Содержание

Список рисунков

Список таблиц

«Видит око, да зуб неймет!»
(с) Народная пословица

1. ВВЕДЕНИЕ

С самых ранних дней существования вычислительных машин программисты-пользователи хотели иметь практически неограниченный объем быстродействующей памяти, но возможности технологии и проблемы высокой стоимости всегда ограничивали реально доступный объем. Что можно было сделать разработчикам вычислительных систем в таких случаях?? Ведь фундаментальное противоречие между «быстродействущий» и «большой объем» отменить в принципе нельзя, если брать в расчет стоимость аппаратуры.

Выход всегда был один – если нельзя сделать в реальности, то нужно попробовать создать иллюзию, что такая память существует. Но эта иллюзорная быстрая память, тем не менее, должна обеспечивать решение тех же самых задач для программиста, желающего иметь неограниченный объем быстродействующей памяти. И что самое главное – сделать его счастливым обладателем уникального компьютера, который позволит нашему программисту забыть (хотя бы на время) о купленных им иллюзиях.

Содержание этого пособия посвящается методам и технологии иллюзионистов от инженерного сообщества, разрабатывавших компьютеры во второй половине 20-го века и делающих это сейчас.

Но прежде чем пытаться создать любую иллюзию необходимо провести анализ реальных жизненных ситуаций у потенциальных клиентов и создать специальные технические средства, которые позволят исполнить задуманное с выгодой для клиента и собственного кармана.

Поговорим о конкретной ситуации пользователя, программа которого обрабатывает большой объем данных, в принципе не помещающийся в быстрой памяти ЦП, но в тоже время ему нужно обработать эту информацию значительно быстрее по сравнению с доступом к данным во внешнюю медленную память из каждой команды ЦП. Попробуем сравнить характер доступа программы пользователя к данным с работой студента в университетской библиотеке во время подготовки реферата.

ПРИМЕР ДОСТУПА К ДАННЫМ:

Шаг 1. **Студент:** *Найти книги из рекомендованного списка литературы по каталогу.*

Программа: *Сгенерировать адреса расположения данных.*

Шаг 2. **Студент:** *Заказать и получить все книги из списка литературы.*

Программа: *Выкачать данные из внешней памяти во внутреннюю регистровую память ЦП.*

Ограничения для обоих: Сколько книг может поднять студент и сколько данных может поместиться в регистровой памяти ЦП.

Шаг 3. **Студент:** *Сесть за стол и начать работать над первой частью реферата, используя первую книгу из списка, читая страница за страницей и часто возвращаясь назад для уяснения деталей..*

Программа: *Начать обрабатывать данные из первого массива, делая частые циклы обращения к последовательным элементам массива.*

Шаг 4. **Студент:** *Повторить шаг 3 для всех источников литературы из списка и закончить первую часть реферата.*

Программа: *Повторить шаг 3 для всех массивов данных, перегружая их во внутреннюю память по мере необходимости.*

Шаг 5. **Студент:** *Перейти к работе над второй частью реферата и вернуться к использованию первой книги из списка, читая страница за страницей и часто возвращаясь назад для уяснения деталей.*

Программа: *Вернуться к обработке данных из первого массива, делая частые циклы обращения к последовательным элементам массива.*

Шаг 6. **Студент:** *Повторить шаг 5 для всех источников литературы из списка и закончить вторую часть реферата.*

Программа: *Повторить шаг 5 для всех массивов данных, перегружая их во внутреннюю память по мере необходимости.*

Дальнейшие действия будут повторением уже описанных с учетом различий для последующих разделов реферата и массивов данных.

Беглый анализ действий студента и программы показывает, что имеется два вида локальности доступа к источникам литературы и данным.

Локальность доступа во времени (temporal locality) – когда книга или массив данных имеют тенденцию быть востребованы для повторного чтения или доступа в течении относительно короткого промежутка времени.

Локальность доступа в пространстве (spatial locality) – когда страницы книги или элементы массива данных имеют тенденцию быть прочитаны последовательно один за другим в предсказуемой манере.

Что это значит с точки жизненных ситуаций пользователя программ? Говоря по-простому, ему абсолютно не нужны все массивы данных для обработки сразу, но в каждом массиве он может обрабатывать соседние элементы либо последовательно, либо в коротких циклах. Иногда ему нужно перегрузить заново уже удаленный массив, чтобы продолжить обработку отдельных элементов, которые по какой-либо причине не были обработаны в первых проходах. Характер действий программы и характеристика доступа к данным в большинстве случаев предсказуемы и имеют последовательный характер.

Локальность доступа в программах происходит из самой природы программ. К примеру, большинство программ содержат циклы, доступ как к командам так и элементам данных будет повторяться во времени, создавая локальность доступа во времени. С другой стороны, программа исполняется всегда последовательно, и мы почти всегда можем предугадать, какие команды будут выбираться для исполнения в ближайших тактах ЦП. Похожая ситуация возникает и с данными при обработке массивов; следущий в последовательности элемент массива с большой вероятностью будет обрабатываться в последующей группе инструкций. Это создает локальность доступа в пространстве.

Как можно использовать данную ситуацию для создания иллюзии «быстрой памяти большого объема», имея в распоряжении быструю и крайне дорогую регистровую память ЦП, более дешевые элементы статической памяти SRAM и еще более дешевые, но медленные элементы динамической памяти DRAM? Можно также включить в рассмотрение дисковую память большого объема (массовую память) и почти исчезнувшую память на магнитных лентах, которые на несколько порядков величины медленнее кремниевых аналогов.

Ответом и главным принципом создания иллюзии «быстрая память большого объема» является иерархическая организация памяти компьютерной системы, в которой с помощью специальных аппаратных

средств соединены все вышеуказанные типы памяти, образуя конфигурацию, вполне приемлемую по цене и быстродействию. Причем, эти специальные аппаратные средства для объединения различных видов памяти в единую иерархическую структуру должны быть абсолютно прозрачны (невидимы) для пользователя программ, который даже и не подозревает, что его откровенно «надувают» иллюзионисты.

Как выглядят с точки зрения стоимости и скорости доступа технические средства, которые можно было бы использовать для создания иллюзии «быстрой памяти большого объема»? В **Табл. 1** приведены типовые данные по технологии памяти на 2004 год.

Табл. 1 . Параметры различных типов памяти.

Технология памяти	**Типичное время доступа**	**Стоимость за Мегабайт**
Register RAM Регистровая память	250 – 1000 пикосекунд	Очень дорого, на порядок величины по сравнению со статической памятью
SRAM Статическая память	0.5 – 5 наносекунд	$4 - $10 , достаточно дорого
DRAM Динамическая память	50 – 70 наносекунд	$0.1 - $0.2 вполне доступно
Магнитные диски	5 – 20 миллисекунд	$0.0005-$0.002 очень недорого
Магнитные ленты	Секунды и минуты	Совсем дешево

Теперь поговорим об инженерных делах и внутреннем устройстве иллюзиона, а также о том, как снизить затраты на его построение.

Главный принцип – иерархическое построение, которое предусматривает предоставление пользователю максимально возможного объема памяти, типичного для наиболее дешевой технологии памяти с одновременным представлением скорости доступа, которую предусматривают наиболее быстрые и дорогие технологии памяти.

Рис. 1 иллюстрирует иерархию памяти с отображением базовых характеристик памяти на каждом уровне и ширины обмена между уровнями.

Можно заметить, что объем доступной памяти в системе возрастает с номером уровня иерархии (уровень иерархии регистров ЦП=0) с одновременным падением стоимости.

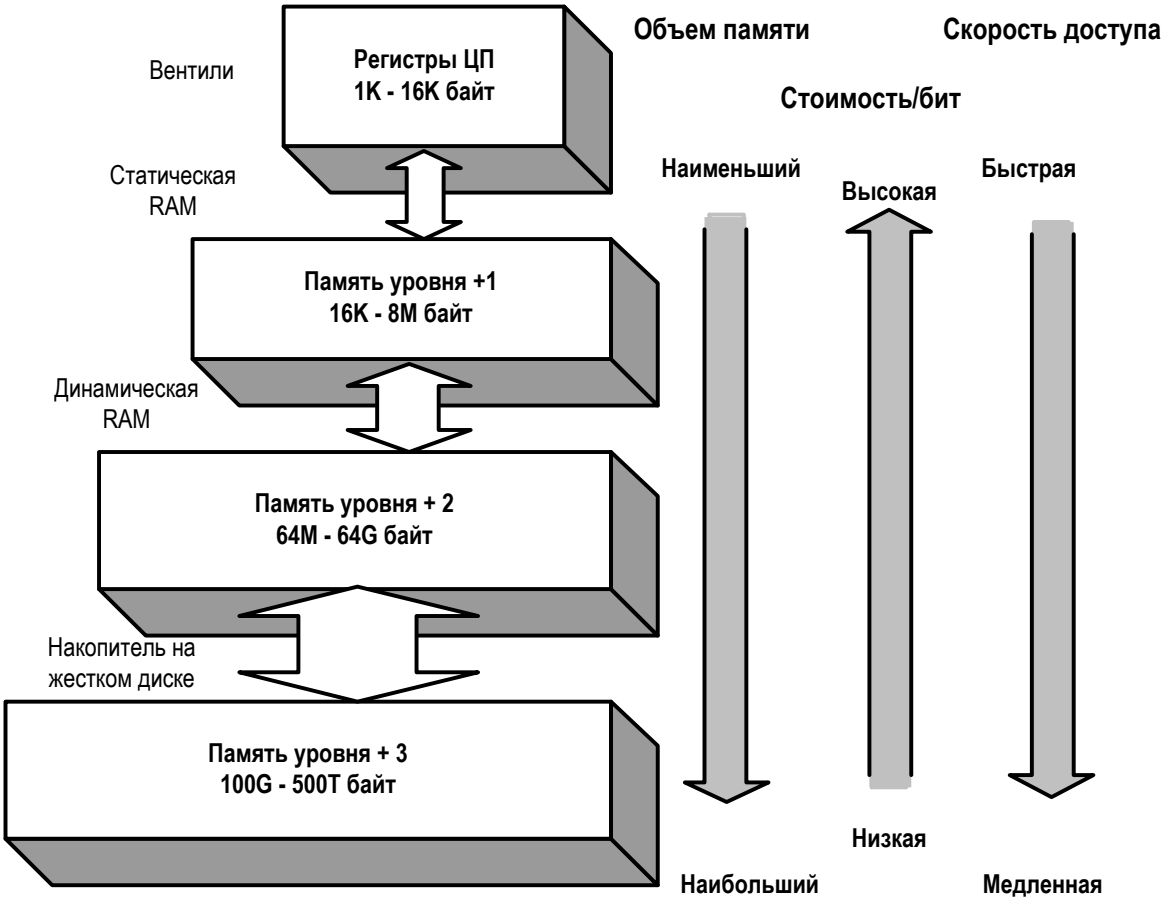


Рис. 1 . Иерархия памяти в компьютерной системе.

Следует отметить наиболее важные особенности иерархической структуры памяти:

- Требование унифицированности доступа к элементу данных на всех уровнях, программа пользователя использует один и тот же унифицированный формат адреса команды или данных для доступа к памяти на любом уровне иерархии;
- Каждый вышестоящий уровень памяти может являться подмножеством нижестоящего уровня с точки зрения содержащихся данных (**multilevel inclusion**), т.е. данные только копируются при обмене между уровнями:
  - Возможен также вариант исполнения с уникальными или эксклюзивными копиями блока данных на каждом уровне памяти (**multilevel exclusion**), когда блок данных полностью перемещается на другой уровень иерархии, освобождая пространство для других данных

- Обмен данными между уровнями памяти полностью скрыт от программы пользователя специальными аппаратными средствами и служебными утилитами операционной системы;
- Обмен данными между уровнями может иметь разную ширину, которая варьируется от нескольких байт до К байт (машинное слово 4-8 байт, блок или строка 16, 32, 64 или 128 байт, страница или сегмент в 4К байт или больше).

Рассмотрим в деталях, что происходит при исполнении программы пользователя на машине с иерархической организацией памяти.

Каждый раз, когда исполняемая команда ЦП нуждается в данных из памяти, она обращается к ближайшему быстрому уровню памяти предполагая, что эти данные там окажутся. Если предположение оправдывается, то ЦП получает данные с минимально возможной задержкой и происходит *«попадание»* (**Hit**) в ближайший к ЦП уровень иерархии памяти. Если ожидание не оправдывается и необходимых данных на ближайшем уровне памяти нет, то происходит *«промах»* (**Miss**). Тогда поиск необходимых данных производится на следующем (+1) уровне иерархии памяти, что приводит к значительному простою ЦП в ожидании необходимых данных.

Мы можем характеризовать любой уровень иерархии памяти, кроме последнего, специальным *«коэффициентом попаданий»* (**Hit rate**), который показывает, какое количество обращений к данному уровню памяти было успешным, когда данные были найдены и считаны ЦП или предшествующим уровнем иерархии памяти. Коэффициент попаданий связан с *«коэффициентом промахов»* (**Miss rate**) следующим выражением:

Miss_Rate= 1 – Hit_Rate

«Коэффициент промахов» = (1 - «Коэффициент попаданий»)

И часто гораздо удобнее пользоваться вторым ввиду его относительно малой величин; предполагается, что система оптимизируется для минимизации промахов.

Другой характеристикой, которая связана с аппаратными возможностями является *«Время успешного доступа»* (**Hit time)**, которое определяется аппаратной задержкой в определении самого факта попадания или промаха, к которой добавляется задержка в считывании самих данных, если данные найдены на этом уровне иерархии памяти.

Для характеристики последствий промахов используется *«Время промаха»* (**Miss penalty**), которое включает в себя время, которое

необходимо для поиска и доставки необходимого блока данных с последующего уровня иерархии памяти.

Пожалуй это самые важные моменты в организации иерархической системы памяти, которая является критическим элементом в любой вычислительной системе, как с точки зрения производительности, так и стоимости. Причем от качества исполнения иллюзиона «Быстродействующая память большого объема» зависит производительность и стоимость многих миллионов больших и малых вычислительных систем.

Построение реальных подсистем памяти для компьютеров следующих поколений является сложной задачей, над которой бьются лучшие инженерные умы и крупные электронные компании всего мира. В данном учебном пособии описаны самые базовые принципы построения иерархических систем памяти в надежде, что в вашей последующей профессиональной деятельности вы сможете значительно глубже изучить все детали и даже разрабатывать аппаратное и программные средства для вычислительных систем следующего поколения.

В разделе 2 будут описаны принципы построения *кэш-памяти* (**Cache memory**), которая является ближайшим уровнем памяти по отношению к ЦП. Уровень детализации примеров исполнения кэш-памяти является вполне достаточным для понимания работы большинства современных ЦП. Три основных способа реализации включают кэши прямого отображения, полностью ассоциативные и частично-ассоциативные кэши. Рассмотрены детали построения каждого типа кэшей с оценкой аппаратных затрат.

В разделе 3 основное внимание будет уделено оценке производительности и улучшению параметров кэш-памяти различной организации. Рассматривается многоуровневая кэш-память как один из типичных вариантов исполнения иерархической структуры памяти в современных микропроцессорах. Приводятся способы оценки быстродействия и уровня аппаратных затрат.

В разделе 4 рассмотрена концепция виртуальной памяти, которая является основой построения практически всех современных компьютерных систем и включает в себя все уровни иерархии памяти.

В разделе 5 рассматривается многоуровневая организация памяти в вычислительной системе, которая включает интеграцию двухуровневой кэш-памяти в в общую систему виртуальной памяти. Рассматриваются механизмы трансляции виртуальных адресов в физические на всех уровнях системы памяти на примере ЦП Alpha 21264.

Раздел 6 предназначен для углубленного изучения реальных вариантов исполнения иерархии памяти в семействе процессоров ARM, Intel Pentium P4 и AMD Opteron, Intel Itanium и Эльбрус E2K. Его изучение является первым шагом на пути совершенствования вашего профессионализма.

Раздел 7 посвящен обзору уровня технологии и особенностям технического исполнения модулей памяти SRAM и DRAM, различных типов энергонезависимой памяти, используемых для реализации различных уровней иерархии памяти.

2. ПРИНЦИПЫ ПОСТРОЕНИЯ КЭШ-ПАМЯТИ: ТРИ ОСНОВНЫХ СПОСОБА РЕАЛИЗАЦИИ

2.1. Принцип построения кэш-памяти

В самом начале попробуем разобраться, что такое кэш-память (**Cache memory, $$-memory**). Слово является составным и первая половина явно заимствована из другого языка. «Кэш» является прямой трансляцией слова “Cache” из английского языка, которое означает «надежное место, куда можно спрятать или где можно хранить вещи». Одно небольшое условие – доступ туда должен быть быстрым и удобным. Житейский пример – внутренний карман пиджака, куда обычно прячут документы и ценности, если таковые имеются.

Есть маленькая проблема с прямой трансляцией англоязычных терминов, когда совершенно разные вещи могут звучать одинаково. Имеется ввиду слово “Cash”, которое ассоциируется с банкнотами. Так что просьба иметь ввиду такое созвучие, причем в рабочих материалах часто используется аналогичное сокращение, когда кэш-память просто обозначается как $$.

Теперь о самой кэш-памяти: так сложилось, что в профессиональном мире компьютерной техники под кэш-памятью понимают любой блок или модуль памяти, который использует принцип локальности обращений. Кэш-память содержит данные, которые являются подмножеством данных, содержащихся в памяти большего объема нижнего уровня иерархии, либо эксклюзивную копию этих данных, переданных с нижнего уровня иерархии памяти.

Наиболее популярным является определение кэш-памяти, как промежуточного уровня памяти между ЦП и основной памятью компьютерной системы. Хотя кэш-память иногда используется также на других уровнях иерархии памяти, в частности между основной и

дисковой памятью системы, при этом принципы функционирования практически не отличаются от кэш-памяти ЦП.

Впервые кэш-память была построена на исследовательских ЭВМ в начале 60-х и годов и отличалась очень высокой стоимостью, затем с развитием технологии производства устройств памяти стоимость стала снижаться и большинство коммерчески используемых машин также получили блоки кэш-памяти. В настоящее время практически все микропроцессоры используют кэш-память, которая является важным фактором повышения производительности.

Для экономии времени и пространства кэш-память принято называть просто кэш. Поэтому далее по тексту местами будет использоваться слово «кэш», имея ввиду кэш-память.

Рассмотрим самую простейшую версию кэш-памяти, которую назовем *кэш прямого отображения* (**Direct-Mapped Cache**).

На Рис. 2 представлена структура простейшего кэша на восемь строк, которая связана с основной памятью, имеющей в два раза больший объем. Содержание основной памяти должно быть отображено в кэш таким образом, чтобы у ЦП не было никаких сомнений, что он обращается в основную память (принцип унифицированности доступа). В приведенной иерархии памяти адрес байта данных основной памяти содержит всего 6 бит, причем 2 бита отводится на адресацию байта внутри 32-разрядного блока данных и 4 бита остается на адресацию блока из 32 бит. В данной структуре мы можем разместить в каждой строке кэша содержание одного из двух потенциально возможных блоков из основной памяти, так как мы должны использовать старшие биты для адресации блока данных унифицированным способом. Поэтому структура адреса данных может быть представлена в виде следущих полей:

- адрес байта в блоке (используется ЦП для выбора нужного байта);
- индекс (адрес) блока данных (адрес строки кэша);
- старшие биты адреса (все оставшиеся биты адреса ЦП, в нашем простейшем случае только 1 бит).

Таблица с раскладкой разрядных полей адреса является упрощенным вариантом для приведенной структуры, однако для основной памяти и кэша гораздо больших объемов поля будут те же самые, только число бит в каждом поле будет соответствовать максимальным размерам памяти обеих уровней.

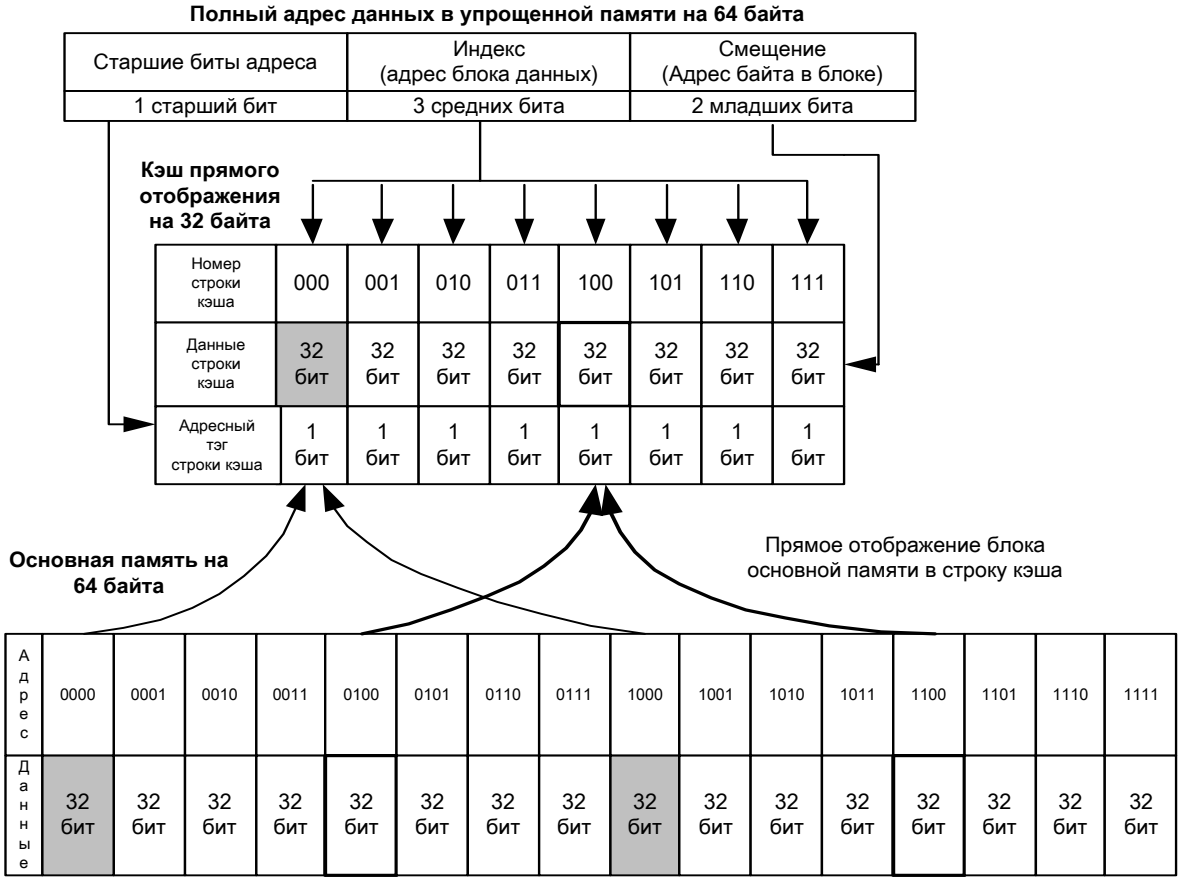


Рис. 2. Пример упрощенного кэша прямого отображения, имеющего вдвое меньший объем по сравнению с основной памятью.

Как видно из вышеуказанной иллюстрации, мы можем разместить в кэше только один из двух возможных блоков данных основной памяти. Допустим, в начальный момент исполнения программы все строки кэша пусты, можно также сказать, что они имеют недостоверные (**invalid**) данные.

Что должно происходить в данной иерархии памяти, если ЦП пытается прочитать данные из основной памяти с адресом «000000»? В этом случае мы имеем промах, так как данный блок данных отсутствует в кэше, и кэш-контроллер должен скопировать данный блок из основной памяти в кэш для обеспечения последующего доступа ЦП к этому блоку данных.

Кэш-контроллер имеет только одну альтернативу резервирования строки кэш-памяти – строка кэша с адресом «000», которая соответствует битам индекса в таблице полного адреса. При этом контроллер должен

отметить данную строку кэша как недостоверную и инициировать запрос в основную память по адресу «000000».

Как только данные будут получены, они записываются в строку кэша «000» и строка отмечается как достоверная. После этого ЦП может использовать любой из 4-х байтов, записанных в данную строку. Аналогичная последовательность действий будет, если ЦП попытается далее прочитать данные с адресом «010000», только при этом данные из основной памяти будут копироваться в строку кэша с номером «100», что соответствует значению индекса в поле адреса памяти. Теперь мы можем проанализировать аппаратную структуру кэш-памяти прямого отображения в **Табл. 2**, которая преставляет собой модуль памяти, в котором имеются два отделения: одно для собственно данных и второе для тэга (**Tag**), который содержит старшие биты адреса (в нашем упрощенном случае только один оставшийся старший бит из 6-разрядного адреса. В дополнение мы должны включить специальный флаг (бит) признака достоверности данных.

ПОЯСНЕНИЕ ИСПОЛЬЗОВАНИЯ ТЕРМИНА ТЭГ (TAG)

> В последующем материале будет интенсивно использоваться слово «Тэг» (**Tag**), которое на русский язык может быть переведено как ярлык, бирка, или карточка, которая прикреплена к какому-либо предмету и содержит краткую информацию о самом предмете, либо о его владельце. Использование слова «тэг» в контексте кэш-памяти совершенно аналогично, определенное значение тэга ассоциируется с блоком данных, загруженным в кэш-память, причем этим значением обычно является адрес расположения этого блока в основной памяти (унифицированный адрес). Тэг может также содержать некоторую дополнительную информацию о статусе данных (достоверность, изменение и так далее).

Табл. 2. Структура модуля памяти упрощенного кэша прямого отображения.

Номер строки кэша	**Флаг действи-тельности данных**	**Адресный тэг блока данных (старший бит адреса)**	**Данные Байт 3**	**Данные Байт 2**	**Данные Байт 1**	**Данные Байт 0**
0	< 1 бит >	< 1 бит > < 27 бит >	<8 бит>	<8 бит>	<8 бит>	<8 бит>
1	< 1 бит >	< 1 бит > < 27 бит >	<8 бит>	<8 бит>	<8 бит>	<8 бит>
...	....	.....	.....	.....	.....	.....
7	< 1 бит >	< 1 бит > < 27	<8 бит>	<8 бит>	<8 бит>	<8 бит>

		бит >				

Рассмотренный модуль памяти имеет 8 ячеек, называемых строками кэша и в каждой из которых хранится 32 бита данных и 2 бита использованы для хранения служебной информации, необходимой для обеспечения унифицированного доступа к данным со стороны ЦП. Попробуем посчитать процент дополнительной памяти: ((2 бита * 8 ячеек) / (8 бит * 4 байта * 8 ячеек)) * 100% = 6.25%.

Данная структура упрощенного кэша прямого отображения без сомнений может быть расширена для обработки 32-разрядных адресов, генерируемых ЦП. Для этого контроллер кэш-памяти должен обрабатывать следующий формат адреса:

Старшие биты адреса	**Индекс (адрес блока данных)**	**Смещение (Адрес байта в блоке)**
Биты адреса [31:5]	Биты адреса [4:2]	Биты адреса [1:0]

Как видно из таблицы формата, поле индекса и смещения остались неизменными, в тоже время в поле старших битов адреса мы имеем 27 бит, которые должны размещаться в адресном тэге каждой строки кэш-памяти прямого отображения вместо только одного бита в предыдущем случае.

Теперь рассмотрим необходимые действия для определения факта попадания или промаха при поиске необходимых ЦП данных в кэш-памяти прямого отображения.

1. ЦП генерирует полный 32-разрядный адрес необходимых данных и посылает этот адрес кэш-контроллеру;
2. Кэш-контроллер выбирает биты индекса и читает содержимое строки кэш-памяти (включая адресный тэг и данные), при этом данные не передаются ЦП.
3. Адресный тэг из кэш-памяти сравнивается со старшими 25 битами адреса, сгенерированного ЦП. Если значения совпадают, то произошло попадание и считанные данные должны быть переданы ЦП с использованием значения поля смещения для выбора нужного байта.
4. Если значение тэга и старших битов адреса от ЦП не совпадают, то произошел промах и данные должны быть загружены из основной памяти в данную строку кэш-памяти. Считанные при промахе данные из кэш-памяти аннулируются и ЦП должен остановиться и ждать, пока требуемые данные будут загружены в кэш из основной памяти. Кэш-контроллер получив запрос, должен считать требуемый блок данных

из основной памяти и послать их назад в кэш (при этом одновременно данные могут быть посланы ожидающему их ЦП).

5. Затем ЦП должен считать данные, на доступе к которым произошел промах и завершив исполнение этой команды, продолжить нормальную последовательность работы.

Таким образом имеется еще один важный элемент в структуре кэша – компаратор адресного тэга и старших битов адреса ЦП.

На **Рис. 3** представлена структура кэш-памяти прямого отображения объемом 4К байт, в которой имеется 256 строк (или блоков), каждый из которых содержит 16 байт данных. Младшие 4 бита адреса используются как смещение для выбора требуемого байта, последующие 8 бит используются в качестве индекса для выборки необходимой строки, и старшие 20 бит - для сравнения со значением адресного тэга.

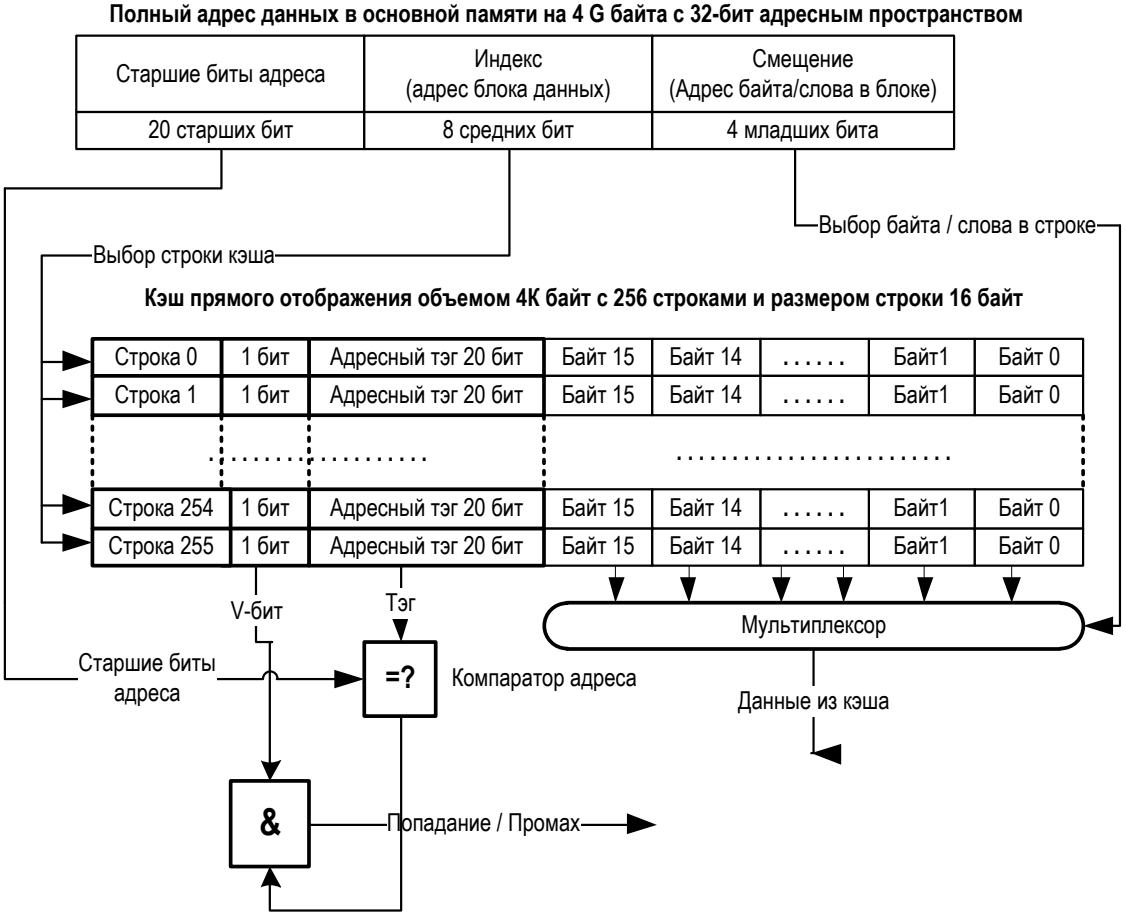


Рис. 3. Кэш-память прямого отображения объемом 4К байт с компаратором адресного тэга.

Попробуем посчитать процент дополнительного объема памяти для служебных разрядов тэга в этой структуре кэш-памяти, которая является типичной для встроенных микропроцессоров.

ПРИМЕР:

Нам необходимо хранить адресный тэг 20 бит, 1 бит для флага достоверности и еще 1 бит флага изменения данных. Флаг изменения необходимо устанавливать каждый раз, когда ЦП производит запись данных в кэш-память для последующей синхронизации содержания кэш-памяти и основной памяти (напоминание о том, что оба уровня памяти должны содержать одни и те же данные). Таким образом имеется 22 бита для каждой строки. Процент дополнительного объема для служебных бит в модуле памяти определяется:

((22 бит * 256 строк) / (8 бит * 16 * 256 строк)) * 100% = 17.1%

Как видно из расчета, расходы на дополнительную память возрастают практически втрое по сравнению с предыдущим расчетом, сделанным для простейшей структуры кэш-памяти и в основном за счет достаточно большого адресного тэга.

Рассмотрим другие численные характеристики кэш-памяти исходя из ее потребительских качеств, которые обеспечивают минимальное возможное среднее время доступа к памяти.

Ключевым является следущее выражение:

T_average = T_hit + MissRate*T_miss

ПРИМЕЧАНИЕ:

По правилам математики при расчете взвешенного среднего надо учитывать вероятность для каждого из слагаемых, т.е. T_{avg} = (1-MissRate)*T_{hit} + MissRate*T_{miss}. В данном случае при расчете мы не принимаем во внимание значение (1 – MissRate), предполагая, что во-первых вероятность промаха достаточно мала и значение выражение (1 – MissRate) будет близко к единице, а во-вторых, T_{hit} значительно меньше T_{miss}, что еще больше приближает значение множителя (1 – MissRate) к единице.

В данном выражении время может считаться как в физических единицах (наносекундах), так и в тактах ЦП, что обычно более удобно.

ПРИМЕР:

Если считать, что ЦП может определить факт попадания и считать данные из кэша за 2 такта (время доступа), коэффициент промахов

составляет 0.05 (или 5%), а задержка промаха составляет 40 тактов, то значение для среднего времени доступа (СВД)

СВД= 2 + 0.05 * 40 = 4 такта ЦП будет в среднем ожидать получения данных в случае обращения к памяти.

Время промаха обычно включает время доступа к более медленной основной памяти (**access time**) и время передачи блока данных (**transfer time**) для строки кэша (в нашем случае 16 байт данных, для которых может потребоваться 4 такта при ширине шины в 32-бита).

2.2. Три основных способа реализации кэш-памяти

Как уже отмечалось выше, кэш прямого отображения является наиболее простым способом реализации кэш-памяти. Платой за эту простоту является ограничение на выбор строки кэша, в которую могут быть отображены блоки памяти – местоположение данных в кэше полностью определяется значением битов адреса, попадающих в поле индекса. В примере на **Рис. 3** блоки данных памяти с адресами «0000» и «1000» могут быть размещены только в строке кэш-памяти «000», а блоки «0110» и «1110» - только в строке кэш-памяти «110». Очевидно что для повышения гибкости иерархии памяти хотелось бы размещать любой блок из основной памяти в любой строке кэш-памяти. Для достижения этой цели разработано два других способа построения кэш-памяти: полно-ассоциативная или просто ассоциативная кэш-память и частично-ассоциативная кэш-память.

Таким образом, можно перечислить все три способа построения кэша с краткой характеристикой каждого из них:

- Кэш прямого отображения (**Direct mapped Cache**), где каждый блок данных из памяти может быть отображен только в одну строку кэша с определенным номером;
- Полно-ассоциативный кэш (**Fully Associative Cache**), где каждый блок данных из памяти может быть отображен в любую из из строк кэша;
- Частично-ассоциативный кэш (**Set Associative Cache**), где каждый блок данных может быть отображен в строку на выбор из нескольких параллельных наборов строк кэша (обычно 2, 4, 8 или 16). Частично-ассоциативный кэш может рассматриваться как набор из нескольких (2, 4, 8 или 16) параллельных кэшей прямого отображения, объединенных общей логикой управления.

ПРИМЕЧАНИЕ: ЧТО ДОРОЖЕ В АППАРАТНОМ ИСПОЛНЕНИИ

Кэш прямого отображения является наиболее экономичным с точки зрения аппаратных затрат, так как он требует наличия только одного адресного компаратора. Наиболее дорогостоящим является полностью-ассоциативный кэш, в котором адресный компаратор требуется для каждой строки кэша. Частично-ассоциативный кэш является компромиссным решением, увеличивающим гибкость кэша прямого отображения частичной ассоциативностью между параллельными блоками кэшей прямого отображения.

На **Рис. 4** проиллюстрированы возможности отображения блоков данных из памяти в строки кэш-памяти всех трех типов.

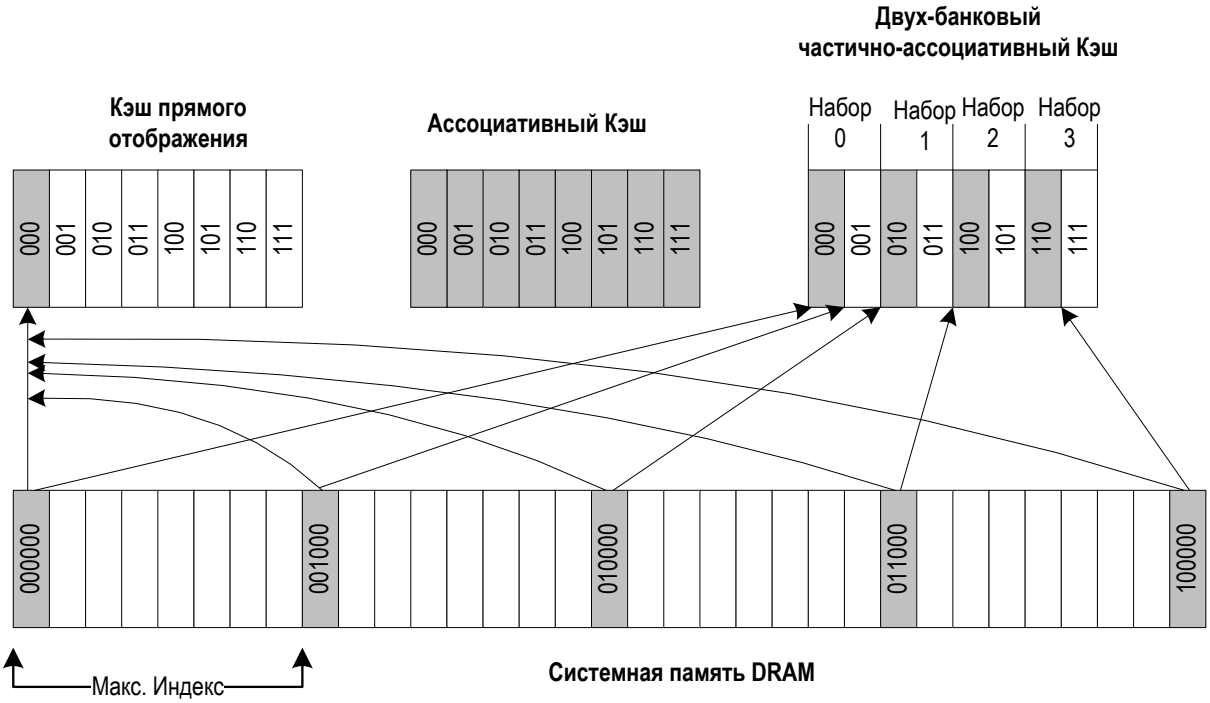


Рис. 4. Отображение блоков данных из основной памяти в кэш-память прямого отображения, ассоциативную кэш-память и частично-ассоциативную кэш-память.

Как видно из **Рис. 4**, ассоциативный кэш не имеет ограничений на отображение блоков данных из памяти в строки кэша, в отличие от кэша прямого отображения и частично-ассоциативного кэша. Казалось бы, это наилучший способ построения кэш-памяти и мы должны забыть про другие способы. Но здесь возникает проблема аппаратных затрат, которые значительно превышают уровень для других типов кэша. Кроме того, сложность аппаратуры снижает возможное быстродействие.

Рассмотрим структуру ассоциативной кэш-памяти, представленной на **Рис. 5**.

Она имеет такой же объем сегмента данных, как пример на **Рис. 3**, но другую структуру.

Сегмент тэгов				**Сегмент данных**				
Компаратор Адресного тэга с адресом ЦПУ (отдельный для каждой строки)	Адресный тэг блока данных (старшие биты адреса)	Флаги досто-верности и изменения данных V-флаг и D-флаг		Байт 3	Байт 2	Байт 1	Байт 0	Номер строки сегмента данных кэша
ЦПУ ?=Тэг 0	<30 бит>	<2 бит>		< 32 бит >				0
.....	.....	.....		.....				....
ЦПУ ?=Тэг 6	<30 бит>	<2 бит>		< 32 бит >				6
ЦПУ ?=Тэг 7	<30 бит>	<2 бит>		< 32 бит >				7

Старшие биты адреса <30 бит>	Смещение (Адрес байта в блоке) <2 бит>
Биты адреса [31:2]	*Биты адреса [1:0]*

Рис. 5. Структура ассоциативной кэш-памяти.

Сегмент данных абсолютно аналогичен кэш-памяти прямого отображения, при этом сегмент тэга существенно отличается и имеет тэг значительно больших размеров, который включает все биты адреса кроме смещения. Кроме этого, для каждой строки добавлены собственные 30-разрядные компараторы для сравнения с адресом, поступающим из ЦП. Два бита достоверности и изменения данных в строке аналогичны используемым в кэше прямого отображения.

Каким образом происходит доступ к данным в ассоциативном кэше? Алгоритм работы несколько отличается от кэша прямого отображения.

1. ЦП генерирует полный 32-разрядный адрес необходимых данных и посылает этот адрес кэш-контроллеру;
2. Кэш-контроллер сравнивает старшие 30 бит адреса со адресными тэгами всех строк кэша одновременно (для этого имеется отдельный компаратор адреса для каждой строки). Если значения с каким-либо из тэгов совпадают, то произошло попадание и кэш-контроллер определяет номер строки кэша, данные из которой должны быть считаны для ЦП.

3. Используя этот номер строки и поле смещения для выбора нужного байта, данные считываются из сегмента данных и должны быть переданы в ЦП.
4. Если значение тэгов во всех строках кэша и старших битов адреса от ЦП не совпадают, то произошел промах и данные должны быть загружены из основной памяти в одну из строк кэш-памяти. ЦП должен остановиться и ждать, пока требуемые данные будут загружены в кэш из основной памяти.
5. Затем ЦП должен считать данные, на доступе к которым произошел промах и завершив исполнение этой команды, продолжить нормальную последовательность работы.

Главными отличиями этого алгоритма от работы кэша прямого отображения являются:

- Не используется индекс для выборки конкретной строки, содержащей как данные, так и адресный тэг;
- Вместо этого строка кэш-памяти, содержащая необходимык данные, определяется путем одновременного сравнения адресных тэгов всех строк кэша с генерацией номера строки, где значение адресного тэга равно значению старших битов адреса ЦП. Поэтому имеется дополнительный шаг по определению факта попадания, результатом которого является генерация номера строки;
- При промахе нужно определить, в какую строку кэш-памяти можно будет загружать требуемые данные из основной памяти. В кэш-памяти прямого отображения строка для загрузки данных всегда однозначно определяется значением индекса в адресе, поступившем от ЦП. В то время, как в ассоциативном кэше все строки кэша равнозначны и нужен специальный алгоритм замещения данных для выбора строки, данные из которой могут быть замещены требуемыми данными из памяти.

На

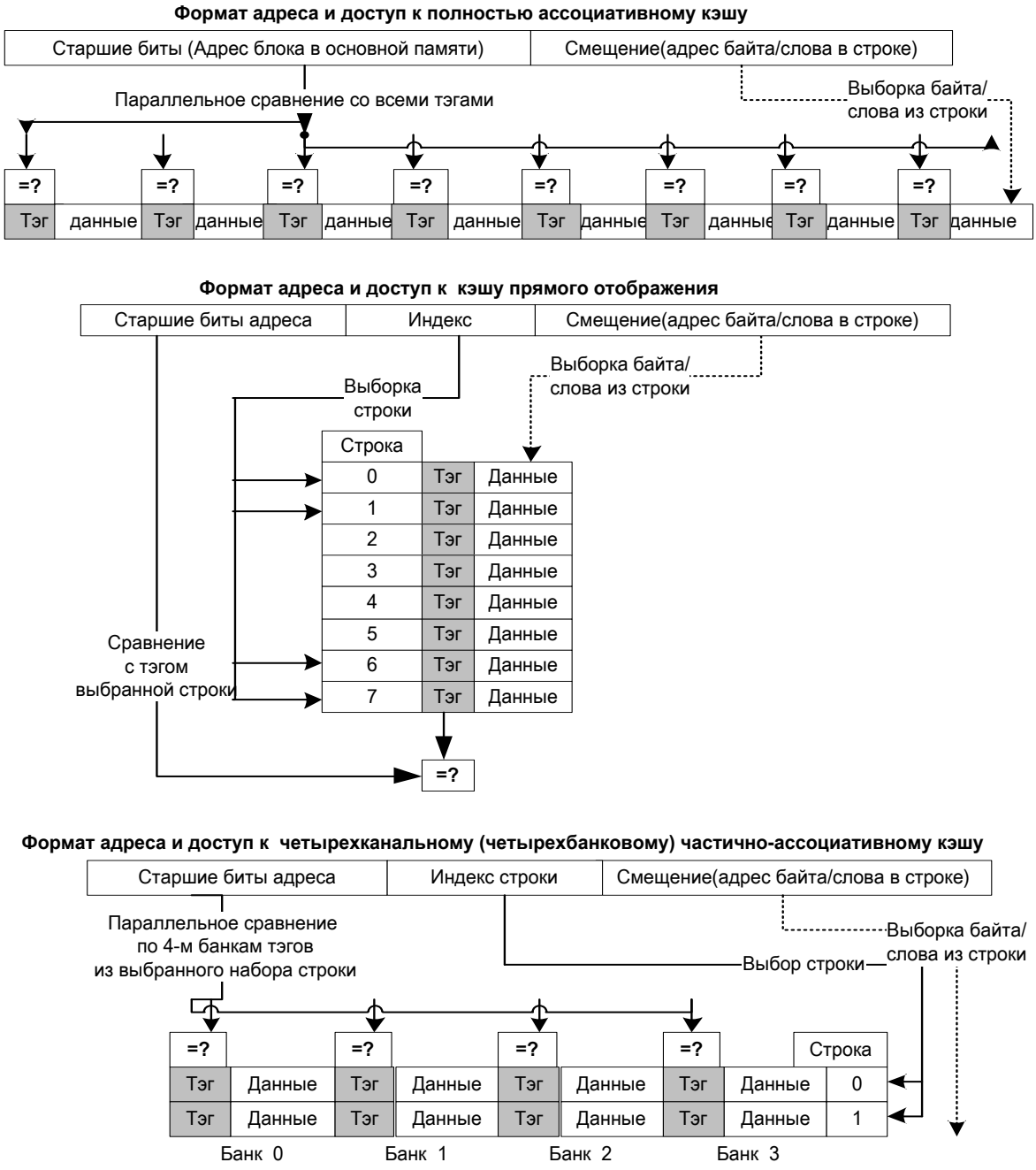


Рис. 6 проиллюстрированы отличия в доступе к ассоциативному кэшу в отличие от кэша прямого отображения. Если представить кэш-память как двумерный массив, то ассоциативный кэш выглядит как одна строка блоков, доступ к которым определяется сравнением полных адресных тэгов всех блоков. А кэш прямого отображения выглядит как столбец блоков, где доступ к отдельному блоку определяется индексом из поля адреса и не требует сравнения всех тэгов, а только одного тэга из блока, определенного значением индекса.

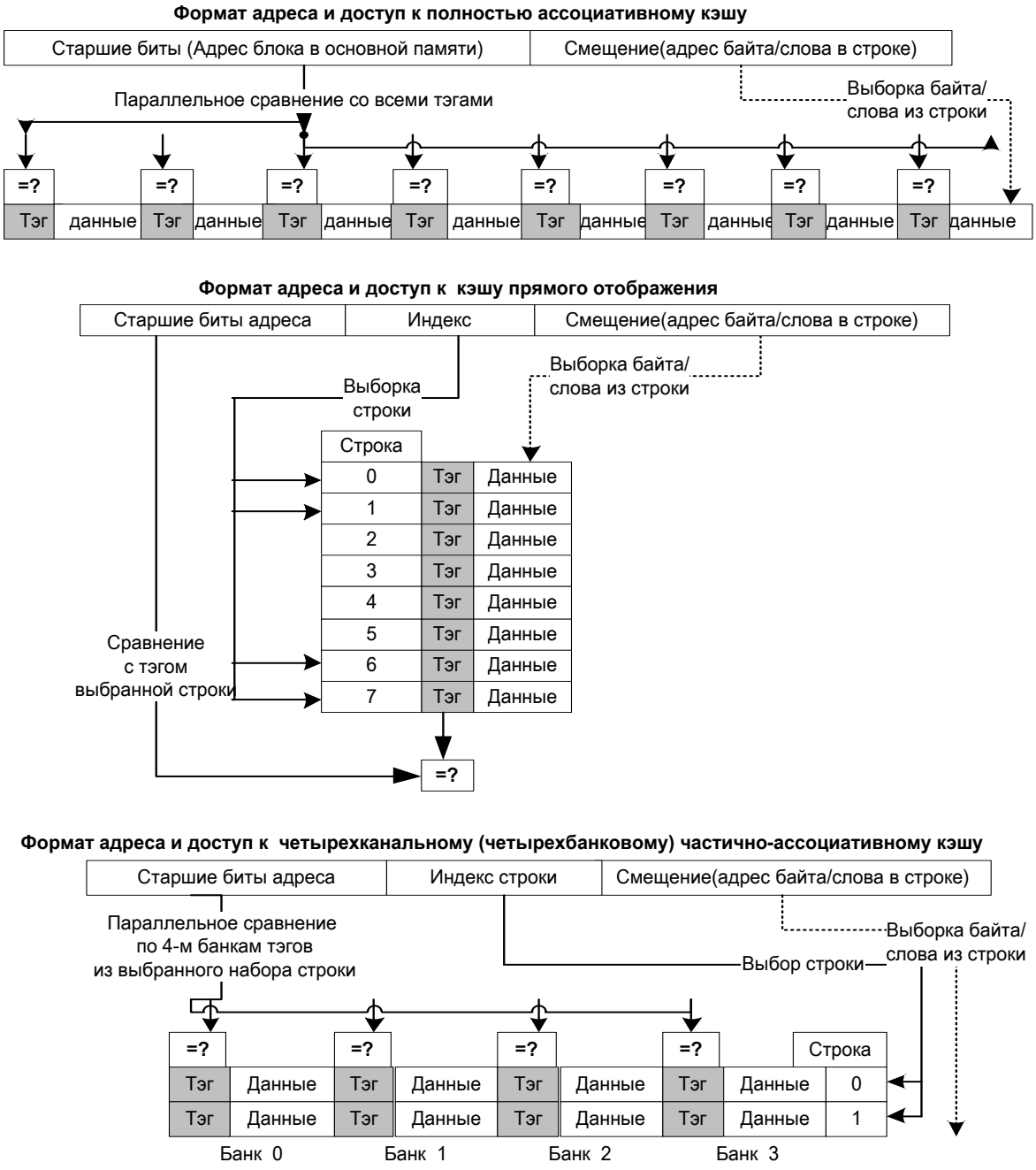


Рис. 6. Иллюстрация доступа к различным типам кэш-памяти на примере кэша с 8 блоками (строками.

В промежутке между ними находится частично-ассоциативный кэш, который может состоять из нескольких каналов (банков) кэш-памяти прямого отображения (2, 4, 8 и более). При этом доступ ко всем банкам этой памяти осуществляется аналогично кэшу прямого отображения с

одним отличием в параллельном сравнении адресных тэгов в каждом канале для определения банка, в котором находятся запрашиваемые ЦП данные. При это число компараторов в таком кэше равно числу банков, что многократно ниже, чем у полностью ассоциативного кэша.

Читатель, знакомый с программированием, может заметить аналогию доступа в кэш прямого отображение с доступом к элементам массива, и доступа в полностью ассоциативный кэш с доступом к элементам списка и доступа в частично-ассоциативный кэш с доступом к элементам хэша.

В англоязычной литературе используется понятие "N-way set associative cache" для определения структуры частично-ассоциативного кэша, при переводе на русский понятие «банк» и «канал» являются взаимозаменяемыми. В дальнейшем мы будем использовать как то, так и другое.

КРАТКО О ТИПАХ КЭША

1) Ассоциативный кэш: выборка блока производится только по значению полного адресного тэга (паралелльная или псевдо-параллельная проверка всех блоков). Поля индекса для таких кэшей не существует.

2) Кэш прямого отбражения: выборка строки (блока) по значению поля индекса, далее считанный из этой строки адресный тэг сравнивается с адресом, полученным от ЦП.

3) Частично-ассоциативный кэш с N банками или каналами: выборка всех банков одновременно производится по значению поля набора (аналогичен индексу), выбор нужного банка производится путем сравнения N адресных тэгов с адресом, полученным от ЦП. Далее по тексту будет использоваться понятие **резидентности** данных по отношению к тому или иному уровню иерархии памяти. Резидентность блока данных означает, что этот блок скопирован (перемещен) в кэш. Понятие резидентности также распространяется на блоки данных в других уровнях иерархии памяти.

2.3. Основные параметры кэш-памяти

Можно выделить следующие основные параметры, которые характеризуют кэш-память различных типов:

- Объем кэш-памяти (сегмент данных и сегмент адресных тэгов)
- Размер строки (блока) кэш-памяти
- Степень (уровень) ассоциативности кэш-памяти (Прямое отображение, ассоциативный, частично-ассоциативный)

- Алгоритм замещения данных (выбор строки для удаления) при необходимости загрузки новых данных в кэш из основной памяти (LRU, Random)
- Алгоритм записи данных в кэш-память из ЦП (кэш с обратной записью и кэш со сквозной записью)

Теперь рассмотрим как образом основные параметры влияют на производительность:

Объем кэш-памяти всегда влияет на общую производительность системы памяти по следующим причинам:

- При большем объеме больше данных может находиться внутри кэш-памяти и вероятность того, что нужные очередной команде данные резидентны в кэше, всегда будет выше (вероятность промахов уменьшается);
- Объем кэш-памяти может варьироваться от сотен байт до десятков Мегабайт в зависимости от области применения (ЦП для сотовых телефонов и рабочих станций, жесткий диск, сетевые контроллеры и файловые системы).

Кэш-память любого типа всегда содержит сегмент данных и сегмент тэгов с адресом и другими битами, отражающими статус соотвествующих строк сегмента данных. Обычно это биты достоверности (**Valid**) и изменения (**Dirty**) данных. Для краткости мы будем называть их **V**-битом и **D**-битом. В некоторых структурах могут быть дополнительные биты статуса. В примере, приведенном ниже, рассмотрены способы расчета требуемого объема памяти для сегмента тэгов.

ПРИМЕР РАСЧЕТА ОБЪЕМА И РАЗМЕРА СТРОКИ КЭША

Здесь действует правило: «Больше – Лучше», однако приемлемость аппаратных затрат всегда ограничивает пожелания. Поэтому будем считать, что объем сегмента данных максимально возможный в каждой конкретной ситуации. Можно говорить о цифрах в десятки килобайт для кэша первого уровня. При этом кэш не оперирует байтами или словами, все транзакции осуществляются на уровне строк/блоков данных, которые содержат некоторое количество байт. Размер строки кэша, как правило, определяется шириной транзакций с основной памятью и ее режимами работы. Например, для шины основной памяти шириной 32 бит и двойным темпом обмена данными, за один цикл

обмена кэш-контроллер может послать или получить 64 бита данных. В пакетном режиме обмена (**burst**) за 4 цикла можно получить 256 бит. Поэтому для 32-разрядных систем размеры строки кэша варьируются между 64 и 256 бит. Далее определяется число строк – обычно от 64 до 1024. В рассмотренном ранее примере кэша прямого отображения было 256 строк по 16 байт=128 бит каждая, что дает общий объем сегмента данных в 4К байт.

Сегмент тэгов содержит адресные тэги и другие служебные биты и как правило, объем этой части памяти никогда не указывается в спецификации для конечного пользователя. Но разработчикам это необходимо знать и типичный кэш с обратной записью имеет следующие поля:

- 1 бит достоверности данных (V-бит)

- 1 бит изменения данных (D-бит)

- Адресный тэг (бит) = $\log_2$ (Объем основной памяти) - $\log_2$ (Число строк кэша) – $\log_2$ (Размер строки кэша)

- Номер банка (бит) = $\log_2$ (Число банков или каналов) – в случае частично-ассоциативного кэша.

Таким образом, если объем основной памяти 1 Гигабайт, число строк кэша 256, размер строки 16 байт, то Адресный Тэг = (30 – 8 – 4) = 18 бит, добавив 2 бита достоверности и изменения, получаем 20 бит. Если кэш частично ассоциативный с 4 банками, то еще 2 бита, получив всего 22 бита. Эти два бита добавляются в адресный тэг из поля индекса, так как каждый банк 4-канального частично-ассоциативного кэша будет содержать только 64 строки и нам необходимо только 6 бит для индекса. Следует отметить что вышеприведенный расчет не является корректным для полностью ассоциативного кэша, где отсутствует поле индекса и число строк кэша не учитывается. Таким образом для ассоциативного кэша:

Адресный тэг= (Объем основной памяти) – log2 (Размер строки кэша) = 30 – 4 = 26 бит +2 = 28 бит

Рис. 3 и **Рис. 5** иллюстрируют вышеописанную методику расчета. Общий объем сегмента тэгов рассчитывается умножением размера тэга на число строк/блоков кэша 22 /28 бит *256 = 5632/7168 бит.

Степень ассоциативности кэша сильно влияет на быстродействие. Кэш прямого отображения является наиболее быстродействующим, так

как тест на попадание/промах производится одновременно с считыванием данных и в случае промаха считанные данные просто аннулируются. В ассоциативном и частично-ассоциативном кэшах данные появляются только после результата теста, что обуславливает дополнительную задержку. На **Рис. 7** представлена структура частично-ассоциативного кэша с 2-мя банками кэшей прямого отображения, где данные выбираются по результатам теста на попадание/промах в двух банках. Четырех-банковые (канальные) и восьми-канальные частично-ассоцитивные кэши будут иметь аналогичную структуру с большим числом модулей памяти и компараторов. Повышение степени ассоциативности уменьшает конфликты размещения блоков данных, возникающие в кэше прямого отображения. При этом блоки данных из памяти имеющих одинаковые биты в поле индекса в адресе могут быть размещены в различных банках кэша.

Когда кэш уже заполнен данными из основной памяти, а программа продолжает обращаться к новым данным, то необходимо освободить место в кэше, замещая старые данные новыми из памяти и копируя старые данные назад в основную память, если они были изменены. Алгоритмы замещения данных являются актуальными только для полно-ассоциативного и частично-ассоциативного кэша. В случае кэша прямого отображения выбора нет, для каждого значения индекса имеется только один блок, который должен быть замещен новыми данными и в этом случае содержание этого блока кэша просто объявляется недостоверным, либо в дополнение копируется в основную память, если данные были изменены.

В случае частичной ассоциативности имеется несколько банков, в которых имеются по одному блоку/строке, соотвествующих данному значению индекса. Здесь необходимо сделать выбор, в каком банке выполнить замещение, что и является целью алгоритмов замещения. Обычно используются два основных алгоритма:

- Давно не используемый блок (**Least Recently Used – LRU**). Специальные счетчики используются для определения числа тактов, когда к данному блоку не было обращения от ЦП. Замещается блок с максимальным значением такого счетчика. Другим, более экономным, способом реализации является использование матрицы или пирамиды битов отношения (j-й канал старше k-го).
- Случайное замещение (используется генератор случайных чисел для определения номера банка).

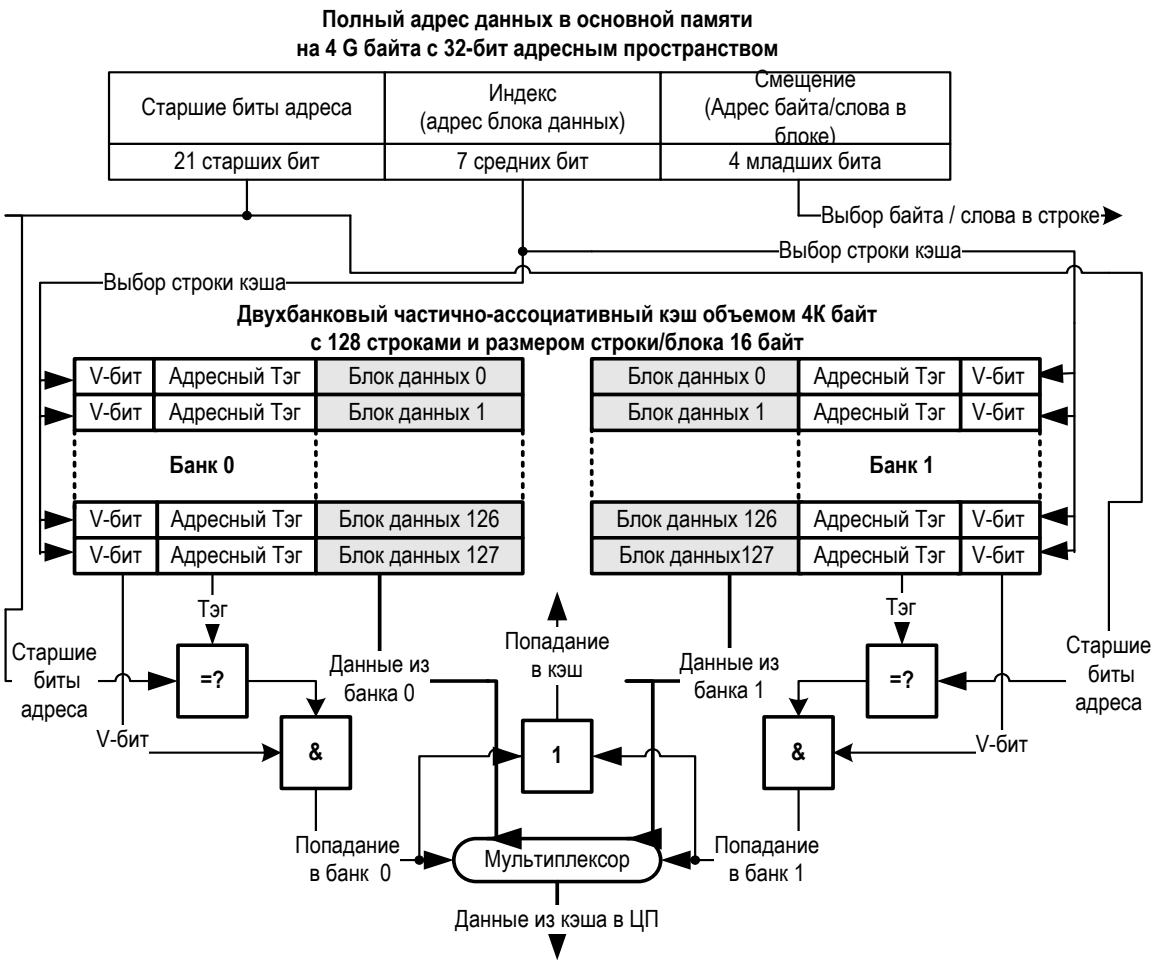


Рис. 7. Структура частично-ассоциативного кэша с двумя банками памяти.

Кроме этого, следует учитывать, что строки кэша с измененными данными должны быть скопированы назад в основную память, обычно это делается при замещении блока. Это уже относится к алгоритмам записи данных в кэш-память.

Алгоритмы записи данных в кэш включают две основные группы:

- Обратная запись (**Write Back**), когда данные пишутся только в соответствующую строку кэша и дублируются в основную память только при замещении данных;
- Сквозная запись (**Write Trough**), когда данные пишутся одновременно в кэш и основную память с использованием FIFO-буфера записи. Буфер записи данных позволяет ЦП записывать последовательные блоки данных, не заботясь о фактическом моменте их появления в основной памяти.

Каждый алгоритм имеет свои достоинства и недостатки. Например, при сквозной записи промахи чтения не блокируют запись блоков в память из-за наличия паралелльного тракта доступа к памяти с буфером записи. В свою очередь, при использовании кэша с обратной записью, последовательность операций записи в один и тот же адрес кэша не вызывает избыточных операций записи в основную память и уменьшает нагрузку на интерфейс основной памяти и общую шину системы.

В следующем разделе будет проанализирована модель поведения кэш-памяти и рассмотрены способы повышения производительности, включая использование многоуровневого кэша.

3. АНАЛИЗ МОДЕЛИ ПОВЕДЕНИЯ И ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ КЭШ-ПАМЯТИ

3.1. Анализ составляющих повышения производительности

Возвратимся немного назад и вспомним, что производительность кэш-памяти характеризует следущее выражение:

T_average = T_hit + Miss_Rate * T_miss

Где *T_average* – Среднее Время Доступа к памяти, *T_hit* – Время доступа при попадании, *Miss_Rate* – Доля промахов при обращении к памяти и *T_miss* – Время выборки строки данных из основной памяти при промахе.

В данном выражении время может считаться как в физических единицах (наносекундах), так в тактах ЦП, что обычно более удобно. Естественно, что нашей целью является уменьшение значения среднего времени доступа до физически возможного предела.

Анализируя данное выражение можно перечислить возможные способы оптимизации:

- Уменьшение времени доступа при попадании за счет специальной быстрой логики и использования скоростной статической памяти;
- Уменьшение значения доли промахов до нескольких процентов от общего числа обращений к памяти;
- Уменьшение задержки на выборку строки данных со следующего уровня иерархии при промахе до физически возможного минимума для конкретного типа используемой памяти DRAM с помощью различных методов ускорения доступа к основной памяти.

Первый и третий способ базируются на особенностях используемой статической памяти для кэша и динамической памяти для основной

системной памяти, а также качестве проектирования контроллеров кэша и основной памяти которые могут включать в себя специальные блоки, ускоряющие выборку данных из кэш-памяти и основной памяти. Второй способ в значительной степени зависит от организации и объема кэша.

3.2. Уменьшение времени доступа при попадании

Уменьшение времени доступа при попадании может быть достигнуто следующими способами:

- Использование простых кэшей небольшого объема.
- Использование аппаратного предсказания доступа к блокам и строкам, которое приближает скорость работы частично-ассоциативных кэшей к простым кэшам прямого отображения.
- Исключение преобразования виртуального адреса на выборке строки кэша по индексу.
- Конвейеризация доступа к кэш-памяти.
- Использование кэша команд, содержащего трассы исполнения команд и не требующих повторных выборок строк при наличии команд перехода.

Растущая тактовая частота ЦП вынуждает разработчиков к уменьшению времени доступа при попадании, которое может быть достигнуто за счет упрощения и уменьшения размеров кэшей первого уровня L1. Кэш прямого отображения является идеальным эталоном, в котором выборка данных и сравнение тэга происходят одновременно, уменьшая время доступа до минимума. Сравнивая размеры кэшей первого уровня L1 у последних поколений микропроцессоров, можно заметить, что роста объема практически нет, в некоторых случаях кэш даже уменьшается (Pentium 3 и Pentium 4).

Табл. 3 иллюстрирует зависимость скорости доступа при попадании в кэш-память от объема и степени ассоциативности кэш-памяти.

Табл. 3. Зависимость скорости доступа в кэш-память от объема и степени ассоциативности

Степень ассоциативности и объем кэш-памяти	**Кэш прямого отображ-я**	**2-банковый частично ассоциатив. кэш**	**4-банковый частично ассоциатив. кэш**	**Полностью ассоциативн. кэш**
4 КВ	3	5.2	5.8	5.8
8 КВ	3	5.4	6	6
16 КВ	4	6	6.3	7
64 КВ	6	7	8.2	8.5

256 KB	9	11.5	11.8	14

Использование предсказания доступа реализовано в качестве специального блока предсказания банка (**way prediction**) и строки (**line prediction**) для следующей выборки в 2-банковом частично-ассоциативном кэше команд процессора Альфа 21264. Использование предсказания банка и строки следующей выборки в частично-ассоциативном кэше использует принцип локальности обращений к строке данных, и если профиль доступа достаточно регулярный (например при выборке команд), то можно предсказать, будет ли блок выборки команд ЦП обращаться в тот же самый банк и блок. Доступ может также производиться в другой банк к тому или другому блоку, что может быть заранее определено при предварительной выборке данных из основной памяти. Для этого в поле тэга добавляются биты предсказания банка и строки. При начальной загрузке каждой строки в кэш эти биты устанавливаются соотвествующим образом с использованием значения программного счетчика предварительной выборки команд. Далее при каждом доступе в случае ошибки предсказания, биты предсказания корректируются специальным аппаратным блоком, использующим текущее значение программного счетчика исполнения команд. Таким образом, аппаратура предсказания в кэше команд процессора Альфа 21264 является обучаемой и адаптируется к динамическим переходам, возникающим при исполнении программы. На **Рис. 8** приведена структура кэша команд процессора Альфа 21264, использующего аппаратное предсказание.

Использование предсказания позволяет повысить скорость работы частично-ассоциативного кэша до скорости кэша прямого отображения и уменьшает задержку попадания практически в два раза. Цена ошибки предсказания достаточно невелика, и к тому же число таких ошибок минимизируется за счет постоянной коррекции в случае программных циклов.

Аналогичные аппаратные решения, возможно, также применить к кэшу данных, только в этом случае профиль доступа будет зависеть от поведения программы и его можно предсказать заранее с гораздо меньшей точностью.

В большинстве микропроцессоров используется система виртуальной памяти, когда ЦП генерирует виртуальные адреса в процессе исполнения программы. В тоже время кэш-память содержит данные из физического пространства адресов, так как отображает реальные блоки памяти. Соответственно адресные тэги тоже содержат физический адрес.

Для того, чтобы произвести сравнение, формально нужно преобразовать виртуальный адрес в физический, что занимает дополнительное время. В деталях данный процесс будет рассмотрен в следующем разделе, поэтому здесь мы ограничимся только тем, что поле индекса для выборки строки всегда должно состоять из адресных битов, являющихся общими для виртуального и физического адресов. Тогда преобразование старших битов адреса из виртуального в физический (трансляция адреса) происходит одновременно с выборкой строки кэша, и на момент сравнения тэгов мы имеем физический адрес, оттранслированный из виртуального. Однако этих битов может не хватить для полного индекса и тогда, в индекс необходимо включить часть битов из виртуального адреса.

Рис. 8. Кэш команд процессора Альфа 21264, использующий предсказание доступа

Возможно также использование виртуальных кэшей, когда тэги содержат виртуальные адреса, но в этом случае возникает проблема адресных клонов, когда две программы, имеющие данные в разных

областях физической памяти генерируют одинаковые виртуальные адреса. Для того, чтобы различить данные, необходимо вводить в адресный тэг специальный идентификатор процесса (**Process-Identifier-Tag**).

Другой проблемой, возникающей при использовании виртуальных КЭШей, является проблема возникновения синонимов или дубликатов, когда разные программы обращаются к одним и тем же блокам физической памяти, которые копируются в разные строки кэша. Используются как аппаратные, так и программные решения для ликвидации синонимов. Это могут быть дополнительные проверки или специальная раскраска страниц средствами операционной системы.

Конвейеризация доступа в кэш в принципе не позволяет уменьшить задержку отдельно взятого обращения, но увеличивает пропускную способность в целом за счет одновременной обработки нескольких обращений на разных ступенях конвейера. Например в семействе процессоров Pentium задержка доступа к кэш-памяти выросла от 1 такта до 4, в тоже время пропускная способность увеличилась за счет роста тактовой частоты и выдачи данных из кэша на каждый такт работы конвейера доступа.

Кэши с трассированием последовательности команд характерны для канала выборки команд и могут иметь гораздо более сложные функции отображения блоков основной памяти с программой в кэш команд, так как учитывают историю переходов и динамически корректируют выборку данных в строку кэша в соотвествии с состоявшимися или несостоявшимися переходами. Для кэшей команд с размером строки в 64 и более байт, статическая функция отображения (типичная для кэша данных) может порождать проблемы неэффективности использования простанства, когда только одна или две команды из строки будут использованы, а далее произойдет переход в другую строку. Поэтому специальные аппаратные блоки поддерживают динамическую функцию отображения. Типичным примером является реализация кэша команд в семействе процессоров Pentium 4 (**NetBurst microarchitecture**), рассмотренный в разделе 6.

3.3. Уменьшение доли промахов

Уменьшение доли промахов базируется на выборе типа и объема кэш-памяти, а также на некоторых особенностях ее структурного построения. Поэтому попробуем проанализировать, какие типы промахов существуют и как они могут влиять на поведение кэш-памяти.

Для кэш-памяти характерны следующие виды промахов:

- **Неизбежные промахи** (**compulsory misses**), когда программа (данные) впервые запускается (считываются) ЦП;
- **Промахи нехватки объема** кэш-памяти (**capacity misses**), когда строка с необходимыми данными была уже загружена в кэш и впоследствии удалена алгоритмом замещения для освобождения места данным, которые обрабатывались позже;
- **Промахи конфликтов отображения** (**conflict misses**), когда аналогично предыдущему случаю необходимая строка данных была удалена из-за того, что новые данные из памяти отображались на ту же самую строку (характерны для кэш-памяти прямого отображения)
- **Промахи потери когерентности данных (coherency misses)** возможны в многопроцессорных системах, когда копия одного и того же блока данных из основной памяти располагается в двух и более кэшах параллельно работающих ЦП. Если один из них производит запись в этот блок, то все другие копии должны быть объявлены недостоверными и попытка доступа к ним вызовет промах. Протоколы и аппаратные средства поддержки когерентности данных в иерархии памяти будут рассмотрены в пособии по многопроцессорным системам.

3.3.1. Неизбежные промахи

Такие промахи также называют промахами холодного старта (**cold start misses**) и промахами первичного доступа (**first reference misses**). Они возникают в самом начале исполнения программы и далее по мере того, как новые данные (команды) из основной памяти должны попасть в кэш для последующей обработки в ЦП. Эти промахи являются неизбежными, так как доступ к данным областям памяти не производился до момента исполнения программы. Число таких промахов соотвествует числу уникальных ссылок к блокам памяти, соотвествующих по размеру строке кэша. Для этого мы должны удалить из полного адреса биты смещения и далее сравнить все оставшиеся биты адресных ссылок.

ПРИМЕР ПОСЛЕДОВАТЕЛЬНОСТИ АДРЕСОВ ДОСТУПА В ПАМЯТЬ

Рассмотрим последовательность 32-разрядных адресов в Табл. 4, генерируемых ЦП для обращения в основную память за данными, подразумевая, что имеется кэш-память с размером строки в 16 байт. 32-разрядный адрес может быть закодирован с использованием восьми шестнадцатиричных цифр, которые будут использоваться для удобства в дальнейшем.

Табл. 4. Пример: последовательность обращений к памяти

Полный адрес байта (последняя цифра – смещение байта в блоке/строке)	**Адрес блока (смещение удалено)**	**Номер уникальной ссылки**	**Примечание**
0x12345678	0x1234567	1	Первая ссылка на 1
0x12345688	0x1234568	2	Первая ссылка на 2
0x02020212	0x0202021	3	Первая ссылка на 3
0x02020214	0x0202021	3	Вторая ссылка на 3
0x12345671	0x1234567	1	Вторая ссылка на 1
0x11111111	0x1111111	4	Первая ссылка на 4
0x1234567f	0x1234567	1	Третья ссылка на 1
0x11111121	0x1111112	5	Первая ссылка на 5
0x11111119	0x1111111	4	Вторая ссылка на 4
0x11111101	0x1111110	6	Первая ссылка на 6

Как видно из таблицы Табл. 4, имеется 6 уникальных ссылок в последовательности 10 обращений к памяти, которые вызовут 6 неизбежных промахов, по одному на каждую уникальную ссылку.

Как видно из примера, бороться с неизбежными промахами непосредственно в конструкции кэша невозможно, они следуют из природы программы. Но существуют способы уменьшения числа неизбежных промахов за счет предварительной выборки (подкачки) данных в кэш из основной памяти, инициированных либо аппаратурой кэш-контроллера, либо специальными командами ЦП, добавленными в код при компиляции программы. Более детально это рассматривается в разделе 3.4.

3.3.2. Промахи нехватки объема

Если кэш-память не может вместить все блоки данных, необходимых при исполнении программы, то кэш-контроллер будет вынужден удалять некоторые блоки данных из кэш-памяти для размещения последующих необходимых блоков данных, несмотря на то, что они могут понадобиться вновь (например, при длинных циклах). Когда программа обращается к некоторому блоку данных во второй раз, а он был уже удален из кэша, происходит промах нехватки объема и этот блок должен быть загружен в кэш во второй раз.

Для того, чтобы подсчитать число промахов нехватки объема, нужно использовать полно-ассоциативный кэш ограниченного объема, в котором не бывает конфликтов отображения. Последовательность адресов обращения к памяти подается на такой кэш и общее число промахов кэша будет включать в себя как неизбежные, так и промахи нехватки объема. Так как мы можем легко подсчитать все неизбежные промахи из последовательности уникальных адресных ссылок, то промахи нехватки объема подсчитываются по формуле:

#Capacity_Misses = Total #misses - #Compulsory_Misses

Где *#Capacity_Misses* - число промахов нехватки объема, *Total #misses* - общее число промахов в кэше, *#Compulsory_Misses* - число неизбежных промахов

Рассмотрим пример простого ассоциативного кэша объемом 32-байт, который имеет только один банк памяти с двумя блоками (строками) по 16 байт каждая.

ПРИМЕР ДОСТУПА В АССОЦИАТИВНЫЙ КЭШ

Наш простейший полно-ассоциативный кэш имеет структуру строки с двумя блоками (см.

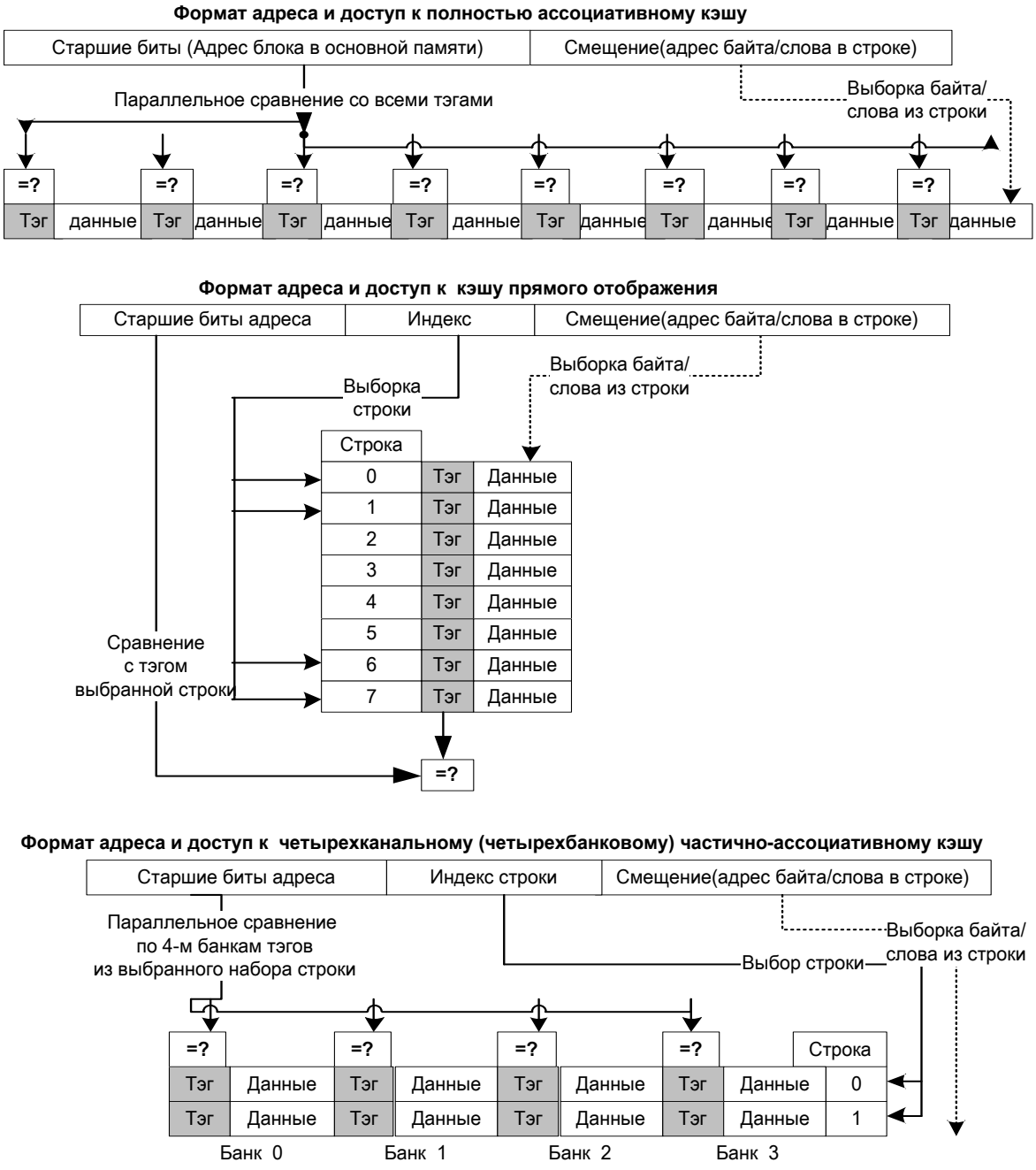


Рис. 6). Для объяснения принципа нам не нужно знать содержание строки, а только значение адресного тэга. Используем в качестве входного потока последовательность адресов из предыдущего примера. При этом адресный тэг будет содержать только 28 бит, так как 4 бита смещения используются для адресации байта в блоке. Считаем что изначально оба блока кэша содержат недостоверные тэги и данные.

Invalid	Invalid

Если мы будем посылать в этот кэш последовательность адресов из предыдущей таблицы, то мы сможем определить число неизбежных промахов и число промахов из-за нехватки объема. Первые две ссылки вызывают два неизбежных промаха и в обоих блоках будут размещены данные со следующими адресами:

0x1234567	0x1234568

Следующая ссылка с адресом 0x0202021 вызовет промах и замещение блока адресным тэгом 0x1234567 по алгоритму LRU (наиболее старого):

0x0202021	0x1234568

Четвертая ссылка по адресу 0x0202021 приведет к попаданию и замещение в кэш-памяти не производится.

0x0202021	0x1234568

Пятая ссылка по адресу 0x1234567 приведет к промаху, так как блок с данным значением тэга был уже удален. Согласно алгоритма LRU блок с тэгом 0x1234568 подлежит замене данными с тэгом 0x1234567:

0x0202021	**0x1234567**

Шестая ссылка к адресу 0x1111111 вызовет промах и замену по LRU блока 0x0202021:

0x1111111	0x1234567

Седьмая ссылка по адресу 0x1234567 вызовет попадание и замещение в кэш-памяти не производится.

0x1111111	0x1234567

Восьмая ссылка по адресу 0x1111112 вызовет промах и замещение по LRU блока с адресом 0x1234567:

0x1111111	**0x1111112**

Девятая ссылка по адресу 0x1111111 вызовет попадание, и замещение в кэш-памяти не производится.

0x1111111	0x1111112

Десятая ссылка по адресу 0x1111110 вызовет промах и замещение по LRU с адресом 0x1111111:

0x1111110	0x1111112

Таким образом полно-ассоциативный кэш с двумя блоками вызовет 7 промахов, в которые включены как неизбежные промахи, так и промахи нехватки объема. Зная из предыдущего примера, что неизбежных промахов в данной последовательности адресов было 6, то на промахи нехватки объема остается всего один.

3.3.3. Промахи конфликтов отображения

Отметим вначале, что полно-ассоциативный кэш в наименьшей степени страдает от промахов и обладает наибольшей гибкостью по сравнению с кэшами других типов. Разумеется, за соотвествующую плату в виде компараторов на каждый блок и сложной схемы определения факта попадания / промаха (**Рис. 5**). Кэш-память таких же размеров, но с меньшей степенью ассоциативности, будет иметь дополнительное число промахов, вызванных проблемами отображения, которых не существует в полно-ассоциативных кэшах. В наибольшей степени эта проблема представлена в КЭШах прямого отображении.

Число этих промахов может быть вычислено с помощью формулы:

#Conflict_Misses = Total #misses - #Compulsory_Misses - #Capacity_Misses

Где *#Capacity_Misses* - число промахов нехватки объема, *Total #misses* - общее число промахов в кэше, *#Compulsory_Misses* - число неизбежных промахов, *#Conflict_Misses* - число промахов конфликтов отображения

ПРИМЕР ДОСТУПА В КЭШ ПРЯМОГО ОТОБРАЖЕНИЯ

Теперь в качестве примера используем простейший кэш прямого отображения, имеющий структуру столбца с двумя блоками и имеющий тот же объем в 32 байта. Для объяснения принципа нам не нужно знать содержание строки, а только значение адресного тэга. Используем в качестве входного потока ту же самую последовательность адресов как и в двух предыдущих примерах. При этом адресный тэг будет содержать уже 27 бит против 28 бит, у полно-ассоциативного, так как один бит используется в качестве индекса для выбора блока (строки) и определяет отображение. 4 бита смещения также используются для адресации байта в блоке. Считаем что изначально оба блока кэша содержат недействительные тэги и данные.

Строка 0
Строка 1

Первые две ссылки вызывают два неизбежных промаха и в обоих блоках будут размещены данные со следующими адресами:

0x1234568
0x1234567

При этом размещение будет произведено в соответствии со значением индексного бита, который является самым младшим битом последней цифры адреса xxxxxx8= «100 **0**» и попадает строку 0 и xxxxxx7= «011 **1**» попадает в строку 1.

Третья ссылка по адресу 0x0202021 приведет к промаху и вызовет замещение строки 1, так как оба блока данных с адресами 0x1234567 и 0x0202021 могут быть отображены только в эту строку из-за значения бита индекса в младшей цифре адреса xxxxxx7 = «011 **1**» и xxxxxx1 = «000 **1**» равного единице в обоих случаях.

0x1234568
0x0202021

Четвертая ссылка 0x0202021 вызовет попадание и содержание кэша останется неизменным.

0x1234568
0x0202021

Пятая ссылка 0x1234567 приведет к промаху и вызовет замещение строки 1 с тэгом 0x0202021:

0x1234568
0x1234567

Шестая ссылка 0x1111111приведет к промаху и вызовет замещение строки 1 по значению бита индекса:

0x1234568
0x1111111

Седьмая ссылка по адресу 0x1234567 приведет к новому промаху и вызовет замещение той же самой строки 1.

0x1234568
0x1234567

Восьмая ссылка по адресу 0x1111112 снова приведет к промаху и вызовет замещение строки 0 с адресом 0x1234568:

0x1111112

0x1234567

Девятая ссылка по адресу 0x1111111 приведет к промаху и вызовет замещение строки 1 с адресом 0x1234567.

0x1111112
0x1111111

Десятая ссылка по адресу 0x1111110 вызовет промах и замещение строки 0 с адресом 0x1111112:

0x1111110
0x1111111

Таким образом общее число промахов в кэше прямого отображения объемом в 32 байта будет равняться 9, тогда согласно вышеприведенной формуле вычтем 6 неизбежных промахов и 1 промах нехватки объема, полученные в предыдущих примерах. Результатом будут 2 промаха, вызванные конфликтами отображения.

3.4. Способы уменьшения доли промахов при обращении в кэш

Можно перечислить следующие общепринятые способы уменьшения доли промахов, которые можно отнести к специальным усовершенствованиям архитектуры:

- Увеличение объема кэш-памяти
- Увеличение размера блока (строки) кэша;
- Увеличение степени ассоциативности кэша;
- Использование дополнительного **кэша замещения** (**victim cache**);
- Использование **псевдо-ассоциативного кэша** (**pseudo-associative cache**);
- Использование **аппаратной предварительной подкачки** данных (**hardware prefetching**);
- **Реорганизация кода** компилятором для повышения локальности доступа
- Использование **предварительной подкачки под управлением компилятора** (**compiler generated prefetching**).
- Использование вспомогательного кэша замещения

3.4.1. Увеличение объема, размера блока и степени ассоциативности

Увеличение объема кэша всегда дает положительный результат в снижении доли промахов, таблица **Табл. 5** иллюстрирует значительное

снижение доли промахов при росте объема кэш-памяти. Однако за это приходится платить значительным ростом аппаратных затрат, которые растут пропорционально с объемом кэша.

Увеличение размера блока (строки) кэша может дать положительный эффект за счет использования большей локальности доступа в пространстве (вероятность попаданий в строку с 64 байтами всегда выше чем в строку с 32 байтами). Однако в случае промаха объем данных, которые нужно передать из кэша и записать в кэш возрастает пропорционально росту размера блока. И хотя доля промахов будет уменьшаться, общее время доступа может возрастать. Эффект от увеличения размера строки кэша зависит от поведения программы и может быть как положительным, так и отрицательным. Второе бывает на программах с нерегулярными адресами (например, компилятор). Кроме того, увеличение размера блока увеличивает вероятность конфликтов потери когерентности данных в многопроцессорной системе.

ПРИМЕР РАСЧЕТА СРЕДНЕГО ВРЕМЕНИ ДОСТУПА ПРИ УВЕЛИЧЕНИИ СТРОКИ КЭША

Возьмем кэш-память с размером строки 32 байта и следущими параметрами:

T_hit = 1 такт, Miss_Rate = 7% = 0.07 , T_miss = 10 тактов, среднее время доступа рассчитывается по формуле

$T\_average = T\_hit + Miss\_Rate * T\_miss = 1 + 0.07 * 10 = 1.7$ такта

Теперь если мы увеличим размер строки кэша в два раза до 64 байт, то величина задержки при промахе возрастет из-за большего объема данных которые необходимо передать в кэш при промахе. Если считать, что за один такт мы можем передать 4 байта, то величина задержки при промахе T_miss возрастет на 8 тактов, необходимых для передачи дополнительных 32 байт. Допустим также, что доля промахов уменьшиласть на 2% и составляет теперь 5%. Тогда среднее время доступа будет

$T\_average = T\_hit + Miss\_Rate * T\_miss = 1 + 0.05 * 18 = 1.9$ такта

Таким образом мы имеем ситуацию, когда доля промахов уменьшается, но каждый промах стоит дороже с точки зрения потерь времени на передачу информации и общее время доступа может возрастать, что не является желательным. Традиционным способом решения этой проблемы является **пересылка требуемого слова данных первым** и продолжение обработки без ожидания размещения в кэше всех остальных данных замещающей строки.

Увеличение степени ассоциативности кэша может уменьшить число промахов, вызванных конфликтами отображения. По мере роста степени

ассоциативности кэша (2-канальный, 4-канальный, 8-канальный, 16-канальный и т.д. до полно-ассоциативного), она будет оказывать все меньшее и меньшее влияние на уменьшение доли промахов до достижения уровня, когда доля промахов уже не меняется с увеличением степени ассоциативности. **Табл. 5** иллюстрирует изменение доли промахов с изменением степени ассоциативности и объема КЭШа.

Табл. 5. Изменение доли промахов в зависимости от степени ассоциативности и объема кэш-памяти

Степень ассоциативности и объем кэш-памяти	**Частично ассоциативный 2-канальный кэш**	**Частично ассоциативный 4-канальный кэш**	**Частично ассоциативный 8-канальный кэш**
16 KB	5.18 %	4.67 %	4.39 %
64 KB	1.88 %	1.54 %	1.39%
256 KB	1.15 %	1.13 %	1.12 %

Побочным эффектом повышения степени ассоциативности кэша является рост сложности аппаратуры для сравнения адресных тэгов, что может привести к увеличению времени доступа при попадании **T_hit**. Это увеличение может привести к резкому росту среднего времени доступа. Поэтому для всех разработчиков частично- и полно-ассоциативной кэш-памяти поддержание минимально возможного времени доступа при попадании с увеличением числа банков (степени ассоциативности) является приоритетной задачей.

3.4.2. Использование вспомогательного кэша замещения

Использование вспомогательного кэша замещения (**victim cache**) предусматривает добавление к основной кэш-памяти дополнительного буфера, куда посылаются блоки, замещенные новыми блоками данных в основной кэш-памяти. Кэш замещения является небольшим (на несколько блоков/строк) кэшем, размещенным между основным кэшем и следующим уровнем иерархии памяти, его основной задачей является уменьшить потери от промахов конфликтов отображения. На **Рис. 9** приведена структура кэша прямого отображения с дополнительным кэшем замещения.

Как видно из приведенной структуры, кэш замещения получает данные только из основного кэша в момент замещения одной из строк. С точки зрения управления, он ничем не отличается от кэша прямого отображения, только имеет дополнительные биты в тэге для значения

индекса, которые используются для теста попадания и возврата данных назад в основной кэш в случае факта попадания. Вначале всегда производится тест на попадание в основном кэше и в случае промаха, затем проверяется кэш замещения. Если имеется попадание в кэш замещения, то данные считываются ЦП, а сама строка возвращается в основной кэш. В этом случае потери на дополнительный тест попадания соизмеримы со временем Thit для кэша без буфера замещения, так что значение T_hit для кэша с буфером замещения примерно удваивается. В случае промаха, блок данных из памяти должен быть загружен в основной кэш, а замещенная строка скопирована в кэш замещения. Очень часто для кэша замещения отводятся несколько дополнительных строк основного кэша и вместо отдельного кэша замещения имеется только буфер замещения с номерами (указателями) строк, которые должны быть замещены. Такая схема позволяет избежать копирования данных и сводится только к манипулированию указателями на строки КЭШа.

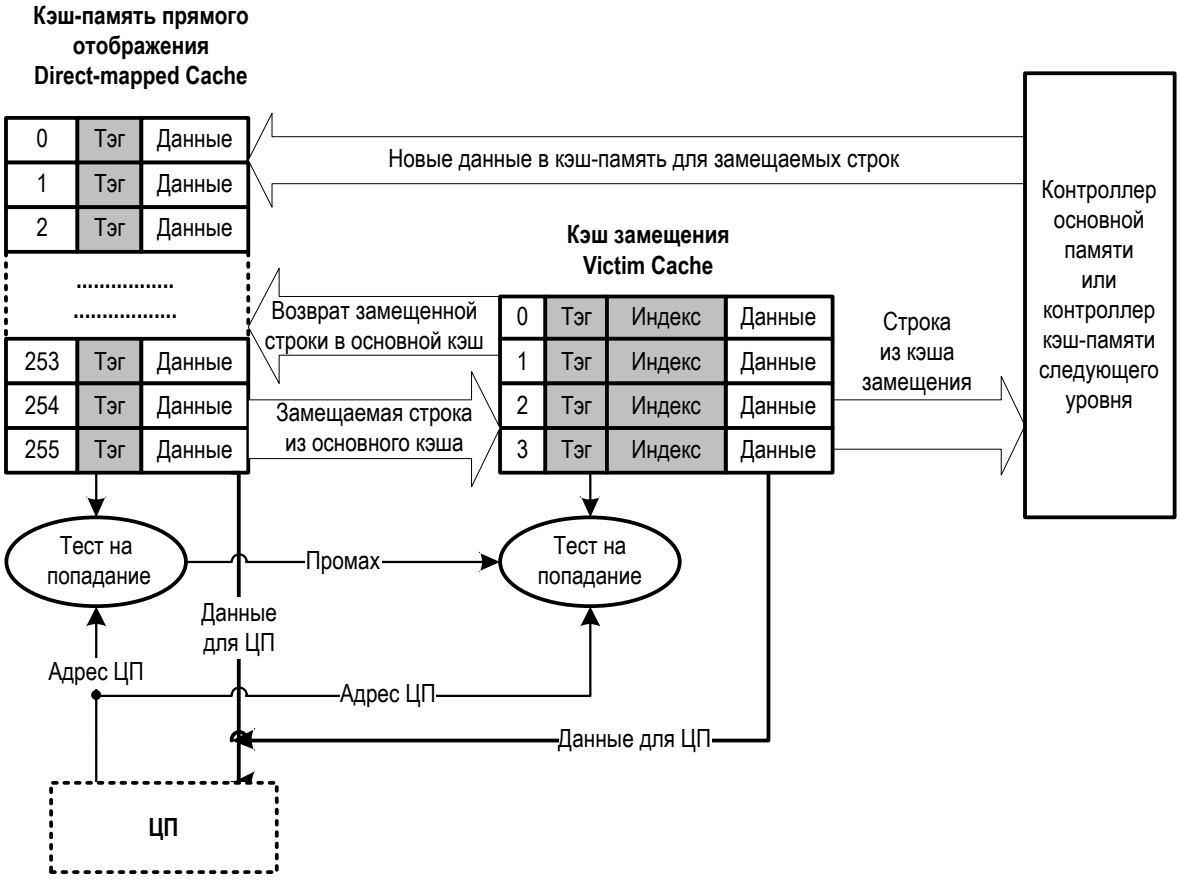


Рис. 9. Использование кэша замещения для уменьшения конфликтов в кэше прямого отображения.

3.4.3. Использование псевдо-ассоциативного кэша

Использование псевдо-ассоциативного кэша тоже является одной из возможностей уменьшить число конфликтов отображения, сохранив простоту и быстродействие кэш-памяти прямого отображения. Наиболее простым способом является разделение кэша прямого отображения на две одинаковые половины с уменьшением поля индекса на один бит. Фактически мы имеем два последовательно проверяемых на попадание кэша прямого отображения. При тесте на попадание вначале тестируется первая половина и в случае промаха – вторая половина. Это увеличивает задержку попадания T_hit обычно на один такт , так как как надо провести только дополнительное сравнение уже считанных тэгов с обеих половин кэша. В принципе вышеописанных подход можно расширить и разделить кэш на 4 части и проводить тест на попадание последовательно начиная первой части. Такая конструкция называется вертикально-ассоциативным кэшем (**column-associative cache**) и имеет один серьезный недостаток: число тактов при доступе может варьироваться от одного до нескольких в зависимости от числа столбцов в таком кэше. Эта особенность не позволяет использовать такие кэши на первом уровне L1 вместе с ЦП. Более приемлемым вариантом является использование вертикально-ассоциативных кэшей на втором уровне L2, когда они не соединены непосредственно с ЦП. Время доступа T_hit в в вертикально-ассоциативном кэше минимально и близко к кэшу прямого отображения, в тоже время доля попаданий такая же, как в частично-ассоциативном кэше с 2-мя или 4-мя банками.

Другим типом псевдо-ассоциативного кэша является кэш с несимметричным отображением, который строится на базе частично-ассоциативного кэша с двумя банками, в котором отдельные банки кэша используют разную функцию отображения. Такой кэш имеет время доступа частично-ассоциативного кэша с 2-мя банками и долю попаданий частично-ассоциативного кэша с 4-мя банками. Это уменьшает конфликты отображения и улучшает производительность. Структуры псевдо-ассоциативных кэшей проиллюстрированы на **Рис. 10**.

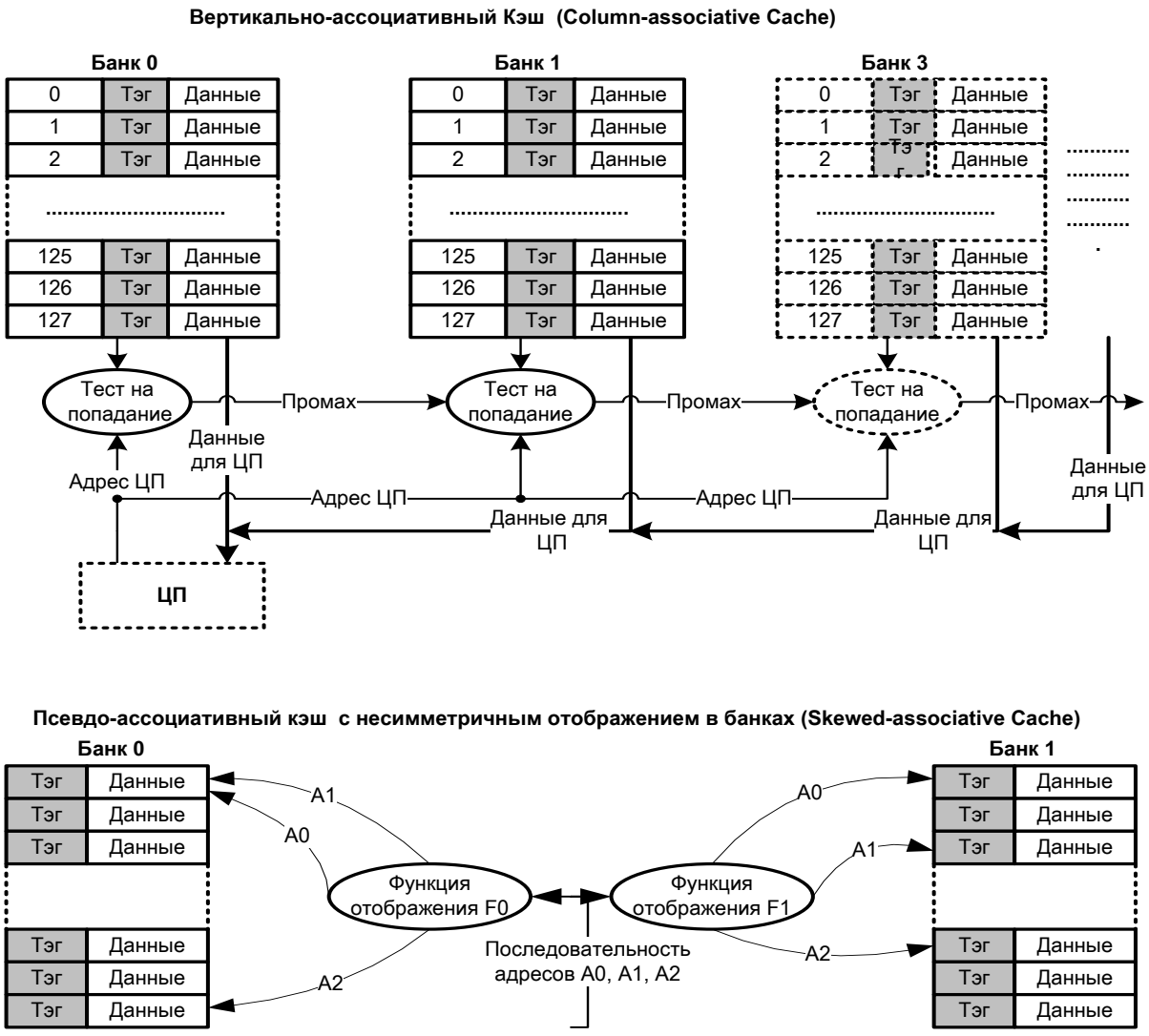


Рис. 10. Структуры псевдо-ассоциативных КЭШей

3.4.4. Использование аппаратной предварительной подкачки данных

Использование аппаратной предварительной подкачки дает возможность уменьшить число промахов как в кэше команд, так и в кэше данных. Предварительная подкачка является единственным способом уменьшения числа неизбежных промахов, при этом она, как правило, должна быть поддержана специальными аппаратными средствами в ЦП. Одной из используемых идей является предварительная подкачка данных из последовательных адресов в надежде, что ЦП использует данные последовательно и запросит вскоре эти данные. Например, при

последовательном исполнении команд (без переходов), так оно и происходит с выборкой команд из основной памяти. В этом случае число неизбежных промахов может быть серьезно уменьшено, так же как и доля промахов при обращении в кэш.

Реализуется аппаратная предварительная подкачка с использованием так называемых потоковых буферов и стратегии выборки одновременно двух или более последовательных блоков из памяти при промахе. При этом последующие блоки остаются в потоковых буферах, не попадая в основной кэш. Проверка содержания буферов проводится в случае промаха и приведена на рисунке. Если требуемые данные находятся в буфере, содержание которого проверяется на факт попадания в случае промаха в основном кэше. Если в потоковом буфере происходит попадание, то данные из него посылаются в ЦП, а сам блок данных копируется в основной кэш, освобождая место для последующих циклов предварительной подкачки.

Рис. 11 Схема обращения к потоковым буферам

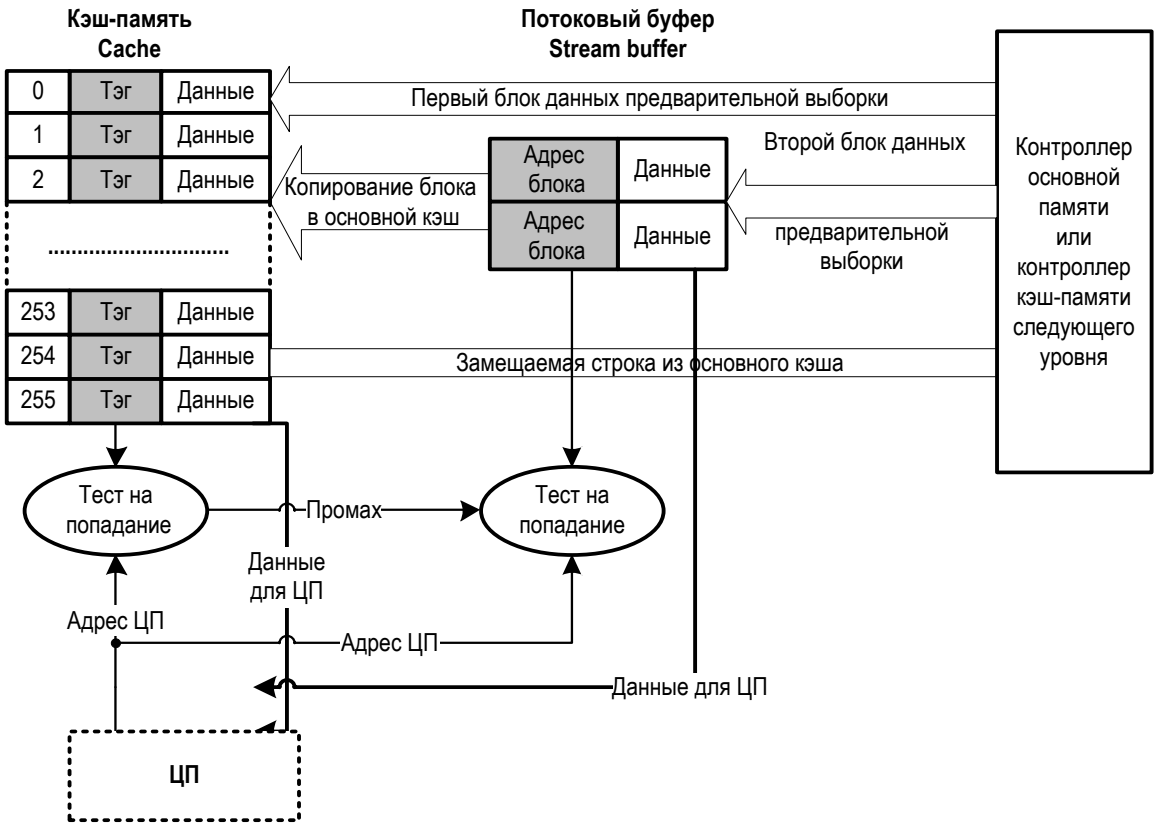


Рис. 11. Структура кэша с потоковым буфером.

В зависимости от типа прикладных программ, такой буфер может уменьшать число промахов на 50-70%. Недостатком такого подхода является требование высокой пропускной способности интерфейса памяти, часть которой будет засоряться перемещением блоков, которые впоследствии могут быть не использованы в кэше.

3.4.5. Реорганизация кода и использование программно-инициированной предварительной подкачки

Анализ кода программы может показать, что в некоторых случаях программа обрабатывая блок данных, снова возвращается к нему через определенный промежуток времени. Однако кэш-память имеет ограниченный объем и в случае больших массивов данных, они удаляются из кэша и заменяются другими, которые необходимы для последующих фрагментов кода. Главной идеей является попытка переставить местами фрагменты кода таким образом, чтобы сгруппировать все обращения к конретным блокам данных из основной памяти и обеспечить локальность обращений во времени. В этом подходе первичным является поток данных, а последовательность действий

программы реорганизуется в зависимости от наличия данных в кэше. Безусловно, данный подход требует более сложного компилятора и не во всех случаях может иметь положительный эффект.

Использование программно-инициированной предварительной подкачки требует наличия специальных команд ЦП, которые исполняются кэш-контроллером и обеспечивают предварительную подкачку данных из основной памяти в кэш. Кроме того, компилятор должен уметь анализировать структуру входного потока данных и автоматически вставлять такие команды в двоичный код. Данная технология также позволяет уменьшить долю неизбежных промахов, что приводит к существенному повышению эффективности кэш-памяти.

В отличие от аппаратных блоков, компилятор имеет гораздо большие возможности и время для обзора кода и анализа входного потока данных, а также может иметь определенную информацию о природе и поведении алогритмов программы. На основе этой информации он встраивает в различные части кода команды предварительной подкачки, управляя входным и выходным потоком данных. Следует отметить, что все это происходит совершенно прозрачно для пользователя программ, который может даже не подозревать о наличии подобных команд в данном ЦП и их поддержке компилятором. Следует отметить, что такие команды (предварительной подкачки) являются одной из форм спекулятивного исполнения команд и не могут вызывать прерываний. Вместе с тем, команды предварительной подкачки увеличивают объем кода и могут, соотвественно, уменьшать скорость исполнения программ. Микропроцессоры PA-RISC (загрузка регистров), MIPS, Power PC, SPARC имеют подобные команды.

3.5. Уменьшение задержки при промахе

В настоящее время существуют следущие способы уменьшения задержки при промахе:

- Использование многоуровневых кэшей, когда сама кэш-память имеет иерархическую организацию с несколькими уровнями (**Multilevel cache**);
- Отдача приоритета чтению из кэша над записью в кэш и слияние операций записи в память (**Priority to read misses over writes**);
- Частичное замещение строки (отдельных подблоков данных) – **Subblock placement**;
- Ранний старт и критическое слово первым (**Early start and critical word first**);
- Неблокирующие кэши (**Non-blocking caches**).

3.5.1. Использование многоуровневых кэшей

Использование многоуровневых кэшей является одним из главных способов уменьшения задержки при промахе и этот подход используется даже в самых недорогих компьютерах. Главная идея исключительно проста: использовать небольшой и очень быстродействующий кэш (L1), непосредственно сопряженный с АЛУ ЦП, который в свою очередь соединен с кэшем второго уровня (L2), имеющего значительно больший объем. При этом подразумевается, что кэш второго уровня L2 имеет меньшее быстродействие по сравнению с кэшем L1, но значительно быстрее, чем основная память. На **Рис. 12** приведена структура двухуровневого кэша с возможным дополнительным уровнем L3, который характерен для последних моделей 64-разрядных ЦП.

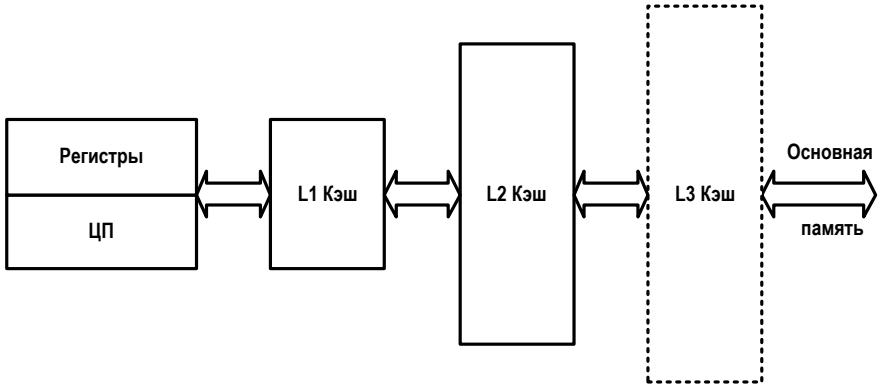


Рис. 12. Использование двух-уровневой кэш-памяти.

Тогда выражение для среднего времени доступа кэша L1 примет вид:

*T_average = T_hit(L1) + Miss_Rate(L1) * T_miss(L1)*

Где *T_average* – среднее время доступа к памяти, *Miss_Rate(L1)* – доля промахов в кэше L1, *T_miss(L1)* – задержка промаха для кэша L1.

Учитывая, что кэш L1 теперь соединен с кэшем L2, можно выразить задержку промаха через аналогичную формулу для кэша L2:

*T_miss(L1) = T_hit(L2) + Miss_Rate(L2) * T_Miss(L2)*

Где *T_miss(L1)* – среднее время доступа к памяти, *Miss_Rate(L2)* – доля промахов в кэше L2, *T_miss(L2)* – задержка промаха для кэша L2.

Подставив данное выражение в уравнение для среднего времени доступа, получаем выражение для среднего времени доступа к памяти в системе с двухуровневым кэшем:

*T_average = T_hit(L1) + Miss_Rate(L1) * [T_hit(L2) + Miss_Rate(L2) * T_miss(L2)]*

Данное выражение является ключевым в определении параметров многоуровневого кэша, оно может быть расширено аналогичным образом в случае добавления следующих уровней L3 и L4.

ОПРЕДЕЛЕНИЯ

Доля локальных промахов (Local Miss Rate) – число промахов в кэш уровня X, поделенное на общее число доступов в данный кэш. Для кэша L1 это будет *Miss_Rate (L1)*, для кэша L2 это будет *Miss_Rate(L2)*.

Доля глобальных промахов (Global Miss Rate) – число промахов в данный кэш, поделенное на общее число доступов к памяти, сгенерированных ЦП. Для кэша L1 это будет тот же *Miss_Rate (L1)*, в то же время для L2 это будет произведение долей промахов в оба кэша *Miss_Rate (L1) * Miss_Rate (L2)*

В некоторых ЦП кэш второго уровня L2 располагается на том же кристалле, что и ЦП (Pentium, Alpha 21164), в других ЦП – соединен с кристаллом ЦП специальным интерфейсом.

Использование двухуровневой кэш-памяти имеет следующие преимущества:

- Скорость работы ЦП не ограничивается кэшем большого объема;
- Выборка строки кэша может выполняться одновременно с транслированием адресов из виртуальных в физические;
- Большая свобода выбора в построении аппаратных средств кэша второго уровня, так как его скорость влияет только на задержку промаха кэша первого уровня;

Возможно два варианта отображения данных основной памяти в двухуровневый кэш:

- Многоуровневое копирование данных (**multilevel inclusion**), когда все данные, хранящиеся в кэше L1 также содержатся в кэше L2 (данные в L1 являются подмножеством L2), что соответствует принципу когерентности данных на всех уровнях;
- Уникальные или эклюзивные копии данных в кэшах первого и второго уровней (**multilevel exclusion**), когда содержание кэша первого уровня не пересекается со кэшем второго уровня.

Первый вариант является более простым в исполнении, однако требует большего объема кэша второго уровня для поддержки низкой доли промахов. Когерентность всех уровней памяти позволяет без особых проблем строить мультипроцессорные системы. Размеры блока в кэшах двух уровней, как правило, одинаковы, в некоторых случаях кэш второго уровня имеет большие размеры, кратные размеру первого уровня.

Второй вариант позволяет использовать кэши второго уровня с относительно небольшим объемом, лишь в несколько раз превышающим

кэш первого уровня, при этом возникает проблема синхронизации по данным между кэшем первого уровня и основной памятью.

ПРИМЕР РАСЧЕТА СРЕДНЕГО ВРЕМЕНИ ДОСТУПА К ПАМЯТИ В СТРУКТУРЕ С ДВУХУРОВНЕВЫМ КЭШЕМ

Допустим мы имеем гипотетическую структуру памяти с двумя кэшами L1 и L2, среднее время доступа в которой описывается выражением, где *T_average* – среднее время доступа к памяти:

$$T\_average = T\_hit_{L1} + Miss\_Rate_{L1} * (T\_hit_{L2} + Miss\_Rate_{L2} * T\_miss_{L2})$$

Параметры в формуле имеют следующие значения:

$T\_hit_{L1} = 1$ *такт* – время доступа в L1 при попадании, *Miss_RateL1=5%* – доля промахов в кэше L1, $T\_hit_{L2} = 4$ *такта* – время доступа в L2 при попадании *Miss_RateL2=10%* – доля промахов в кэше L2, $T\_miss_{L2} = 40$ *тактов* – задержка промаха для кэша L2, когда необходимо считать данные из основной памяти.

Подставив в данное выражение вышеуказанные параметры, получаем значение среднего времени доступа к памяти в системе с двухуровневым кэшем:

$$T\_average_{L1+L2} = 1 \text{ такт} + 0.05 * [\ 4 \text{ такта} + 0.1 * 40 \text{ тактов}] = 1 + 0.4 = 1.4 \text{ такта}$$

Теперь уберем кэш L2 из этой гипотетической структуры и соединим кэш L1 напрямую с сновной памятью. Тогда среднее время доступа будет определяться выражением:

$$T\_average_{L1\ only} = T\_hit_{L1} + Miss\_Rate_{L1} * T\_miss_{L1}$$

Где $T\_miss_{L1}=40$ *тактов* - задержка промаха для кэша L1, когда необходимо считать данные из основной памяти, ввиду отсутствия кэша L2.

$$T\_average_{L1\ only} = 1 \text{ такт} + 0.05 * 40 \text{ тактов} = 1 + 2 = 3 \text{ такта}$$

Сравнение результатов показывает, что наличие кэша второго уровня позволяет уменьшить среднее время доступа более чем в два раза, что и является основной причиной использования многоуровневых кэшей.

3.5.2. Отдача приоритета чтению перед записью

Отдача приоритета чтению перед записью в кэш-памяти с сквозной записью может значительно увеличить общую производительность. При использовании FIFO-буфера обратной записи, он содержит все данные и адреса, по которым должна быть произведена запись в основную память. Все работает абсолютно гладко до момента, когда ЦП запрашивает для чтения данные по адресу, модифицированный блок которых до сих пор находится в буфере записи и еще не попал в основную память. В этой ситуации кэш-контроллер может считать из памяти неверные данные, порождая **RAW**(Read After Write)-конфликт по зависимости данных в случае промаха. Можно предложить два выхода из такой ситуации:

1. При промахе чтения подождать, пока FIFO-буфер записи будет пустым (данные размещены в основной памяти), и затем считать данные из памяти в кэш. Однако такое ожидание может увеличить задержку промаха на 50%.
2. При промахе чтения проверить содержание буфера записи на предмет наличия требуемого слова данных и если оно там, то считать копию для ЦП. Если требуемого слова нет, то считать требуемый блок данных из памяти в кэш и затем считать из кэша слово для ЦП.

В кэше обратной записи приоритет чтения над записью может быть выполнен при небольшой модификации апаратных средств. Например, если в случае промаха нам нужно освободить место в кэше и при этом мы должны удалить модифицированый блок, который должен быть в обязательном порядке скопирован в основную память. Обычный способ подразумевает вначале запись удаляемого блока в основную память и затем чтение нового блока данных, требуемого для обработки промаха. При этом время ожидания данных для ЦП существенно увеличится (практически в два раза), так как запись блока в память будет занимать практически столько же времени, как и чтение нового блока. Если ввести в состав такого кэша буфер записи на строку, то можно сбросить удаляемый блок в буфер записи и затем сразу произвести чтение из памяти на освободившееся место. И только после этого произвести запись удаленного блока в основную память.

Слияние операций записи необходимо для оптимизации загрузки на интерфейс памяти и уменьшения холостых циклов ЦП, вызываемых переполнением буфера записи. Каждый элемент буфера записи расширяется до нескольких 32- или 64-разрядных слов с несколькими битами маски, показывающими наличие действительных данных в каждом слове. Каждый раз когда ЦП производит запись слова, оно

пишется в один из элементов в позицию, определямую младшими битами. Пример реализации буфера для слияния операций приведен на **Рис. 13**, где верхний блок показывает обычный буфер шириной в одно слово, а нижний - буфер для слияния операций записи.

Адрес записи		V	Данные записи 32-бит
100		1	x00996875
104		1	x11883000
108		1	x00996801
112		1	x00556891

100		1	x00996875	1	x11883000	1	x00996801	1	x0055689
		0		0		0		0	
		0		0		0		0	

Рис. 13. Использование слияния операций записи.

3.5.3. Частичное замещение строки или использование подблоков

Частичное замещение строки или использование подблоков характерно в кэшах с большим размером блока/строки, что обычно получается при желании разработчика уменьшить длину адресного тэга и соответственно объем памяти тэгов. Доля промахов в такой кэш снижается, но за счет роста задержки промаха, когда при замещении строки нужно передавать гораздо больший объем данных. Этот эффект можно уменьшить, добавив дополнительные биты достоверности для для каждого подблока данных и позволив контроллеру кэша читать и записывать такие подблоки независимо. Это уменьшает задержку промаха за счет уменьшения объема данных до размера одного подблока, передаваемого в кэш при промахе. **Рис. 14** иллюстрирует упрощенную структуру кэша с подблоками.

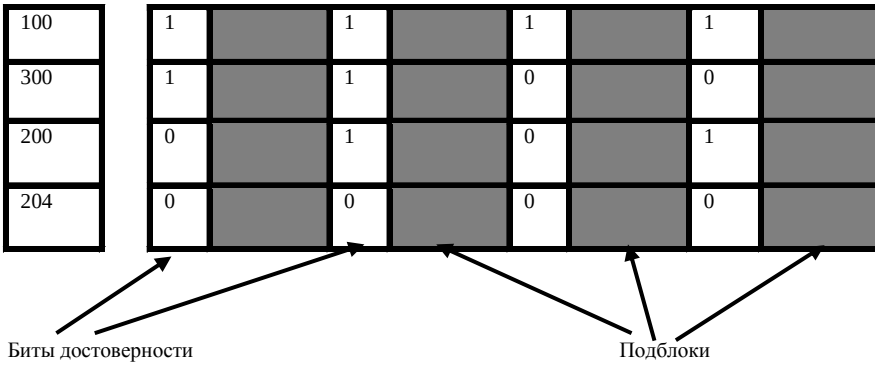


Рис. 14. Упрощенная структура кэша с подблоками

3.5.4. Ранний старт и критическое слово первым

Ранний старт и критическое слово первым базируются на принципе, когда ЦП не должен ожидать полной загрузки блока/строки в кэш-память и немедленно продолжить вычисления в момент получения требуемого слова данных. При этом подразумевается, что размер блоков кэш-памяти достаточно велик и загрузка блока данных из памяти требует определенного числа тактов.

Ранний старт предусматривает возможность для ЦП немедленно продолжить вычисления с момента загрузки в кэш необходимого слова данных. К примеру, строка кэш-памяти содержит 8 слов и ЦП ожидает второе слово, которое в момент поступления немедленно передается в ЦП для обработки, в то время как кэш-контроллер продолжает загрузку оставшихся 6 слов в строку кэша.

Критическое слово первым предусматривает запрос на требуемое слово из памяти первым вне зависимости от положения слова в строке кэша и посылку данных ЦП немедленно по получении. Это очень похоже на ранний старт с отличием в порядке получения слов строки кэша из памяти. В случае раннего старта порядок получения слов из памяти всегда один и тот же (начиная с первого), здесь же порядок будет меняться в зависимости от положения требуемого слова в строке, которое всегда должно выбираться первым.

3.5.5. Неблокирующие кэши

С появлением суперскалярных микропроцессоров с произвольным порядком исполнения команд и значительным уровнем параллелизма, традиционная технология построения кэшей, когда ЦП останавливался в ожидании данных из последовательной подсистемы памяти, стало неприемлемым. В ЦП с произвольным порядком исполнением команд, подсистема памяти также должна обеспечивать параллелизм доступа к

памяти, не блокируя процесс доступа другим командами в случае промаха, вызванной одной из команд ЦП. Если одна из команд при попытке доступа к данным получила промах, то другие команды могут иметь попадание в кэш и продолжить обработку. Работа таких кэшей обеспечивается по принципу обработки пакетных трансакций, когда множество запросов не мешают друг другу.

Для поддержки такого режима необходим кэш-контроллер с несколькими параллельными трактами, который может выполнять обработку промаха для одной команды и выдавать данные в случае попадания для других команд, позволяя ЦП продолжить обработку данных. Такая особенность функционирования называется «Попадание после промаха» (**Hit under Miss**), позволяя значительно уменьшить время простоя ЦП в ожидании данных. Более развитая фунциональность таких кэшей позволяет обрабатывать «Попадание после множества промахов» (**Hit under multiple misses**), обрабатывая множество промахов одновременно.

Безусловно, требование многотрактности и параллельной обработки усложняет кэш-контроллер, который должен быть способным генерировать множество запросов к памяти и уметь правильно распределять поток поступающих из памяти данных. На **Рис. 15** приведена структура простейшего неблокирующего кэша (канал чтения). Кэш-контроллер при каждом обращении к памяти после проверки на попадание распределяет промахи и попадания в разные конвейеры, обеспечивая считывание данных для попаданий без блокирования предшествующими промахами.

НЕКОТОРЫЕ ПОЛЕЗНЫЕ ФОРМУЛЫ ДЛЯ РАСЧЕТА ПРОИЗВОДИТЕЛЬНОСТИ КЭШ-ПАМЯТИ

Ниже приведены некоторые формулы, которые могут быть полезны для расчета производительности кэш-памяти:

Расчет числа битов индекса, необходимы для доступа в кэш прямого отображения и частично-ассоциативный кэш:

$$2^{Index} = Cache\_Size / (Block\_size \times Set\_Assoc)$$

$$Index = \log_2 (Cache\_Size / (Block\_size \times Set\_Assoc)$$

Где *Cache_Size* – объем кэш-памяти в байтах, *Block_size* – размер строки/блока кэша в байтах, *Set_Assoc* – число каналов или банков в частично-ассоциативном кэше, имеет значение единицы для кэша прямого отображения.

CPU_time = (CPU_exe_cycles + Mem_stall_cycles) x Cycle_time

Где *CPU_time* - общее время работы ЦП, *CPU_exe_cycles* - число тактов, в течении которых команды исполнялись, *Mem_stall_cycles* - число тактов, в течении которых ЦП простаивал, ожидая данные из памяти, *Cycle_time* - период тактового сигнала ЦП

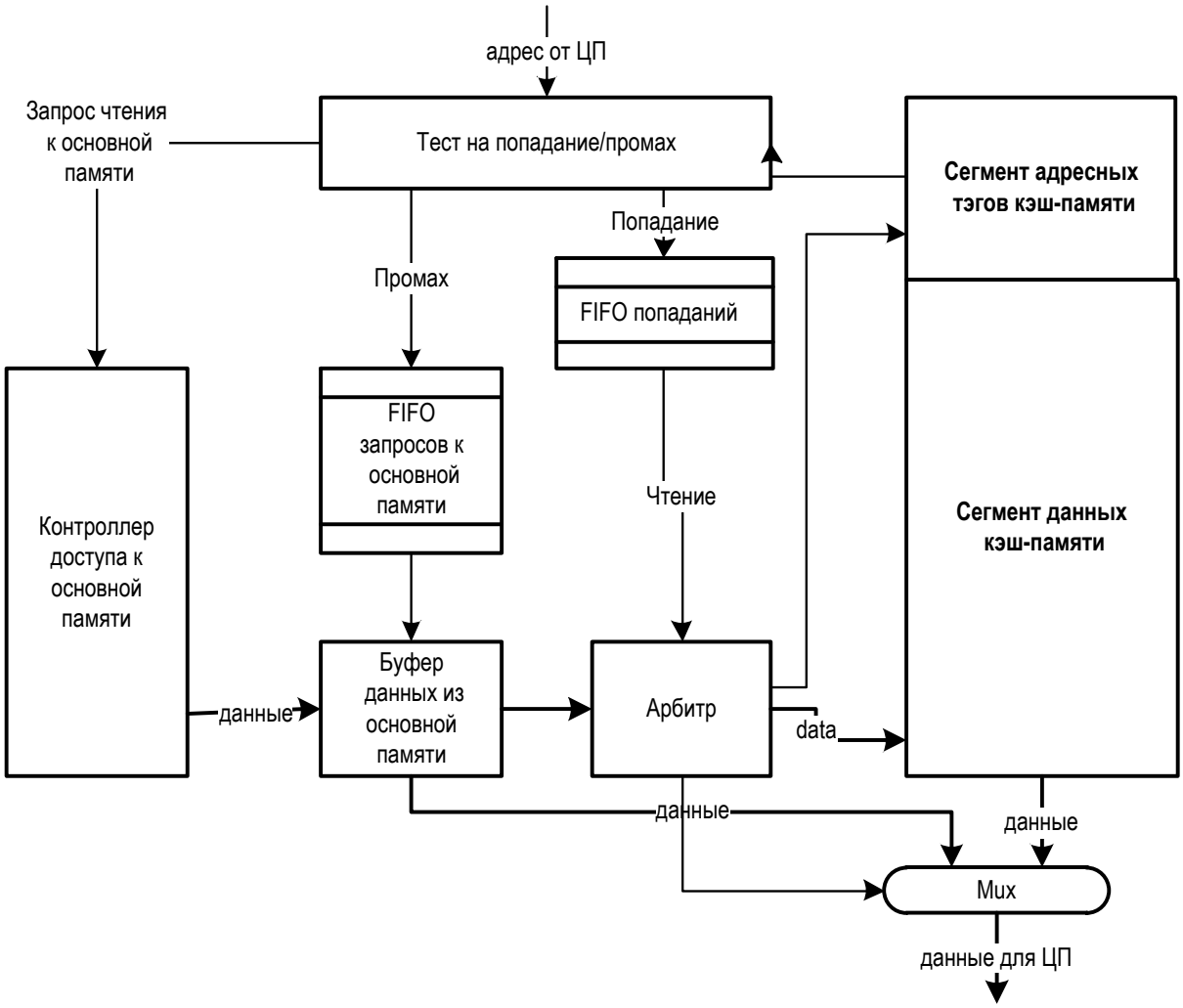


Рис. 15. Структура тракта чтения простейшего неблокирующего КЭШа.

Mem_stall_cycles = (#Reads x Rd_miss_rate x Rd_miss_penalty + #Writes x Wr_miss_rate x Wr_miss_penalty)

Где *#Reads, #Writes* – число выполненных операций чтения/записи памяти, *Rd_miss_rate, Wr_miss_rate* - доли промахов при чтении/записи в память, *Rd_miss_penalty, Wr_miss_penalty* - время выборки/записи строки в память в случае промаха. Можно сложить значения этих парных операндов и тогда число тактов простоя ЦП в оожидании данных из памяти будет выглядеть проще:

Mem_stall_cycles = #Mem_Accesses x Miss_rate x Miss_penalty

Где *#Mem_Accesses=(#Reads + #Writes), Miss_rate= (Rd_miss_rate + Wr_miss_rate), and Miss_penalty= =(Rd_miss_penalty + Wr_miss_penalty)*

Общее время работы ЦП может быть выражено через другую формулу

CPU_time = IC x (CPI + MAPI x Miss_rate x Miss_Penalty) x Cycle_time

Где IC (Instruction Count) – число команд в тестовой программе, CPI (Cycles per Instruction) – среднее число тактов на исполнение одной команды, MAPI (Memory Acess per Instruction) – среднее число доступов в память на одну команду,

Misses per instruction *MPI = MAPI x Miss_rate*

CPU_time = IC x (CPI + MPI x Miss penalty) x Cycle_time

Вышеприведенные формулы, при всей их простоте, позволяют понять особенности той или иной структуры памяти и оценить ее поведение при наличии некоторых статистических данных о работе тех или иных тестовых программ. Причем время работы ЦП, как правило, известно по содержанию специальных счетчиков, считываемых тестовой программой и интерес представляют другие параметры в формулах, которые рассчитываются на основании статистических данных.

4. ВИРТУАЛЬНАЯ ПАМЯТЬ В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

4.1. Концепция виртуальной памяти.

Этот раздел посвящен одному из базовых постулатов концепции иерархической памяти в вычислительной системе. Механизм виртуальной памяти существует практически во всех компьютерных системах, начиная с портативных персональных компьютеров. Встроенные системы в новых моделях сотовых телефонов и карманных компьютеров также используют механизм виртуальной памяти.

Мы рассмотрим основные особенности механизма виртуальной памяти (в дальнейшем **ВП** для краткости), базовые определения и параметры. Затем будут рассмотрены простые примеры построения виртуальной памяти со страничной и сегментной организацией с детальным анализом работы этих структур. Проблемы повышения производительности ВП будут рассмотрены в контексте взаимодействия с кэш-памятью и внешней памятью на магнитных дисках.

ОПРЕДЕЛЕНИЯ ВИРТУАЛЬНОЙ И ФИЗИЧЕСКОЙ ПАМЯТИ

Виртуальную память можно определить как специальный механизм, который позволяет ЦП генерировать виртуальный адреса, которые далее траслируются (или отображаются) в физические адреса для доступа к данным в иерархии памяти компьютерной системы. При этом для программиста создается иллюзия предоставления каждой отдельной программе полного адресного пространства и имеющихся ресурсов памяти. Механизм ВП реализуется как аппаратными средствами ЦП, так и программными модулями операционной системы, обеспечивающих загрузку необходимых данных в основную память из внешней памяти на магнитных дисках.

Далее мы будем использовать несколько терминов, которые надо определить:

- Виртуальная память (**Virtual memory**), как рабочее пространство адресов для конкретной программы, ассоциируется также с виртуальным адресом. К примеру, для 32-разрядных ЦП виртуальное пространство или максимальный объем виртуальной памяти будет 2^{32} =4 Гигабайта,

- Физическая память (**Physical memory**), как реально имеющийся в компьютерной системе объем основной памяти, рабочее пространство которой может быть меньше виртуального. К примеру типичный объем

основной памяти в персональных компьютерах с 32-разрядным ЦП составляет 256-512 Мегабайт, что в 8-16 раз меньше чем максимальный объем виртуальной памяти. Максимальный размер физической памяти определается особенностями реализации адресной шины процессора в сочетании с ограничениями, накладываемыми особенностями реализации контроллера памяти.

Какие побудительные причины были за созданием механизма ВП? Можно перечислить следующие основные пункты:

- Компьютерная система в каждый момент времени исполняет множество программ (процессов в терминах операционной системы), каждый из которых требует своего собственного адресного пространства.
- Так как большинство программ в каждый взятый момент времени не используют всего отведенного адресного пространства, то резервирование полного физического адресного пространства для каждой программы было бы крайне расточительным.
- Если разделить физическую (основную) память на блоки (сегменты) или страницы, то можно резервировать физическую (основную) память для программ по мере необходимости в ходе их исполнения.
- Если множество программ одновременно используют физическую память, то необходим специальный механизм, обеспечивающий защиту данных одной программы от случайного или преднамеренного доступа другой программы.

Рис. 16 иллюстрирует размещение данных в пространстве основной памяти с использованием разделения пространства памяти на блоки (страницы или сегменты), которые могут размещаться в основной памяти (становиться резидентными) или находиться на магнитном диске в зависимости от стадии исполнения той или иной программы.

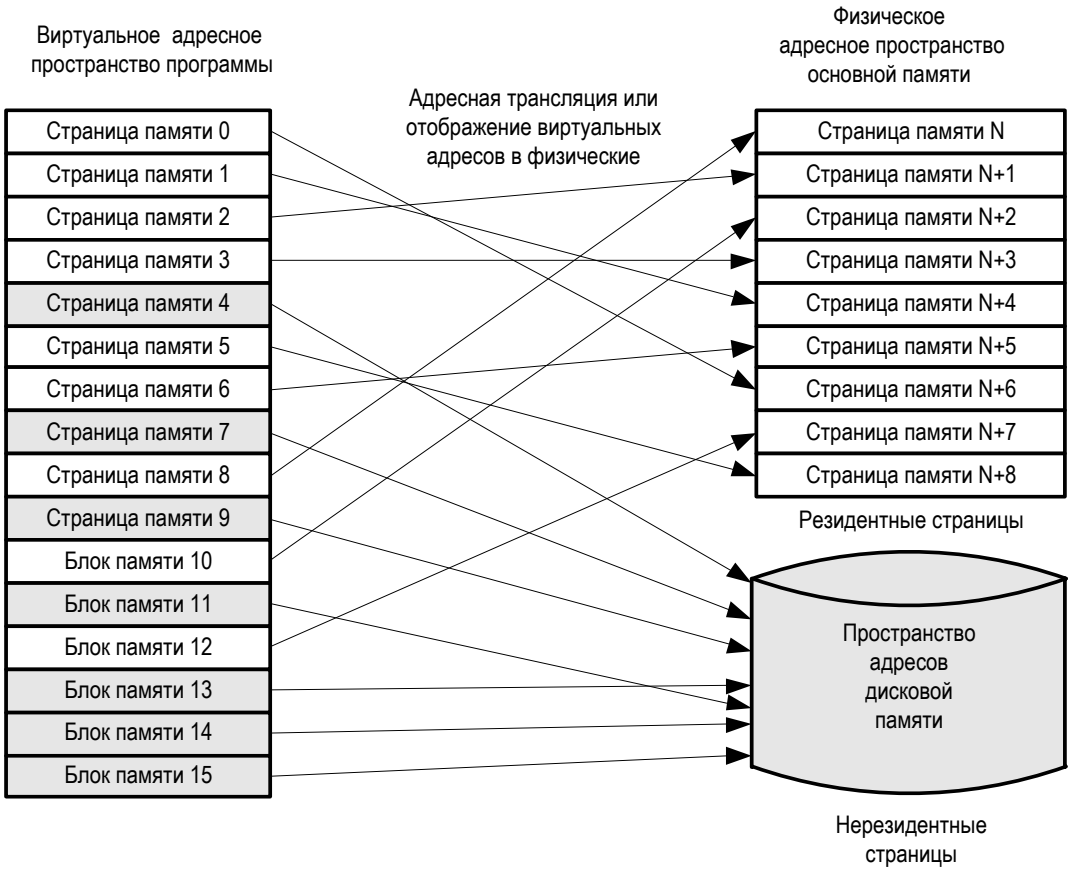


Рис. 16. Размещение блоков (страниц) данных в основной памяти и внешней дисковой памяти.

Использование механизма ВП обеспечивает следующие преимущества:

1. ВП позволяет разделять физическую (основную) память компьютерной системы между множеством программ и обеспечивать реальный мультипрограммный режим.
2. ВП предоставляет каждой программе одинаковое адресное пространство для кода и данных.
3. ВП обеспечивает схему защиты данных одной программы в памяти от доступа другой программы.
4. Виртуальная память позволяет программам использовать значительно большее адресное пространство , чем имеется в основной (физической) памяти. До появления механизма ВП программисты были вынуждены поддерживать режим разделения программ на резидентную и дисковую часть (overlay). ВП делает этот режим полностью прозрачным (невидимым) для прикладного программиста.

5. ВП упрощает загрузку программ, все перемещения программ контролируются этим механизмом. При других подходах необходимо модифицировать исполняемый код и регистры перемещения.

Однако все эти достоинства сопровождаются некоторыми издержками – дополнительным временем, необходимым для трансляции адреса и дополнительной памятью и аппаратурой, необходимой для поддержки трансляции адресов.

Поэтому встроенные системы, как правило не используют систему ВП, так как они исполняют небольшие прикладные программы реального времени, загрузка которых происходит крайне редко. При этом нет требований к разделению ресурсов и защите памяти из-за использования статического принципа размещения кода в памяти. Хотя это не относится к новейшим многофункциональным сотовым телефонам и MP-3 плейрам, которые могут загружать новые программы и данные. В этом случае нужна полномасштабная система ВП для обеспечения гибкости системы.

В некоторых сверхбыстродействующих компьютерах, в которых временные затраты на трансляцию адресов считаются неприемлемыми, также отсутствует система ВП.

Упрощенная иллюстрация работы механизма виртуальной памяти представлена на **Рис. 17**. Исполняемая прикладная программа генерирует только виртуальные адреса своего собственного адресного пространства и не имеет представления, где на самом деле находятся данные, к которым она должна обращаться. Операционная система, используя идентификатор программы, обеспечивает размещение данных в основной физической памяти. Программа может получить доступ к своим данным только после процедуры адресной трансляции, когда виртуальный адрес заменяется физическим, указывающим точное расположение данных в памяти. Но требуемых данных в основной памяти может не оказаться по причине нехватки места, и эти данные находятся на внешней памяти НМД. Тогда аналогично принципам работы кэш-памяти, мы имеем промах в доступе к основной памяти, и данные должны быть загружены с НМД.

Ввиду того, промах в механизме виртуальной памяти имеет несколько иную природу, его принято называть ошибкой доступа к странице (**page fault**). Такие промахи обрабатываются программными модулями операционной системы, которые поддерживают специальные таблицы размещения страниц (**page tables**) в основной и внешней памяти, используемые для трансляции адресов. Данные таблицы должны быть размещены в основной памяти и для того, чтобы узнать отображение

каждого виртуального адреса в физический, необходимо произвести доступ к этой таблице.

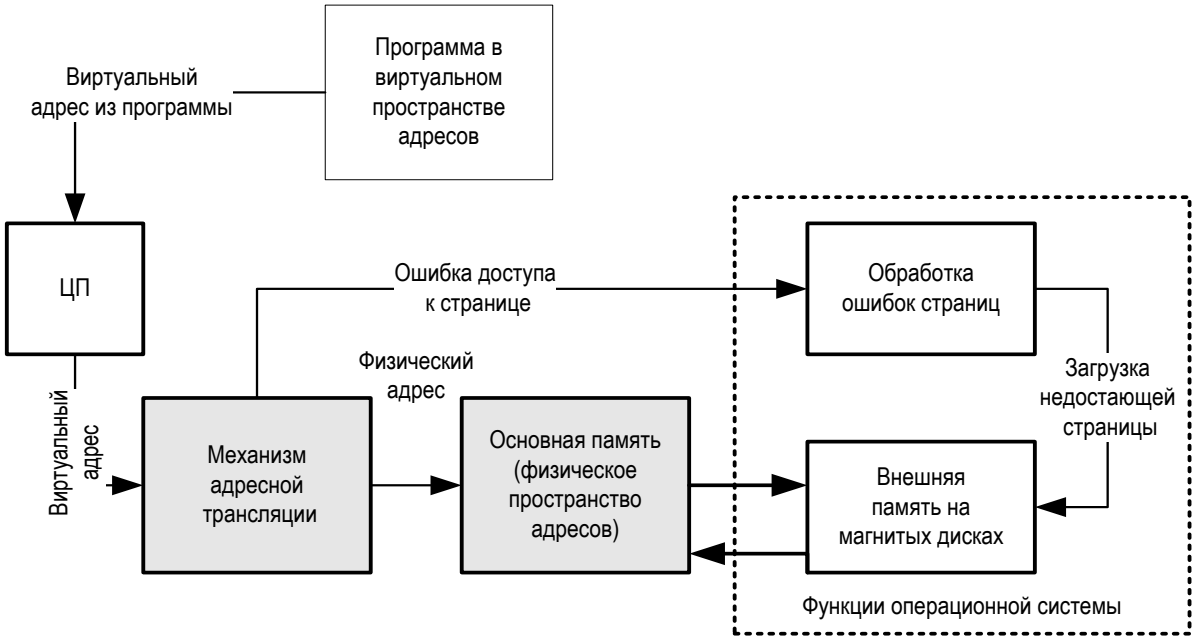


Рис. 17. Механизм виртуальной памяти

ОЦЕНКА СРЕДНЕГО ВРЕМЕНИ ДОСТУПА К ВИРТУАЛЬНОЙ ПАМЯТИ

Рассмотрим как влияет наличие виртуальной памяти на среднее время доступа, учитывая следующие параметры:

- Время доступа ВП (**virtual memory hit time**) в диапазоне 40-100 тактов
- Задержка промаха ВП (**miss time**) в диапазоне 700 000 – 6 000 000 тактов, так как в этом случае производится доступ к относительно медленному НМД
- Доля промахов ВП в диапазоне 0.00001% - 0.001%

Тогда, подставив эти значения в известную формулу *Tavg = Thit + MR*Tmiss* , получаем

*Tavg_min1 = 40 + 0.00001%*700,000 = 40.07 тактов*

*Tavg_min2 = 40 + 0.00001%*6,000,000 = 40.6 тактов*

*Tavg_max1 = 40 + 0.001%*700,000 = 47 тактов*

*Tavg_max2 = 40 + 0.001%*6,000,000 = 100 тактов*

По результатам приведенных расчетов можно сделать вывод, что наибольший эффект оказывает доля промахов, при росте которой среднее время доступа резко возрастает. Единственными способами уменьшить долю промахов являются увеличение объема основной памяти и предварительная подкачка страниц.

4.2. Характеристики и параметры механизма виртуальной памяти

Параметры механизма виртуальной памяти очень похожи на параметры кэш-памяти, сохраняется практически полная аналогия, только это другой уровень иерархии памяти:

- Страница памяти очень похожа на строку или блок кэш-памяти и это является наименьшим фрагментом памяти, с которым имеет дело механизм виртуальной памяти. Отметим, что в отличие от строки кэша, размер страницы или сегмента может быть переменным от 4К байт до десятков мегабайт.
- Также должны быть использованы алгоритмы замещения для удаления страниц из основной памяти. В кэш-памяти это выполняется аппаратными средствами контроллера, в механизме ВП из-за очень большой задержки промаха могут быть использованы программные утилиты операционной системы.
- Доступ к ВП также может вызывать промахи, которые называются ошибками доступа к странице или ошибками адреса, и они очень похожи на промахи в кэш-памяти.
- Объем ВП частично определяется объемом адресного пространства используемого ЦП.
- Внешний накопитель на магнитных дисках (НМД) является самым нижним уровнем иерархии при использовании механизма ВП. Он используется как память файловой системы, так и для хранения страниц (сегментов) ВП.

Ключевые характеристики механизма ВП:

- Размеры блоков данных передаваемых из основной памяти во внешнюю и наоборот, размеры страниц могут быть постоянными (4 - 8 Кбайт) или варьироваться в случае сегментов или страниц переменных размеров.

- Также как и для кэш-памяти в механизме ВП должны использоваться алгоритмы замещения (**replacement policy**).
- В дополнение к алгоритмам замещения необходимы также алгоритмы размещения (**placement policy**), так как в отличие от кэша все области основной памяти равнозначны и отображение виртуальной страницы может быть выполнено на любую страницу физической памяти за исключением зарезервированных для операционной системы.
- Алгоритмы управления загрузкой из дисковой памяти (**demand load policy**), когда недостающая страница загружается только в случае ошибки доступа к странице, либо заранее загружается множество страниц, необходимых для работы программы.

Существуют три основных типа механизма ВП:

1) ВП с страничной организацией, где используются страницы фиксированного и относительно небольшого размера
2) ВП с сегментной организацией, где размеры сегментов могут варьироваться.
3) Комбинированная ВП с сегментами и страницами различного размера (число размеров при этом ограничено).

В механизме ВП со страничной организацией виртуальная и физическая память разбивается на блоки одинакового размера в диапазоне от 4К байт до 64 Кбайт. Причем в виртуальном пространстве эти блоки называются страницами, а в физическом – страничными фреймами (**page frame**). Эти страницы адресуются практически таким же способом как и блоки данных в кэше, когда адресное поле разбивается на два поля, включающих номер страницы (**page number**) и поле смещения внутри страницы (**offset**).

Второй и третий тип ВП использует сегменты разного размера от 1К байт до 4Г байт. Сегментация программ требует сложного компилятора, который должен разбивать данные на модули разного размера в зависимости от их использования программой.

Дискуссия о преимуществах и недостатках обоих типов ВП продолжается по сей день, в разных приложениях они показывают разную эффективность. Мы тоже присоединимся к этой дискуссии и попробуем сравнить оба механизма в **Табл. 6**.

Табл. 6. Сравнение сегментной и страничной организации механизма виртуальной памяти

Характеристики / Тип механизма ВП	**ВП с страничной организацией**	**ВП с сегментной организацией**

Число слов в адресе	Одно слово	Два (сегмент и адрес)
Доступ прикладного программиста	Нет доступа, механизм прозрачен	Доступ к механизму ВП возможен
Алгоритм замещения блоков и страниц	Простой, так как все страницы одного размера	Сложный, так как в физической памяти должны быть найдены непрерывные фрагменты нужного размера
Эффективность использования памяти	Внутренняя фрагментация (Неиспользованные части страниц в памяти)	Внешняя фрагментация (Неиспользуемые фрагменты памяти между сегментами)
Эффективность пересылки данных между основной и внешней памятью	Размер страницы может быть подогнан для повышения эффективности пересылки данных в/из внешней памяти	Сегменты малого размера могут снижать скорость пересылки данных

Чисто сегментная организация ВП используется крайне редко в современных системах, более популярным становится использование страниц переменного размера с минимальным размером 4К байт и максимальным в размере 4М байт, кратным числам в степени двойки. Ограничения на минимальный размер страницы решают проблему ввода/вывода, в тоже время большие страницы эффективно решают проблему отображения ресурсов памяти для операционной системы и крупноразмерных структур данных.

Для иллюстрации работы механизма ВП рассмотрим структуру на **Рис. 18**.

В данной структуре используется 32-разрядный виртуальный адрес в котором имеется два поля, 12 младших бит предназначены для смещения внутри страницы размером в 4К байт и 20 старших бит используются для номера виртуальной страницы. Адрес физической памяти содержит 28 бит, в которых те же самые 12 младших бит используются для смещения и оставшиеся 16 старших для номера физической страницы или фрейма страницы. Таким образом, имеется виртуальное пространство адресов размерностью в 4 Г байт и физическое пространство адресов размерностью в 256 М байт, что 16 раз меньше. Таблица страниц размещается в специальной области основной памяти, отведенной для операционной системы, и обеспечивающей защиту от доступа прикладных программ.

Каждый раз, когда прикладная программа генерирует виртуальный адрес, старшие 20 бит используются в качестве индекса, который прибавляется к значению базового адреса для доступа к соотвествующему элементу таблицы. Каждый элемент таблицы содержит набор некоторых данных, которые включают физический адрес страницы (16 бит), права доступа и идентификатор программы, а также специальный бит V (**Valid**), указывающий наличие данной страницы в основной памяти. Если страницы в основной памяти нет (V=0), то генерируется ошибка доступа к странице, при которой ОС должна загрузить недостающую страницу в основную память из внешней памяти. Если права доступа или идентификатор не совпадают, то генерируется прерывание по защите доступа к основной памяти.

Работа данной структуры в целом похожа на механизм косвенной адресации, когда для определения истинного адреса данных необходимо вначале извлечь из основной памяти сам адрес и затем произвести доступ к данным. Поэтому использование ВП теоретически предполагает снижение скорости доступа к памяти из-за дополнительной выборки физического адреса из таблицы страниц в памяти.

Несколько слов о размере таблицы страниц, для 20-разрядного номера страниц в данном случае должно быть 1 М элементов таблицы, при размере элемента в 4 байта общий объем занимаемой памяти будет 4М байт. Если размер виртуального адреса возрастает больше 40 бит, что характерно для всех современных 64-разрядных ЦП, то мы можем иметь серьезные проблемы при размещении таблицы страниц в памяти. Как решаются эти проблемы, рассмотрено в следующем разделе.

Таким образом, ЦП может генерировать виртуальные адреса, размерность которых может превышать размерность физической памяти, механизм ВП будет обеспечивать поиск необходимых страниц во вторичной памяти на НМД и их загрузку в свободные фрагменты основной (физической памяти). Механизм ВП построен на группе алгоритмов, управляющих отображением и размещением страниц как в основной памяти , так и памяти НМД.

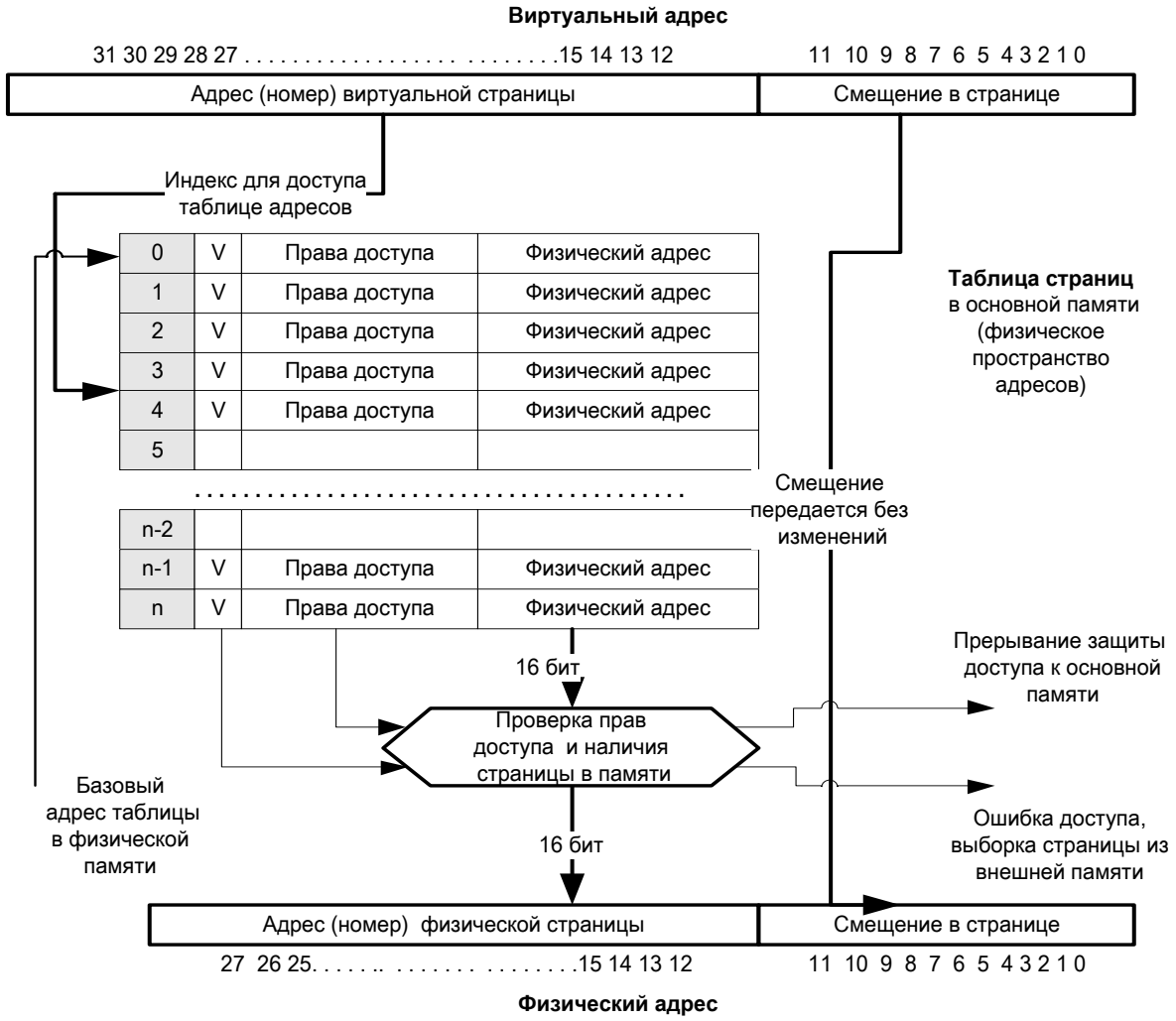


Рис. 18. Таблица страниц в механизме виртуальной памяти

Две группы основных алгоритмов управляют работой механизма ВП (вспоминаем кэш-память):

1) Алгоритмы размещения страниц в основной и вторичной памяти;
2) Алгоритмы замещения страниц в основной памяти;

Алгоритмы размещения страниц решают следующие задачи:

- Каким образом виртуальная страница может быть размещена в физической памяти? Причем с учетом ограничений на отображение страниц с целью минимизации доли промахов, так как задержка при промахе в механизме ВП может достигать десятков миллисекунд. Из опыта изучения кэш-памяти можно сказать, что наиболее эффективным с этой точки зрения является полно-ассоциативное отображение.

- Каким образом блок данных может быть найден в основной памяти? Адресная трансляция с использованием таблицы страниц или специального буфера быстрой трансляции является ключевым элементом. Индексом в эту таблицу адресов являются старшие биты виртуального адреса.
- Утилиты операционной системы должны обеспечивать изменение таблицы страниц в соответствии с текущим расположением всех данных.

Алгоритмы замещения по своей природе совершенно аналогичны используемым в кэш-памяти, только более сложные, так как могут быть реализованы на уровне программ ОС.

Основные особенности:

- Наиболее эффективным эвристическим алгоритмом является все тот же знакомый алгоритм LRU;

- В механизме ВП алгоритм LRU реализуется через использование специально бита использования или произошедшего доступа к странице (**use or reference bit**), ассоциированного с каждой страницей ВП, отображенной в физическую память. Этот бит также включается в запись в таблице страниц вместе с битом наличия страницы в физической памяти;

- Бит доступа выставляется в момент доступа к данной странице, биты доступа всех страниц периодически сбрасываются специальной утилитой ОС;

- Когда необходимо замещение и нужно выбрать страницу для перемещения на НМД, то утилита ОС выбирает страницу с сброшенным битом доступа. Это может быть неточное исполнение LRU, но вполне достаточное приближение.

Небольшой комментарий по алгоритмам записи. Ввиду очень большой задержки при записи данных на НМД, используются только алгоритмы обратной записи, сквозная запись в механизме ВП не применяется.

Рассмотрим пример выполнения механизма виртуальной в 64-разрядном ЦП Alpha 21264.

4.3. Пример механизма виртуальной памяти со страничной организацией

Вначале следует отметить, что механизм ВП в Alpha 21264 использует смешанную сегментно-страничную организацию, где на самом верхнем уровне представлены три сегмента: seg0, kseg и seg1. Сегмент 0 и kseg используются используется для прикладных программ, в то время как сегмент seg1 для системных программ ОС. Разделение на сегменты производится по значению старших битов адреса (63-46): seg0[63-46] = 00…00, kseg[63-46]= 00…010, seg1[63-46]=11…11. Каждый сегмент в свою очередь подразделяется на страницы, которые адресуются 43-разрядным виртуальным адресом. Отметим, что биты 43, 44, 45 были зарезервированы для будущего увеличения размеров страниц. При таком адресном пространстве таблица страниц, выполненная традиционным способом будет иметь огромные размеры, что является неприемлемым. Поэтому используется иерархический способ организации таблицы страниц, проиллюстрированный на **Рис. 19**. Виртуальный 64-разрядный адрес разделяется на пять компонент:

- Старшие 21 бит используются для идентификации используемого сегмента;

- Следующие 10 бит используются для индексирования таблицы страниц уровня L1;

- Следующие 10 бит используются для для индексирования таблицы уровня L2 с использованием базового адреса, считанного из таблицы L1;

- Следующие 10 бит используются для индексирования таблицы уровня L3 с использованием базового адреса, считанного из таблицы L2;

- Самые младшие 13 бит используются для адресации внутри отдельной страницы размером 8 К байт.

Заметим, что каждая таблица содержит 2^{10} = 1024 элементов и каждый элемент содержит 64 бит = 8байт, что определяет размер таблицы в 8 К байт, который равен размеру страницы. Каждый элемент таблицы содержит 32-разрядный физический адрес фрейма, определяющий базовый адрес следующей таблицы и биты прав доступа. Оставшиеся биты неопределены и могут быть использованы утилитами ОС для поддержки алгоритмов замещения и других целей.

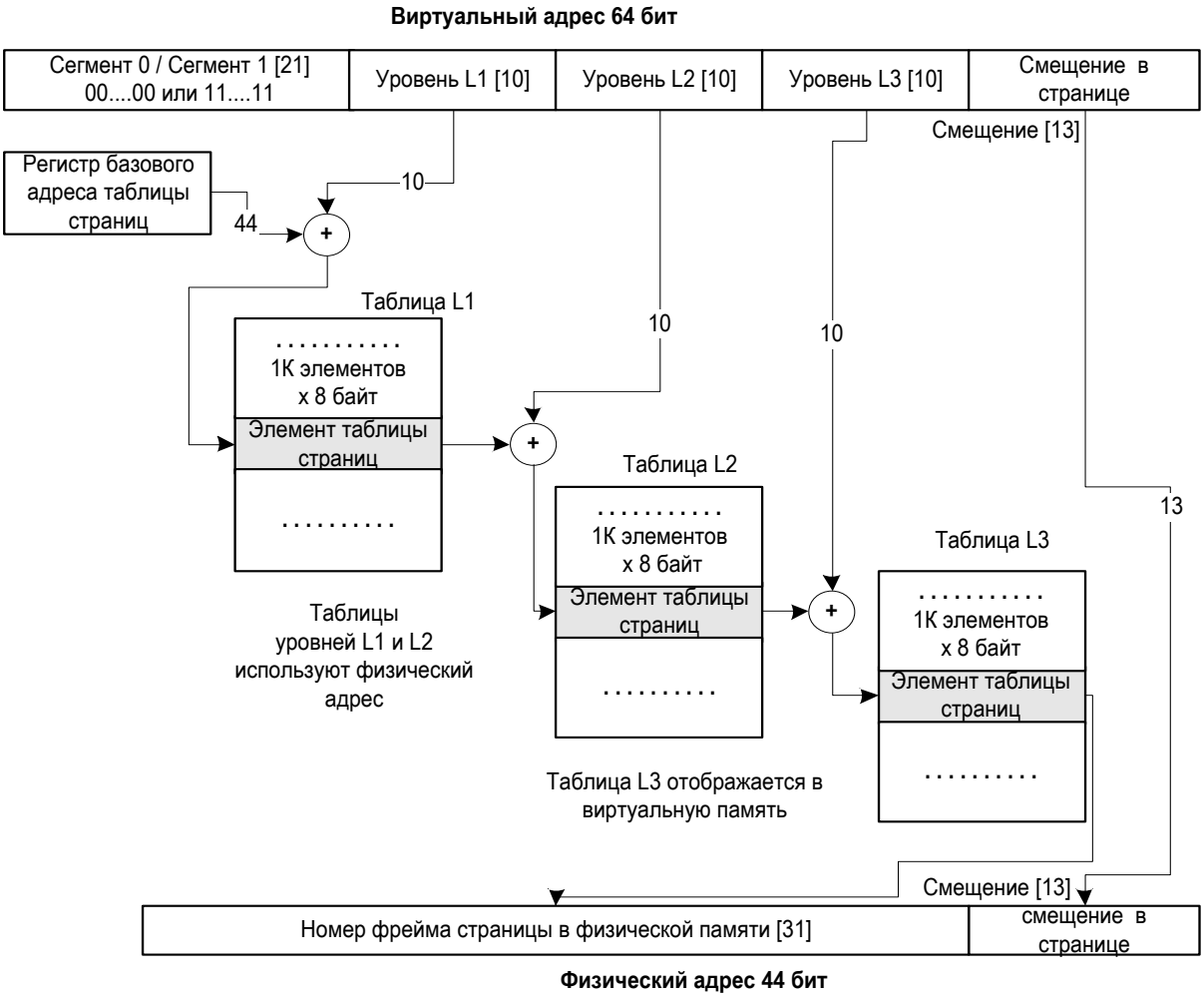


Рис. 19. Организация таблицы страниц в механизме ВП ЦП Альфа 21264

Процесс поиска физического адреса исключительно прост:

1. Значение базового адреса таблицы (страницы) и 10-разрядный индекс L1 объединяются, чтобы найти нужный элемент в таблице L1,

2. Выбранный элемент в таблице L1 содержит базовый адрес таблицы L2 и биты прав доступа, показывающие, где в данный момент находится необходимая таблица L2 (резидентна в основной памяти или на НМД),

3. Если таблица резидентна в основной памяти, то процесс продолжается и считывается элемент таблицы L2 с использованием

10 бит уровня L2 в качестве индекса. Элемент этой таблицы L2 содержит базовый адрес таблицы последнего уровня L3.

4. На последнем уровне базовый адрес из таблицы L2 объединяется с индексом L3 и считывается элемент таблицы L3, в котором должны быть проверены права доступа,

5. Если права доступа программы соответствуют записанным в элементе таблицы, то базовый адрес из таблицы L3 объединяется с полем смещения виртуального адреса и производится доступ к данным по конечному физическому адресу.

Структура битового поля права доступа в элементе таблицы страниц в механизме ВП ЦП Альфа 21264 представлена в Табл. 7.

Табл. 7. Структура прав доступа в ЦП Альфа 21264

Название битового поля в оригинале	**Название в переводе**	**Назначение**
Valid	Достоверность	Данный номер физического фрейма достоверен и может быть использован для адресной трансляции. Если бит сброшен, то данная страница находится во вторичной памяти на НМД
Use or reference	Использование или доступ	Данный номер физического фрейма использовался с момента последнего циклического сброса битов использования страниц
User read enable	Право пользователя на чтение	Позволяет прикладной программе читать данные из страницы памяти
Kernel Read enable	Право ядра ОС на чтение	Позволяет ядру ОС читать данные из страницы памяти
User write enable	Право пользователя на запись	Позволяет прикладной программе

		записывать данные в страницу памяти
Kernel write enable	Право ядра ОС на запись	Позволяет ядру ОС записывать данные в страницу памяти

Теперь проанализируем, что происходит в вышеописанном механизме ВП при чтении элемента таблицы страниц, где бит достоверности (V или Valid) оказался в сброшенном состоянии. Это означает, что запрошенная страница не представлена в физической памяти и произошла ошибка доступа к странице (**page fault**), который по аналогии с кэш-памятью также можно назвать промахом. Ошибка доступа приводит к очень большой задержке доступа к данным и является крайне нежелательным событием.

Запрашиваемая страница может быть таблицей L2 или L3, либо страницей с конечными данными для программы. Отметим, что таблица L1 всегда находится в физической памяти, так как нет никаких указателей, где она может находиться и механизм ВП использует ее как исходную точку для поиска данных. Это не создает проблем, так как размер таблицы L1 всего 8 К байт.

В случае ошибки доступа, элемент таблицы страниц не содержит достоверного адреса, но эти адресные биты могут использоваться для индикации расположения страницы во вторичной памяти на НМД. Ошибка доступа к странице всегда вызывает прерывание с вызовом утилит ОС, которые отвечают за обработку прерывания от механизма ВП. Утилиты ОС должны выполнить следующую последовательность действий:

1. Проверить таблицу страниц в физической памяти на предмет наличия элементов, которые не были использованы с момента последнего сброса всех битов использования (Use=0) и записать удаляемую страницу на НМД;

2. Модифицировать соотвествующую запись в таблице страниц (V=0), показывающей, что страница удалена из физической памяти и указать в поле адреса расположение на диске для удаленной страницы;

3. Прочитать элемент таблицы страниц, вызвавшей прерывание по ошибке доступа и найти требуемую страницу во вторичной памяти на НМД, используя данные из поля адреса;

4. Считать данную страницу в физическую память на место удаленной и модифицировать элемент таблицы страниц, вызвавший ошибку доступа. При этом бит достоверности должен быть установлен (V=1) и стартовый адрес страницы в физической памяти должен быть записан в поле адреса.

5. После этого команда или доступ к данным, вызвавшис прерывание по ошибке доступа, должны быть перезапущены.

В данном механизме ВП возможны ошибки доступа к таблицам L2 и L3, а также к самой требуемой странице данных и в случае промахов на всех трех уровнях запрос на получение данных или команд может обрабатываться достаточно долго. Поэтому одной из важных проблем является повышение быстродействия и производительности механизма ВП.

4.4. Повышение производительности механизма виртуальной памяти

Возвращаясь к использованию кэш-памяти в структуре иерархии памяти компьютерной системы можно попробовать встроить кэш в механизм виртуальной памяти, как показано на **Рис. 20**. Здесь данные кэша являются подмножеством данных в физической памяти и для доступа к ним необходимо использовать физический адрес. И здесь возникает проблема с быстродействием адресной трансляции. Так как таблица страниц находится в основной памяти, то преимущества использования кэш-памяти для данных и команд полностью исчезают по причине необходимости доступа к основной памяти для получения физического адреса страницы.

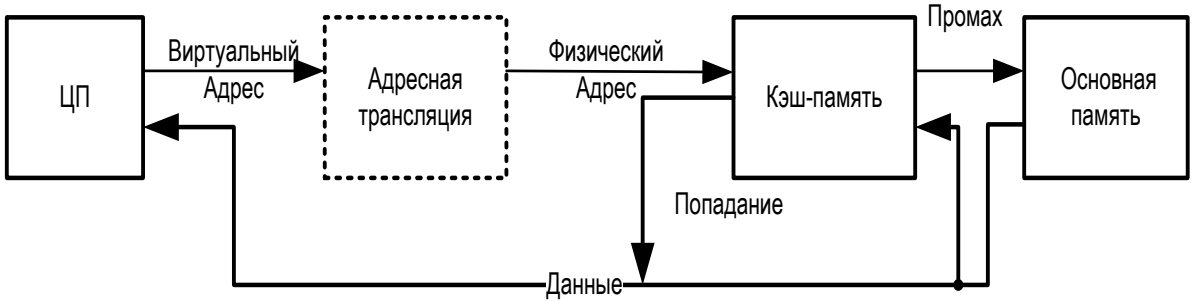


Рис. 20. Механизм виртуальной памяти с кэшем, использующим физические адреса.

Выходом из такой ситуации является введение в структуру еще одного специализированного кэша, в котором будут храниться недавно использованные элементы таблицы страниц, при повторном обращении к этим страницам доступ к таблице страниц в основной памяти не производится и физический адрес считывается из этого кэша. При этом задержка доступа к такому кэшу многократно меньше задержки доступа к основной памяти. Ввиду специализированности кэша для хранения только элементов таблицы страниц, его принято называть буфером быстрой трансляции (**Translation Look-aside Buffer, TLB**). Структура механизма виртуальной памяти с буфером быстрой трансляции (далее **ББТ**) представлена на **Рис. 21**.

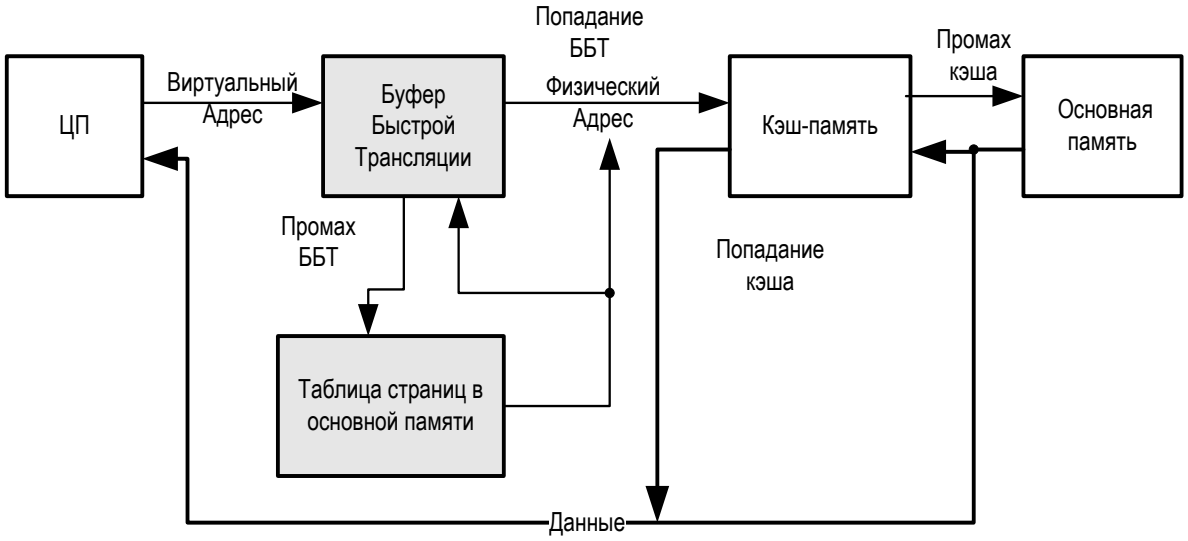


Рис. 21. Механизм виртуальной памяти с кэшем и буфером быстрой трансляции адресов

ББТ, как правило, имеет небольшие размеры в пределах от 64 до 256 строк и характеризуется высокой степенью ассоциативности, часто ББТ выполняются как полно-ассоциативные буферы. Каждая строка ББТ содержит виртуальный номер страницы для поиска, физический номер страницы и биты прав доступа из таблицы страниц в основной памяти. Упрощенная структура ББТ представлена на **Табл. 8**.

Табл. 8. Упрощенная структура Буфера Быстрой Трансляции (ББТ)

Виртуальный адрес начала страницы (эквивалент адресного тэга)	**Расположение страницы в основной памяти**	**Права доступа и достоверность данных**			
Виртуальный	Физически	Данные	Да	Дан	До

номер страницы (Поиск в буфере производится по виртуальному адресу)	й номер страницы	изменены Dirty	нные использо-ваны Use	ные достоверны Valid	ступ Access
04050607	5566788	0	1	1	1
……	……..	….	….	…..	….
12345678	678943A	1	1	0	0

Как работает механизм ВП с ББТ? ЦП генерирует виртуальный адрес, который подается на ББТ, который может быть полностью ассоциативным, частично-ассоциативным и буфером прямого отображения (как любая кэш-память). Далее выполнятся следующая последовательность:

1. Производится поиск по виртуальному адресу (номеру) страницы совершенно аналогично обычному кэшу, механизм поиска зависит от ассоциативности ББТ.

2. Если виртуальный адрес (ВА) ЦП совпадает с ВА одной из строк ББТ (произошло попадание), то ББТ возвращает физический адрес (номер) страницы и далее производится доступ к данным по иерархии памяти.

3. В случае промаха, необходимый элемент таблицы страниц должен быть извлечен из главной таблицы страниц в основной памяти и скопирован в ББТ

4. Для этого из ББТ удаляется наиболее давняя использованная строка и размещается новая строка с необходимым для адресной трансляции элементом таблицы страниц.

5. Производится адресная трансляция и физический адрес подается на кэш данных для теста на попадание/промах.

В связи с высокой локальностью обращений к памяти, использование ББТ уменьшает негативное влияние механизма ВП на среднее время доступа в иерархии памяти. Тем не менее, требуются некоторые

дополнительные меры для уменьшения времени трансляции и ее совмещения с доступом в кэш данных. Можно перечислить следующие стандартные способы:

- ЦП с механизмом ВП и ББТ должны генерировать адрес как можно раньше момента чтения (записи) данных для уменьшения числа тактов задержки при доступе в кэш-память,

- Доступ в кэш должен быть по возможности совмещен с доступом в ББТ и это возможно в связи с тем что биты смещения в странице (младшие биты адреса) могут быть использованы в качестве индекса для доступа к кэшу.

Однако это порождает некоторые проблемы, связанные с перекрытием поля индекса кэша и физического адреса. Это приводит к тому, что старшие биты индекса кэша должны быть получены из физического адреса в результате адресной трансляции, и совмещение процесса доступа к ББТ и кэшу становится невозможным. Поэтому это принуждает к использованию в комплексе с механизмом ВП кэшей прямого отображения небольшого объема или кэшей с высокой степенью ассоциативности, если мы хотим увеличить объем кэша.

ПРИМЕР НЕСОВМЕСТИМОСТИ КЭША И МЕХАНИЗМА ВИРТУАЛЬНОЙ ПАМЯТИ

К примеру, при использовании механизма ВП с 4 К байт страницами и размерностью 4 Г байт попробуем совместить его с кэшем прямого отображения объемом 8 Кбайт и размером блока 32 байт.

Структура виртуального адреса выглядит следующим образом:

31 Номер виртуальной/физической страницы 12	11 Смещение в странице 0

Структура адреса при доступе к кэшу имеет несколько отличную структуру:

31 Адресный тэг 13	12 Индекс 5	4 Смещение в строке 0

Как видно из приведенных форматов, бит 12 является старшим битом индекса и необходим для доступа к кэшу, в тоже время этот же бит 12 является частью номера страницы и должен быть определен результатом адресной трансляции. Это не позволяет совместить процесс адресной трансляции с

доступом в кэш по индексу, так как старший бит индекса будет неопределен до конца адресной трансляции. Можно предложить несколько вариантов выхода из сложившейся ситуации:

1) Увеличить размер страницы виртуальной памяти до 8 К байт;

2) Использовать двухканальный частично-ассоциативный кэш (уменьшив индекс на один бит)

3) Программное обеспечение может гарантировать, что бит 12 в виртуальном и физическом адресе будет иметь одно и тоже значение.

Промахи ББТ оказывают крайне негативное влияние на производительность ЦП, это приводит к дополнительным обращениям в память для замены строк ББТ. Что характерно, многие ЦП не могут обеспечить доступ ко всему пространству кэша L2 без промахов в ББТ.

Использование ББТ в целом повторяет стратегию использования кэш-памяти, если ЦП имеет раздельные кэши команд и данных (что является типичным), то для каждого кэша уровня L1 необходимо обеспечить раздельные ББТ, работающие в паре с кэшем. В следующем разделе подробно рассматривается комбинирование ББТ и двухуровневого кэша.

5. МНОГОУРОВНЕВАЯ ОРГАНИЗАЦИЯ ПАМЯТИ С ТРАНСЛЯЦИЕЙ АДРЕСОВ

5.1. Интеграция двухуровневой кэш-памяти в общую систему виртуальной памяти

Попробуем разобраться, каким образом можно скомбинировать двухуровневый кэш и систему виртуальной памяти. На **Рис. 22** представлена абстрактная структура адресации ЦП, использующего 48-разрядный виртуальный адрес, на основе которого генерируется 40-разрядный физический адрес, по которому адресуются данные в памяти и двухуровневом кэше. Размер страниц фиксированный и равен 8К байт. На диаграмме можно выделить два тракта:

1) Тракт преобразования адреса, включающий в себя буфер быстрой трансляции адресов (ББТ)

2) Тракт выборки данных, включающий в себя кэши двух уровней L1 и L2 .

Кэш первого уровня L1 имеет объем 8 К байт и построен на принципе прямого отображения так же как и кэш второго уровня L2 объемом 1М байт. Виртуальный адрес состоит из двух полей: поля виртуального номера страницы и поля смещения страницы (35 и 13 бит соответственно). Следует помнить, что поле смещения соответствует размеру страницы 2^{13}=8К байт. Виртуальный номер страницы должен быть послан в тракт преобразования адреса для получения из ББТ номера физического фрейма или физического адреса страницы. Поле смещения страницы посылается напрямую в тракт выборки данных и попадает непосредственно в кэш первого уровня.

На входе тракта преобразования адреса виртуальный номер страницы разделяется на поля индекса для выборки строки ББТ и поле адреса тэга для последующего сравнения с тэгом строки. На входе тракта доступа к данным смещение в странице также разделяется на поля смещения строки кэша и поле индекса для выборки строки кэша L1.

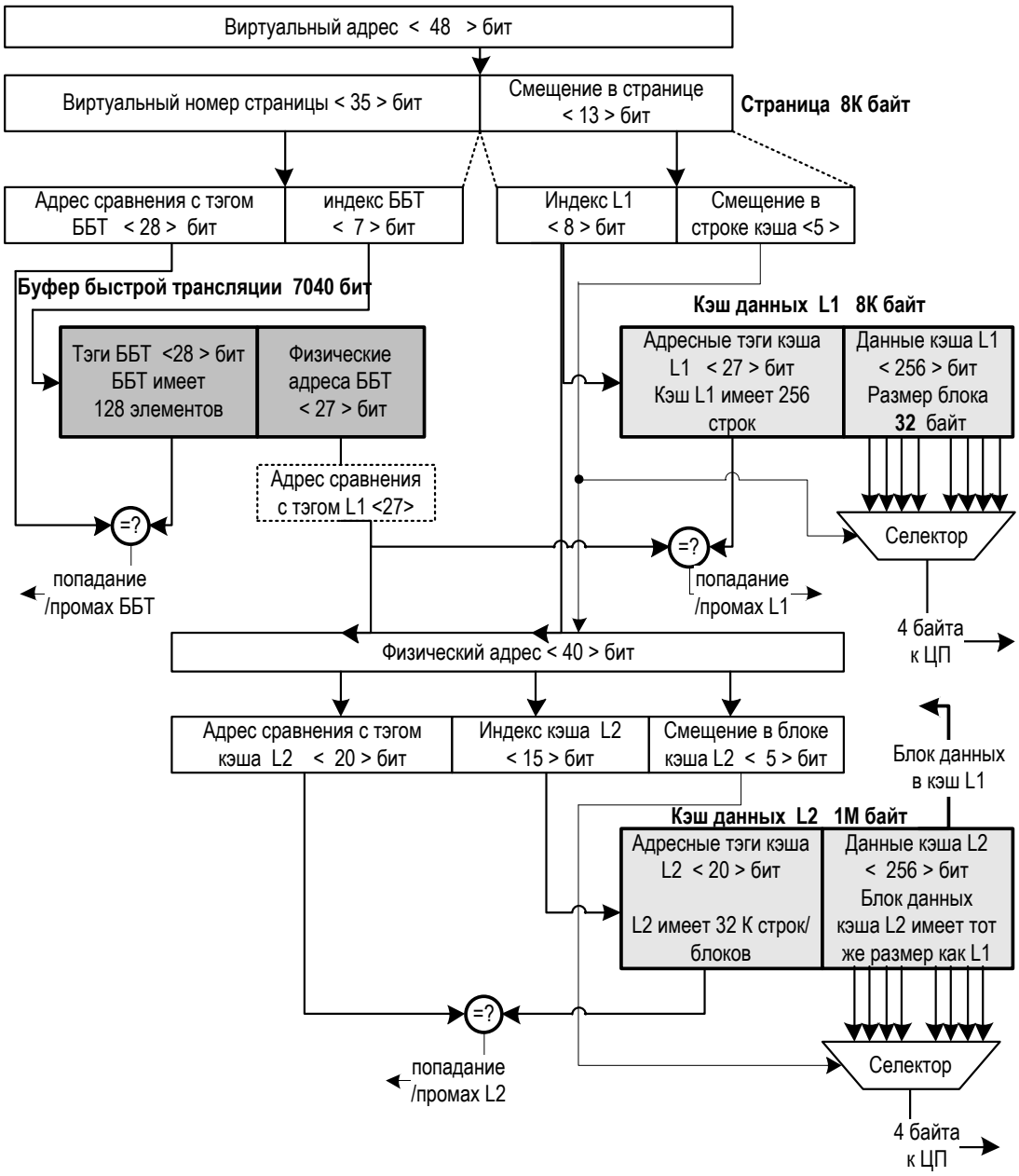


Рис. 22. Абстрактная структура двухуровневого кэша, комбинированного с ББТ адресов

Теперь рассмотрим как производится доступ к данным в такой структуре. Отметим вначале, что кэш L1 и L2 используют физические адреса и для достоверного доступа вначале необходимо произвести преобразование адресов. Последовательность работы механизма следующая:

1. Используя значение поля индекса из виртуального номера страницы, в ББТ производится выборка элемента и сравнение адресного тэга элемента с адресным тэгом виртуального номера страницы. В случае попадания ББТ выдает физический номер страницы, старшие биты которого подаются на кэш L1 для сравнения адресных тэгов.

2. В это же время в кэше L1 производится выборка строки по значению поля индекса. При этом одновременно считываются как сегмент данных, так и сегмент тэга, при этом сегмент данных не выдается до момента проверки считываемого значения из сегмента тэга на совпадение с адресным тэгом, считанным из ББТ.

3. В контроллере кэша L1 производится сравнение адресных тэга, полученного из ББТ с адресным тэгом, считанным из строки L1. В случае попадания данные выдаются в ЦП, в случае промаха номер физической страницы объединяется с полем смещения страницы и выдается в кэш-память второго уровня L2.

4. Кэш L2 имеет значительно больший объем при том же размере блока данных и поле индекса будет составлять 15 бит в отличие от кэша L1. Соотвественно уменьшится и размер адресного тэга, который составит 20 бит по сравнению с 27 бит в кэше L1.

5. Производится выборка строки L2 по значению поля индекса и сравнение считанного тэга со старщими битами адреса из поля тэга. При попадании данные выдаются в кэш L1, при промахе необходим доступ к основной памяти и загрузка данных в кэш L2 с последующим копированием в кэш L1.

Данная структура является наиболее простым вариантом реализации двухуровневой структуры кэш-памяти, комбинированной с ББТ и поддерживающей работу механизма виртуальной памяти. В реальных ЦП, как правило, используется частично-ассоциативный кэш с множеством банков, а ББТ часто реализуется как полностью ассоциативный блок памяти. Кроме того, кэши команд и данных уровня L1 выполняются как раздельные и не связанные блоки, каждый из которых использует собственную ББТ, так как особенности доступа к командам и данным сильно различаются.

Соответственно, при увеличении степени ассоциативности и сохранении прежнего объема как ББТ, так и кэшей обоих уровней размеры полей индекса уменьшатся пропорционально числу банков. Число компараторов также возрастет и будет равно числу банков каждого кэша.

В качестве упражнения предлагается модифицировать поля индексов и тэгов в случае размера виртуального адреса в 64 бита и размера физического адреса в 44 бит, как представлено на **Рис. 19**. Объем кэш-памяти и ББТ для упрощения можно оставить тем же самым, как и на **Рис. 22**. Предлагается затем сравнить результаты с кратким описанием подсистемы памяти ЦП Альфа 21264, представленной в следующей секции.

5.2. Иерархия памяти ЦП Альфа 21264

Альфа 21264 имеет раздельные тракт выборки команд и тракт чтения/записи данных с раздельными ББТ и блоками кэш-памяти уровня L1. На уровне L2 оба тракта объединяются и кэш L2 хранит как команды, так и данные.

На Рис. 23 приведена структура тракта выборки команд в ЦП Альфа 21264. Специфической особенностью является то, что кэш команд является виртуальным, т.е. при доступе в кэш команд L1 трансляции адресов не производдится. В некоторых случаях при исполнении служебных программ, строка кэша может содержать физический адрес, который отмечается в тэге специальным флагом. Другой особенностью является комбинирование кэша команд с механизмов предсказания переходов – предиктором доступа. На Рис. 23 проиллюстрировано объединение кэша команд с предиктором доступа. Структура тэга кэша команд отображает его функциональные возможности и включает следующие поля:

ASN – Address Space Number, номер виртуального адресного пространства, возможно 256 пространств для разных задач;

ASM – Address Space Match, совпадение адресного пространства, используется в комбинации с ASN;

PHY – Физический адрес в тэге, также может называться “**PAL code**”;

Valid – Строка с действительными данными;

KESU – Kernel-Executive-Supervisor-User, биты показывающие статус доступа кода программы, загруженного в строку кэша (Ядро – Загрузчик– Супервизор – Пользователь)

Parity – Четность тэга и четность блока данных

Адресный тэг – Виртуальный адресный тэг (если не установлен бит PHY)

Банк (следующей выборки) – Предсказанный банк следущей выборки из кэша команд (устанавливается и корректируется предиктором)

Строка (следующей выборки) – Предсказанная строка следующей выборки из кэша команд (устанавливается и корректируется предиктором)

Объем кэша данных составляет 1024 строки, разделенных на два банка частично-ассоциативной структурой, при этом поле индекса кэша будет 9 бит (параллельная выборка двух банков по 512 строк) . Размер блока данных 64 байта (16 команд), на поле смещения отводится 6 бит соотвественно. Важным моментом является перекрытие полей индекса и смещения строки (9+6=15 бит) с полем смещения страницы, которое составляет 13 бит. Для выборки команд из кэша нам необходимо 15 бит, поэтому 2 старших бита необходимо взять из виртуального номера страницы. Поэтому размер адресного тэга в строке уменьшен на 2 бита и равен 33 битам. В случае использования физических адресов в кэше команд возникает проблема трансляции этих двух битов (см. секцию 4.4) из виртуальных в физические перед выборкой строки кэша, что сильно замедляет работу. Поэтому основным режимом является использование виртуальных адресов, которые могут быть идентифицированы также дополнительными кодами номера адресного пространства ASN. Только в случае промаха в кэше команд L1 производится преобразование виртуального адреса в физический с использованием полно-ассоциативного ББТ, который генерирует физический адрес для доступа в буфер предварительной выборки, кэш L2 или в основную память.

Полно-ассоциативный ББТ содержит физические адреса (номера страниц) и каждый элемент может поддерживать страницу размером 8К, или непрерывные фрагменты 8х8К, 64х8К, 512х8К (8, 64 и 512 страниц соответственно). Адресный тэг здесь полный, так как доступ является полностью ассоциативным и индекс не используется. Для повышения скорости работы считываются все 128 элементов и на основе кода попадания (позиция компаратора, где произошло совпадение), мультиплексор выбирает нужный адрес.

Последним блоком является буфер предварительной выборки, который срабатывает в случае промаха кэша L1 и процесса адресной трансляции. Как только в результате промаха ББТ выдал физический адрес страницы, полный физический адрес (номер физической страницы и поле смещения в странице) подается на буфер предварительной

выборки для проверки на совпадение адресного тэга. Если есть попадание, то строка данных из буфера должна быть перемещена в кэш команд. В случае отсутствия данных в буфере необходимо считать данные из кэша L2 и полный физический адрес направляется туда. Кроме этого в случае промаха буфер автоматически добавляет к адресу промаха константу +64 и считывает еще одну строку в надежде, что программа будет также использовать последующие 16 команд . Буфер может считывать до 4 дополнительных блоков на каждый случай промаха при наличии места в буфере. При этом необходимо производить проверку на пересечение границы виртуальной страницы, если адрес выходит за пределы, то предварительная подкачка не производится.

Тракт выборки команд соединяется с мультиплексором адреса кэша L2, куда также заводятся адресные линии и линии данных из тракта чтения/записи данных.

Тракт чтения/записи данных отличается от тракта выборки команд последовательностью расположения блоков и режимом адресации кэша данных L1. Структура тракта представлена на рис.6.24. Кэш данных является физическим, а не виртуальным в отличие от кэша команд, поэтому ББТ должен преобразовать виртуальный адрес и только затем можно считать данные. ББТ тракта данных имеет точно такую же структуру и размеры как и ББТ в тракте выборки команд и работает без каких-либо отличий. Кэш данных также имеет аналогичный объем и двухканальную частично-ассоциативную структуру, как и кэш команд.

Имеются только отличия в полях тэга:

R-robin – Round-Robin allocation, бит управляющий размещением блоков данных в четном и нечетном банке частично-ассоциативного кэша (один на два блока/строки)

Dirty – Флаг модификации данных ЦП, данные должны быть скопированы в следующий уровень иерархии при удалении;

Modif – Флаг модификации блока данных в системной памяти или кэше уровня L2 другим агентом (другим ЦП или контроллером ПДП), используется для поддержки когерентности данных;

Valid – Строка с действительными данными;

Shared – Данные являются общими для нескольких ЦП;

Parity – Бит четности

Адресный тэг – Физический адрес блока данных

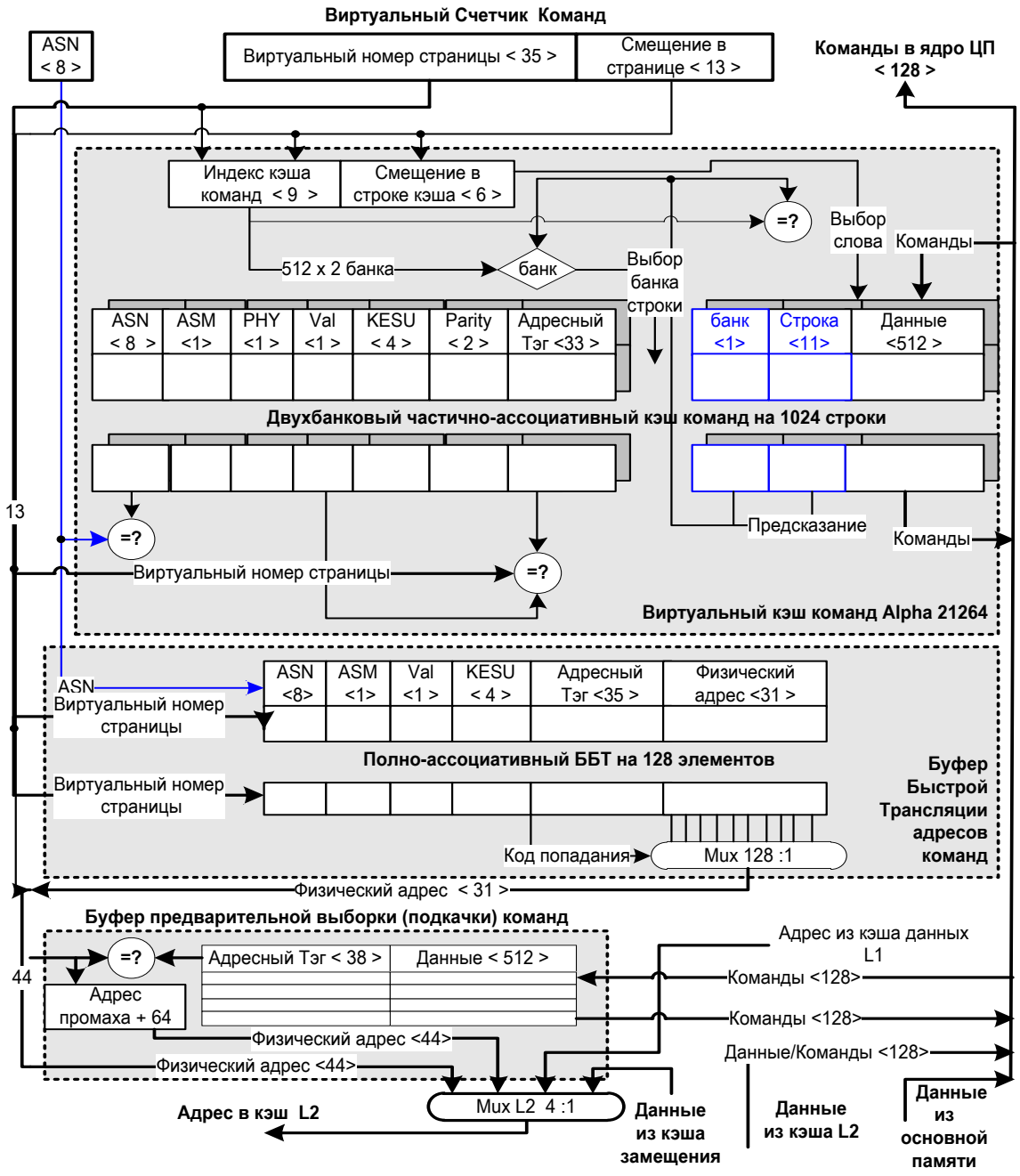


Рис. 23. Тракт выборки команд в ЦП Альфа 21264

В отличие от полностью виртуального кэша команд, кэш данных L1 индексируется виртуальным адресом, точнее его частью, совпадающей с полем смещения страницы. Но адресный тэг при этом является полным физическим адресом. Такой подход ведет к проблеме возникновения виртуальных «синонимов» данных из-за необходимости трансляции 2 старших бит адреса при индексировании кэша данных. При считывании

строки имеются 4 возможных варианта, из которых надо выбрать один после полного преобразования адреса. Это снижает скорость работы и для решения этой проблемы аппаратные средства разрешают иметь в кэше в конкретный момент времени только один из возможных вариантов, обеспечивая использование только виртуальных битов адреса для считывания строки, и последующей проверки совпадения физического адресного тэга с адресом, сгенерированным на основе содержимого элемента ББТ. Если контроллер кэша обнаруживает два или более синонимов с разными значениями старших битов индекса, то все кроме одного, отмечаются как недействительные.

Что происходит при промахе чтения данных в кэше данных L1? Контроллер должен заместить одну из строк кэша новыми данными, которые могут быть считаны из кэша L2 или из основной памяти. При этом данные из замещаемой строки копируются в кэш замещения (**victim buffer**) в надежде, что они могут понадобиться при последующих доступах (принцип локальности доступа в пространстве). Этот кэш замещения работает по принципу FIFO в режиме записи из основного кэша, и как обычный полно-ассоциативный кэш в режиме чтения в случае промаха в основном кэше. Размер кэша замещения небольшой и он содержит всего восемь строк. Если в случае промаха строка данных найдена в кэше замещения, то она может быть скопирована назад в основной кэш. Те строки кэша замещения, к которым не было повторного доступа, переносятся в основную память или кэш L2.

Отдельно следует рассмотреть ситуацию, когда происходят промахи в ББТ или ошибки доступа к странице. В ББТ обоих трактов выборки команд и чтения данных в таких случаях инициируют прерывания и вызывают специальные утилиты системного PAL кода, которые обрабатывают прерывание и загружают недостающие ссылки на физические адреса фрагментов программы и данных. Задержка при ошибке доступа является достаточно большой и поэтому крайне нежелательной.

Детали реализации кэш-памяти в ЦП Альфа 21264 рассмотрены в прилагаемых источниках литературы и технических описаниях.

К сожалению, после слияния компаний Hewlett-Packard и Compaq линия развития процессоров Альфа была приостановлена и ЦП Альфа 21464 не был выпущен на рынок в связи с реализацией совместного проекта Intel и Hewlett Packard по разработке ЦП Itanium IA-64. Особенности построения иерархии памяти в этом ЦП рассматриваются в следующем разделе.

Тем не менее ЦП Альфа является классическим примером отлично разработанной архитектуры, обеспечивающей максимальную производительность при минимально возможных издержках. Вероятно, архитектура этого ЦП будет интенсивно использоваться в образовательных целях, как пример реализации основных принципов современной компьютерной архитектуры.

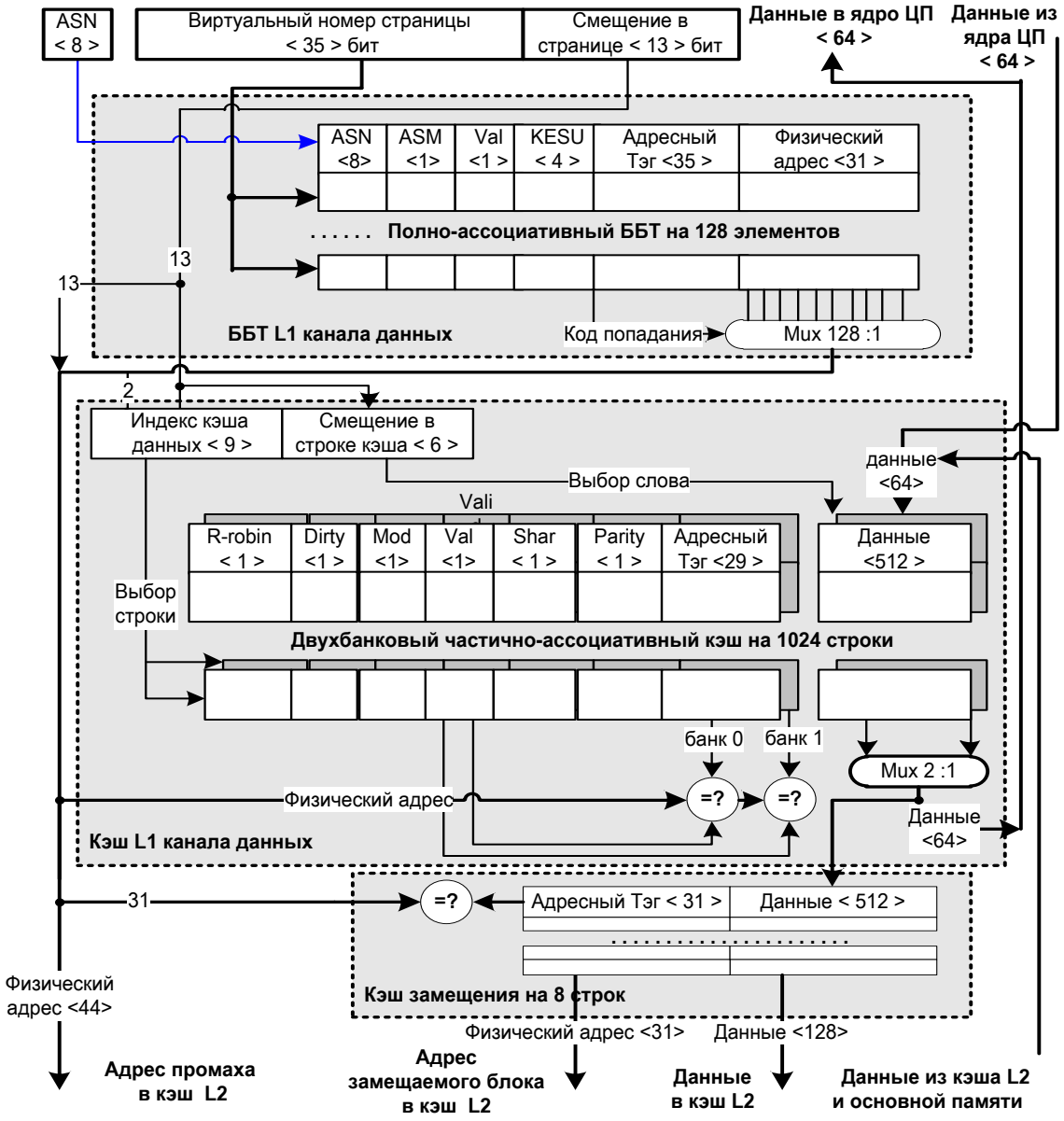


Рис. 24. Тракт чтения/записи данных в ЦП Альфа 21264

6. ПРИМЕРЫ ИЕРАРХИИ ПАМЯТИ СОВРЕМЕННЫХ МИКРОПРОЦЕССОРОВ

6.1. Иерархия памяти в процессорах семейства ARM

Рассмотрим 64-канальный кэш процессора ARM (Рис.25). Для сокращения стоимости памяти тэгов применяются строки, состоящие из четырех слов. На производительность это решение практически не влияет.

При реализации ассоциативного кэша требуется большое количество CAM-памяти тэгов, которая потребляет много энергии. CAM-память (Content Adressable Memory) – это такая RAM-память, в которой каждая ячейка имеет встроенный компаратор, что позволяет осуществлять параллельный поиск по всей памяти. Энергопотребление можно уменьшить, разбив память на блоки, однако тем самым мы уменьшаем ассоциативность.

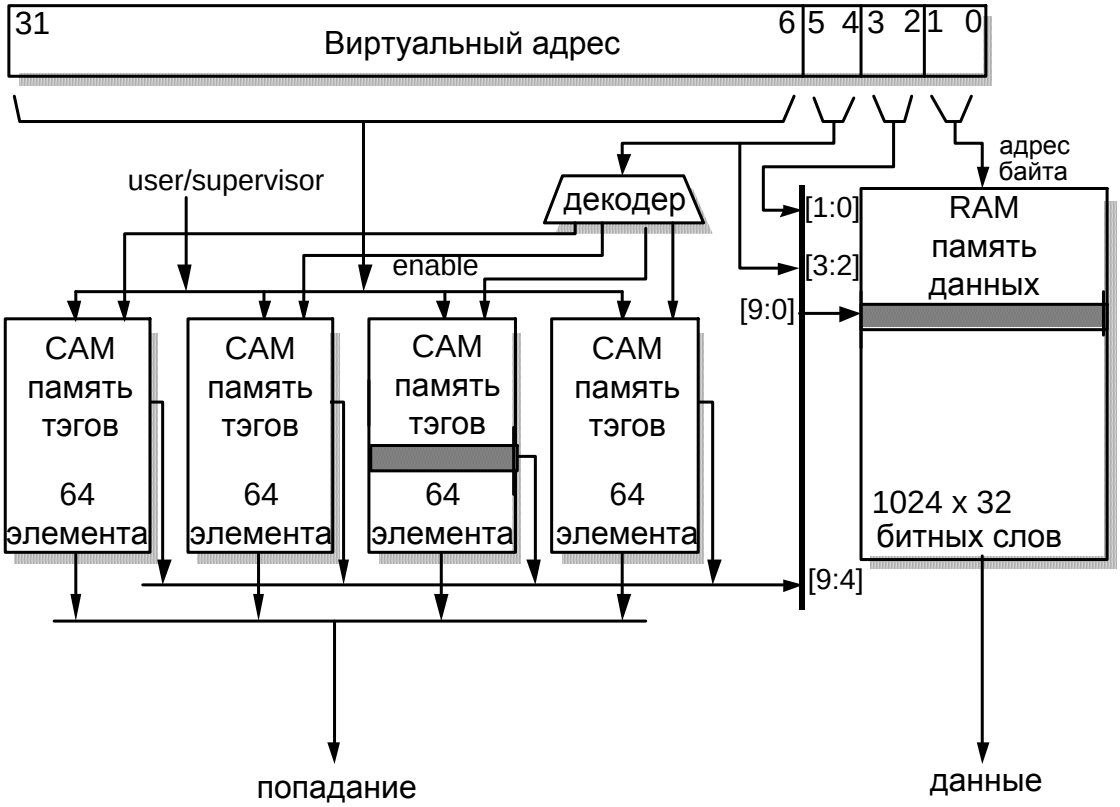


Рис. 25. Структура кэша процессора ARM3

Теперь коротко опишем организацию данного кэша. Младшие два бита виртуального адреса используется для выборки байта из 32-битного слова, следующие два бита – для выборки слова из строки кэша, и следующие два бита для выборки одного из четырех блоков тэгов. Оставшаяся часть виртуального адреса передается в выбранный блок тэгов (остальные блоки выключаются в целях экономии) для проверки нахождения данных в кэше. В случае попадания также формируется адрес данных в памяти данных кэша.

Для того чтобы показать как работает логика управления кэша, рассмотрим конечный автомат управления ARM600. В этом процессоре используются два тактовых генератора – быстрый при взаимодействии с кэшем или при записи в буфер записи, и медленный при обращениях к внешней памяти. Переключение между двумя источниками тактовых испульсов происходит динамически. Также следует заметить, что сами по себе они могут быть асинхронными. Однако, если медленные тактовые импульсы получены из быстрых, то уменьшаются затраты на синхронизацию.

Обычно процессор работает от быстрого источника, однако, в случае кэш-промаха происходит переключение на медленный истоник тактовых сигналов, и после следует обращение к памяти. Поскольку переключение между двумя источниками приводит к определенным накладным расходам на синхронизацию, перед тем как переключиться назад на быстрый источник, проводится проверка следующего адреса памяти.

Тепер рассмотрим управляющий конечный автомат, представленный на Рис.26.

После инициализации процессор попадает в состояние *Проверка тэга* и использует быстрый истоник тактовых импульсов. В зависимости от того, находятся ли данные в кэше, возможно следющие варианты развития событий:

До тех пор пока адреса не последовательны (non-sequential), нет ошибки MMU (MMU fault), происходит буферизованная запись (buffered write), либо читаемые данные находятся в кэше, автомат будет находиться в состоянии *Проверка тэга*, а данные будут возвращаться или записываться каждый цикл.

Если происходит последовательное чтение (из той же строки кэша), либо последовательная буферизованная запись (buffered write), то происходит переход в состояние *Ускоренное обращение*, где доступ к данным осуществляется без проверки тегов и использования MMU (в целях экономии энергии). Данные также читаются или записываются каждый цикл.

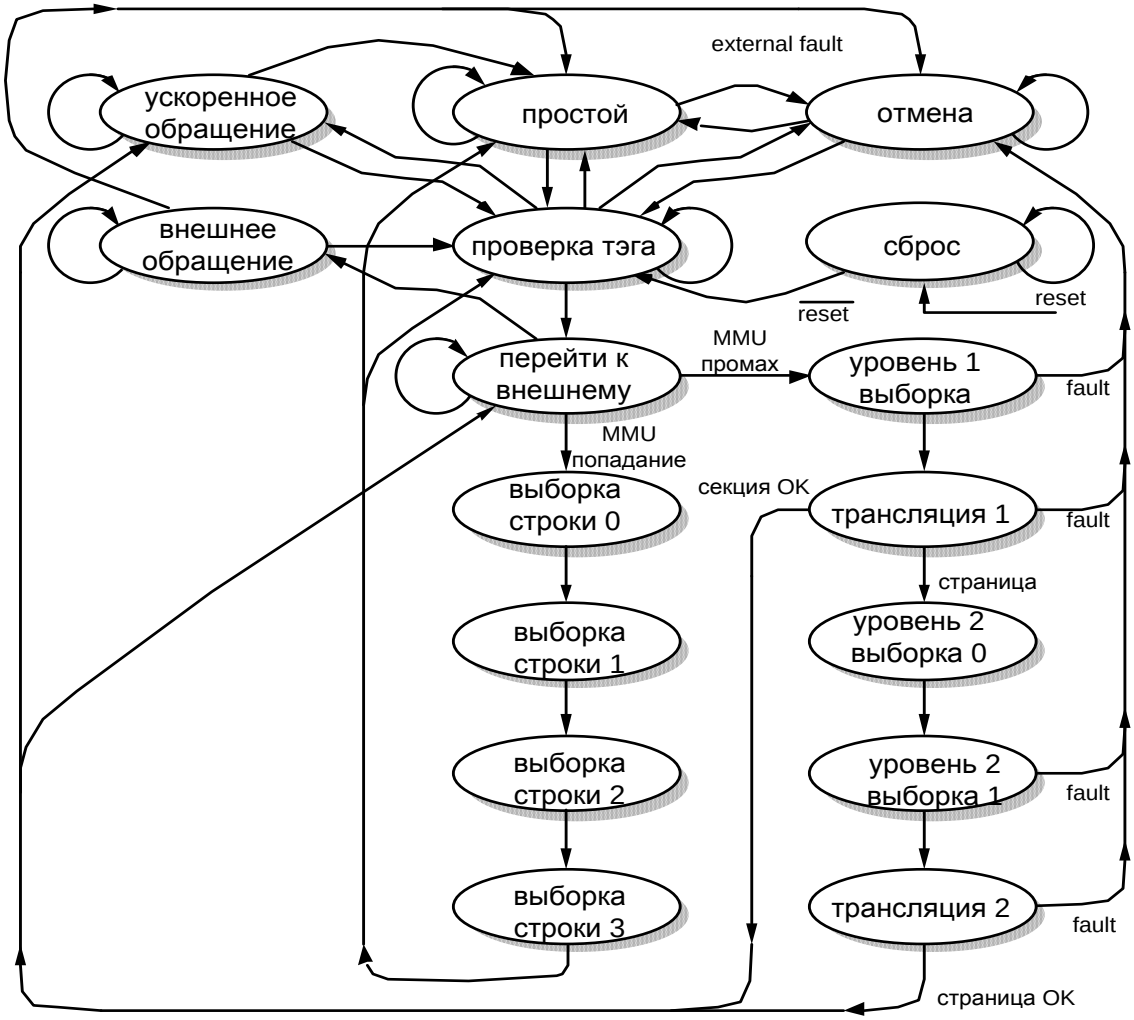


Рис. 26. Логика управляющего автомата кэша ARM

Если данных нет в кэше, либо происходит не буферизованная запись (unbuffered write), то требуется доступ к внешней по отношению к кэшу памяти. Просиходит переход к состоянию *Перейти к внешнему*. Далее чтение или запись осуществляются в состоянии *Внешнее обращение*. При чтении в кэш происходит необходимой для трансляции страниц/секций информации (если она еще не находится в MMU) и далее строки, состоящей из учетверенного слова (quad-word).

Если процессору не нужная память, то происходит переход в состояние *Простой*.

В некоторых случаях может происходить прерывание процесса доступа к памяти. Эти случаи соответствуют состоянию *Отмена*. В случае чтения данных, не находящихся в кэше, либо при

небуферизованной записи, прерывание может просходить из-за внешних событий.

Далее следует кратко описать трансляцию адресов. Трансляция нового виртуального адреса всегда начинается выборки первого уроня (first-level fetch). Правда, в данный момент в целях упрощения мы исключаем из рассмотрения TLB. При трансляции адресов используется базовый адрес таблицы адресов, хранящийся в регистре 2 сопроцессора CP15. Его биты [31:14] объединяются с битами [31:20] виртуального адреса для формирования адреса памяти (Рис.27), который испольуется для доступа к дескриптору первого уровня.

Дескриптор первого уровня может являться как дескриптором сегмента, так и указателем на таблицу страниц второго уровня. Это зависит от двух младших битов: «11» и «01» - «тонкая» и «грубая» таблица страниц второго уровня соответственно (second-level fine page table и second-level coarse page table), «10» - дескриптор секции, «00» - дескриптор, вызывающий ошибку трансляции (translation fault). Далее происходит дальнейшая трансляция секции или страницы.

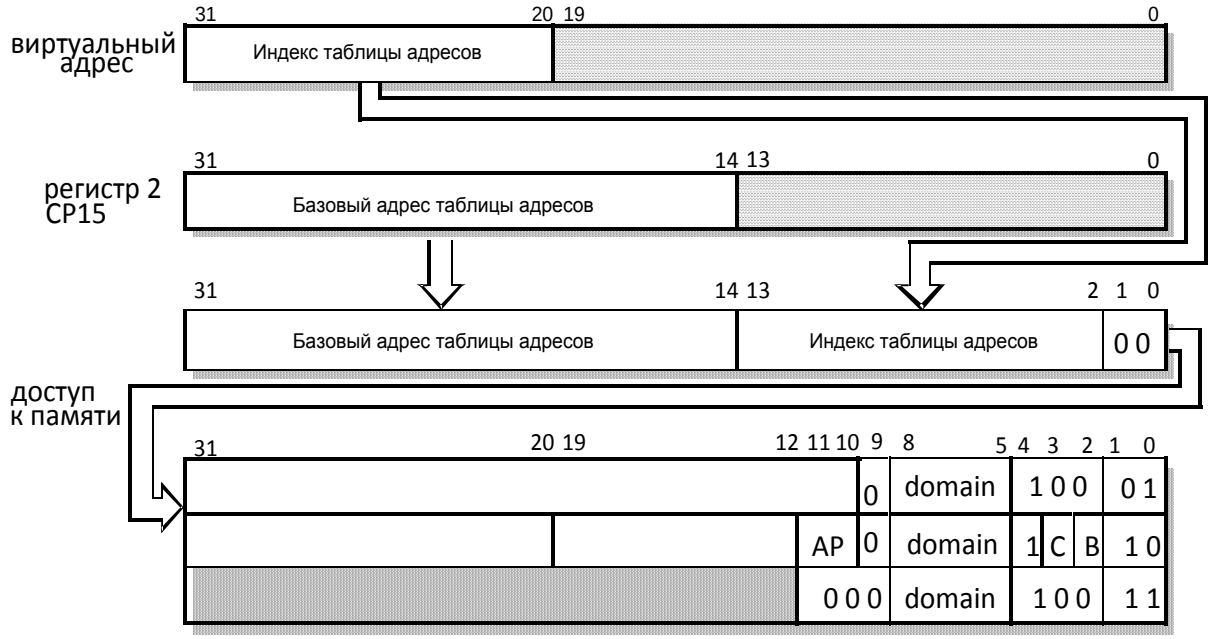


Рис. 27. Трансляция адресов в устройстве управления памятью процессора ARM

На рисунке 28 приведена схема проверки прав доступа в устройстве управления памятью процессора ARM.

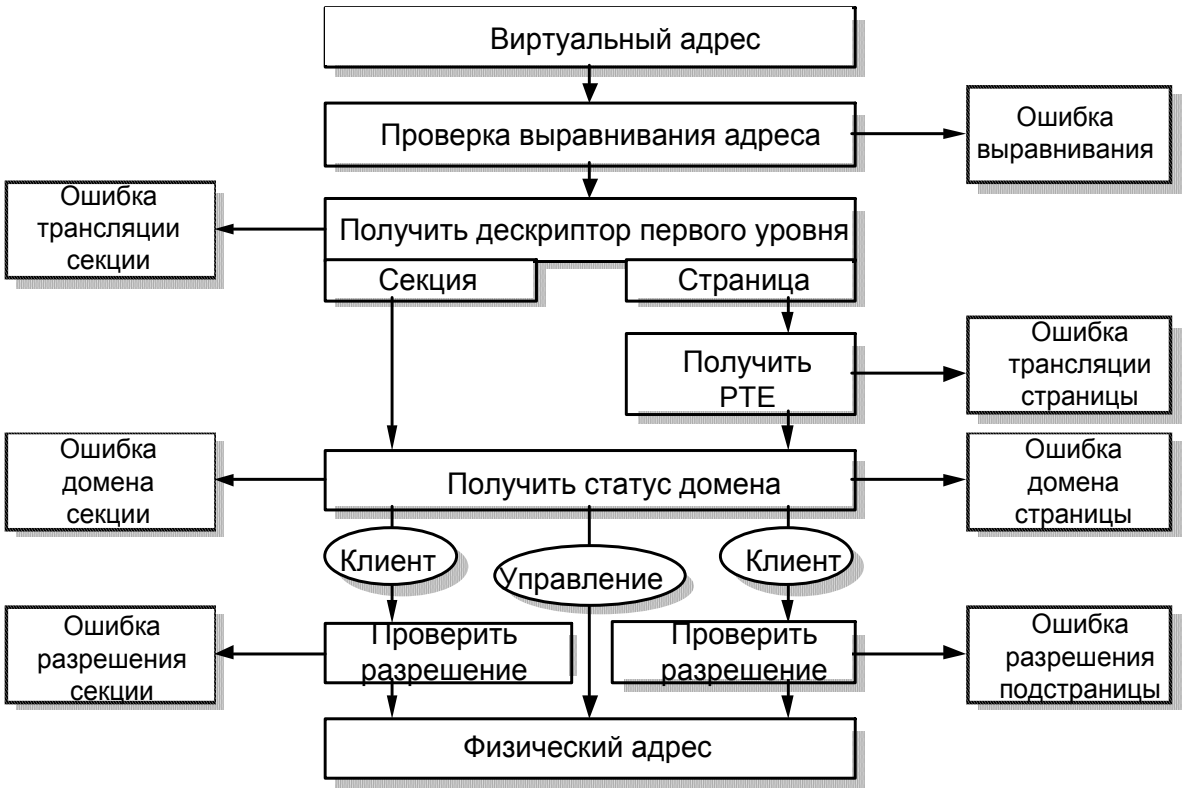


Рис. 28. Схема проверки прав доступа в устройстве управления памятью процессора ARM

6.2. Иерархия памяти в ЦП Intel Pentium P4 и AMD Opteron

Оба рассматриваемых ЦП относятся к классу массовых платформ для персональных компьютеров, рабочих станций и серверов. Хотя оба ЦП реализуют базовую систему команд IA-32, имеется существенная разница в построении иерархии памяти и возможностях адресации.

Если Opteron уже является 64-разрядным процессором, то Pentium P4 остается классическим 32-разрядным процессором с с расширением общего адресного пространства до 36 бит, ограничивающим каждую отдельную программу в пространстве 4 Гбайт. В тоже время Opteron не использует в полной мере возможности 64-разрядных адресных регистров, оставляя часть битов зарезервированными для будущих модификаций. В **Табл. 9** приводится сравнение механизмов адресной трансляции обоих ЦП. Opteron имеет двухуровневый ББТ для каждого тракта в отличие от одноуровневой у Pentium P4. В случае промаха (ошибки доступа к странице), специальный аппаратный блок

обеспечивает загрузку элемента ББТ в обоих ЦП. Оба ЦП поддерживают одинаковые форматы доступа к страницам.

Табл. 9. Сравнение механизмов адресной трансляции в ЦП Pentium P4 и Opteron

Характеристика	**Intel Pentium P4**	**AMD Opteron**
Виртуальный адрес	32 бита	48 бит
Физический адрес	36 бит	40 бит
Размер страницы	4 КВ, 2/4 МВ	4 КВ, 2/4 МВ
Организация ББТ	1 ББТ для тракта команд и 1 ББТ для тракта данных Оба имеют 4-канальную частично-ассоциативную структуру Оба используют алгоритм замещения псевдо-LRU Оба имеют по 128 элементов, в ББТ тракта данных 64 элемента для 4К страниц и 64 элемента для больших страниц Промахи обоих ББТ обрабатываются аппаратными средствами	2 ББТ (L1, L2) для тракта команд и 2 ББТ (L1, L2) для тракта данных Оба ББТ L1 имеют полностью ассоциативную структуру с алгоритмом замещения LRU Оба ББТ уровня L2 имеют 4-канальную частично-ассоциативную структуру с круговым алгоритмом замещения LRU Оба ББТ L1 имеют по 40 элементов Оба ББТ L2 имеют по 512 элементов Промахи обоих трактов ББТ обрабатываются аппаратными средствами

Механизм иерархии памяти в Opteron является классическим и в значительной степени похож на аналогичную структуру в ЦП Альфа, в обоих случаях имеются кэши команд и данных. В ЦП Pentium P4 в тракте выборки команд реализована несколько другая схема, связанная с декодированием и преобразованием команд IA-32 в микрооперации формата RISC-ядра. Это сделано для повышения эффективности ядра ЦП. На **Рис. 29** представлена упрощенная структура ЦП Pentium P4 в которой особый интерес вызывает кэш трасс выполняемых последовательностей или цепочек микрокоманд. В оригинале его называют кэшем трасс микрокоманд (Trace cache), который заменяет традиционный кэш команд уровня L1. Команды формата IA-32

выбираются из кэша второго уровня L2 специальным блоком предварительной подкачки (Instruction Prefetch), скомбинированным с собственным ББТ. При этом используется блок предсказания переходов, содержащий данные о 4096 возможных переходах в основной программе. После того, как команды IA-32 декодируются, они превращаются в одну микрокоманду или цепочку из нескольких микрокоманд формата RISC, и затем записываются в кэш цепочек микрокоманд, где они упаковываются в строках последовательно по 6 микроопераций согласно порядка исполнения. Причем фрагменты циклов кодируются наиболее эффективно, ликвидируя необходимость повторной предварительной выборки и декодирования команд. Таким образом вместо кэша команд L1, копируюшего область памяти с размещенной там программой в оригинальных машинных кодах IA-32, в ЦП Pentium P4 реализован кэш трасс с цепочками микрокоманд, которые являются результатом декодирования команд IA-32, размещенных в унифицированном кэше L2.

Кэш трасс микрокоманд совмещен со специальным блоком памяти микропрограмм для поддержки многоцикловых команд IA-32, в случае декодирования такой команды управление передается блоку микропрограммного управления, который выполняет микропрограмму, соответствующую многоцикловой команде. По завершении исполнения микропрограммы кэш трасс микрокоманд получает управление назад и продолжает снабжать ядро ЦП микрокомандами в обычном режиме.

Благодаря использованию кэша трасс микрокоманд резко снижается нагрузка на блок предварительной выборки и команды IA-32 можно выбирать и декодировать по одной за такт. Блок предсказания переходов построен на основе специального буфера с адресами переходов (**Branch Target Buffer - BTB**), который имеет 4К элементов, содержащих адрес пресказанного перехода. Для доступа в буфер используются 12 младших бит программного счетчика, который управляет предварительной выборкой.

Аналогичный буфер с предсказанными адресами переходов имеет блок предсказания кэша трасс микрокоманд, он содержит 512 элементов и адресуется микропрограммным блоком управления. Так как адресация в кэше трасс микрокоманд не имеет ничего общего с программным счетчиком и доступом к памяти, то нет необходимости в трансляции адресов на этом уровне. Трансляция адресов производится только при доступе в кэш L2 при предварительной выборке команд формата IA-32, поэтому ББТ команд (**Instruction TLB**) используется только в паре с кэшем L2.

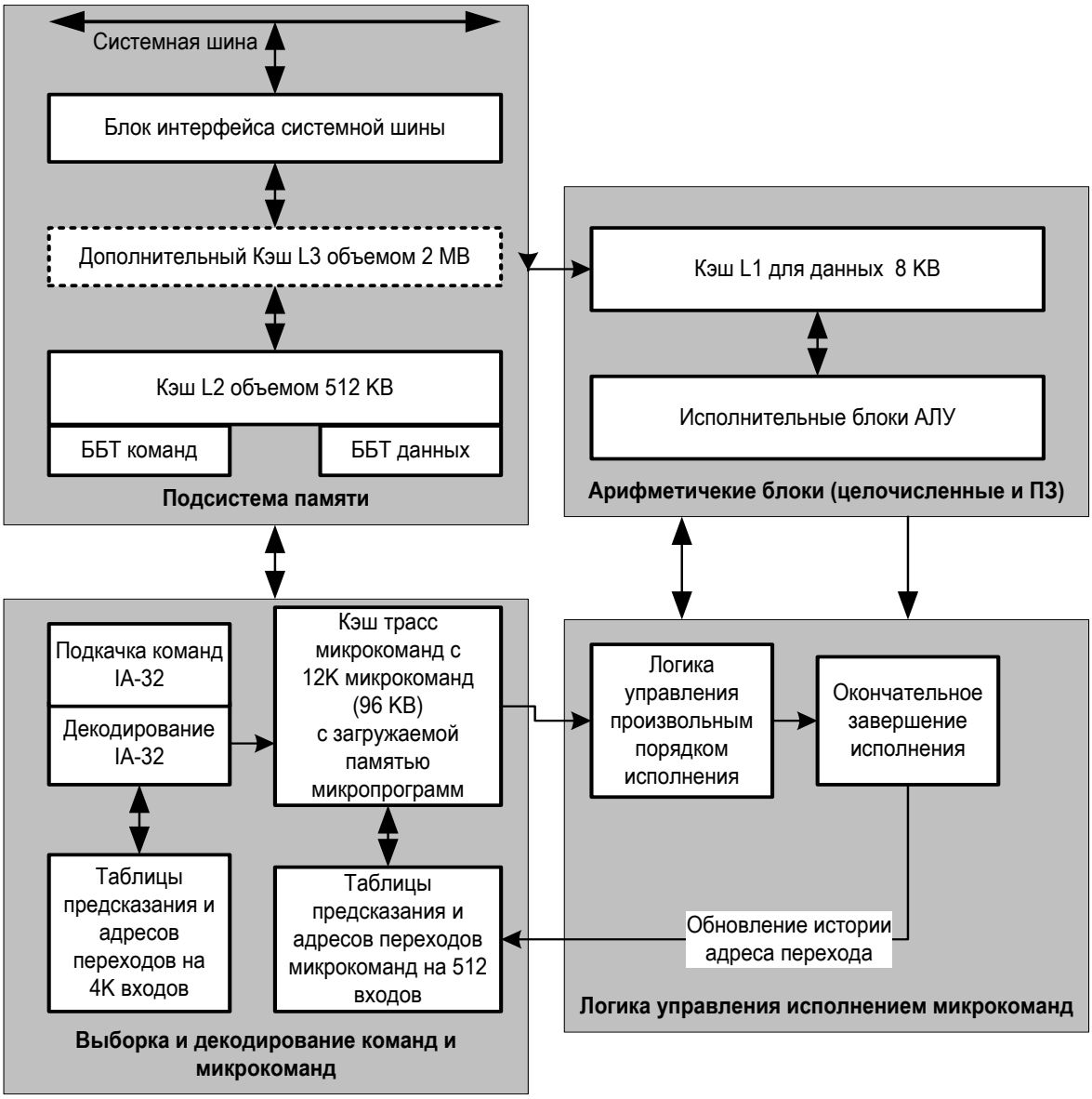


Рис. 29. Упрощенная структура ЦП Intel Pentium P4

Кэш данных имеет очень небольшой объем в 8 КВ и подключен к исполнительному арифметическому блоку, использование сквозной записи позволяет избежать многих промахов при записи, потенциально возникающих из-за малого размера кэша. Требование высокого быстродействия включают использование ББТ L1 и кэш данных L1 работает в физическом пространстве адресов, преобразование адресов в физический производится только в случае промаха и доступа в кэш L2, который является унифицированным для команд и данных. В **Табл. 10**

приведено сравнение реализации механизма кэш-памяти ЦП Пентиум с ЦП Оптерон.

Табл. 10 . Сравнение кэшей L1 и L2 ЦП Pentium P4 и Opteron

Характеристика	**Intel Pentium P4**	**AMD Opteron**
Организация кэша L1	Отдельные блоки кэш-памяти для команд и данных	Отдельные блоки кэш-памяти для команд и данных
Объем кэша L1	8 КВ физический кэш данных, 96 КВ трассирущий кэш для RISC микрокоманд (12 К RISC микрокоманд)	64 КВ физический кэш данных 64 КВ физический кэш команд
Ассоциативность кэша L1	4-канальный частично-ассоциативный кэш данных 8-канальный частично-ассоциативный трассирующий кэш	2-канальный частично-ассоциативный
Алгоритм замещения кэша L1	Приближенный LRU	LRU
Размер блока кэша L1	64 байт	64 байт
Алгоритм записи кэша L1	Сквозная запись	Обратная запись
Организация кэша L2	Унифицированный для команд и данных	Унифицированный для команд и данных
Объем кэша L2	256-512 КВ	1024 КВ (1 МВ)
Ассоциативность кэша L2	8-канальный частично-ассоциативный	16-канальный частично-ассоциативный
Алгоритм замещения кэша L2	Приближенный LRU	Приближенный LRU
Размер блока кэша L2	Блок 64 байт, сектор 128 байт	64 байт
Алгоритм записи кэша L2	Обратная запись	Обратная запись
Организация кэша L3 (только модель Xeon)	Унифицированный для команд и данных	Отсутствует

Объем кэша L3	512 KB – 1MB	Отсутствует
Ассоциативность кэша L3	8-канальный частично-ассоциативный	Отсутствует
Алгоритм замещения кэша L3	Приближенный LRU	Отсутствует
Размер блока кэша L3	Блок 64 байт, сектор 128 байт	Отсутствует

ЦП Оптерон использует более классическую организацию кэш-памяти с симметричными трактами выборки команд и чтения/записи данных на уровне L1, которые затем объединяются в унифицированный кэш L2, использующий раздельные ББТ для выборки команд и доступа к данным. Оба кэша команд и данных уровня L1 сблокированы с собственными быстродействующими полностью ассоциативными ББТ на 40 элементов, обеспечивающими трансляцию виртуальных адресов в физические. Кэш данных использует обратную запись, так как имеет достаточно большой объем. Кэш команд L1 содержит команды в формате IA-32 и выборка производится блоком предварительной выборки, также использующего предсказание переходов. Далее команды декодируются с разделением на быстрые микрокоманды и микропрограммы, которые исполняются с помошью блока микропрограммного управления. Дополнительная информация по построению этого ЦП может быть найдена на веб-сайте компании AMD.

6.3. Иерархия памяти в ЦП Intel-HP Itanium и Эльбрус E2K

Новым поколением 64-разрядных ЦП является семейство Itanium - совместная разработка Intel и HP. Данный ЦП, как и Эльбрус E2K, относится к классу EPIC (**Explicit Parallelism Instruction Coding**) в отличие от классических суперскалярных ЦП, рассмотренных ранее. Эти архитектуры являются развитием архитектур VLIW (**Very Long Instruction Word** – Сверхдлинное командное слово (макс: Командное Слово Большого Размера или Большой Длины) и имеют свою специфику требований к иерархии памяти, которая должна обеспечивать минимальное время доступа для поддержки эффективности.

На **Рис. 30** приведена структура иерархии кэш-памяти ЦП Itanium 2 с трехуровневой кэш-памятью и двухуровневыми ББТ тракта команд и тракта данных. Кэши команд и данных уровня L1 имеют практически одинаковую структуру с небольшим различием в числе портов доступа. Спецификой этих кэшей является их комбинирование с ББТ уровня L1,

когда блоки тэгов ББТ и кэша собраны в общий блок с целью уменьшения задержки доступа до одного такта. Структура и детали работы кэша такого типа будут рассмотрены ниже. Спецификой кэша данных L1 является его использование только для хранения целочисленных данных, все остальные типы данных хранятся на других уровнях.

ALAT (Advanced Load Address Table) – Таблица адресов предварительной загрузки данных является специальным кэшем, который позволяет реализовать спекулятивную обработку данных. Он содержит информацию обо всех спекулятивных загрузках данных из памяти, выполненных ЦП и всех операциях записи по синонимам этих адресов. В какой-то степени эта таблица является аналогом станций резервирования, используемых в суперскалярных машинах и позволяет динамическое разрешение зависимости по данным, которые неизвестны на этапе компиляции. Эта структура является полностью ассоциативной и содержит 32 элемента, может обрабатывать одновременно две операции загрузки и две операции записи. Каждый элемент содержит поле для номера регистра, куда были загружены данные и физический адрес загрузки данных. Запись, сделанная по физическому адресу, перекрывающему физические адреса в одном или нескольких элементах таблицы аннулирует содержание элементов таблицы, и содержание регистров не может быть использовано.

Если выполняется команда, которая требует загрузки данных с определенного физического адреса и действительная ссылка на таковые имеется в таблице, то данные считываются из регистра без задержки в том же такте. Это позволяет иметь нулевую задержку при спекулятивной загрузке. Если действительной ссылки не найдено, то команда загрузки выполняется заново и команда использования этих данных также выполняется заново после получения данных.

Унифицированный кэш второго уровня L2 содержит как команды, так и все типы данных, включая целые и вещественные числа, семафоры и таблицы страниц. Для трансляции адресов используются раздельные ББТ для трактов команд и данных, работающие по классической схеме «Трансляция адреса – Доступ в кэш». Промахи ББТ всех уровней обрабатываются аппаратным блоком загрузки страниц (**HW Page Walker**).

Кэш третьего уровня L3 аналогичен кэшу второго уровня за исключением большего объема, также он не может содержать данных для команд обработки семафоров, что является исключительно прерогативой кэша L2. Кэш L3 сблокирован с интерфейсом системной шины, через которую осуществляется доступ в основную память.

В **Табл. 11** приведены основные характеристики модулей кэш-памяти ЦП Itanium.

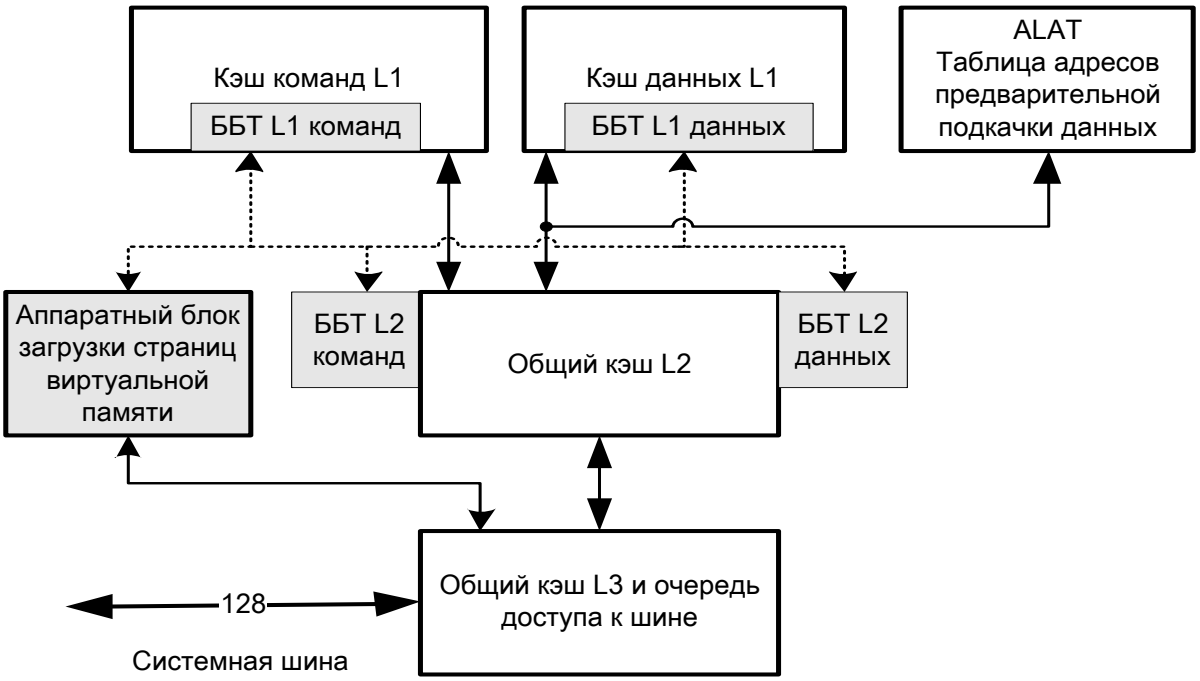


Рис. 30. Иерархия кэш-памяти в ЦП Intel-HP Itanium 2

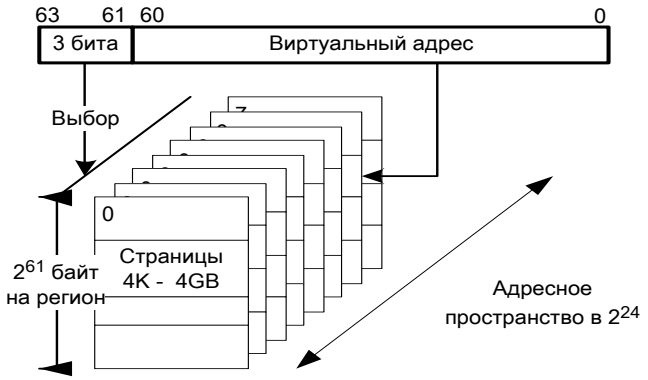
Можно отметить очень большой совокупный объем кэш-памяти, расположенной на кристалле и малое время доступа кэшей L1, непосредственно определяющих производительность ЦП в целочисленных операциях. Для обеспечения высокой пропускной способности кэш L2 построен на основе 16 банков, которые обеспечивают параллельный доступ для 4 вещественных чисел одновременно с доступом для обоих кэшей уровня L1, кэша L3 и системной памяти.

Табл. 11. Характеристики кэш-памяти в ЦП Itanium 2

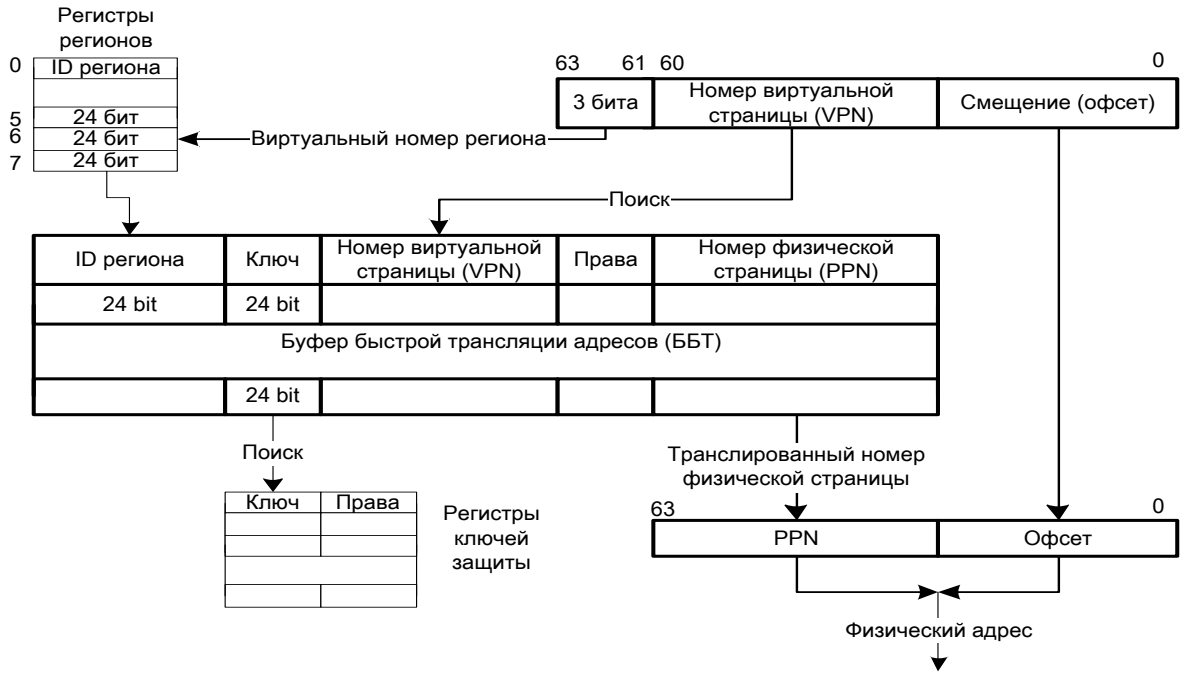
Характеристика кэш-памяти	**Кэш команд L1**	**Кэш данных L1**	**Унифицированный кэш L2**	**Унифицированный кэш L3**
Размер (объем)	16 КВ	16 КВ	256 КВ	1.5 – 3.0 МВ
Размер строки (блока)	64 байт	64 байт	128 байт	128 байт
Индексирование и тип тэгов	Физический индекс, Физический тэг в ББТ	Физический индекс, Физический тэг в ББТ	Физический индекс, Физический тэг	Физический индекс, Физический тэг
Задержка	1 такт/ 6	1 такт	5-9 тактов для	До 12 тактов

данных, блокирование	команд, Блокирующий или неблокирующий режим	Блокирующий или неблокирующий режим	целочисленных данных, +1 такт для вещественных данных Неблокирующий	для всех типов данных Неблокирующий
Степень ассоциативности и число банков	частично-ассоциативный 4-банка	частично-ассоциативный 4 банка	частично-ассоциативный, 16 банков	частично-ассоциативный 12-банков
Число портов памяти тэгов	1R 1W	2R 2W	4R 2W	1RW
Число портов данных	1R 1W	2R 2W	1R 1W x 16 банков	1RW
Типы данных в кэше	Только команды (6 команд за такт)	Только целочисленные данные	Команды, целочисленные и вещественные данные, семафоры, таблица страниц	Команды, целочисленные и вещественные данные, таблица страниц
Алгоритм записи	Нет	Сквозная запись без копирования в L1	Обратная запись с копированием в L2	Обратная запись с копированием

Механизм виртуальной памяти ЦП Itanium обеспечивает защиту данных для множества программ и является более сложным, чем рассмотренный ранее механизм в ЦП Альфа. На Рис. 31 представлена структура виртуального адресного пространства в ЦП Itanium, которое подразделяется на восемь регионов, каждый из которых может быть выбран из 2^{24} общего числа виртуальных адресных пространств, определяемых значением 24-разрядных регистров региона. ББТ имеет специальные поля для номера адресного пространства (региона), которое сравнивается при доступе в ББТ. Дополнительным полем является поле ключа, обеспечивающее защиту данных от несанкционированного доступа. В этом случае реализуется двухступенчатый поиск сначала в ББТ, а затем по значению ключа в в файле регистра защиты данных.



А) Виртуальное адресное пространство в процессоре Intel-HP Itanium



Б) Защита памяти через механизм ВП в процессоре Intel-HP Itanium

Рис. 31. Механизм защиты памяти и трансляции адресов в ЦП Intel-HP Itanium

Основные характеристики механизма трансляции адресов ЦП Itanium приведены в **Табл. 12**. Все ББТ являются полноассоциативными и ББТ первого уровня сблокированы с кэш-памятью. Часть элементов ББТ второго уровня могут работать как фиксированные регистры трансляции, т.е. их содержание не может быть удалено из ББТ в процессе замещения физических адресов. Это позволяет хранить в ББТ наиболее часто используемые страницы. Другой особенностью является поддержка

страниц переменного размера в ББТ второго уровня, что уменьшает число требуемых элементов в ББТ.

Табл. 12. Характеристики механизма трансляции адресов в ЦП Itanium 2

Характеристика механизма	**ББТ команд L1 ITLB 1**	**ББТ данных L1 DTLB 1**	**ББТ команд L2 ITLB 2**	**ББТ данных L2 DTLB 2**
Число элементов ББТ	32	32	128	128
Степень ассоциативности	Полно-ассоциативный	Полно-ассоциативный	Полно-ассоциативный	Полно-ассоциативный
Размеры поддерживаемых страниц	4 КВ	4 КВ, x4 КВ	4 КВ до 4 ГВ	4 КВ до 4 ГВ
Максимальное число регистров трансляции	Нет	Нет	64	64
Число портов ББТ	1R 1W	2R 1W	1R 1W	2R 2W

Особенностью ББТ первого уровня является их объединение с кэш-памятью первого уровня (**Prevalidated Tag Cache**), который можно назвать транслирующим кэшем, так как процесс адресной трансляции фактически перекрывается с выборкой данных из кэша.

Структура транслирующего кэша L1 ЦП Itanium 2 представлена на **Рис. 32**, где имеются два блока тэгов: тэги с виртуальным адресом ББТ и тэги с действительным указателем на строку ББТ (**prevalidated tags**) в 4 банках кэша данных. Доступ к обоим блокам тэгов производится одновременно, старшие биты виртуального адреса используются для доступа к тэгам ББТ, младшие биты в качестве индекса используются для доступа к 4 банкам кэша данных. Ввиду небольшого числа элементов ББТ для указателя необходимо только 5 бит на каждый банк, указатели всех четырех банков подаются на компаратор, куда также поступает номер элемента ББТ, где произошло совпадение старших битов виртуального адреса. Номер элемента ББТ сравнивается со всеми четырьмя указателями из кэша, при наличии совпадения имеется факт попадания и номер банка выдается на выходной мультиплексор данных частично-ассоциативного кэша, имеющего 4 банка. Управление таким кэшем несколько отличается от обычного, когда в тэге размещается явный физический адрес, но в целом никаких особых сложностей нет.

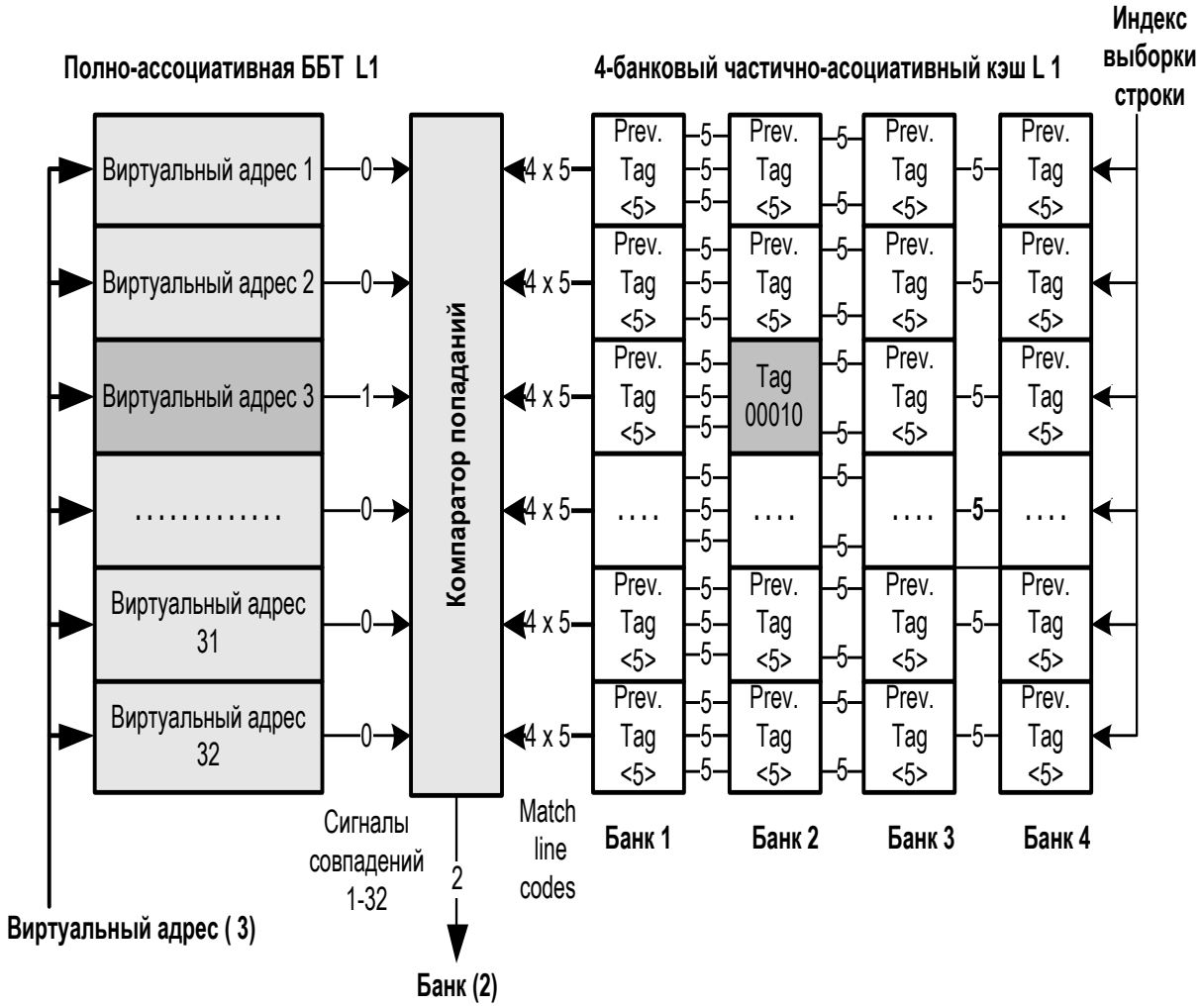


Рис. 32. Транслирующий кэш L1 в ЦП Itanium 2

Теперь можно перейти к интеграции иерархии памяти и исполнительных устройств в архитектуре Itanium 2. На **Рис. 33** представлена конфигурация памяти и блоков обработки целых и вещественных числовых данных. Входы M0 и M1 являются входами чтения данных из кэша L1D в блок целочисленной арифметики. Следует отметить, что блок целочисленной арифметики является единственным клиентом кэша L1D. Это кэш является транслирующим, объединяя ББТ и тэги кэша в один блок, управляющий доступом к массиву данных L1D. Входы M2 и M3 являются входами записи данных в кэш (обычно только в L2), в особых случаях данные копируются также в L1D с использованием буфера записи (**store buffer**) и ББТ L2D для генерации физического тэга для транслирующего кэша данных L1D. Так как унифицированный кэш L2 имеет много клиентов в отличие от L1, то все адреса записи данных

сохраняются в буфере заполнения (**Fill buffer**) в ожидании резервированной строки для данных, которые в свою очередь ожидают в буфере записи. Как только кэш контроллер выделил строку, данные из буфера записи и физический адресный тэг из буфера заполнения копируются в сегмент данных и сегмент тэга кэша L2.

Блок вещественной арифметики может читать данные только из кэша L2, соответственно необходимо обеспечить многопортовость за счет множества независимых банков памяти, так кроме этого данные должны читаться из кэша L3 и буфера записи. Кэш L3 имеет наибольший объем и скомбинирован с интерфейсом доступа к памяти, все промахи из L1 и L2 попадают в L3 и в большинстве случаев данные возвращаются на верхние уровни иерархии с приемлемой задержкой порядка 12-15 тактов.

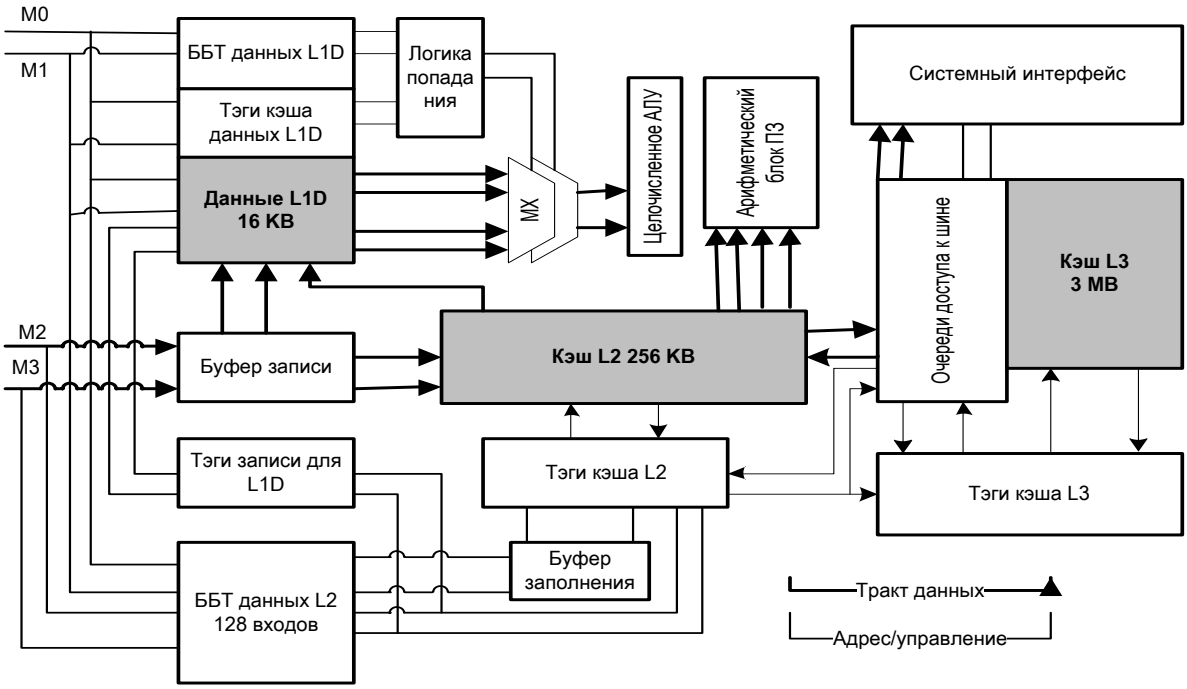


Рис. 33. Интеграция блоков памяти и исполнительных устройств в ЦП Itanium 2

Российский ЦП Эльбрус Е2К концептуально близок к ЦП Itanium 2, только был разработан значительно раньше на основе опыта разработки суперкомпьютеров «Эльбрус». Упрощенная структура иерархии памяти Е2К представлена на **Рис. 34**, где имеется два уровня иерархии кэша с унифицированным кэшем L2 для команд и данных, и раздельными кэшами команд и данных уровня L1. Каждый кэш снабжен собственным ББТ для трансляции адресов. Если тракт выборки команд имеет классическое исполнение с 64 КВ кэшем и 64-элементным полно-

ассоциативным ББТ, то спецификой тракта выборки данных является дублирование 8 КВ кэша данных L1 для двух исполнительных блоков, причем каждый из них имеет собственный 16-элементный полно-ассоциативный ББТ. Данные всегда пишутся в оба кэша, чтение производится в зависимости от команд, подаваемых на исполнительные блоки. При этом нет различия между блоками целочисленной и вещественной арифметики, как в ЦП Itanium. Блок управления памятью имеет собственный частично-ассоциативный 4-канальный ББТ, обеспечивающий трансляцию адресов для кэша данных L2 или системной памяти. Кэш данных L2 имеет объем 256 КВ, является частично-ассоциативным 2-канальным и должен поддерживать многопортовый доступ для клиентов. Следует также отметить наличие специального буфера для выборки массивов, что унаследовано от суперкомпьютерных предков E2K.

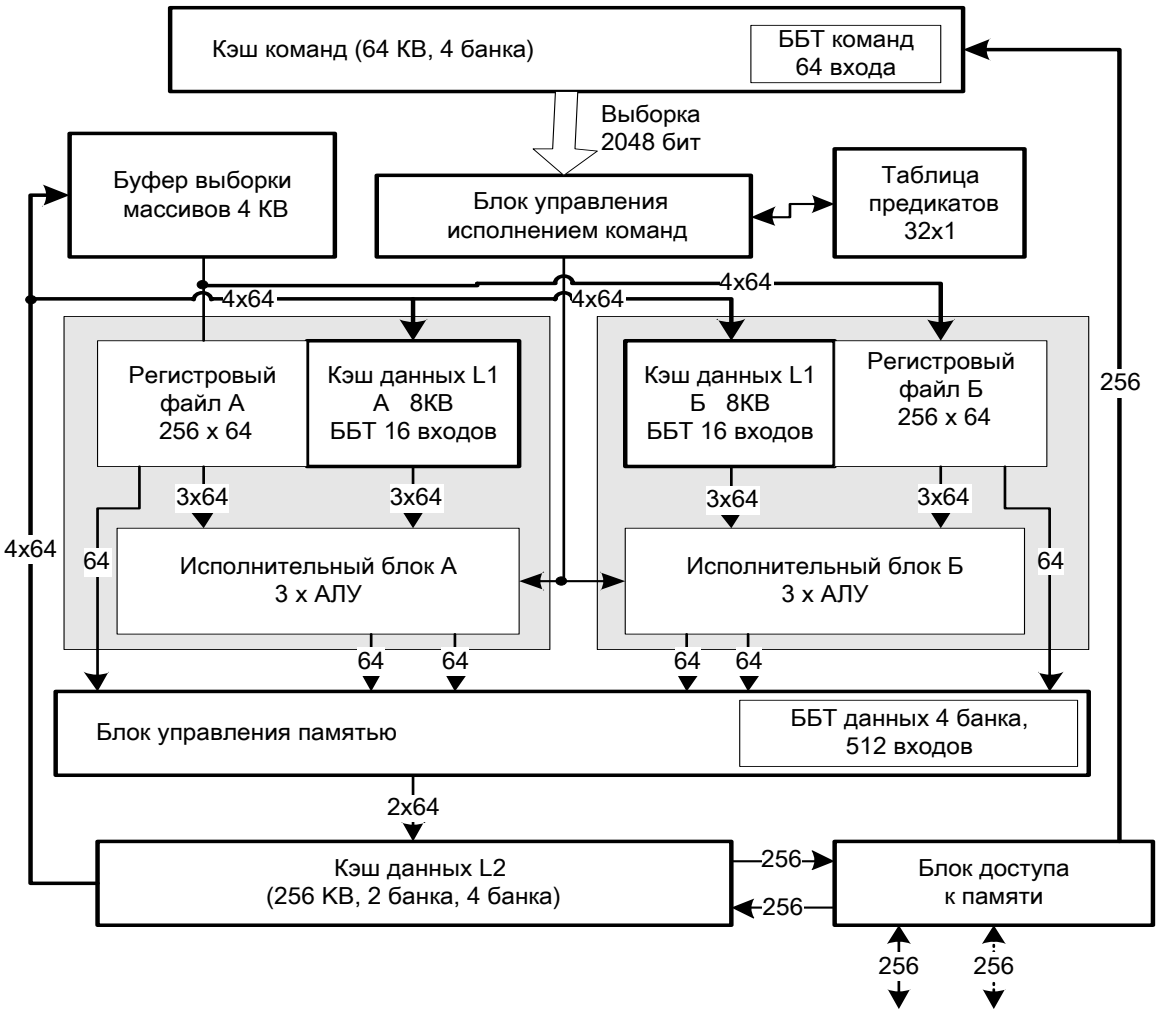


Рис. 34. Иерархия памяти в ЦП Эльбрус E2K.

Детальная информация по архитектуре E2K представлена в приложении, кроме того, можно получить дополнительную информацию на веб-сайте разработчиков.

7. КРАТКИЙ ОБЗОР ОРГАНИЗАЦИИ И МОДУЛЕЙ, ИСПОЛЬЗУЕМЫХ В ИЕРАРХИИ ПАМЯТИ

7.1. Организация интерфейса в иерархии памяти

Возвращаясь к структуре иерархии памяти на Рис. 1, рассмотрим способы организации интерфейса между уровнями памяти для обеспечения максимальной пропускной способности. Существует несколько способов организации, которые применяются в зависимости от требований прикладной области компьютерной системы, Рис. 35:

1) Интерфейс шириной в одно машинное слово через все уровни,
2) Широкий интерфейс в несколько слов между основной памятью и кэшем, между кэшем L1 и L2,
3) Интерфейс с использованием многобанковой памяти и «расслоением» доступа, называют также доступом с чередованием банков
4) Интерфейс с использованием многобанковой памяти с независимыми контроллерами на каждый банк (многоканальная память).

Интерфейс шириной в одно машинное слово (32 или 64 бит) в зависимости от типа ЦП является базовой версией и наиболее прост в исполнении, см. Рис. 35**а**. Однако такая версия интерфейса порождает проблемы с задержкой промаха, которая будет иметь наибольшую величину для такого соединения. Так как строка кэш-памяти может состоять из определенного числа слов, то нужно соотвественное число тактов шины интерфейса для передачи данных.

Если предположить, что 4 такта необходимы для посылки адреса, 4 такта для посылки/получения слова данных и 24 такта для доступа к медленной динамической памяти, то задержка промаха для кэша L1 с 4 словами в строке будет T_miss(L1) = 4 *(4+4+24)=128 тактов. Пропускная способность интерфейса памяти будет 32/128 = ¼ байт за такт.

Ширина интерфейса в одно слово нужна только для уровня {ЦП ←→Кэш L1}, на остальных уровнях иерархии {Кэш L1 ←→ Кэш L2 ←→ Основная память} ширину интерфейса можно увеличить, к примеру в четыре раза до 128 бит, Рис. 35**б**. При этом вместо четырех последовательных циклов доступа для выборки строки кэша, будет необходим только один (4+4+24)=32 такта. Пропускная способность интерфейса будет 32/32=1 байт за такт, что в четыре раза выше. Увеличение ширины шина основной памяти и кэша приводит к необходимости использования быстродействующих мультиплексоров на выходе кэша L1, так как ЦП нужно считывать только одно слово из передаваемой строки.

Еще одной проблемой создаваемой основной памятью с широким интерфейсом является рост минимального шага при увеличении объема памяти пользователем, шаг увеличивается кратно росту ширины интерфейса относительно слова ЦП. Дополнительные проблемы с широким интерфейсом могут возникнуть при использовании аппаратной коррекции ошибок, когда код коррекции рассчитывается на всю ширину доступа, каждый раз нужно считывать и записывать группу слов, даже если изменения проводятся только в одном из них.

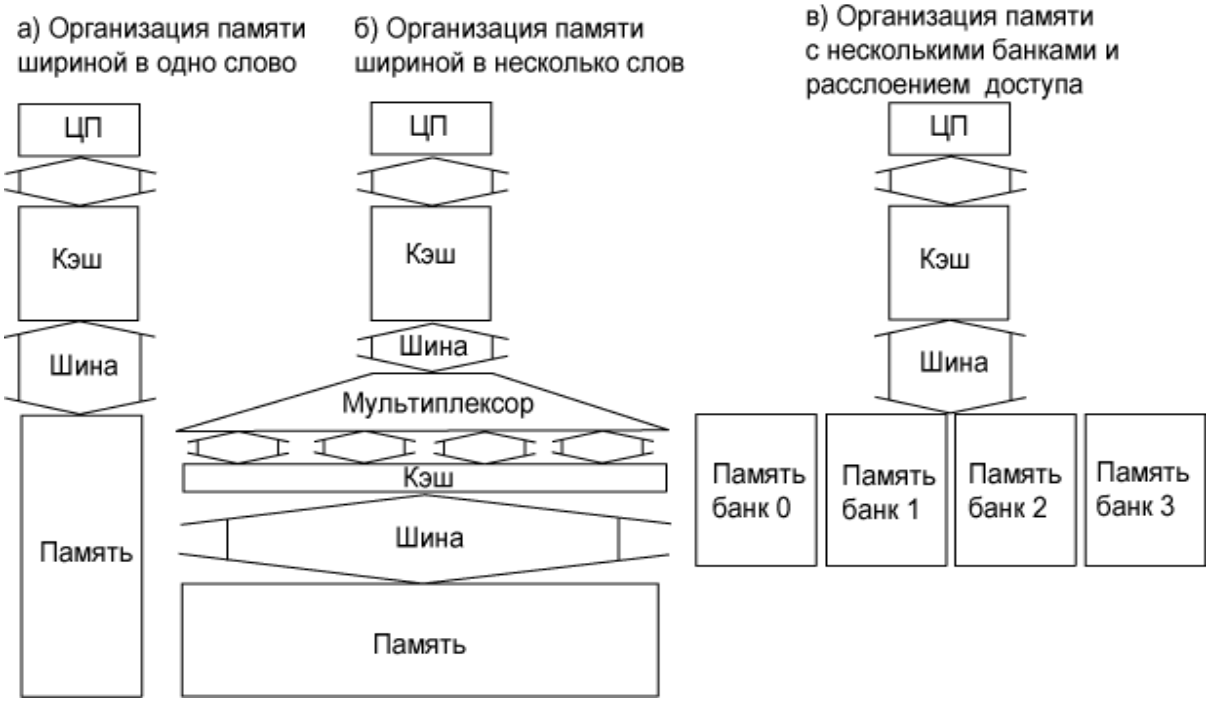


Рис. 35. Увеличение пропускной способности памяти за счет ширины интерфейса

Компромиссным вариантом является использование многоканальной памяти с расслоением доступа (**Interleaved access**) , Рис. 35**в**. Здесь основная память разделяется на несколько банков с последовательным чередованием адресов, Рис. 36**а**. Слово с адресом 0 располагается в банке 0, с адресом 1- в банке 1, с адресом 2 – банке 2, с адресом 3 в банке 3 и затем все повторяется в той же последовательности. Модуль отображения адресов в банки памяти называется коэффициентом расслоения (**interleaving factor**). Все банки имеют ширину в одно слово и если одновременно послать адрес во все 4 банка, то время доступа доступа сократится до величины (4+ 4*4 +24)=44 такта против 128 тактов в изначальной базовой версии. При этом не нужно иметь широкую шину данных, так как данные из банков выбранных одновременно, считываются в четырех последовательных группах тактов по четыре каждый.

Такая организация памяти с расслоением доступа также дает преимущества при записи, если производится запись в только в один из банков, то если следующая запись производится в другой банк, то нет необходимости ждать завершения предыдущей записи. Использование основной памяти с расслоением доступа особенно эффективно, когда размер строки кэша совпадает с шагом ширины банков памяти и каждый раз, когда кэш либо читает данные или производит запись, доступ производится ко всем банкам памяти, что в максимальной степени использует пропускную способность такого интерфейса памяти.

Другим вариантом построения интерфейса памяти является использование независимых банков памяти, каждый из которых имеет собственный контроллер доступа, Рис. 36**б**. Специфика такого подхода к построению интерфейса обусловлена супескалярными ЦП, которые имеют неблокирующие кэши с несколькими паралелльными трактами, а также блоки прямого доступа, которые являются клиентами интерфейса доступа к памяти. При этом промахи чтения кэша могут обрабатываться в одном тракте, запись производиться через другой, а контроллер прямого доступа может записывать данные через третий тракт. Такое построение интерфейса памяти является типичным для современных суперскалярных ЦП и графических процессоров.

Теперь кратко рассмотрим основные элементы, которые используются при построении современных подсистем памяти.

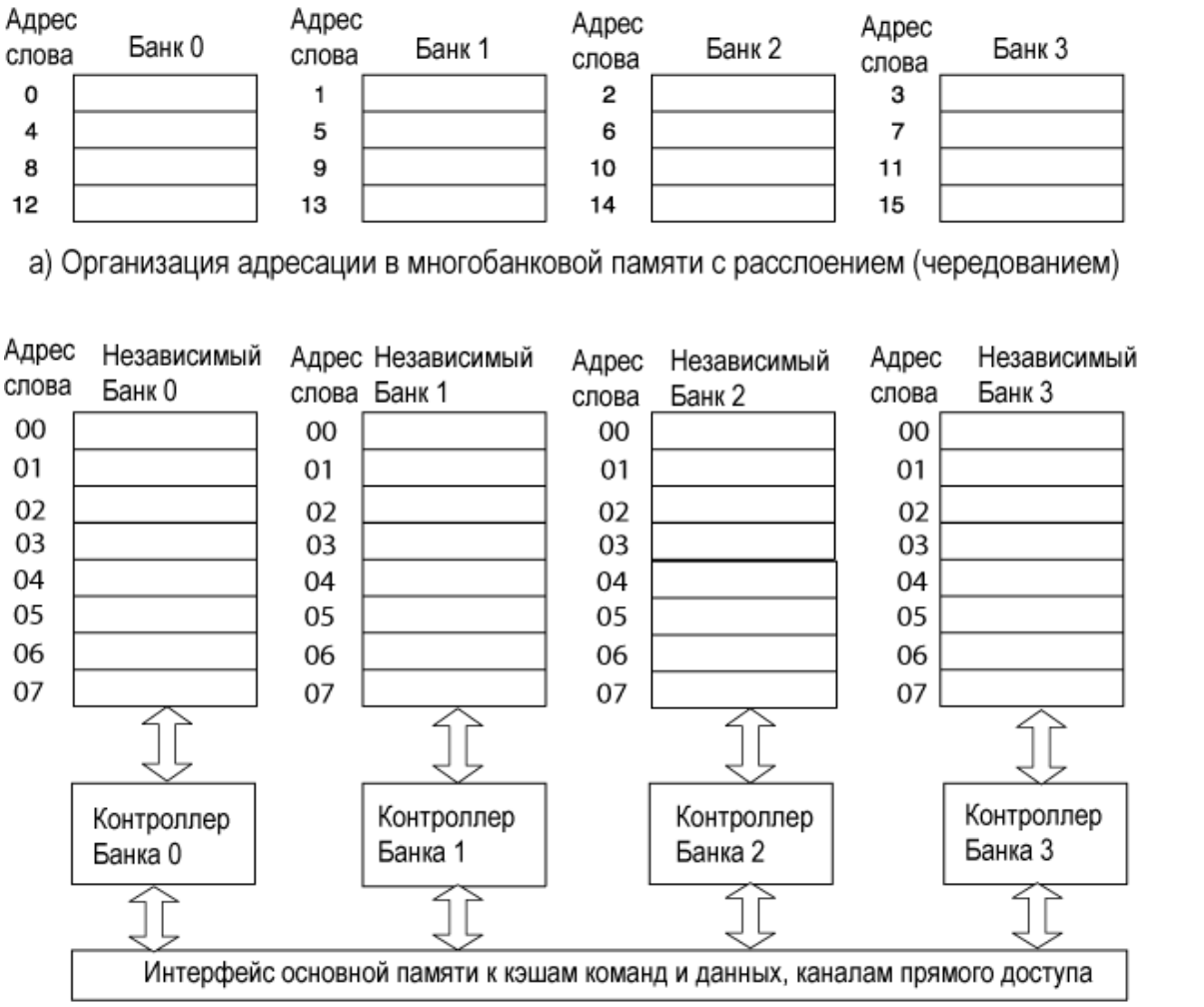
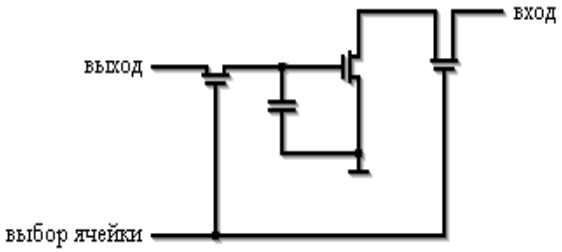


Рис. 36. Организация интерфейса с многобанковой памятью.

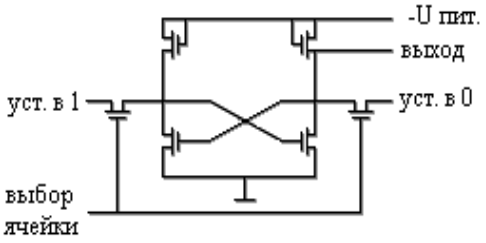
7.2. Модули динамической памяти DRAM

Ядро микросхемы динамической памяти состоит из множества элементов, каждый из которых хранит всего один бит информации. На физическом уровне элементы объединяются в прямоугольную матрицу, горизонтальные линейки которой называются строками (**Row**), а вертикальные - столбцами (**Column**), которые объединены в страницы (**Page**).

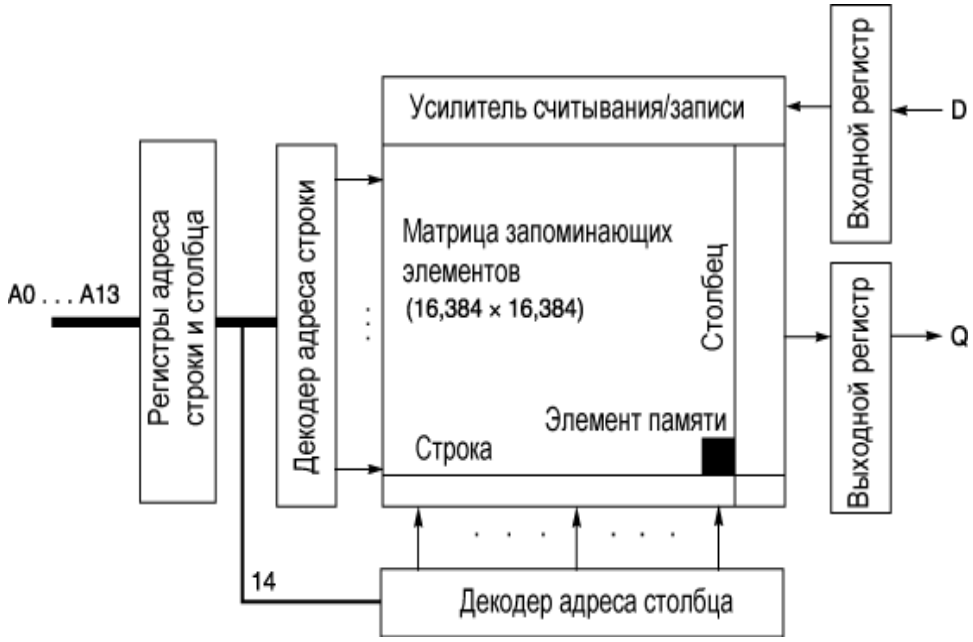
На пересечении строк и столбцов, являющихся обычными проводниками, находятся сами элементы памяти состоящие из одного транзистора и одного конденсатора, Рис. 37**а.**



А) Элемент динамической памяти DRAM



Б) Элемент статической памяти SRAM



В) Структура модуля (микросхемы) статической SRAM и динамической памяти DRAM

Рис. 37. Базовые элементы и структура модулей динамической DRAM и статической SRAM памяти.

Конденсатор хранит информацию в виде заряда, что соответствует одному биту. Отсутствие заряда соответствует логическому нулю, а его наличие - логической единице. Транзистор является вентилем, удерживающим конденсатор от разряда. В обычном состоянии транзистор закрыт, но при подаче на соответствующую строку матрицы электрического сигнала транзистор откроется, соединяя конденсатор с соответствующим столбцом.

Усилитель, подключенный к каждому из столбцов матрицы, считывает всю страницу целиком. Это происходит потому, что проводник строки подключен ко всем транзисторам в странице и при активации

страницы все транзисторы открываются. При чтении вся информация в строке разрущается, так как все конденсаторы будут разряжены, поэтому во избежание потери информации считанную строку нужно перезаписать вновь. Внутренние схемы модуля памяти обеспечивают эту функцию.

Ввиду малых размеров и емкости конденсатора, записанная на нем информация хранится недолго – десятки миллисекунд из-за саморазряда конденсатора. Несмотря на использование высококачественных диэлектриков с огромным удельным сопротивлением, ток утечки ликвидировать полностью не удается и малый заряд конденсатора стекает достаточно быстро. Поэтому необходима регенерация памяти - периодическое считывание всех строк с последующей перезаписью. В зависимости от конструктивных особенностей схема регенерации может находиться как в контроллере памяти, так и в самой DRAM. Нынешний уровень технологии позволяет выполнить регенерацию локально в самой микросхеме, и для избежания блокировки доступа со стороны ЦП используется специальный буфер временного хранения строки на период регенерации.

7.2.1. Организация модуля динамической памяти обычного типа (Page Mode DRAM)

Структура модуля динамической памяти представлена на **рис.6.33в**. Интерфейс модуля памяти включает три группы портов:

1) Порт адреса для выборки данных (Address)

2) Порт данных для записи или чтения данных (Data)

3) Порт управления, определяющий режим работы (Запись или чтение – Write Enable), и другие сигналы упарвления.

Низкий уровень сигнала WE определяет режим записи данных в соответствующую ячейку, а высокий – режим чтения данных из ячейки памяти. Совмещение линий в портах для использования передачи разных сигналов в последовательных тактах является стандартным способом снижения числа выводов микросхемы и снижения ее стоимости в производстве. Адреса столбца и строки матрицы памяти также совмещены в единых адресных линиях. Для квадратной матрицы число адресных линий сокращается вдвое, но и выборка занимает вдвое больше тактов, так как адреса столбца и строки приходится передавать последовательно. Для идентификации адреса строки или стобца в порт управления введены дополнительные сигналы RAS (Row Address Strobe) - строб адреса строки и CAS (Column Address Strobe) - строб адреса столбца соответственно. Активным является низкий уровень сигнала.

Последовательность работы контроллера памяти при доступе к обычной DRAM:

1. Контроллер памяти преобразует полный физический адрес из ЦП в раздельные адрес строки и адрес столбца

2. Контроллер выставляет адрес строки на шину и после некоторой задержки на стабилизацию, активирует сигнал RAS (низкий уровень или логический нуль), по которому микросхема памяти считывает адрес строки в буфер адреса и декодер строки производит выборку всей строки матрицы, содержимое которой запоминается в усилителе чтения/записи.

3. Далее контроллер должен выставить на шину адрес столбца для выборки элемента строки, и после стабилизации сигнала активирует сигнал CAS (низкий уровень или логический нуль), по которому микросхема памяти считывает адрес столбца в буфер адреса. Содержимое этого буфера теперь используется декодером столбца для выборки элемента строки, который подается на выходной порт данных.

4. Контроллер считывает данные и дезактивирует стробы RAS и CAS, затем он должен дать определенное время для перезарядки и перезаписи строки для восстановления информации.

Задержка между подачей номера строки и номера столбца называется tRCD (RAS 2 CAS delay). Задержка между подачей номера столбца и получением данных на выходе - tCAS (CAS delay), а задержка между чтением последнего элемента данных строки ячейки и возможным моментом подачи номера новой строки - tRP (RAS precharge).

7.2.2. Эволюция динамической памяти

В **Табл. 13** представлены параметры микросхем динамической памяти, выпускаемых с конца 70-х годов по настоящее время.

Табл. 13. Параметры микросхем динамической памяти.

Год выпуска	**Объем в битах**	**Строб RAS в наносекундах, медленная**	**Строб RAS в наносекундах, быстрая**	**tCAS в наносекундах (рабочий**	**Полный цикл доступа в**

		память	**память**	**цикл)**	**наносекундах**
1980	64К бит	180	150	75	250
1983	256К бит	150	120	50	220
1986	1М бит	120	100	25	190
1989	4М бит	100	80	20	165
1992	16М бит	80	60	15	120
1996	64М бит	70	50	12	110
1998	128М бит	70	50	10	100
2000	256М бит	65	45	7	90
2002	512М бит	60	40	5	80
2004	1G бит	55	35	5	70
2006	2 G бит	50	30	2.5	60

Столь существенный рост характеристик обеспечивался не только за счет технологии производства, но также и за счет создания новых типов DRAM. В последние 10 лет были созданы следующие новые типы DRAM:

1) Память быстрого страничного доступа FPM DRAM (Fast Page Mode DRAM)
2) Память с усовершенствованным выводом данных EDO DRAM (Extended Data Out)
3) Память EDO RAM с пакетным режимом P/BEDO (Burst EDO)
4) Синхронная SDRAM (Synchronous DRAM)
5) Усовершенствованная синхронная ESDRAM

Далее развитие DRAM пошло двумя путями, одним из которых являлось повышение пропускной способности:

A. Память с удвоенной скоростью выдачи данных DDR SDRAM, SDRAM II (Double Data Rate SDRAM – DDR SDRAM)
B. Память типа Rambus RDRAM (Rambus DRAM)

И другим путем являлось уменьшение задержки для встроенной памяти DRAM:

C. Синхронная память с виртуальным каналом VC-SDRAM (Virtual channel)
D. Синхронная память с быстрым циклом FC-DRAM (Fast Cycle)

E. Синхронная память с уменьшенной задержкой RLDRAM (Reduced Latency)
F. MOSYS-DRAM

Краткая информация об особенностях вышеприведенных типов памяти приведена ниже.

7.2.3. Память быстрого страничного доступа FPM DRAM (Fast Page Mode DRAM)

FPM DRAM была разработанная в 1985 году. Основным отличием от памяти предыдущего поколения стала поддержка последовательной выборки элементов одной строки, которая при считывании сохранялась в специальном быстродействующем буфере на 1024-2048 битов. Если очередной элемент данных находится в той же самой строке, что и предыдущая, ее адрес однозначно определяется одним лишь номером столбца и передача номера строки уже не требуется. При последовательном чтении элементов строки передается только адрес столбца памяти, и время доступа сокращается на 40-50% (при обработке компактных одно-двух килобайтовых структур данных).

При произвольном доступе к памяти, а также при перекрестных запросах коротких фрагментов данных из различных страниц невозможно воспользоваться преимущенствами такого типа памяти и все работает в режиме обычной DRAM. Если очередной фрагмент данных лежит вне текущей (так называемой, открытой) строки, контроллер должен дезактивировать RAS, выдержать паузу «RAS precharge» на перезарядку, передать номер строки, выдержать паузу «RAS to CAS delay» и лишь затем он сможет приступить к передаче номера столбца.

Если запрашиваемый фрагмент данных находится в уже открытой строке, то происходит "попадание в страницу" (**Page Hit**), в противном случае происходит промах (**Page Miss**). Поскольку при промахе возникает дополнительная задержка на открытие новой строки, то эти особенности архитектуры FPM-DRAM должны учитываться при разработке модуля интерфейса памяти ЦП.

Другой проблемой является непостоянство времени доступа, которое затрудняет оценку производительности микросхем памяти и сравнение их друг с другом. В худшем случае обращение к ячейке составляет «RAStoCAS Delay + CAS Delay + RAS precharge», а в лучшем «CAS Delay». Произвольное, но не слишком интенсивное обращение к памяти (с учетом перезарядки) требует не более «RAStoCAS Delay + CAS Delay».

Поскольку, величины задержек «RAStoCAS Delay», «CAS Delay» и «RAS precharge» непосредственно не связаны друг с другом и в принципе могут принимать любые значения, достоверная оценка производительность микросхемы требует для своего выражения как минимум трех чисел. Однако производители микросхем обычно приводят только два показателя:

1) «RAStoCAS Delay + CAS Delay», называемый так же "циклом полного доступа", характеризует время доступа к произвольному фрагменту данных
2) «CAS Delay», называемый так же "рабочим циклом" - время доступа к последовательным фрагментам уже открытой строки.

Причем время, необходимое для регенерации микросхемы памяти (т.е. «RAS precharge»), из цикла полного доступа исключено.

7.2.4. Схематическое описание численных параметров памяти

К середине девяностых среднее значение RAS to CAS Delay составляло порядка 30 нс., CAS Delay - 40 нс., а RAS precharge - менее 30 нс. (наносекунд). Если взять в качестве базы частоту работы системной шины в 60 МГц (т.е. ~17 нс.), то на открытие и доступ к первому элементу страницы уходило около 6 тактов, а на доступ к остальным элементам открытой страницы - около 3 тактов. Схематически это записывается как < **6-3-3-3** > и называется формулой памяти.

Таким образом, формула имеет следующие позиции: Полный цикл доступа – Рабочий цикл считывания/записи последовательных элементов данных.

Формула памяти упрощает сравнение различных микросхем друг с другом, однако для сравнения необходимо знать преобладающий тип обращений к памяти в каждой программе: последовательный или произвольный. Поэтому, практическое значение имеет сравнение лишь вторых цифр - времени рабочего цикла. Совершенствуя ядро памяти, производители сократили его сначала до 35, а затем и до 30 нс, достигнув практически семикратного превосходства над микросхемами прошлого поколения. Например, сейчас формула современной DRAM <4-1-1-1>.

7.2.5. Память с усовершенствованным выводом данных EDO-DRAM (Extended Data Out)

Если оснастить DRAM специальным регистром-защелкой, удерживающим значения данных после исчезновения сигнала CAS,

станет возможным дезактивировать CAS до окончания процесса чтения данных, подготавливая в это время микросхему к приему адреса следующего столбца.

У обычной FPM DRAM низкий уровень CAS удерживается до окончания считывания данных, затем CAS дезактивируется, выдерживается небольшая пауза на перезарядку внутренних цепей, и только после этого на адресную шину подается номер столбца следующего элемента. В новом типе памяти EDO DRAM, сигнал CAS дезактивируется еще в процессе чтения данных, что дает время на перезарядку внутренних цепей еще до окончания считывания данных. Благодаря такому подходу адрес следующего столбца может подаваться до завершения считывания линий данных. Длительность рабочего цикла EDO DRAM (в зависимости от качества микросхемы) составляла 30, 25 и 20 нс., что соответствовало всего двум тактам в 66 МГц системной шины. Совершенствование производственных технологий сократило и полное время доступа. На частоте 66 МГц формула лучших EDO-микросхем выглядела так: **5-2-x-x**. Простой расчет позволяет установить, что пиковый прирост производительности (в сравнении с FPM-DRAM) составляет около 30%.

7.2.6. P/BEDO (Packet/Burst EDO) - пакетная EDO RAM

Двукратное увеличение производительности было достигнуто лишь в BEDO-DRAM (Burst EDO). Добавив в микросхему аппаратный счетчик адреса столбца, инженеры ликвидировали задержку CAS Delay, сократив время цикла до 15 нс. После обращения к произвольному элементу данных микросхема BEDO автоматически увеличивает адрес столбца на единицу, не требуя его явной передачи со стороны контроллера. Из-за ограниченной разрядности адресного счетчика максимальная длина пакета не могла превышать четырех элементов (2 * 2 = 4).

ЦП Intel 80486 и Pentium в силу пакетного режима обмена с памятью никогда не обрабатывают менее четырех смежных элементов за транзакцию. Поэтому, независимо от порядка обращения к данным, BEDO всегда работает на максимально возможной скорости и для частоты 66 Мгц ее формула выглядит так: 5-1-1-1, что на 40% быстрее EDO-DRAM.

Все же, несмотря на свои скоростные показатели, BEDO не получила массового распространения. Проблема состояла в том, что BEDO, как и все ее предшественники, оставалась асинхронной памятью. Это накладывало жесткие ограничения на максимально достижимую

тактовую частоту, ограниченную 66-75 МГц. Например, если время рабочего цикла составляет 15 нс (1 такт в 66 МГц системе), то можно ожидать, что память так и будет выдавать данные каждый такт. На самом деле, поскольку тактирование контроллера памяти и самой микросхемы памяти не синхронизованы, нет никаких гарантий, что начало рабочего цикла микросхемы памяти совпадет с началом тактового импульса контроллера, вследствие чего минимальное время ожидания составляет два такта. Вернее, если быть совсем точным, рабочий цикл микросхемы памяти никогда не совпадает с началом тактового импульса. Несколько наносекунд уходит на формирование контроллером управляющего сигнала RAS или CAS, за счет чего он уже не совпадет с началом тактирующего импульса. Еще несколько наносекунд требуется для стабилизации сигнала и его считывания микросхемой памяти. Это не позволяет добиться высокого быстродействия.

7.2.7. Синхронная SDRAM (Synchronous DRAM) и усовершенствованная ESDRAM

Появление микропроцессоров с системной шиной на 100 МГц и выше привело к радикальному пересмотру механизма управления памятью, и подтолкнуло инженеров к созданию синхронной динамической памяти - SDRAM (Synchronous DRAM). Как и следует из названия, микросхемы SDRAM памяти работают синхронно с контроллером, что гарантирует завершение цикла в строго заданный срок. Кроме того, адреса строк и столбцов подаются одновременно с таким расчетом, чтобы к приходу следующего тактового импульса сигналы уже успели стабилизироваться и были готовы к считыванию.

Кроме того, в SDRAM реализован усовершенствованный пакетный режим обмена. Контроллер может запросить как одну, так и несколько последовательных элементов данных, либо всю строку целиком. Это стало возможным благодаря использованию полноразмерного адресного счетчика, не ограниченного, как в BEDO DRAM, двумя битами.

Другим важным усовершенствованием является увеличение числа матриц (банков) памяти в SDRAM с одного до двух-четырех. Это позволяет обращаться к элементам данных одного банка параллельно с перезарядкой внутренних цепей другого банка, что вдвое увеличивает предельно допустимую тактовую частоту. Кроме этого реализована возможность одновременного открытия двух (четырех) страниц памяти, причем открытие одной страницы (т.е. передача адреса строки) может происходить во время считывания информации с другой страницы, что

позволяет обращаться по новому адресу столбца памяти на каждом тактовом цикле.

В отличие от предшествующих FPM-DRAM, EDO-DRAM, BEDO-DRAM, выполняющих перезарядку внутренних цепей при закрытии страницы (т.е. при дезактивации сигнала RAS), синхронная память выполняет эту операцию автоматически, позволяя держать страницы открытыми неограниченно долго.

Формула чтения произвольного элемента из закрытой строки для SDRAM обычно выглядит как 5-1-x-x, а открытой как **3-1-x-x.**

ESDRAM является вариантом структурного усовершенствования модуля памяти, в котором совмещаются такты считывания данных и подачи команд и адреса для следующего цикла считывания, что позволяет сэкономить 3-4 такта между транзакциями. Как правило, используется в встроенном исполнении.

7.2.8. SDRAM с удвоенной скоростью передачи данных DDR, DDR II, III (Double Data Rate SDRAM)

Дальнее развитие синхронной памяти привело к появлению DDR-SDRAM - Double Data Rate SDRAM (SDRAM удвоенной скорости передачи данных). Удвоение скорости достигается за счет передачи данных и по фронту, и по спаду тактового импульса (в SDRAM передача данных осуществляется только по фронту). Благодаря этому эффективная частота увеличивается в два раза - 100 МГц DDR-SDRAM по своей производительности эквивалента 200 МГц SDRAM. Правда, по маркетинговым соображениям, производители DDR-микросхем стали маркировать их не тактовой частой, а максимально достижимой пропускной способностью, измеряемой в мегабайтах в секунду. Т.е. DDR-1600 работает не на частоте 1,6 ГГц, а всего лишь 100 МГц. Соответственно, DDR 2100 работает на частоте 133 МГц.

Изменилась также структура управления матрицами (банками) памяти. Во-первых, количество банков увеличилось до четырех, а, во-вторых, каждый банк получил собственный контроллер, в результате чего получилось четыре, работающих относительно независимо друг от друга канала памяти. Соответственно, максимальное количество элементов данных, обрабатываемых за один такт, возросло с одного до четырех.

7.2.9. Память типа Rambus или RDRAM (Rambus DRAM)

С вышеописанной DDR-SDRAM конкурирует Direct RDRAM, разработанная компанией Rambus. Основных отличий от памяти предыдущих поколений всего три:

а) увеличение тактовой частоты за счет сокращения разрядности шины,

б) одновременная передача адресов строки и столбца ячейки

в) увеличение числа банков памяти для повышения параллелизма.

Повышение тактовой частоты вызывает усиление электромагнитных помех, интенсивность которых в общем случае пропорциональна квадрату частоты, а на частотах свыше 350 МГц вообще приближается к кубической. Это обстоятельство налагает чрезвычайно жесткие ограничения на топологию и качество изготовления печатных плат модулей микросхемы, что значительно усложняет технологию производства и себестоимость памяти. С другой стороны, уровень помех можно значительно понизить, если сократить количество проводников, т.е. уменьшить разрядность микросхемы. Именно по такому пути компания Rambus и пошла, компенсировав увеличение частоты до 400 MHz (с учетом технологии DDR эффективная частота составляет 800 MHz) уменьшением разрядности шины данных до 16 бит (плюс два бита на ECC). Таким образом, Direct RDRAM в четыре раза обгоняет DDR-1600 по частоте, но во столько же раз отстает от нее в разрядности. А от DDR 2100, Direct RDRAM даже отстает. Следует отметить, что себестоимость DDR заметно ниже RDRAM.

Второе преимущество RDRAM - одновременная передача адресов строки и столбца элемента данных является просто конструктивной особенностью. Это не уменьшает задержки доступа к произвольному элементу данных (т.е. интервала времени между подачей адреса и получения данных), т.к. эта задержка в большей степени определяется скоростью матрицы и декодеров с усилителями, которые в RDRAM являются такими же как и в других типах памяти. Из спецификации RDRAM следует, что время полного доступа составляет 38,75 нс (для сравнения время доступа 100 MHz SDRAM составляет 40 нс.). Так в чем же особенность? Этой особенностью является глубокая конвейеризация обращений у памяти.

Большое количество банков позволяет (теоретически) достичь идеальной конвейеризации, несмотря на то, что данные поступают на шину лишь спустя 40 нс. после подачи первого запроса (что соответствует 320 тактам в 800 MHz системе), сам поток данных далее становится непрерывным и каждый такт передается один элемент данных.

Для алгоритмов последовательной обработки массивов данных в памяти это удобно, но во всех остальных случаях RDRAM не покажет никаких преимуществ перед DDR-SDRAM, а то и обычной SDRAM. К тому же, объем кэш-памяти современных процессоров позволяет

обрабатывать подавляющее большинство запросов локально, вообще не обращаясь к основной памяти. Производительность памяти реально ощущается лишь при обработке больших объемов данных, например, редактировании изображений полиграфического качества.

Таким образом, использование RDRAM в домашних и офисных компьютерах в значительной степени не оправдано. Для высокопроизводительных рабочих станций лучший выбор - DDR-SDRAM, не уступающей RDRAM в производительности, но значительно выигрывающей у последней в стоимости.

7.2.10. VC-SDRAM, FCRAM, MOSYS

VC-SDRAM с виртуальным каналом включает в себя специальный кэш с сегментами, куда производится предварительная выборка сегментов данных из страниц SDRAM, затем если этот сегмент данных используется несколько раз, то доступ к самой матрице данных не производится, а производится выборка фрагмента данных из кэша. Этот тип памяти снят с производства ввиду дороговизны и малого спроса.

FCRAM с быстрым циклом доступа имеет вдвое меньшую задержку считывания из матрицы памяти за счет снижения ширины чтения, при этом экономится такт, необходимый для исправления перекоса данных на усилителе считывания/записи.

RLDRAM используется в качестве встроенной памяти или модулей на микросборках, и сравнима с FCRAM по задержке доступа. При этом имеет более высокую частоту работы и обычно не мультиплексированную шину адреса аналогично SRAM. Может применяться в качестве основы для построения L3 кэша, так как имеет высокую плотность размещения информации, низкое энергопотребление и малую стоимость одного бита. В настоящее время данная технология интенсивно развивается.

1T-SRAM внешне выглядит как обычная статическая память, структурно же она отличается от традиционной DRAM наличием большого числа независимых банков, которые автономно регенерируются. Кроме того, внешний интерфейс имитирует стандартный интерфейс статической памяти и все детали работы структуры такой памяти скрыты от пользователя.

7.2.11. Перспективы использования DRAM

В Табл. 14 приведены перспективные данные по развитию процессоров и DRAM, а также их технологическому исполнению.

Табл. 14. Возможная эволюция процессоров и DRAM с развитием технологии.

Год выпуска	**2004**	**2007**	**2010**	**2013**	**2016**
Поколение полупроводниковой технологии (nm)	90	65	45	32	22
Частота ЦП в MHz	3990	6740	12000	19000	29000
Миллионов транзисторов на кв.см	77.2	154.3	309	617	1235
Число выводов в высокопроизводительных процессорах	2263	3012	4009	5335	7100
Стоимость высокопроизводительных процессоров в центах за один вывод)	1.88	1.61	1.68	1.44	1.22
Стоимость микросхем памяти в центах за один вывод	0.34 - 1.39	0.27 - 0.84	0.22 - 0.34	0.19 - 0.39	0.19-0.33
Число выводов в микросхемах памяти	48 -160	48 -160	62 - 208	81- 270	105 - 351

В настоящее время не существует практической альтернативы использованию DRAM в большинстве видов компьютерных систем. Технология изготовления будет совершенствоваться в сторону уменьшения потреблямой мощности и повышения скорости. Серьезной проблемой станет защита информации от повреждения элементарными частицами, в результате практически все устройства будут использовать специальную схемотехнику. Практически все системы будут использовать одну из четырех возможных конфигураций интерфейса ЦП и модулей DRAM, представленных на Рис. 38.

Первая конфигурация интерфейса на **Рис. 38а** предусматривает прямое соединение ядра процессора с DRAM в заказной конфигурации, которая может быть выполнена на одном кристалле с ядром и иметь очень широкий интерфейс с наивысшей пропускной способностью и малой задержкой доступа. Область применения такой конфигурации – процессорные блоки для суперкомпьютеров и мощных игровых консолей, где требуется обработка больших объемов данных. Стоимость таких блоков из-за заказной конфигурации и комбинированной технологии изготовления на одном кристалле логики процессора и матриц DRAM

будет достаточно высокой, и возможно, только массовый выпуск таких устройств для индустрии развлечений может изменить ситуацию.

Вторая конфигурация на **Рис. 38б** также использует прямое соединение с полузаказными коммерческими DRAM, возможно в рамках системы «кристалл-на-кристалле», когда кристаллы процессора и полузаказной памяти DRAM соединяются через шариковые выводы путем их наложения друг на друга (сэндвич). В такой конфигурации пропускная способность будет высокой, задержка доступа также малой или умеренной, так как длины межсоединения не превышают милиметров. Ширина интерфейса будет ограничиваться максимальным числом шариковых выводов, которые могут быть размещены на обоих кристаллах.

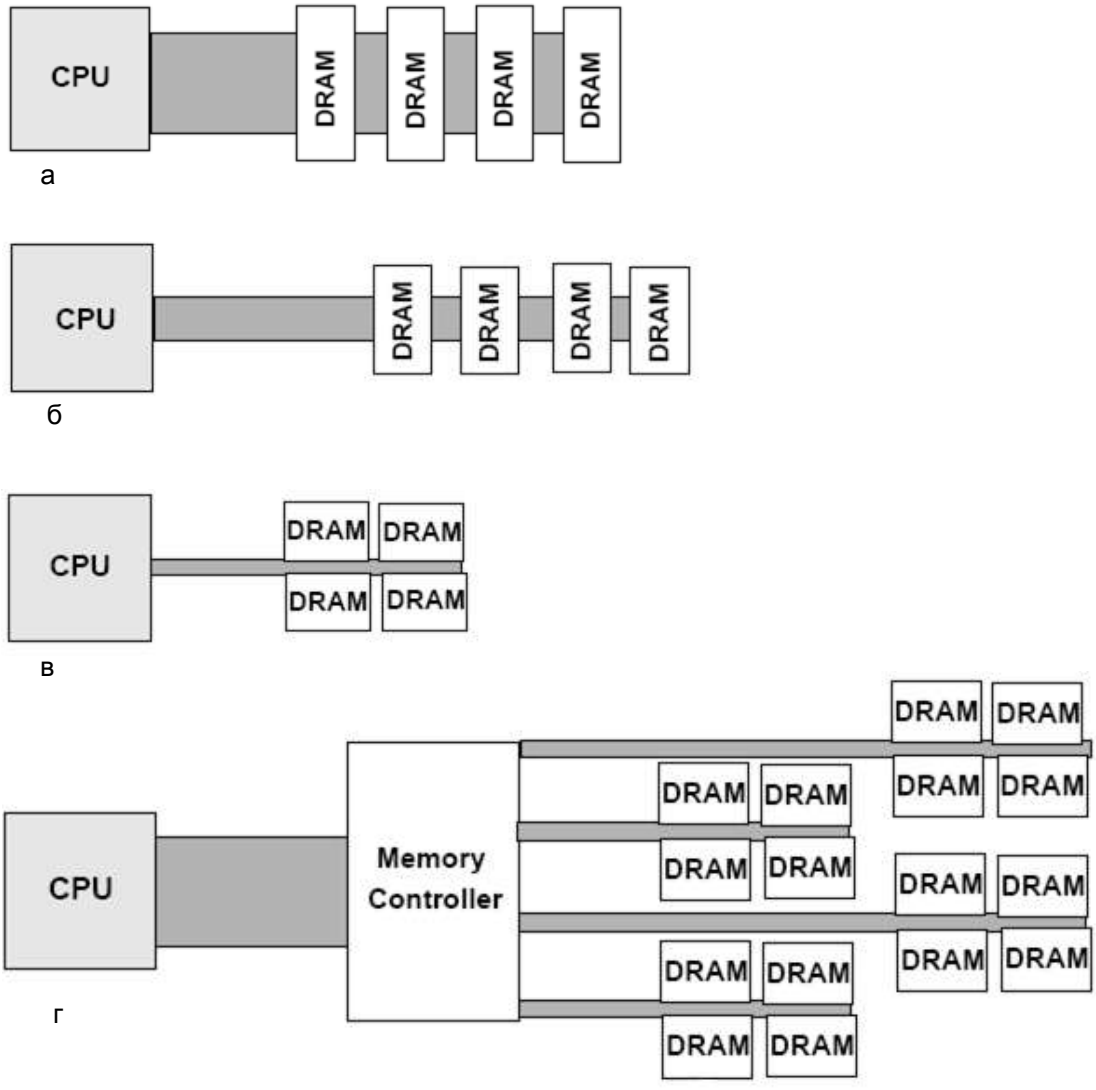


Рис. 38. Структуры интерфейса ЦП и модулей динамической памяти DRAM различных типов.

Третья конфигурация на **Рис. 38в** предусматривает прямое соединение ЦП и модулей коммерческих DRAM через внешний интерфейс стандартной ширины. В такой конфигурации пропускная способность будет низкой из-за ограничений в ширине внешнего интерфейса и тактовой частоты его работы. Задержка доступа не будет повышаться, так как модули памяти подсоединены непосредственно к ЦП. Реализация этой конфигурации возможна при создании микросборки с процессором и модулями памяти. Использование массовых коммерчески доступных компонент позволяет снизить стоимость таких конфигураций до минимальной.

Четвертая конфигурация на **Рис. 38г** характерна для большинства существующих настольных и серверных компьютерных систем, когда ЦП соединен с модулями памяти не напрямую, а через специальный многоканальный контроллер памяти, который обеспечивает доступ к множеству модулей недорогой коммерческой DRAM. Интерфейс между контроллером памяти и ЦП может иметь высокую пропускную способность и его насыщение данными будет обеспечиваться многоканальностью доступа к модулям памяти. Особенностью такой конфигурации является очень высокая задержка доступа к данным через контроллер памяти, который должен собирать/рассортировывать данные по множеству каналов.

7.3. Модули статической памяти SRAM

Память первых релейных компьютеров по своей природе была статической и долгое время не претерпевала практически никаких изменений, с развитием технологии менялась лишь элементная база: на смену реле пришли электронные лампы, впоследствии вытесненные сначала транзисторами, а затем TTL- и CMOS-микросхемами: но идея, лежащая в основе статической памяти, была и остается прежней – использование двух логических вентилей. Динамическая память, изобретенная значительно позднее, в силу фундаментальных физических ограничений, так и не смогла сравняться со статической памятью в скорости работы.

Основой микросхемы статической оперативной памяти (SRAM - Static Random Access Memory) является набор триггеров - логических устройств, имеющих два устойчивых состояния, одно из которых условно

соответствует логическому нулю, а другое - логической единице. Каждый триггер хранит один бит информации, - ровно столько же, сколько и ячейка динамической памяти с конденсатором. Триггер превосходит конденсатор по двум позициям:

а) состояния триггера устойчивы и при наличии питания могут сохраняться бесконечно долго, в то время как конденсатор из-за тока утечки требует периодической регенерации;

б) триггер, обладая малой инертностью, без проблем работает на частотах вплоть до нескольких ГГц, в то время как конденсаторы выдерживают на порядок меньшие частоты 75-100 МГц.

К недостаткам триггеров следует отнести их высокую стоимость и низкую плотность хранения информации. Если для создания ячейки динамической памяти достаточного всего одного транзистора и одного конденсатора, то ячейка статической памяти состоит как минимум из четырех, а в среднем 6 - 8 транзисторов, поэтому мегабайт статической памяти оказывается по меньшей мере в несколько раз дороже динамической памяти такого же объема. Подобно ячейкам динамической памяти, базовые элементы -триггеры объединяются в единую матрицу, состоящую из строк (**row**) и столбцов (**column**), последние из которых так же называются битами (**bit**). В отличии от ячейки динамической памяти, для управления которой достаточно всего одного транзисторного ключа, ячейка статической памяти управляется как минимум двумя. Триггер, в отличии от конденсатора, имеет раздельные входы для записи логического нуля и единицы соответственно. Таким образом, на ячейку статической памяти расходуется шесть транзисторов - четыре идут на сам триггер и еще два - на управляющие вентили, **Рис. 37б.**

Причем шесть транзисторов на ячейку является минимальной конфигурацией для однопортовой памяти. Основной недостаток 6-транзисторной ячейки заключается в том, что в каждый момент времени может обрабатываться всего лишь одна строка матрицы памяти. Параллельное чтение ячеек, расположенных в различных строках одного и того же банка невозможно, также как невозможно и чтение одной ячейки одновременно с записью другой. Для обеспечения одновременного доступа нескольких клиентов к разным фрагментам данных используется либо многопортовая память либо память с множеством банков с независимым доступом к каждому банку. Каждая

ячейка многопортовой памяти содержит один-единственный триггер, но имеет несколько комплектов управляющих транзисторов, каждый из которых подключен к независимым линиям ROW и BIT, благодаря чему различные ячейки матрицы могут обрабатываться независимо. Такой подход более удобен, чем деление памяти на банки, хотя значительно дороже. При многобанковой организации памяти возможны конфликты, когда разные клиенты пытаются считать считать информацию с одного и того же банка, поэтому необходимо использование специальной логики для арбитража доступа. Наиболее часто встречается двухпортовая память SRAM, используемая в кэш-памяти ЦП. Нетрудно подсчитать, что для создания одной ячейки двухпортовой памяти расходуется восемь транзисторов (6+2). При емкости SRAM 32 Кб на одну только матрицу триггеров потребуется свыше двух миллионов транзисторов. Однако, для построения внутренних буферов на микросхеме, SDRAM не имеет альтернатив, по крайней мере в настоящее время.

Интерфейс матрицы статической памяти по своему устройству практически ничем не отличается от аналогичного интерфейса матрицы динамической памяти. При значительно меньшей емкости и жестких требованиях к быстродействию SRAM не использует мультиплексирование адреса строк и столбцов, так как для встроенного исполнения это не является необходимым. Если статическая память выполнена в виде самостоятельной микросхемы, а не располагается непосредственно на кристалле ЦП, часто используется двунаправленная шина данных и режим работы шины определяется специальным сигналом WE (**Write Enable**). Высокий уровень сигнала WE соотвествует чтению данных, а низкий - записи. Встроенная SRAM, размещенная на одном кристалле вместе с ЦП, обычно имеет раздельные шины ввода-вывода большой ширины для повышения быстродействия.

Номера столбцов и строк поступают на декодеры столбца и строки соответственно (см. **Рис. 37в**). После декодирования расшифрованный номер строки поступает на дополнительный декодер, вычисляющий принадлежащую ей матрицу. Оттуда он попадает непосредственно на схему выборки строки, которая открывает вентили требуемой страницы. В зависимости от выбранного режима работы усилитель, подсоединенный к разрядным строкам матрицы, либо считывает состояние триггеров соответствующей строки, либо устанавливает их в состояние, соответствующее записываемому фрагменту данных.

Существует как минимум три типа статической памяти:

1) асинхронная SRAM (только что рассмотренная выше),

2) синхронная SRAM и

3) конвейерная SRAM.

Асинхронная статическая память работает независимо от контроллера и потому, контроллер не может быть уверен, что окончание цикла обмена совпадет с началом очередного тактового импульса. В результате, цикл обмена удлиняется по крайней мере на один такт, снижая тем самым эффективную производительность.

Синхронная статическая память выполняет все операции одновременно с тактовыми сигналами, в результате чего время доступа к элементу данных укладывается в один такт. Именно на синхронной статической памяти реализуется кэш-память первого уровня современных ЦП, а также все внутренние буферы.

Конвейерная статическая память представляет собой синхронную статическую память, оснащенную специальными регистрами, фиксирующими данные между тактами, что позволяет читать (записывать) содержимое одного элемента данных параллельно с передачей адреса другого. Кроме того, конвейерная память может обрабатывать несколько смежных элементов данных за один рабочий цикл. Достаточно передать лишь адрес первого фрагмента пакета, а адреса остальных микросхема вычислит самостоятельно. За счет большей аппаратной сложности конвейерной памяти, время доступа к первому элементу пакета увеличивается на один такт, однако это не снижает производительности, т.к. все последующие элементы пакета обрабатываются без задержек.

В целом, сравнивая технологии DRAM и SRAM, можно сказать, что первая фокусируется на емкости памяти и стоимости на один бит данных, вторая фокусируется на скорости и также емкости памяти. Средняя емкость памяти микросхем DRAM в 4-8 раз выше микросхем SRAM, выполненных по той же технологии. При этом быстродействие микросхем SRAM в 8-16 раз выше, но и стоимость хранения бита данных будет также в 8-16 раз выше. Особенность технологии производства DRAM не позволяют совмещать ее на одном кристалле со стандартной логикой ЦП, для этого нужен специальный технологический процесс, которым обладают только компании IBM и Samsung. Специфика и

дороговизна такой технологии стали препятствием ее широкого распространения.

7.4. Встроенная память ROM и FLASH

Большинство встроенных систем не имеют магнитных дисков для хранения информации в случае отключения питания, для решения этой проблемы используются технологии постоянной памяти с масочным программированием при изготовлении (ROM или ПЗУ), а также технологии энергонезависимой электрически перезаписываемой постоянной памяти (ППЗУ) типа FLASH.

Программируемые при изготовлении ПЗУ имеют важное достоинство постоянства содержащихся данных, они не могут изменены программой. Так как большиство встроенных компьютеров не обладают системой защиты памяти, то эта особенность позволяет защитить систему от несанкционированной модификации. Они обладают очень высокой плотностью размещения данных, так как требуют только один транзистор на бит, соответственно стоимость также является низкой и сопоставима с DRAM. Она используется для записи системных программ и констант, и обычно выполняется в встроенном варианте в специализированном микропроцессоре.

ППЗУ типа FLASH могут быть перепрограммированы многократно и фактически используются в качестве заменителей магнитных дисков небольшого объема. Технология FLASH памяти непрерывно развивается и скорость работы устройств памяти этого типа уже достигла скорости работы DRAM в режиме чтения, в режиме записи она работатет в 10-100 раз медленнее. Массовое коммерческое использование этой памяти связано с цифровыми камерами и MP3 проигрывателями, в которых она является основным носителем информации.

СПИСОК ЛИТЕРАТУРЫ

1. David Patterson, John Hennessy «Computer Organization and Design: The Hardware/Software Interface», 3rd Edition, 2004, 656 Pages, ISBN: 978-1-55860-604-3, ISBN10: 1-55860-604-1
2. John L. Hennessy, David A. Patterson. "Computer Architecture: A Quantitative Approach". 2011. ISBN 012383872X, ISBN 9780123838728. page B-9.
3. Harvey G. Cragon. "Memory systems and pipelined processors". 1996. ISBN 0867204745, ISBN 9780867204742.

4. В.М. Фельдман «Микропроцессорные вычислительные комплексы отечественной разработки», Сб. научно-техн. трудов "Высокопроизводительные вычислительные системы и микропроцессоры". - М.: ИМВС РАН, №5, 2003. - С.3-15.
5. “A Performance Study of Contemporary DRAM Architectures,”*Proc. ISCA ’99.* V. Cuppu, B. Jacob, B. Davis, and T. Mudge.
6. “Organizational Design Trade-Offs at the DRAM, Memory Bus, and Memory Controller Level: Initial Results,” University of Maryland Technical Report UMD-SCA-TR-1999-2. V. Cuppu and B. Jacob.
7. Timour Paltashev “Lecture Notes on graduate course EE504: Advanced Computer Organization and Design”, College of Engineering, Northwestern Polytechnic University, Fremont, California, year 2009.
8. R.E. Kessler. The Alpha 21264 Microprocessor. IEEE Micro. March/April 1999.
9. D. Leibholz and R. Razdan. The Alpha 21264: A 500 MHz Out-of-order Execution Microprocessor. COMPCON97, 1997.
10. Compaq Computer Corporation. Alpha 21264/EV6 Hardware Reference Manual.
11. R. Kessler, E. McLellan, and D. Webb. The Alpha 21264 microprocessor architecture. International Conference on Computer Design, October 1998
12. The Russians Are Coming: Supercomputer Maker Elbrus Seeks to Join x86/IA-64 Melee, MICROPROCESSOR REPORT, VOLUME 13, NUMBER 2, FEBRUARY 15, 1999
13. Itanium™ Processor Microarchitecture Reference, 2000
14. Intel Itanium 2 Processor Reference Manual, 2004
15. ARM System-on-Chip Architecture (2nd Edition), Steve Furber, 2002, Published by Addison Wesley, ISBN 0-201-67519-6.
16. “Mobile Intel Pentium 4 Processor-M Datasheet". Intel Corp.
17. "Intel's Mobile Pentium 4". Intel Corp.

УПРАЖНЕНИЯ И ЗАДАЧИ

1) Построение КЭШа прямого отображения аналогично Табл.2:

Постройте кэш прямого отображения (таблицу с размещением данных и тегов, а также формат адреса для размеров КЭШа 16 К байт и 64 К байт с размером строки КЭШа в 8 и 16 байт для каждого из вариантов. Обратите внимание на изменение полей индекса и смещения в адресном слове.

Подсчитайте объем памяти тэгов для каждого из вариантов реализации и процент дополнительной памяти по сравнению с обычной памятью данных RAM.

2) Среднее время доступа к памяти в иерархии памяти:

Используя выражение ***T_average = T_hit + MissRate*T_miss*** постройте графики среднего времени доступа к памяти для ***T_hit*** = 1, 2 и 3 такта с ***MissRate***= 2, 5 и 10% для ***T_miss*** в диапазоне от 50 до 250 тактов.

3) Построение ассоциативного КЭШа аналогично Рис.5:

Постройте (полно)ассоциативный кэш (таблицу с размещением данных, компараторов и тегов, а также формат адреса для размеров КЭШа 16 К байт и 64 К байт с размером строки КЭШа в 8 и 16 байт для каждого из вариантов. Обратите внимание на изменение полей смещения в адресном слове.

Подсчитайте объем памяти тэгов для каждого из вариантов реализации и процент дополнительной памяти по сравнению с обычной памятью данных RAM.

4) Построение частично-ассоциативного КЭШа аналогично Рис.7:

Постройте двухбанковый частично-ассоциативный кэш (таблицу с размещением данных и тегов, а также формат адреса для размеров КЭШа 16 К байт и 64 К байт с размером строки КЭШа в 8 и 16 байт для каждого из вариантов. Обратите внимание на изменение полей индекса и смещения в адресном слове.

Подсчитайте объем памяти тэгов для каждого из вариантов реализации и процент дополнительной памяти по сравнению с обычной памятью данных RAM. Сколько счетчиков нужно для реализации LRU в каждом случае?

5) Типы промахов, проценты (доли) промахов в обращении к памяти:

Определите долю промахов в последовательности обращений к памяти в табл.4.

Объясните отличие неизбежных промахов от промахов нехватки объема и конфликтов отображения. Постройте таблицу с последовательностью адресов обращения к КЭШу прямого отображения, где будут представлены неизбежные промахи и промахи конфликтов отображения. Попробуйте сделать аналогичную таблицу обращения к 2-х банковому частично-ассоциативному КЭШу с четырьмя строками по 4 байта, где будут представлены все три вида промахов.

6) Зависимость времени доступа от длины строки КЭШа:

Используя выражение ***T_average = T_hit + MissRate*T_miss*** постройте графики среднего времени доступа к памяти при ***T_hit*** = 1 такт, ***MissRate***= 2, 5% и ***T_miss*** = 30 тактов для строки КЭШа 4, 8, 16, 32 и 64 байта. Ширина шины данных 32 бита и за один такт может передано только 4 байта. Для блоков данных более 4-х байт нужно добавить к ***T_miss*** необходимое число тактов.

7) Среднее время доступа в многоуровневом КЭШе:

Расширьте выражение $T\_average = T\_hit_{L1} + Miss\_Rate_{L1} * (T\_hit_{L2} + Miss\_Rate_{L2} * T\_miss_{L2})$ для трехуровневой системы КЭШей. Используя численные данные из примера в тексте постройте графики среднего времени доступа к памяти для T_hit_{L3} = 8 тактов с $MissRate_{L3}$= 0.5% для T_miss_{L3} = 100, 150, 200, 250, 300 тактов.

8) Сравнение различных конфигураций КЭШей:

Используя формулу $Index = \log_2 (Cache\_Size / (Block\_size \times Set\_Assoc))$, посчитайте конфигурацию адреса (размеры полей тэга, индекса и смещения) при адресном пространстве в 40 бит, размере строки КЭШа в 8 байти для КЭШей размерностью в 128 К байт, реализованных в виде КЭШа прямого отображения, 4-банкового частично-ассоциативного КЭШа и полно-ассоциативного КЭШа. Насколько процентов увеличивается дополнительный расход памяти на тэги по сравнению с КЭШем прямого отображения?

9) Форматы виртуального адреса для различных размеров страниц:

Разработайте форматы виртуальных и физических адресов для виртуального адресного пространства в 16 Т байт и физического адресного пространства в 256 Г байт. Размеры страниц 4К, 8К, 64К и 256 К байт.

10) Практическое построение виртуальной памяти и время доступа:

Используя пример построения виртуальной памяти в процессоре Альфа на рис.19, попробуйте подсчитать объем памяти, необходимый для хранения всей таблицы страниц с линейной адресацией. Сколько всего таблиц 1К х 8 байт можно поместить в этом объеме памяти? Насколько уменьшаются требования к объему памяти при иерархической или списковой виртуальной адресации памяти, представленной на этом

рисунке?? Для примера если у нас всего четыре задачи с 64 М байт пространством памяти каждая??

11) Буфер быстрой трансляции виртуальных адресов в физические:

Постройте структуру ББТ аналогично табл.8, который имеет 64 строки и является полностью ассоциативным. Размерность виртуального адреса 48 бит, размер физического адреса 40 бит. Оцените соотношение между памятью тэгов и памятью данных (физическим адресом).

12) Коллизия номеров виртуальных страниц и индексов КЭШа:

Определите границу коллизии индекса физического КЭШа прямого отображения L1 с размером строки в 8 байт с виртуальной страницей в 4К байта, 8 К байт и 16 К байт. Какие максимальные размеры физического КЭШа могут быть, чтобы индекс не выходил за пределы размера страницы??

13) Комбинация двухуровневого КЭШа и виртуальной памяти

Используя аналогию с структурой на рис.22, постройте диаграмму, комбинирующую виртуальную память с ББТ и двухуровневый кэш для виртуального адресного пространства в 64 бита, физического адресного пространства в 48 бит с размером страниц в 16 К байт. Размеры КЭШей определяются максимально возможными значениями индекса L1.

14) Кэш-контроллер

Используя описание логики кэш-контроллера ARM на рис.26, постройте простые управляющие автоматы для КЭШей прямого отображения и частично-ассоциативных КЭШей, рассмотренных в тексте.

15) Транслирующий кэш в Itanium

Тщательно изучите построение транслирующего КЭШа в микропроцессоре Itanium на Рис.32. Разработайте функционально аналогичную структуру, используя подход, описанный на Рис.22 в задаче 13.

16) Кэш память в Itanium

Используя данные в табл.11 постройте структуру КЭШей и формат адреса для КЭШей всех уровней. Посчитайте среднее время доступа к памяти, если доступ к основной памяти занимает 200 тактов, а доля промахов в L1=2.1%, L2=1.2% , L3=0.1%.

17) Сравнение архитектур иерархии памяти в процессорах Itanium и Эльбрус

Сравните архитектуры иерархии памяти в обоих процессорах, посчитайте число битов памяти, требуемых для построения в обоих случаях.

18) Доступ к основной памяти в режиме расслоения:

Определите, насколько быстрее будет доступ в режиме расслоения к основной памяти, состоящей из восьми банков, при тех же параметрах доступа, как и для структуры на рис.35.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие наиболее важные особенности иерархической структуры памяти можно отметить?
2. Какой принцип построения кэш-памяти?
3. Какие способы реализации кэш-памяти вам известны?
4. Влияет ли степень ассоциативности КЭШа на быстродействие? Каким образом?
5. Как можно повысить производительность кэш-памяти?
6. Приведите определения физической и виртуальной памяти.
7. Приведите причины создания механизма виртуальной памяти.
8. Какие ключевые характеристики механизма виртуальной памяти?
9. Какие типы виртуальной памяти существуют? Их отличия?
10. Каким образом можно повысить производительность виртуальной памяти?
11. Иерархия памяти в процессорах семейства ARM.
12. Приведите основные характеристики модулей кэш-памяти ЦП Itanium.
13. Что содержит таблица ALAT (Advanced Load Address Table)?
14. За счет чего можно увеличить пропускную способность памяти?
15. Приведите определения динамической DRAM и статической SRAM памяти.
16. Базовые элементы и структура модулей динамической DRAM и статической SRAM памяти.
17. Перспективы использования DRAM, от DDR-2 до DDR-5.
18. Сколько типов статической памяти существуют? Назовите их.
19. Что такое VC-SDRAM, FCRAM и MOSYS ?
20. Встроенная память ROM и FLASH.