для организации кэша инструкций первого уровня (*Рис. Схема организации кэш-памяти в МП Motorola MC68020*). В данном микропроцессоре реализован кэш прямого отображения из 64 строк по 4 байт. Тег строки, кроме 24 бит, задающих адрес кэшированного блока, содержит бит значимости, определяющий действительность строки (если бит значимости 0, данная строка считается недействительной и не вызовет кэш-попадания). Обращения к данным не кэшируются.



*Рис. Схема организации кэш-памяти в МП Motorola MC68020*

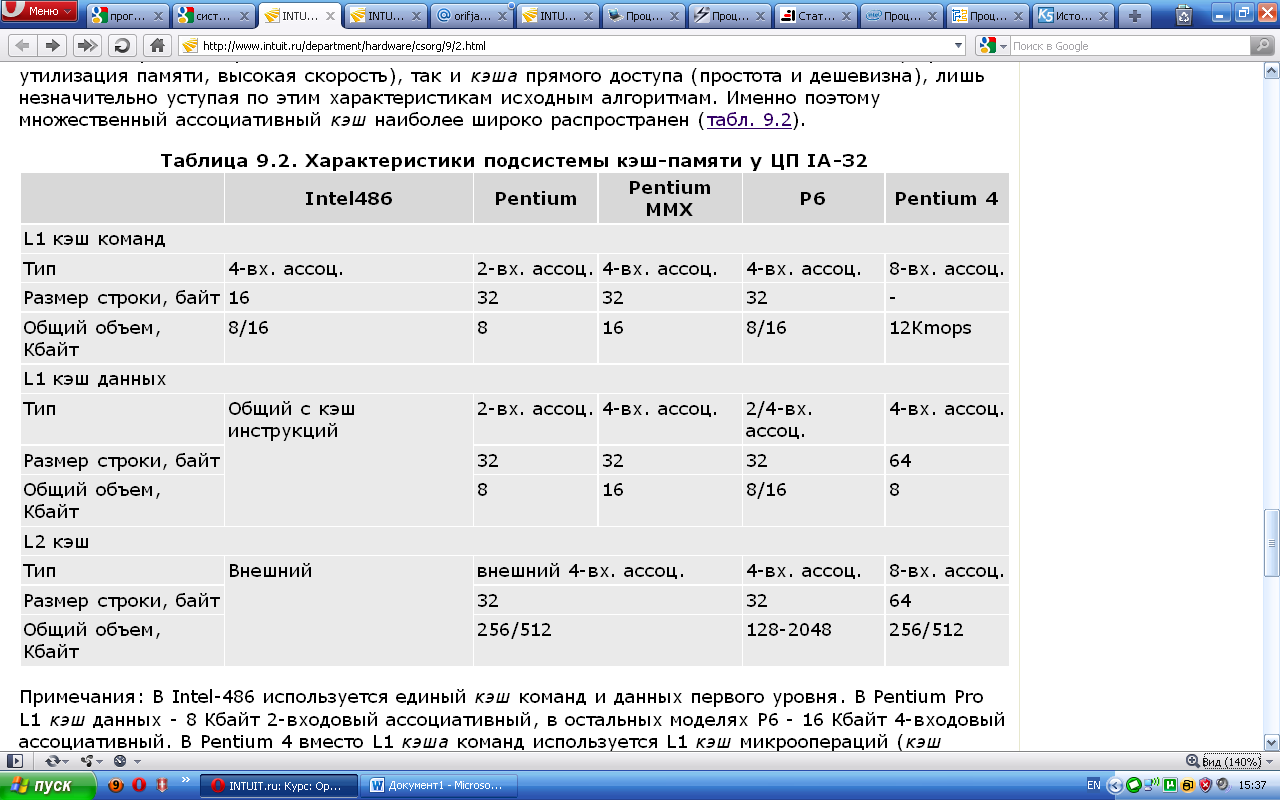
Компромиссным вариантом между первыми двумя алгоритмами является множественный ассоциативный кэш или частично-ассоциативный кэш (*Рис. Двухвходовый ассоциативный кэш 8х8 для 10-битного адреса*). При этом способе организации кэш-памяти строки объединяются в группы, в которые могут входить 2, 4, : строк. В соответствии с количеством строк в таких группах различают 2-входовый, 4-входовый и т.п. ассоциативный кэш. При обращении к памяти физический адрес разбивается на три части: смещение в блоке (строке кэша), номер группы (набора) и тег. Блок памяти, адрес которого соответствует определенной группе, может быть размещен в любой строке этой группы, и в теге строки размещается соответствующее значение. Очевидно, что в рамках выбранной группы соблюдается принцип ассоциативности. С другой стороны, тот или иной блок может попасть только в строго определенную группу, что перекликается с принципом организации кэша прямого отображения. Для того чтобы процессор смог идентифицировать кэш-промах, ему надо будет проверить теги лишь одной группы (2/4/8/: строк).



*Рис. Двухвходовый ассоциативный кэш 8х8 для 10-битного адреса*

Данный алгоритм отображения сочетает достоинства как полностью ассоциативного кэша (хорошая утилизация памяти, высокая скорость), так и кэша прямого доступа (простота и дешевизна), лишь незначительно уступая по этим характеристикам исходным алгоритмам. Именно поэтому множественный ассоциативный кэш наиболее широко распространен (табл. Характеристики подсистемы кэш-памяти у ЦП IA-32).

***Таблица . Характеристики подсистемы кэш-памяти у ЦП IA-32***



Примечания: В Intel-486 используется единый кэш команд и данных первого уровня. В Pentium Pro L1 кэш данных - 8 Кбайт 2-входовый ассоциативный, в остальных моделях P6 - 16 Кбайт 4-входовый ассоциативный. В Pentium 4 вместо L1 кэша команд используется L1 кэш микроопераций (кэш трассы).

Для организации кэш-памяти можно использовать принстонскую архитектуру (смешанный кэш для команд и данных, например, в Intel-486). Это очевидное (и неизбежное для фон-неймановских систем с внешней по отношению к ЦП кэш-памятью) решение не всегда бывает самым эффективным. Разделение кэш-памяти на кэш команд и кэш данных (кэш гарвардской архитектуры) позволяет повысить эффективность работы кэша по следующим соображениям:

* *многие современные процессоры имеют конвейерную архитектуру, при которой блоки конвейера работают параллельно. Таким образом, выборка команды и доступ к данным команды осуществляется на разных этапах конвейера, а использование раздельной кэш-памяти позволяет выполнять эти операции параллельно.*
* *кэш команд может быть реализован только для чтения, следовательно, не требует реализации никаких алгоритмов обратной записи, что делает этот кэш проще, дешевле и быстрее.*

Именно поэтому все последние модели IA-32, начиная с Pentium, для организации кэш-памяти первого уровня используют гарвардскую архитектуру.

Критерием эффективной работы кэша можно считать уменьшение среднего времени доступа к памяти по сравнению с системой без кэш-памяти. В таком случае среднее время доступа можно оценить следующим образом:

Tср = (Thit x Rhit) + (Tmiss x (1 Rhit))

где Thit - время доступа к кэш-памяти в случае попадания (включает время на идентификацию промаха или попадания), Tmiss - время, необходимое на загрузку блока из основной памяти в строку кэша в случае кэш-промаха и последующую доставку запрошенных данных в процессор, Rhit - частота попаданий.

Очевидно, что чем ближе значение Rhit к 1, тем ближе значение Tср к Thit. Частота попаданий определяется в основном архитектурой кэш-памяти и ее объемом. Влияние наличия и отсутствия кэш-памяти и ее объема на рост производительности ЦП показано в табл. *Размер и эффективность кэш-памяти*

***Таблица. Размер и эффективность кэш-памяти***

|  |  |  |
| --- | --- | --- |
| Размер кэш-памяти | Частота попаданий, % | Рост производительности, % |
| Нет кэш-памяти, DRAM с 2 TW | - | 0 |
| 16 Кб | 81 | 35 |
| 32 Кб | 86 | 38 |
| 64 Кб | 88 | 39 |
| 128 Кб | 89 | 39 |
| Нет кэш-памяти, SRAM без TW | - | 47 |

**Стратегия размещения.**

На сложность этого механизма существенное влияние оказывает   
стратегия размещения, определяющая, в какое место кэш-памяти   
следует поместить каждый блок из основной памяти.   
В зависимости от способа размещения данных основной памяти в кэш-памяти существует три типа кэш-памяти:

* кэш с прямым отображением (размещением);
* полностью ассоциативный кэш;
* множественный ассоциативный кэш или частично-ассоциативный.

**Кэш с прямым отображением** (размещением) является самым   
простым типом буфера. Адрес памяти однозначно определяет строку   
кэша, в которую будет помещен блок информации. При этом предпо-   
лагается, что оперативная память разбита на блоки и каждому та-   
кому блоку в буфере отводится всего одна строка. Это простой и недорогой в реализации способ отображения. Основной его недостаток – жесткое закрепление за определенными блоками ОП одной строки в кэше. Поэтому, если программа поочередно обращается к словам из двух различных блоков, отображаемых на одну и ту же строку кэш-памяти, постоянно будет происходить обновление данной строки и вероятность попадания будет низкой.

**Кэш с полностью ассоциативным отображением** позволяет преодолеть недостаток прямого, разрешая загрузку любого блока ОП в любую строку кэш-памяти. Логика управления выделяет в адресе ОП два поля: поле тега и поле слова. Поле тега совпадает с адресом блока ОП. Для проверки наличия копии блока в кэш-памяти, логика управления кэша должна одновременно проверить теги всех строк на совпадение с полем тега адреса. Ассоциативное отображение обеспечивает гибкость при выборе строки для вновь записываемого блока. Принципиальный недостаток этого способа – в необходимости использования дорогой ассоциативной памяти.

**Множественно-ассоциативный тип или частично-ассоциативный тип отображения –** это один из возможных компромиссов, сочетающий достоинства прямого и ассоциативного способов. Кэш-память ( и тегов и данных) разбивается на некоторое количество модулей. Зависимость между модулем и блоками ОП такая же жесткая, как и при прямом отображении. А вот размещение блоков по строкам модуля произвольное и для поиска нужной строки в пределах модуля используется ассоциативный принцип. Этот способ отображения наиболее широко распространен в современных микропроцессорах.

**Отображение секторов ОП в кэш-памяти.**

Данный тип отображения применяется во всех современных ЭВМ и состоит в том, что вся ОП разбивается на секторы, состоящие из фиксированного числа последовательных блоков. Кэш-память также разбивается на секторы, содержащие такое же количество строк. Расположение блоков в секторе ОП и секторе кэша полностью совпадает. Отображение сектора на кэш-память осуществляется ассоциативно, те любой сектор из ОП может быть помещен в любой сектор кэша. Таким образом, в процессе работы АЛУ обращается в поисках очередной команды к ОП, в результате чего, в кэш загружается( в случае отсутствия там блока, содержащего эту команду), целый сектор информации из ОП, причем по принципу локальности, за счет этого достигается значительное увеличение быстродействия системы.

**Иерархическая модель кэш-памяти**

Как правило, кэш-память имеет многоуровневую архитектуру. Например, в компьютере с 32 Кбайт внутренней (в ядре ЦП) и 1 Мбайт внешней (в корпусе ЦП или на системной плате) кэш-памяти первая будет считаться кэш-памятью 1-го уровня (L1), а вторая - кэш-памятью 2-го уровня (L2). В современных серверных системах количество уровней кэш-памяти может доходить до четырех, хотя наиболее часто используется двух- или трехуровневая схема.

В некоторых процессорных архитектурах кэш-память 1-го уровня разделена на кэш команд (Instruction Cache, I-cache) и кэш данных (Data Cache, D-cache), причем необязательно одинаковых размеров. С точки зрения схемотехники проще и дешевле проектировать раздельные I-cache и D-cache: выборку команд проводит I-box, а выборку данных - Е-box и F-box, хотя в обоих случаях задействуются А-box и С-box. Все эти блоки велики, и обеспечить им одновременный и быстрый доступ к одному кэшу проблематично. Кроме того, это неизбежно потребовало бы увеличения количества портов доступа, что также усложняет задачу проектирования.

Так как I-cache и D-cache должны обеспечивать очень низкие задержки при доступе (это справедливо для любого кэша L1), приходится жертвовать их объемом - обычно он составляет от 16 до 32 Кбайт. Ведь чем меньше размер кэша, тем легче добиться низких задержек при доступе.

Кэш-память 2-го уровня, как правило, унифицирована, т. е. может содержать как команды, так и данные. Если она встроена в ядро ЦП, то говорят о S-cache (Secondary Cache, вторичный кэш), в противном случае - о B-cache (Backup Cache, резервный кэш). В современных серверных ЦП объем S-cache составляет от одного до нескольких мегабайт, a B-cache - до 64 Мбайт. Если дизайн ЦП предусматривает наличие встроенной кэш-памяти 3-го уровня, то ее именуют T-cache (Ternary Cache, третичный кэш). Как правило, каждый последующий уровень кэш-памяти медленнее, но больше предыдущего по объему. Если в системе присутствует B-cache (как последний уровень модели кэш-памяти), то он может контролироваться как ЦП, так и набором системной логики.

Если в момент выполнения некоторой команды в регистрах не окажется данных для нее, то они будут затребованы из ближайшего уровня кэш-памяти, т. е. из D-cache. В случае их отсутствия в D-Cache запрос направляется в S-cache и т. д. В худшем случае данные будут доставлены непосредственно из памяти. Впрочем, возможен и еще более печальный вариант, когда подсистема управления виртуальной памятью операционной системы (ОС) успевает вытеснить их в файл подкачки на жесткий диск. В случае доставки из оперативной памяти потери времени на получение нужных данных могут составлять от десятков до сотен тактов ЦП, а в случае нахождения данных на жестком диске речь уже может идти о миллионах тактов.

**Ассоциативность кэш-памяти**

Одна из фундаментальных характеристик кэш-памяти - уровень ассоциативности - отображает ее логическую сегментацию. Дело в том, что последовательный перебор всех строк кэша в поисках необходимых данных потребовал бы десятков тактов и свел бы на нет весь выигрыш от использования встроенной в ЦП памяти. Поэтому ячейки ОЗУ жестко привязываются к строкам кэш-памяти (в каждой строке могут быть данные из фиксированного набора адресов), что значительно сокращает время поиска. С каждой ячейкой ОЗУ может быть связано более одной строки кэш-памяти: например, n-канальная ассоциативность (n-way set associative) обозначает, что информация по некоторому адресу оперативной памяти может храниться в п мест кэш-памяти.

Выбор места может проводиться по различным алгоритмам, среди которых чаще всего используются принципы замещения LRU (Least Recently Used, замещается запись, запрошенная в последний раз наиболее давно) и LFU (Least Frequently Used, запись, наименее часто запрашиваемая), хотя существуют и модификации этих принципов. Например, полностью ассоциативная кэшпамять (fully associative), в которой информация, находящаяся по произвольному адресу в оперативной памяти, может быть размещена в произвольной строке. Другой вариант - прямое отображение (direct mapping), при котором информация, которая находится по произвольному адресу в оперативной памяти, может быть размещена только в одном месте кэш-памяти. Естественно, этот вариант обеспечивает наибольшее быстродействие, так как при проверке наличия информации контроллеру придется "заглянуть" лишь в одну строку кэша, но и наименее эффективен, поскольку при записи контроллер не будет выбирать "оптимальное" место. При одинаковом объеме кэша схема с полной ассоциативностью будет наименее быстрой, но наиболее эффективной.

Полностью ассоциативный кэш встречается на практике, но, как правило, у него очень небольшой объем. Например, в ЦП Cyrix 6x86 использовалось 256 байт такого кэша для команд перед унифицированным 16-или 64-Кбайт кэшем L1. Часто полноассоциативную схему применяют при проектировании TLB (о них будет рассказано ниже), кэшей адресов переходов, буферов чтения-записи и т. д. Как правило, уровни ассоциативности I-cache и D-cache довольно низки (до четырех каналов) - их увеличение нецелесообразно, поскольку приводит к увеличению задержек доступа и в итоге негативно отражается на производительности. В качестве некоторой компенсации увеличивают ассоциативность S-cache (обычно до 16 каналов), так как задержки при доступе к этому кэшу неважны. Например, согласно результатам исследований часто используемых целочисленных задач, у Intel Pentium III 16 Кбайт четырехканального D-cache было достаточно для покрытия около 93% запросов, а 16-Кбайт четырехканального I-cache - 99% запросов.

**Размер строки и тега кэш-памяти**

Немаловажная характеристика кэш-памяти - размер строки. Как правило, на одну строку полагается одна запись адреса (так называемый тег), которая указывает, какому адресу в оперативной памяти соответствует данная линия. Очевидно, что нумерация отдельных байтов нецелесообразна, поскольку в этом случае объем служебной информации в кэше в несколько раз превысит объем самих данных. Поэтому один тег обычно полагается на одну строку, размер которой обычно 32 или 64 байта (реально существующий максимум 1024 байта), и эквивалентен четырем (иногда восьми) разрядностям системной шины данных. Кроме того, каждая строка кэш-памяти сопровождается некоторой информацией для обеспечения отказоустойчивости: одним или несколькими битами контроля четности (parity) или восемью и более байтами обнаружения и коррекции ошибок (ЕСС, Error Checking and Correcting), хотя в массовых решениях часто не используют ни того, ни другого.

Размер тега кэш-памяти зависит от трех основных факторов: объема кэш-памяти, максимального кэшируемого объема оперативной памяти, а также ассоциативности кэш-памяти. Математически этот размер рассчитывается по формуле:

***Stag=log2(Smem\*A/Scache),***

где Stag - размер одного тега кэш-памяти, в битах; Smem - максимальный кэшируемый объем оперативной памяти, в байтах; Scache - объем кэш-памяти, в байтах; А - ассоциативность кэш-памяти, в каналах.

Отсюда следует, что для системы с 1-Гбайт оперативной памятью и 1-Мбайт кэш-памятью с двухканальной ассоциативностью потребуется 11 бит для каждого тега. Примечательно, что собственно размер строки кэш-памяти никак не влияет на размер тега, но обратно пропорционально влияет на количество тегов. Следует понимать, что размер строки кэш-памяти не имеет смысла делать меньше разрядности системной шины данных, но многократное увеличение размера приведет к чрезмерному засорению кэш-памяти ненужной информацией и излишней нагрузке на системную шину и шину памяти. Кроме того, максимально кэшируемый объем кэш-памяти не обязан соответствовать максимально возможному устанавливаемому объему оперативной памяти в системе. Если возникнет ситуация, когда оперативной памяти окажется больше, чем может быть кэшировано, то в кэш-памяти будет присутствовать информация только из нижнего сегмента оперативной памяти. Именно такой была ситуация с платформой Socket7/Super7. Наборы микросхем для этой платформы позволяли использовать большие объемы оперативной памяти (от 256 Мбайт до 1 Гбайт), в то время как кэшируемый объем часто был ограничен первыми 64 Мбайт (речь идет о B-cache, находящемся на системной плате) по причине использования дешевых 8-бит микросхем теговой SRAM (2 бита из которых резервировалось под указатели действительности и измененности строки). Это приводило к ощутимому падению производительности.

Какая информация содержится в тегах кэш-памяти? Это информация об адресах, но как можно точно отобразить расположение строки кэш-памяти на всем пространстве кэшируемого объема оперативной памяти, используя столь незначительное количество адресных битов? Это понятие является фундаментальным в понимании принципов функционирования кэш-памяти. Рассмотрим предыдущий пример, с 11-бит тегами. Учитывая логическое сегментирование благодаря двухканальной ассоциативности, можно рассматривать данную кэш-память как состоящую из двух независимых сегментов по 512 Кбайт каждый. Представим оперативную память как состоящую из "страниц" по 512 Кбайт каждая - их будет соответственно 2048 штук. Далее, Iog2 (2048) = 11 (основание логарифма равно 2, так как возможны только два логических состояния каждого бита). Это означает, что фактически тег - не номер отдельной строки кэш-памяти, а номер "страницы" памяти, на которую отображается та или иная строка. Другими словами, в пределах "страницы" сохраняется прямое соответствие ее "строк" с соответствующими строками кэш-памяти, т. е. п-я строка кэш-памяти соответствует n-й "строке" данной "страницы" оперативной памяти.

Рассмотрим механизм работы кэш-памяти разных видов ассоциативности. Допустим, имеется абстрактная модель с восемью строками кэш-памяти и 64 эквивалентными строками оперативной памяти. Требуется поместить в кэш строку 9 оперативной памяти (заметим, что все строки нумеруются от нуля и по возрастающей). В модели с прямым отображением эта строка может занять только одно место: 9 mod 8=1 (вычисление остатка от деления нацело), т. е. место строки 1. Если взять модель с двухканальной ассоциативностью, то эта строка может занять одно из двух мест: 9 mod 4=1, т. е. строку 1 любого канала (сегмента). Полноассоциативная модель предоставляет свободу для размещения, и данная строка может занять место любой из восьми имеющихся. Другими словами, фактически имеется 8 каналов, каждый из которых состоит из 1 строки.

Ни одна из вышеуказанных моделей не позволит, разумеется, поместить в кэш больше строк, чем он физически в состоянии разместить, они лишь предлагают различные варианты, различающиеся балансом эффективности использования кэша и скорости доступа к нему.

**Типы подключения кэш-памяти**

Количество портов чтения-записи кэш-памяти - показатель того, сколько одновременных операций чтения-записи может быть обработано. Хотя жестких требований и нет, определенное соответствие набору функциональных устройств ЦП должно прослеживаться, так как отсутствие свободного порта во время исполнения команды приведет к вынужденному простою.

Существует два основных способа подключения кэшпамяти к ЦП для чтения: сквозной и побочный (Look-Through и Look-Aside). Суть первого в том, что при необходимости данные сначала запрашиваются у контроллера кэш-памяти самого высокого уровня, который проверяет состояние подключенных тегов и возвращает либо нужную информацию, либо отрицательный ответ, и в этом случае запрос перенаправляется в более низкий уровень иерархии кэш-памяти или в оперативную память. При реализации второго способа чтения запрос одновременно направляется как кэш-контроллеру самого высокого уровня, так и остальным кэш-контроллерам и контроллеру оперативной памяти. Недостаток первого способа очевиден: при отсутствии информации в кэше высокого уровня приходится повторять запрос, и время простоя ЦП увеличивается. Недостаток второго подхода - высокая избыточность операций и, как следствие, "засорение" внутренних шин ЦП и системной шины ненужной информацией. Логично предположить, что если для кэшей L1 оптимальна сквозная схема, то для T-cache или B-cache побочная схема может оказаться более выгодной. Для S-cache выбор неоднозначен.

Различают также локальный и удаленный кэш. Локальным называют кэш, находящийся либо в ядре ЦП, либо на той же кремниевой подложке или в корпусе ЦП, удаленным - размещенный на системной плате. Соответственно локальным кэшем управляет контроллер в ядре ЦП, а удаленным - НМС системной платы. Локальный кэш с точки зрения быстродействия предпочтительнее, так как интерфейс к удаленному кэшу обычно мультиплексируется с системной шиной. С одной стороны, когда другой ЦП захватывает общую системную шину или какой-либо периферийный контроллер обращается к памяти напрямую, удаленный кэш может оказаться временно недоступным. С другой - такой кэш легче использовать в многопроцессорных системах.

Существуют два распространенных способа записи в кэш: сквозной (Write-Through) и обратной (Write-Back) записи. В первом случае информация одновременно сохраняется как в текущий, так и в более низкий уровень иерархии кэш-памяти (или прямо в оперативную память при отсутствии такового). Во втором - данные сохраняются только в текущем уровне кэш-памяти, при этом возникает ситуация, когда информация в кэше и оперативной памяти различается, причем последняя становится устаревшей. Для того чтобы при сбросе кэша информация не была необратимо потеряна, к каждой строке кэша добавляется "грязный" бит (dirty bit, иначе известный как modified). Он нужен для обозначения того, соответствует ли информация в кэше информации в оперативной памяти, и следует ли ее записать в память при сбросе кэша.

Также следует упомянуть способ резервирования записи (write allocation). При записи данных в оперативную память часто возникает ситуация, когда записываемые данные могут скоро понадобиться, и тогда их придется довольно долго подгружать. Резервирование записи позволяет частично решить эту проблему: данные записываются не в оперативную память, а в кэш. Строка кэша, вместо которой записываются данные, полностью выгружается в оперативную память. Так как вновь записанных данных обычно недостаточно для формирования полной строки кэша, из оперативной памяти запрашивается недостающая информация. Когда она получена, новая строка записывается, и тег обновляется. Определенных преимуществ или недостатков такой подход не имеет - иногда это может дать незначительный прирост производительности, но также и привести к засорению кэша ненужной информацией.

**Увеличение кэша и тестирование INEL&AMD**

Первичная причина увеличения объема встроенного кэша может заключаться в том, что кэш-память в современных процессорах работает на той же скорости, что и сам процессор. Частота процессора в этом случае никак не меньше 3200 MГц. Больший объем кэша позволяет процессору держать большие части кода готовыми к выполнению. Такая архитектура процессоров сфокусирована на уменьшении задержек, связанных с простоем процессора в ожидании данных. Современные программы, в том числе игровые, используют большие части кода, который необходимо извлекать из системной памяти по первому требованию процессора. Уменьшение промежутков времени, уходящих на передачу данных от памяти к процессору, - это надежный метод увеличения производительности приложений, требующих интенсивного взаимодействия с памятью. Кэш L3 имеет немного более высокое время ожидания, чем L 1 и 2, это вполне естественно. Хоть он и медленнее, но все-таки он значительно более быстрый, чем обычная память. Не все приложения выигрывают от увеличения объема или скорости кэш-памяти. Это сильно зависит от природы приложения.

Если большой объем встроенного кэша - это хорошо, тогда что же удерживало Intel и AMD от этой стратегии ранее? Простым ответом является высокая себестоимость такого решения. Резервирование пространства для кэша очень дорого. Стандартный 3.2GHz Northwood содержит 55 миллионов транзисторов. Добавляя 2048 КБ кэша L3, Intel идет на увеличение количества транзисторов до 167 миллионов. Простой математический расчет покажет нам, что EE - один из самых дорогих процессоров.

Сайт AnandTech провел сравнительное тестирование двух систем, каждая из которых содержала два процессора – Intel Xeon 3,6 ГГц в одном случае и AMD Opteron 250 (2,4 ГГц) – в другом. Тестирование проводилось для приложений ColdFusion MX 6.1, PHP 4.3.9, и Microsoft .NET 1.1. Конфигурации выглядели следующим образом:

AMD

- Dual Opteron 250;

- 2 ГБ DDR PC3200 (Kingston KRX3200AK2);

- системная плата Tyan K8W;

- ОС Windows 2003 Server Web Edition (32 бит);

- 1 жесткий IDE 40 ГБ 7200 rpm, кэш 8 МБ

Intel

- Dual Xeon 3.6 ГГц;

- 2 ГБ DDR2;

- материнская плата Intel SE7520AF2;

- ОС Windows 2003 Server Web Edition (32 бит);

- 1 жесткий IDE 40 ГБ 7200 rpm, кэш 8 МБ

На приложениях ColdFusion и PHP, не оптимизированных под ту или иную архитектуру, чуть быстрее (2,5-3%) оказались Opteron’ы, зато тест с .NET продемонстрировал последовательную приверженность Microsoft платформе Intel, что позволило паре Xeon’ов вырваться вперед на 8%. Вывод вполне очевиден: используя ПО Microsoft для веб-приложений, есть смысл выбрать процессоры Intel, в других случаях несколько лучшим выбором будет AMD.

**Вывод**

Анализ изложенного выше материала позволяет сделать заключение, что в соответствии с каноническими теориями, современные производители широко используют кэш-память при построении новейших процессоров. Во многом, их превосходные характеристики по быстродействию достигаются именно благодаря применению кэш-памяти второго и даже третьего уровня. Этот факт подтверждает теоретические выкладки Гарвардского университета о том, что ввиду действия принципа локальности информации в современных компьютерах применение кэш-памяти смешанного типа позволяет добиться превосходных результатов в производительности процессоров и снижает частоту необходимых обращений к основной памяти.

Налицо широкие перспективы дальнейшего применения кэш-памяти в машинах нового поколения, однако существующая проблематика невозможности бесконечного увеличения кэша, а также высокая себестоимость изготовления кэша на одном кристалле с процессором, ставит перед конструкторами вопросы о некоем качественном, а не количественном видоизменении или скачке в принципах, либо огранизации кэш-памяти в процессорах будущего.

**К данной работе были использованы материалы:**

1. [**http://www.intuit.ru/department/hardware/csorg/9/2.html**](http://www.intuit.ru/department/hardware/csorg/9/2.html)

**---- 9. Лекция: Организация памяти вычислительной системы**

1. **Э.Танненбаум,Современные операционные системы, Питер 2002.1024 с.**
2. **Р.Столлинз Операционные системы. М.: Вильямз, 2002. – 600 с.**
3. **В.Г.Олифер, Н.А.Олифер Сетевые операционные системы. Питер, 2001.- 554 с.**