Общие принципы функционирования кэш-памяти

Кэш-память (КП), или **кэш**, представляет собой организованную в виде ассоциативного *запоминающего устройства* (АЗУ) быстродействующую *буферную память* ограниченного объема, которая располагается между регистрами процессора и относительно медленной основной памятью и хранит наиболее часто используемую информацию совместно с ее признаками (тегами), в качестве которых выступает часть адресного кода.

В процессе работы отдельные блоки информации копируются из основной памяти в *кэш-память*. При обращении процессора за командой или данными сначала проверяется их наличие в КП. Если необходимая *информация* находится в кэше, она быстро извлекается. Это **кэш-попадание**. Если необходимая *информация* в КП отсутствует (**кэш-промах**), то она выбирается из основной памяти, передается в *микропроцессор* и одновременно заносится в *кэш-память*. Повышение быстродействия вычислительной системы достигается в том случае, когда **кэш-попадания реализуются намного чаще, чем кэш-промахи**.

Зададимся вопросом: "А как определить наиболее часто используемую информацию? Неужели сначала кто-то анализирует ход выполнения программы, определяет, какие команды и данные чаще используются, а потом, при следующем запуске программы, эти данные переписываются в *кэш-память* и уже тогда *программа* выполняется эффективно?" Конечно нет. Хотя в современных микропроцессорах имеется определенный механизм, который позволяет в некоторой степени реализовать этот принцип. Но в основном, конечно, *кэш-память* сама отбирает информацию, которая чаще всего используется. Рассмотрим, как это происходит.

Механизм сохранения информации в кэш-памяти

При включении микропроцессора в работу вся информация в его кэш-памяти недостоверна.

При обращении к памяти микропроцессор, как уже отмечалось, сначала проверяет, не содержится ли искомая информация в кэш-памяти.

Для этого сформированный им физический адрес сравнивается с адресами ячеек памяти, которые были ранее кэшированы из ОЗУ в КП.

При первом обращении такой информации в *кэш*-памяти, естественно, нет, и это соответствует **кэш-промаху**. Тогда *микропроцессор* проводит обращение к оперативной памяти, извлекает нужную информацию, использует ее в своей работе, но одновременно записывает эту информацию в *кэш*.

Если бы в *кэш-память* заносилась только востребованная микропроцессором в данный момент *информация*, то, скорее всего, при следующем обращении вновь произошел бы **кэш-промах**: вряд ли следующее обращение произойдет к той же самой команде или к тому же самому операнду. **Кэш-попадания** происходили бы лишь после того, как в КП накопится достаточно большой фрагмент программы, содержащий некоторые циклические участки кода, или фрагмент данных, подлежащих повторной обработке. Для того чтобы уже следующее обращение к КП приводило как можно чаще к **кэш-попаданиям**, передача из оперативной памяти в *кэш-память* происходит не теми порциями (байтами или словами), которые востребованы микропроцессором в данном обращении, а так называемыми **строками**. То есть *кэш-память* и оперативная *память* с точки зрения кэширования организуются в виде строк. *Длина строки* превышает максимально возможную длину востребованных микропроцессором данных. Обычно она составляет от 16 до 64 *байт* и выровнена в памяти по границе соответствующего раздела (рис. 4.1).

Рис. 4.1. Организация обмена между оперативной и кэш-памятью

Высокий *процент* **кэш-попаданий** в этом случае обеспечивается благодаря тому, что в большинстве случаев программы обращаются к ячейкам памяти, расположенным вблизи от ранее использованных. Это свойство, называемое **принципом локальности ссылок**, обеспечивает эффективность использования КП. Оно подразумевает, что при исполнении программы в течение некоторого относительно малого интервала времени происходит обращение к памяти в пределах ограниченного диапазона адресов (как по коду программы, так и по данным).

Например, микропроцессору для своей работы потребовалось 2 байта информации. Если строка имеет длину 16 *байт*, то в *кэш* переписываются не только нужные 2 байта, но и некоторое их окружение. Когда *микропроцессор* обращается за новой информацией, в силу локальности ссылок, скорее всего, обращение произойдет по соседнему адресу. Затем опять по соседнему, опять по соседнему и т. д. Таким образом, ряд следующих обращений будет происходить непосредственно к *кэш*-памяти, минуя оперативную *память* **(кэш-попадания)**. Когда очередной сформированный микропроцессором *физический адрес* выйдет за пределы строки *кэш*-памяти (произойдет **кэш-промах**), будет выполнена подкачка в *кэш* новой строки, и вновь ряд последующих обращений вызовет **кэш-попадания**.

Чем длиннее используемая при обмене между оперативной и кэшпамятью строка, тем больше *вероятность* того, что следующее обращение произойдет в пределах этой строки. Но в то же время чем длиннее строка, тем дольше она будет перекачиваться из оперативной памяти в *кэш*. И если очередная *команда* окажется *командой перехода* или *выборка* данных начнется из нового массива, то есть следующее обращение произойдет не по соседнему адресу, то время, затраченное на передачу длинной строки, будет использовано напрасно. Поэтому при выборе длины строки должен быть разумный *компромисс* между соотношением времени обращения к оперативной и *кэш*-памяти и вероятностью достаточно удаленного перехода от текущего адреса при выполнении программы. Обычно *длина строки* определяется в результате моделирования аппаратно-программной *структуры системы*.

После того как в КП накопится достаточно большой объем информации, увеличивается *вероятность* того, что формирование очередного адреса приведет к **кэш-попаданию**. Особенно велика *вероятность* этого при выполнении циклических участков программы.

Старая *информация* по возможности сохраняется в *кэш*-памяти. Ее замена на новую определяется емкостью, организацией и стратегией обновления кэша.

Типы кэш-памяти

Если каждая строка *ОЗУ* имеет только одно фиксированное *место*, на котором она может находиться в *кэш*-памяти, то такая *кэш-память* называется памятью **с прямым отображением**.

Предположим, что *ОЗУ* состоит из 1000 строк с номерами от 0 до 999, а *кэш-память* имеет емкость только 100 строк. В *кэш*-памяти с прямым отображением строки *ОЗУ* с номерами 0, 100, 200, ..., 900 могут сохраняться только в строке 0 КП и нигде иначе, строки 1, 101, 201, …, 901

ОЗУ - в строке 1 КП, строки *ОЗУ* с номерами 99, 199, …, 999 сохраняются в строке 99 *кэш*-памяти (рис. 4.2). Такая организация *кэш*-памяти обеспечивает быстрый *поиск* в ней нужной информации: необходимо проверить ее наличие только в одном месте. Однако емкость КП при этом используется не в полной мере: несмотря на то, что часть *кэш*-памяти может быть не заполнена, будет происходить *вытеснение* из нее полезной информации при последовательных обращениях, например, к строкам 101, 301, 101 *ОЗУ*.

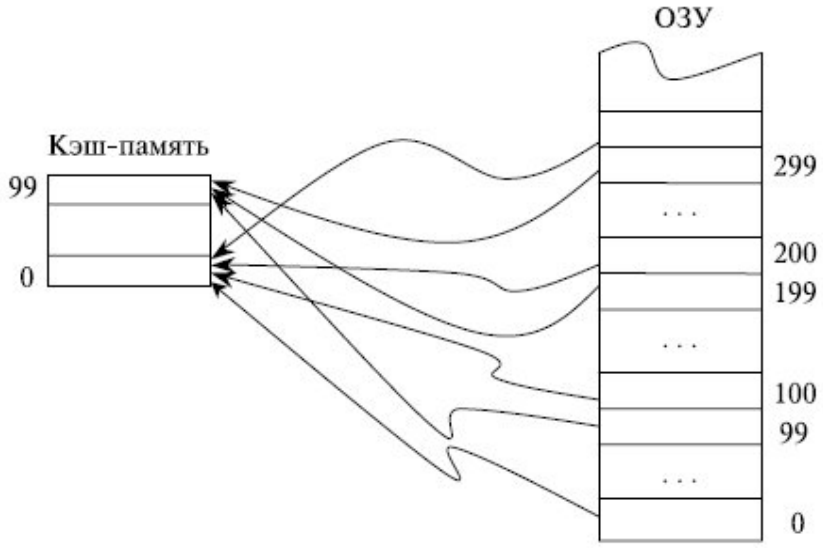

Рис. 4.2. Принцип организации кэш-памяти с прямым отображением

Кэш-память называется **полностью ассоциативной**, если каждая строка *ОЗУ* может располагаться в любом месте *кэш*-памяти.

В полностью ассоциативной *кэш*-памяти максимально используется весь ее объем: *вытеснение* сохраненной в КП информации проводится лишь после ее полного заполнения. Однако *поиск* в *кэш*-памяти, организованной подобным образом, представляет собой трудную задачу.

Компромиссом между этими двумя способами организации *кэш*-памяти служит **множественно-ассоциативная** КП, в которой каждая строка *ОЗУ* может находиться по ограниченному множеству мест в *кэш*-памяти.

При необходимости замещения информации в *кэш*-памяти на новую используется несколько **стратегий замещения**. Наиболее известными среди них являются:

1. *LRU* - замещается строка, к которой дольше всего не было обращений;
2. FIFO - замещается самая давняя по пребыванию в кэш-памяти строка;
3. Random - замещение проходит случайным образом.

Последний вариант, существенно экономя *аппаратные средства* по сравнению с другими подходами, в ряде случаев обеспечивает и более эффективное использование *кэш*-памяти. Предположим, например, что КП имеет объем 4 строки, а некоторый циклический участок программы имеет длину 5 строк. В этом случае при стратегиях *LRU* и FIFO *кэш-память* окажется фактически бесполезной ввиду отсутствия *кэш*-попаданий. В то же время при использовании стратегии случайного замещения информации часть обращений к КП приведет к *кэш*-попаданиям.

Некоторые эвристические оценки вероятности *кэш*-промаха при разных *стратегиях замещения* (в процентах) представлены в табл. 4.1.

Таблица 4.1. Вероятность кэш-промаха для различной кэш-памяти

Размер кэша,Кбайт	**Организация кэш-памяти**					
	2-канальная ассоциативная		**4-канальная ассоциативная**		**8-канальная ассоциативная**	
	LRU	**Random**	***LRU***	**Random**	***LRU***	**Random**
16	5.2	5.7	4.7	5.3	4.4	5.0
64	1.9	2.0	1.5	1.7	1.4	1.5
256	1.15	1.17	1.13	1.13	1.12	1.12

Анализ таблицы показывает, что:

- увеличением емкости кэша, естественно, уменьшается вероятность **кэш-промаха**, но даже при незначительной на сегодняшний день емкости кэш-памяти в 16 Кбайт около 95 % обращений происходят к КП, минуя оперативную память;
- чем больше степень ассоциативности кэш-памяти, тем больше вероятность **кэш-попадания** за счет более полного заполнения КП (время поиска информации в КП в данном анализе не учитывается);
- механизм *LRU* обеспечивает более высокую вероятность **кэш-попадания** по сравнению с механизмом случайного замещения Random, однако этот выигрыш не очень значителен.

Соответствие между данными в оперативной памяти и в *кэш*-памяти обеспечивается внесением изменений в те области *ОЗУ*, для которых данные в *кэш*-памяти подверглись изменениям. Существует два основных способа реализации этих действий: со сквозной записью (writethrough) и с обратной записью (write-back).

При считывании оба способа работают идентично. При записи **кэширование со сквозной записью** обновляет *основную память* параллельно с обновлением информации в КП. Это несколько снижает *быстродействие* системы, так как *микропроцессор* впоследствии может вновь обратиться по этому же адресу для записи информации, и предыдущая пересылка строки *кэш*-памяти в *ОЗУ* окажется бесполезной. Однако при таком подходе содержимое соответствующих друг другу строк *ОЗУ* и КП всегда идентично. Это играет большую роль в мультипроцессорных *системах с общей* оперативной памятью.

Кэширование с обратной записью модифицирует строку *ОЗУ* лишь при *вытеснении* строки *кэш*-памяти, например, в случае необходимости освобождения места для записи новой строки из *ОЗУ* в уже заполненную КП. *Операции* обратной записи также инициируются механизмом поддержания согласованности *кэш*-памяти при работе мультипроцессорной системы с общей оперативной памятью.

Промежуточное положение между этими подходами занимает способ, при котором все строки, предназначенные для передачи из КП в *ОЗУ*, предварительно накапливаются в некотором буфере. Передача осуществляется либо при *вытеснении* строки, как в случае **кэширования с обратной записью**, либо при необходимости согласования *кэш*-памяти нескольких микропроцессоров в мультипроцессорной системе, либо при заполнении буфера. Такая передача проводится в пакетном режиме, что более эффективно, чем передача отдельной строки.

Организация внутренней кэш-памяти микропроцессора

Внутренний *кэш* 32-разрядного универсального микропроцессора является общим при обращении как к командам, так и к данным. Обращение ведется по физическим адресам.

Кэш-память обычно реализуется в виде ассоциативного ЗУ, в котором для каждой строки сохраняются дополнительные сведения, называемые тегом, или признаком, в качестве которого выступает адресный код или его часть. Когда в АЗУ подается *адрес*, с ним одновременно сравниваются все теги.

Внутренняя *кэш-память* в микропроцессоре i486 реализует **сквозную запись**. Начиная с МП Pentium используется **сквозная** или **обратная запись**.

Во внешней КП применяется любой способ записи или их комбинация.

Внутренняя *кэш-память* МП i486 имеет емкость 8 Кбайт и организована в виде 4-канальной ассоциативной памяти. Это означает, что данные из какой-либо строки *ОЗУ* могут храниться в любой из 4 строк *кэш*-памяти.

КП состоит из следующих блоков (рис. 4.3):

- блока данных,
- блока тегов,

- блока достоверности и *LRU*.

Блок данных содержит 8 Кбайт данных и команд. Он разделен на 4 массива (направления), каждый из которых состоит из 128 строк.

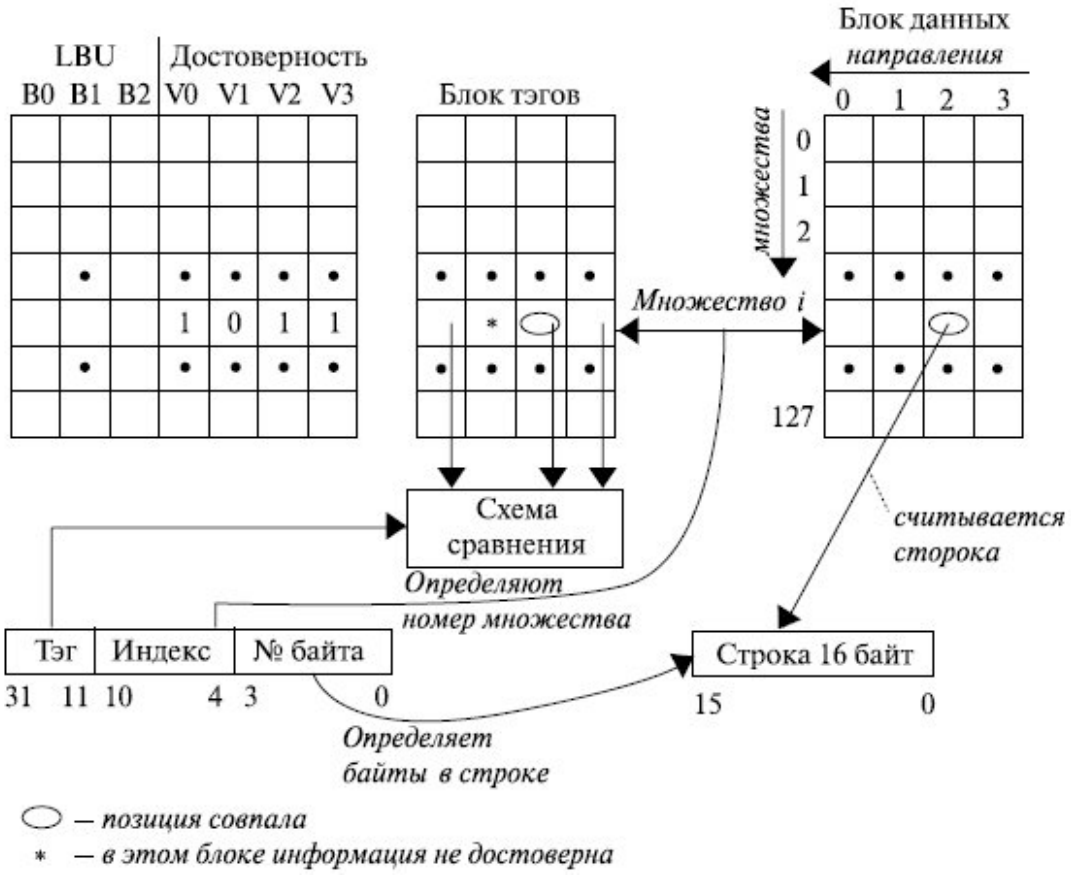

Рис. 4.3. Структура внутренней кэш-памяти МП i486

Строка содержит данные из 16 последовательных адресов памяти начиная с адреса, кратного 16. *Индекс массивов* блока данных, состоящий из 7 *бит*, соответствует 4 строкам КП, по одной из каждого массива. Четыре строки КП с одним и тем же индексом называются множеством.

В **блоке тегов** имеется один *тег* длиной 21 *бит* для каждой строки данных в КП. Блок тегов также разделен на 4 массива по 128 тегов. *Тег* содержит старшие 21 *бит* физического адреса данных, находящихся в соответствующей строке КП.

В **блоке достоверности** и *LRU* содержится по одному 7-разрядному значению для каждого из 128 множеств строк КП: 4 бита достоверности (V) по одному на каждую строку *множества* и 3 бита (B0 ... B2), *управляющие* механизмом *LRU*. Биты достоверности показывают, содержит ли строка достоверные (V = 1) или недостоверные (V = 0) данные. При программной очистке КП и аппаратном сбросе процессора все биты достоверности сбрасываются в 0.

Адресация кэш-памяти осуществляется путем разделения старших 28 *бит* физического адреса на 2 части. Младшие 7 *бит* из этих разрядов (разряды 10…4 физического адреса) образуют *поле* индекса и определяют множество, в котором могут храниться данные. Старшие 21 *бит* (разряды 31…11 физического адреса) служат полем тега и применяются для определения того, находится ли *информация* с данным физическим адресом в какой-либо строке выбранного *множества*.

Поиск в *кэш*-памяти информации с заданным физическим адресом выполняется следующим образом:

1. Физический адрес, по которому происходит обращение, разбивается на 3 поля: Тег, Индекс, № байта. 7 разрядов A10…A4 поля индекса определяют одно из 128 множеств.
2. В выбранном множестве содержатся 4 строки с информацией.

Чтобы определить, присутствует ли нужная информация в одной из строк этого множества, проводится сравнение старших 21 бита физического адреса (поле Тег) с тегами строк выбранного множества. Сравнение проводится только для достоверных строк, то есть тех, у которых в блоке достоверности установлен бит достоверности V = 1.

3. Если для одной из строк ее тег и разряды A31...A11 физического адреса совпали, то это означает, что произошло **кэш-попадание** и необходимая информация есть в кэш-памяти.
4. Считывается найденная строка из 16 байт. Искомый байт в ней определяется 4 младшими разрядами физического адреса (A3…A0).
5. Если на этапе 3 совпадения не произошло или все строки множества недостоверны, эта ситуация определяется как **кэш-промах**. В этом случае по сформированному микропроцессором физическому адресу выполняется обращение к оперативной памяти. Из *ОЗУ* извлекается нужная информация, и содержащая ее строка записывается в свободную строку выбранного множества. Старшие 21бит физического адреса записываются в поле тега этой строки. Если все строки в выбранном множестве достоверны, то замещается строка, к которой дольше всего не было обращений согласно механизму *LRU*. Этот механизм действует точно так же, как и при *вытеснении* строк из **буфера ассоциативной трансляции** *TLB*.

Режим работы *кэш*-памяти определяется программно установкой разрядов CD (запрет кэширования) и NW (запрет сквозной записи) в управляющем регистре CR0. *Кэширование* можно разрешить (это состояние после инициализации при сбросе), можно запретить при наличии достоверных строк (в этом режиме КП действует как быстрое внутреннее *ОЗУ*) или, наконец, *кэширование* может быть полностью запрещено.

Управление работой кэш-памяти на уровне страниц

В **элементах каталога страниц и таблиц страниц** имеются 2 бита, которые применяются для управления выходными сигналами процессора и участвуют в кэшировании страниц.

Бит PCD запрещает (*PCD* = 1) или разрешает (*PCD* = 0) *кэширование* страницы. Запрещение кэширования необходимо для страниц, которые содержат порты ввода/вывода с отображением на *память*. Оно также полезно для страниц, *кэширование* которых не дает выигрыша в быстродействии, например, страниц, содержащих программу инициализации.

Бит PWT определяет метод обновления *ОЗУ* и внешней *кэш*-памяти (*кэш* 2-го уровня). Если PWT = 1, то для данных в соответствующей странице определяется *кэширование* со сквозной записью, при PWT = 0 применяется способ обратной записи. Используется в микропроцессорах начиная с Pentium. Так как внутренняя *кэш-память* в МП i486 работает со сквозной записью, состояние бита PWT на нее не влияет. *Бит* PWT в этом случае действует только на внешнюю КП.

Обеспечение согласованности кэш-памяти микропроцессоров в мультипроцессорных системах

Рассмотрим особенности работы *кэш*-памяти в том случае, когда одновременно несколько микропроцессоров используют общую оперативную *память* (рис. 4.4). В этом случае могут возникнуть проблемы, связанные с кэшированием информации из оперативной памяти в *кэш-память* микропроцессоров.

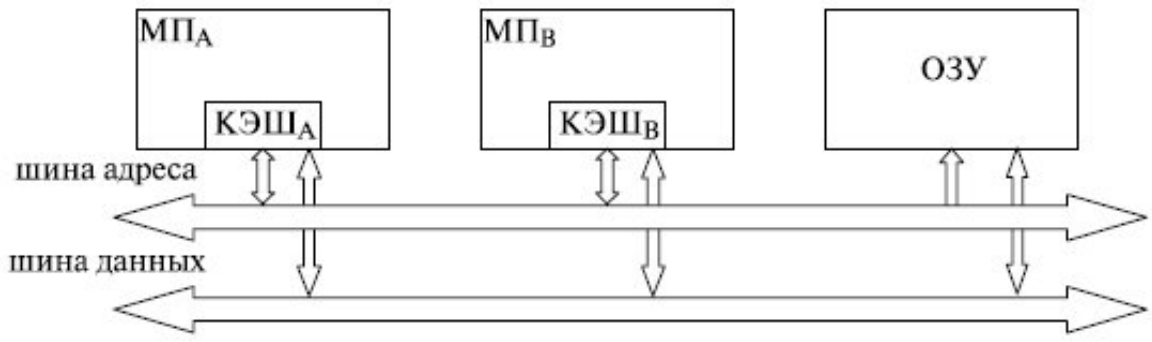

Рис. 4.4. Структура мультимикропроцессорной системы с общей оперативной памятью

Предположим, что МП А считал некоторую строку данных из *ОЗУ* в свою внутреннюю КП и изменил данные в этой строке в процессе работы.

Мы отмечали, что существует два основных механизма обновления оперативной памяти:

1. **сквозная запись**, которая подразумевает, что как только изменилась информация во внутренней кэш-памяти, эта же информация копируется в то же место оперативной памяти, и
2. **обратная запись**, при которой микропроцессор после изменения информации во внутреннем кэше отражает это изменение в оперативной памяти не сразу, а лишь в тот момент, когда происходит *вытеснение* данной строки из кэш-памяти в оперативную. То есть существуют определенные моменты времени, когда информация, предположим, по адресу 2000 имеет разные значения: микропроцессор ее обновил, а в оперативной памяти осталось старое значение. Если в этот момент другой микропроцессор (МП В), использующий ту же оперативную память, обратится по адресу 2000 в *ОЗУ*, то он прочитает оттуда старую информацию, которая к этому времени уже не актуальна.

Для обеспечения согласованности (*когерентности*) памяти в мультипроцессорных системах используются аппаратные *механизмы*, позволяющие решить эту проблему. Такие *механизмы* называются **протоколами когерентности кэш-памяти**. Эти протоколы призваны гарантировать, что любое считывание элемента данных возвращает последнее по времени записанное в него *значение.*

Существует два класса протоколов *когерентности*:

- **протоколы на основе справочника** (directory based): информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы);
- **протоколы наблюдения** (*snooping*): каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии; централизованная система записей отсутствует; обычно кэши расположены на *общей шине*, и контроллеры всех кэшей наблюдают за шиной (просматривают ее), чтобы определять, какие обращения по адресам в пределах этого блока происходят со стороны других микропроцессоров.

В мультипроцессорных *системах с общей* памятью наибольшей популярностью пользуются **протоколы наблюдения**, поскольку для опроса состояния кэшей они могут использовать уже существующее физическое соединение - шину памяти.

Для поддержания *когерентности* применяется два основных метода.

Один из методов заключается в том, чтобы гарантировать, что *процессор* должен получить исключительные *права доступа* к элементу данных перед выполнением записи в этот *элемент данных*. Этот тип протоколов называется протоколом **записи с аннулированием** (write invalidate protocol), поскольку при выполнении записи он аннулирует другие копии. Это наиболее часто используемый протокол как в **схемах на основе справочников**, так и в **схемах наблюдения**. Исключительное *право доступа* гарантирует, что во *время выполнения* записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы.

Альтернативой протоколу записи с *аннулированием* является обновление всех копий элемента данных в случае записи в этот *элемент данных.*

Этот тип протокола называется протоколом *записи с обновлением* (write update protocol), или протоколом *записи с трансляцией* (write *broadcast* protocol).

Эти две схемы во многом похожи на схемы работы *кэш*-памяти со сквозной и с обратной записью. Ключевым моментом реализации в *многопроцессорных системах* с небольшим числом процессоров как схемы записи с *аннулированием*, так и схемы записи с обновлением данных, является использование для выполнения этих операций механизма шины. Для выполнения *операции* обновления или аннулирования *процессор* просто захватывает шину и транслирует по ней *адрес*, по которому должно производиться обновление или *аннулирование* данных. Все процессоры непрерывно наблюдают за шиной, контролируя появляющиеся на ней адреса.

Процессоры проверяют, не находится ли в их *кэш*-памяти *адрес*, появившийся на шине. Если это так, то соответствующие данные в кэше либо аннулируются, либо обновляются в зависимости от используемого протокола.

Рассмотрим один из наиболее распространенных протоколов, обеспечивающих согласованную работу *кэш*-памяти нескольких микропроцессоров и основной памяти в мультимикропроцессорных системах, **протокол MESI**, который относится к группе **протоколов наблюдения с аннулированием**. Будем знакомиться с ним на примере двухпроцессорной системы, состоящей из микропроцессоров A и B.

Этот протокол использует 4 признака состояния строки *кэш*-памяти микропроцессора, по первым буквам которых и называется протокол:

- измененное состояние (Modified): информация, хранимая в кэш-памяти микропроцессора A, достоверна только в этом кэше; она отсутствует в оперативной памяти и в кэш-памяти других микропроцессоров;
- исключительная копия (Exclusive): информация, содержащаяся в кэше A, содержится еще только в оперативной памяти;
- разделяемая информация (Shared): информация, содержащаяся в кэше A, содержится в кэш-памяти по крайней мере еще одного МП, а также в оперативной памяти;
- недостоверная информация (*Invalid*): в строке кэш-памяти находится недостоверная информация.

Таким образом, состояние признаков потокола *MESI* отражает следующие состояния (по отношению к МПА) строки *кэш*-памяти (табл. 4.2):

Таблица 4.2. Формирование признаков состояния протокола *MESI*

остояние признака протокола	**Состояние строки памяти**		
	Кэш A	**Кэш B**	***ОЗУ***
Modified	Д	НД	НД
Shared	Д	Д	Д
Exclusive	Д	НД	Д
Invalid	НД	Х	Х

При работе микропроцессора A с точки зрения обеспечения *когерентности* памяти возможны следующие ситуации:

- RH (Read *Hit*) - **кэш-попадание** при чтении;
- WH (Write *Hit*) - **кэш-попадание** при записи;
- RME (Read Miss Exclusive) - **кэш-промах** при чтении;
- *RMS* (Read Miss Shared) - **кэш-промах** при чтении, но соответствующий блок есть в кэш-памяти другого микропроцессора;
- WM (Write Miss) - **кэш-промах** при записи;
- SHR (Snoop *Hit* Read) - обнаружение копии блока при прослушивании операции чтения другого кэша;
- SHW (Snoop *Hit* Write) - обнаружение копии блока при прослушивании операции записи другого кэша.

Наибольший интерес здесь представляют две последние позиции.

Современные микропроцессоры имеют двунаправленную шину адреса.

Выдавая информацию на эту шину, *микропроцессор* адресует ячейки оперативной памяти или устройства ввода-вывода. В силу того, что в рассматриваемой мультипроцессорной системе микропроцессоры связаны общей шиной, в том числе и шиной адреса, принимая информацию по адресным линиям, *микропроцессор* определяет, было ли обращение по адресам, содержащимся в его *кэш*-памяти, со стороны других микропроцессоров. При обнаружении такого обращения меняется состояние строки *кэш*-памяти микропроцессора.

Изменения признака состояния блока *кэш*-памяти МП в зависимости от различных ситуаций в его работе и работе мультимикропроцессорной системы в целом представлены на рис. 4.5.

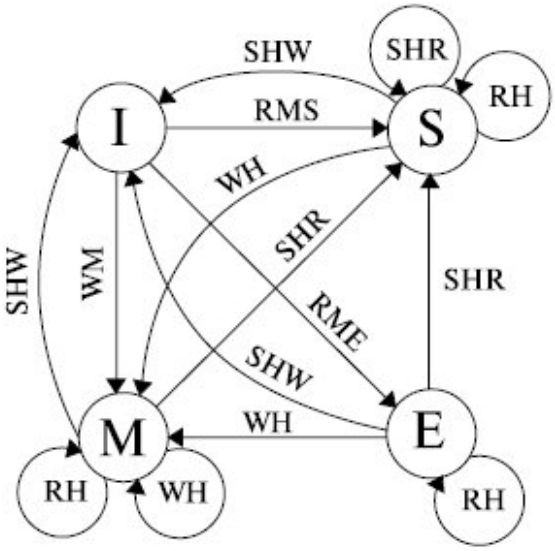

Рис. 4.5. MESI-диаграмма обеспечения когерентности кэш-памяти

Проиллюстрируем некоторые из представленных переходов.

Пусть блок *кэш*-памяти находится в состоянии Modified, то есть достоверная *информация* находится только в *кэш*-памяти данного МП. Тогда в случае обнаружения при прослушивании адресной шины обращения со стороны другого микропроцессора для чтения информации по входящим в данную строку адресам *микропроцессор* должен передать эту строку кэшпамяти в *ОЗУ*, откуда она уже будет прочитана другим микропроцессором.

При этом состояние строки в *кэш*-памяти рассматриваемого микропроцессора изменится с модифицированного на разделяемое (Shared).

Если строка *кэш*-памяти находилась в состоянии *Invalid*, то есть *информация* в ней была недостоверной, то по отношению к этой строке следует рассматривать только ситуации, связанные с **кэш-промахами**. Так, если произошел **кэш-промах** при выполнении *операции* записи, то необходимая строка будет занесена в *кэш-память* данного МП, в эту строку будут записаны измененные данные, и она приобретет статус исключительного владельца новой информации (Modified).