10.1 Ассоциативная память

В рассмотренных ранее видах запоминающих устройств доступ к информации требовал указания адреса ячейки. Зачастую значительно удобнее искать информацию не по адресу, а опираясь на какой-нибудь характерный признак, содержащийся в самой информации. Такой принцип лежит в основе ЗУ, известного как *ассоциативное запоминающее устройство* (АЗУ). В литературе встречаются и иные названия подобного ЗУ: память, адресуемая по содержанию (content addressable memory); память, адресуемая по данным (data addressable memory); память с параллельным поиском (parallel search memory); каталоговая память (catalog memory); информационное ЗУ (information storage); тегированная память (tag memory). Ассоциативное ЗУ — это устройство, способное хранить информацию, сравнивать ее с некоторым заданным образцом и указывать на их соответствие или несоответствие друг другу. Признак, по которому производится поиск информации, будем называть *ассоциативным признаком,* а кодовую комбинацию, выступающую в роли образца для поиска, — *признаком поиска.* Ассоциативный признак может быть частью искомой информации или дополнительно придаваться ей. В последнем случае его принято называть *тегом* или *ярлыком.*

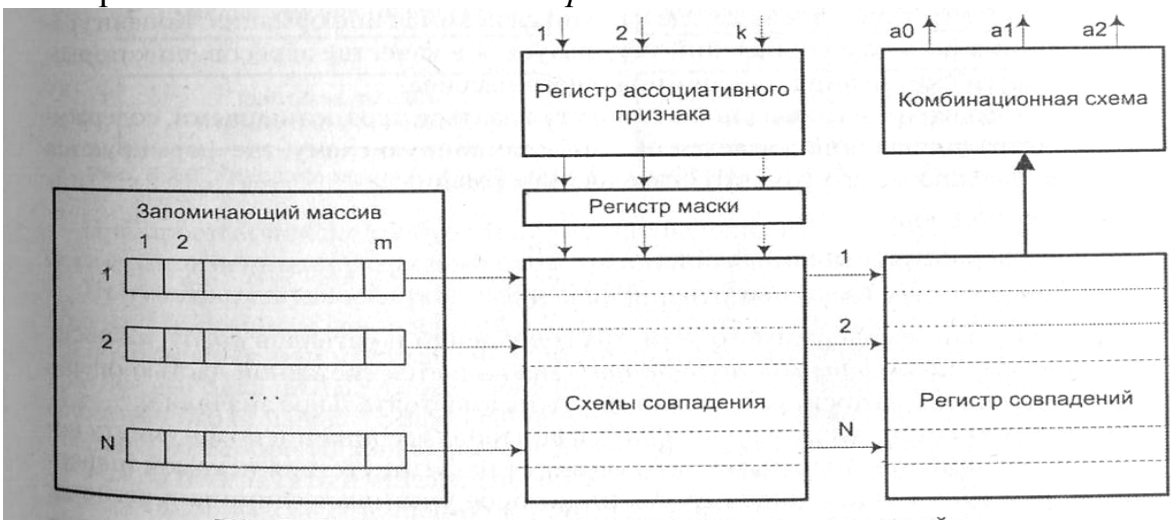

Рис. 10.1 Структура ассоциативного запоминающего устройства

Один из вариантов построения ассоциативной памяти показан на рис. 10.1 АЗУ включает в себя:

- запоминающий массив для хранения *N* m -разрядных слов, в каждом из которых несколько младших разрядов занимает служебная информация;
- регистр ассоциативного признака, куда помещается код искомой информации (признак поиска). Разрядность регистра *k* обычно меньше длины слова *m;*
- схемы совпадения, используемые для параллельного сравнения каждого бита всех хранимых слов с соответствующим битом признака поиска и выработки сигналов совпадения;
- регистр совпадений, где каждой ячейке запоминающего массива соответствует один разряд, в который заносится единица, если все разряды

соответствующей ячейки совпали с одноименными разрядами признака поиска;

- регистр маски, позволяющий запретить сравнение определенных битов;

- комбинационную схему, которая на основании анализа содержимого регистра совпадений формирует сигналы, характеризующие результаты поиска информации.

При обращении к АЗУ сначала в регистре маски обнуляются разряды, которые не должны учитываться при поиске информации. Все разряды регистра совпадений устанавливаются в единичное состояние. После этого в регистр ассоциативного признака заносится код искомой информации (признак поиска) и начинается ее поиск, в процессе которого схемы совпадения одновременно сравнивают первый бит всех ячеек запоминающего массива с первым битом признака поиска. Те схемы, которые зафиксировали несовпадение, формируют сигнал, переводящий соответствующий бит регистра совпадений в нулевое состояние. Так же происходит процесс поиска и для остальных незамаскированных битов признака поиска. В итоге единицы сохраняются лишь в тех разрядах регистра совпадений, которые соответствуют ячейкам, где находится искомая информация. Конфигурация единиц в регистре совпадений используется в качестве адресов, по которым производится считывание из запоминающего массива.

Из-за того что результаты поиска могут оказаться неоднозначными, содержимое регистра совпадений подается на комбинационную схему, где формируются сигналы, извещающие о том, что искомая информация:

a0 — не найдена;

a1 — содержится в одной ячейке;

a2 — содержится более чем в одной ячейке.

Формирование содержимого регистра совпадений и сигналов *aO, a1, a2* носит название *операции контроля ассоциации.* Она является составной частью операций считывания и записи, хотя может иметь и самостоятельное значение.

При считывании сначала производится контроль ассоциации по аргументу поиска. Затем, при *a0* = 1 считывание отменяется из-за отсутствия искомой информации, при *a1* = 1 считывается слово, на которое указывает единица в регистре совпадений, а при *a2* = 1 сбрасывается самая старшая единица в регистре совпадений и извлекается соответствующее ей слово. Повторяя эту операцию, можно последовательно считать все слова.

Запись в ассоциативную память (АП) производится без указания конкретного адреса, в первую свободную ячейку. Для отыскания свободной ячейки выполняется операция считывания, в которой не замаскированы только служебные разряды, показывающие, как давно производилось обращение к данной ячейке, и свободной считается либо пустая ячейка, либо та, которая дольше всего не использовалась.

Главное преимущество ассоциативных ЗУ определяется тем, что время поиска информации зависит только от числа разрядов в признаке поиска и скорости опроса разрядов и не зависит от числа ячеек в запоминающем массиве.

Общность идеи ассоциативного поиска информации отнюдь не исключает разнообразия архитектур АЗУ. Конкретная архитектура определяется сочетанием четырех факторов: вида поиска информации; техники сравнения признаков; способа считывания информации при множественных совпадениях и способа записи информации.

В каждом конкретном применении АЗУ задача поиска информации может формулироваться по-разному (рис. 10.1.2).

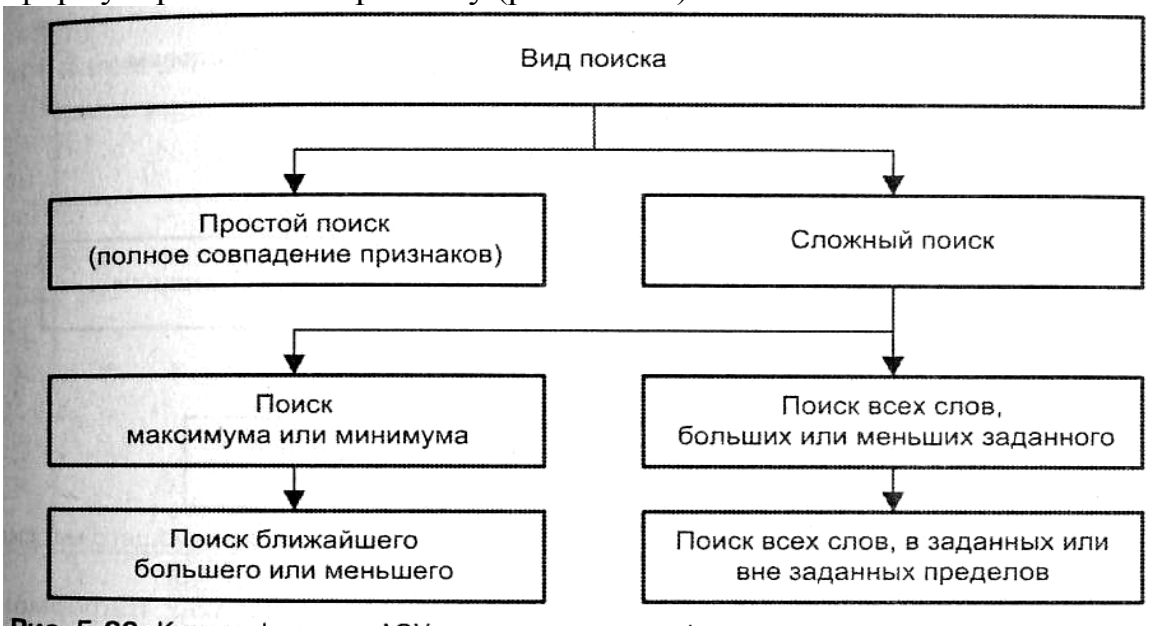

Рис. 10.1.2 Классификация АЗУ по виду поиска информации в запоминающем массиве

При простом поиске требуется полное совпадение всех разрядов признака поиска с одноименными разрядами слов, хранящихся в запоминающем массиве.

При соответствующей организации АЗУ способно к более сложным вариантам поиска, с частичным совпадением. Можно, например, ставить задачу поиска слов с максимальным или минимальным значением ассоциативного признака. Многократное выборка из АЗУ слова с максимальным или минимальным значением ассоциативного признака (с исключением его из дальнейшего поиска), по существу, представляет собой упорядоченную выборку информации. Упорядоченную выборку можно обеспечить и другим способом, если вести поиск слов, ассоциативный признак которых по отношению к признаку опроса является ближайшим большим или меньшим значением.

Еще одним вариантом сложного поиска может быть нахождение слов, численное значение признака в которых больше или меньше заданной величины. Подобный подход позволяет вести поиск слов с признаками, лежащими внутри или вне заданных пределов.

Очевидно, что реализация сложных методов поиска связана с соответствующими изменениями в архитектуре АЗУ, в частности с усложнением схемы ЗУ и введением в нее дополнительной логики.

При построении АЗУ выбирают из четырех вариантов организации опроса содержимого памяти (рис. 10.1.3). Варианты эти могут комбинироваться параллельно по группе разрядов и последовательно по группам. В плане времени поиска наиболее эффективным можно считать параллельный опрос как по словам, так и по разрядам, но не все виды запоминающих элементов допускают такую возможность.

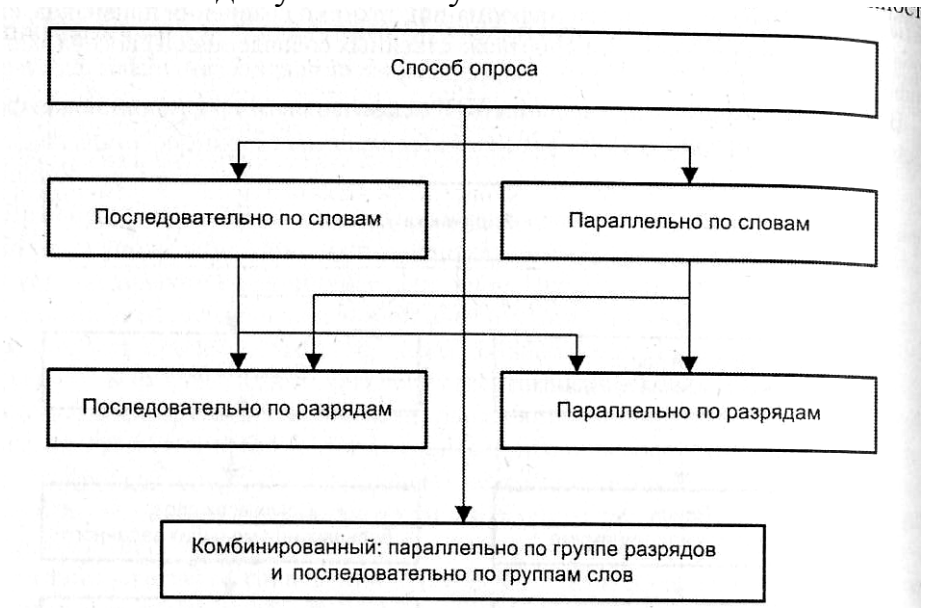

Рис. 10.1.3 Классификация АЗУ по способу опроса содержимого запоминающего массива

Важным моментом является организация считывания из АЗУ информации, когда с признаком поиска совпадают ассоциативные признаки нескольких слов. В этом случае применяется один из двух подходов (рис. 10.1.4).

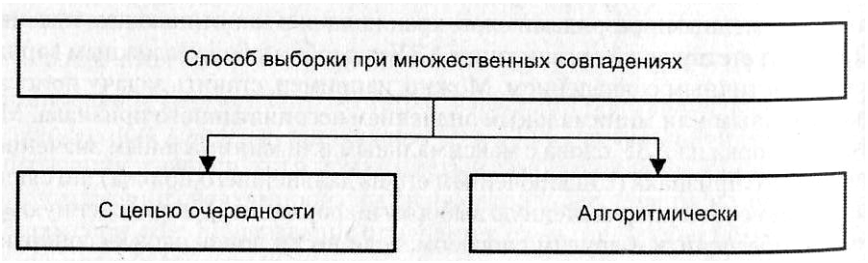

Рис. 10.1.4 Классификация АЗУ по способу выборки совпавших слов при множественных совпадениях

Цепь очередности реализуется с помощью достаточно сложного устройства, где фиксируются слова, образующие многозначный ответ. Цепь очередности позволяет производить считывание слов в порядке возрастания номера ячейки АЗУ независимо от величины ассоциативных признаков.

При алгоритмическом способе извлечения многозначного ответа выборка производится в результате серии опросов. Серия опросов может формироваться путем упорядочивания тех разрядов, которые были замаскированы и не участвовали в признаке поиска. В другом варианте для

этих целей выделяются специальные разряды. Существует целый ряд алгоритмов, позволяющих организовать упорядоченную выборку из АЗУ.

Существенные отличия в архитектурах АЗУ могут быть связаны с выбранным принципом записи информации. Ранее был описан вариант с записью в незанятую ячейку с наименьшим номером. На практике применяются и иные способы (рис. 10.1.5), из которых наиболее сложный — запись с сортировкой информации на входе АЗУ по величине ассоциативного признака. Здесь местоположение ячейки, куда будет помещено новое слово, зависит от соотношения ассоциативных признаков вновь записываемого слова и уже хранящихся в АЗУ слов. Из-за относительно высокой стоимости АЗУ редко используется как самостоятельный вид памяти.

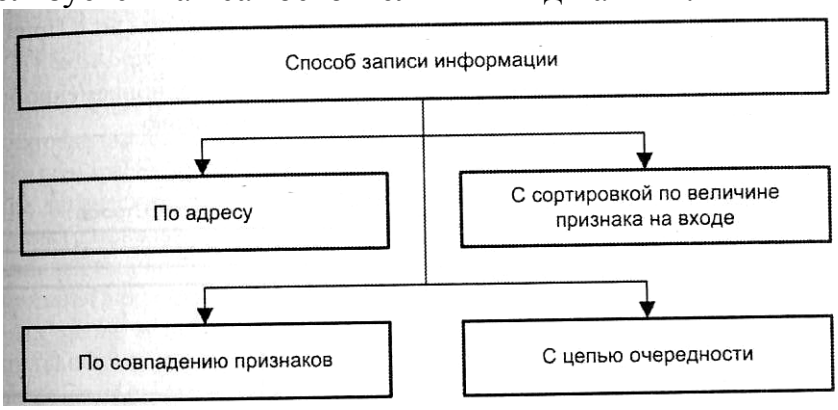

Рис. 10.1.5. Классификация АЗУ по способу записи информации

10.2 Кэш-память

В качестве элементной базы основной памяти в большинстве ВМ служат микросхемы ОЗУ на порядок уступающие по быстродействию центральному процессору. В результате процессор вынужден простаивать несколько тактовых периодов, пока информация из ИМС памяти установится на шине данных ВМ. Решение этой проблемы было предложено в 1965 году в процессе разработки ВМ Atlas и заключается оно в использовании двухуровневой памяти, когда между ОП и процессором размещается небольшая, но быстродействующая буферная память. В процессе работы такой системы в буферную память копируются те участки ОП, к которым производится обращение со стороны процессора. В общепринятой терминологии — производится *отображение* участков ОП на буферную память. Выигрыш достигается за счет свойства локальности — если отобразить участок ОП в более быстродействующую буферную память и переадресовать на нее все обращения в пределах скопированного участка, можно добиться существенного повышения производительности ВМ.

Сначала рассматриваемую буферную память называли подчиненной (slave memory). Позже распространение получил термин *кэш-память* (от английского слова *cache* — убежище, тайник), поскольку такая память обычно скрыта от программиста в том смысле, что он не может ее адресовать и может даже вообще не знать о ее существовании. Впервые кэш-системы появились в машинах модели 85 семейства IBM 360.

В общем виде использование кэш-памяти поясним следующим образом. Когда ЦП пытается прочитать слово из основной памяти, сначала

осуществляется поиск копии этого слова в кэше. Если такая копия существует, обращение к ОП не производится, а в ЦП передается слово, извлеченное из кэш-памяти. Данную ситуацию принято называть успешным обращением или *попаданием* (hit). При отсутствии слова в кэше, то есть при неуспешном обращении — *промахе* (miss),- требуемое слово передается в ЦП из основной памяти, но одновременно из 01 в кэш-память пересылается блок данных, содержащий это слово.

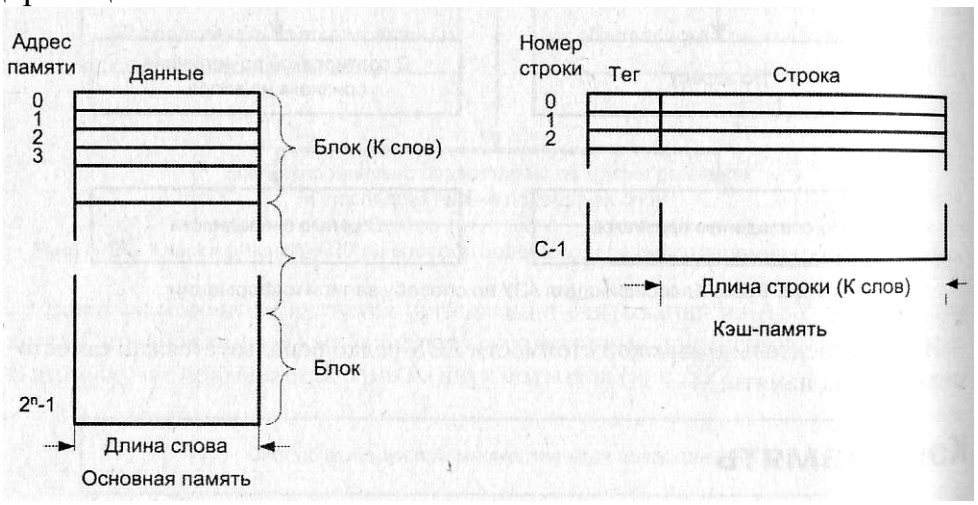

Рис. 10.2.1 Структура системы с основной и кэш-памятью

На рис. 10.2.1 приведена структура системы с основной кэш-памятью. ОП состоит из 2^n адресуемых слов, где каждое слово имеет уникальный n-разрядный адрес. При взаимодействии с кэшем эта память рассматривается как *M* блоков фиксированной длины по *K* слов в каждом *($M = 2^n/K$)*. Кэш-память состоит из С блоков аналогичного размера (блоки в кэш-памяти принято называть *строками),* причем их число значительно меньше числа блоков в основной памяти *(C<<M)*. При считывании слова из какого-либо блока ОП этот блок копируется в одну из строк кэша. Поскольку число блоков ОП больше числа строк, отдельная строка не может быть выделена постоянно одному и тому же блоку ОП. По этой причине каждой строке кэш-памяти соответствует *тег* (признак), содержащий сведения о том, копия какого блока ОП в данный момент хранится в данной строке. В качестве тега обычно используется часть адреса ОП.

На эффективность применения кэш-памяти в иерархической системе памяти влияет целый ряд моментов. К наиболее существенным из них можно отнести:

- емкость кэш-памяти;
- размер строки;
- способ отображения основной памяти на кэш-память;
- алгоритм замещения информации в заполненной кэш-памяти;
- алгоритм согласования содержимого основной и кэш-памяти;
- число уровней кэш-памяти.

Выбор **емкости кэш-памяти** — это всегда определенный компромисс. С одной стороны, кэш-память должна быть достаточно мала, чтобы ее стоимостные показатели были близки к величине, характерной для ОП. С

другой — она должна быть достаточно большой, чтобы среднее время доступа в системе, состоящей из основной и кэш-памяти, определялось временем доступа к кэш-памяти. В пользу уменьшения размера кэш-памяти имеется больше мотивировок. Так, чем вместительнее кэш-память, тем больше логических схем должно участвовать в ее адресации. Как следствие, ИМС кэш-памяти повышенной емкости работают медленнее по сравнению с микросхемами меньшей емкости, даже если они выполнены по одной и той же технологии.

Реальная эффективность использования кэш-памяти зависит от характера решаемых задач, и невозможно заранее определить, какая ее емкость будет действительно оптимальной. Установлено, что для большинства задач близкой к оптимальной является кэш-память емкостью от 1 до 512 Кбайт.

Еще одним важным фактором, влияющим на эффективность использования кэш памяти, служит **размер строки**. Когда в кэш-память помещается строка, вместе с требуемым словом туда попадают и соседние слова. По мере увеличения размера строки вероятность промахов сначала падает, так как в кэш, согласно принцип локальности, попадает все больше данных, которые понадобятся в ближайшее время. Однако вероятность промахов начинает расти, когда размер строки становится излишне большим Объясняется это тем, что:

- большие размеры строки уменьшают общее количество строк, которые можно загрузить в кэш-память, а малое число строк приводит к необходимости частой их смены;

- по мере увеличения размера строки каждое дополнительное слово оказывается дальше от запрошенного, поэтому такое дополнительное слово менее вероятно понадобится в ближайшем будущем.

Зависимость между размером строки и вероятностью промахов во многом определяется характеристиками конкретной программы, из-за чего трудно рекомендовать определенное значение величины строки. Исследования показывают, что наиболее близким к оптимальному можно признать размер строки, равный 4-8 адресуемым единицам (словам или байтам). На практике размер строки обычно выбирают равным ширине шины данных, связывающей кэш-память с основной памятью.

10.3 Способы отображения оперативной памяти на кэш-память

Сущность отображения блока основной памяти на кэш-память состоит в копировании этого блока в какую-то строку кэш-памяти, после чего все обращения к блоку в ОП должны переадресовываться на соответствующую строку кэш-памяти. Удачным может быть признан лишь такой способ отображения, который одновременно отвечает трем требованиям:

- обеспечивает быструю проверку кэш-памяти на наличие в ней копии блока основной памяти;

- обеспечивает быстрое преобразование адреса блока ОП в адрес строки кэша;

- реализует достижение первых двух требований наиболее экономными средствами.

Известные варианты отображения основной памяти на кэш можно свести к трем видам: прямому, полностью ассоциативному и частично-ассоциативному, причем последний имеет две модификации — множественно-ассоциативное отображение и отображение секторов.

При **прямом отображении** адрес строки *i* кэш-памяти, на которую может быть отображен блок j из ОП, однозначно определяется выражением: $i=j \bmod m$, где *m* — общее число строк в кэш-памяти (рис. 10.3.1).

Прямое отображение — простой и недорогой в реализации способ отображения. Основной его недостаток — жесткое закрепление за определенными блоками ОП одной строки в кэше. Поэтому если программа поочередно обращается к словам из двух различных блоков, отображаемых на одну и тут же строку кэш-памяти, постоянно станет происходить обновление данной строки и вероятность попадания будет низкой.

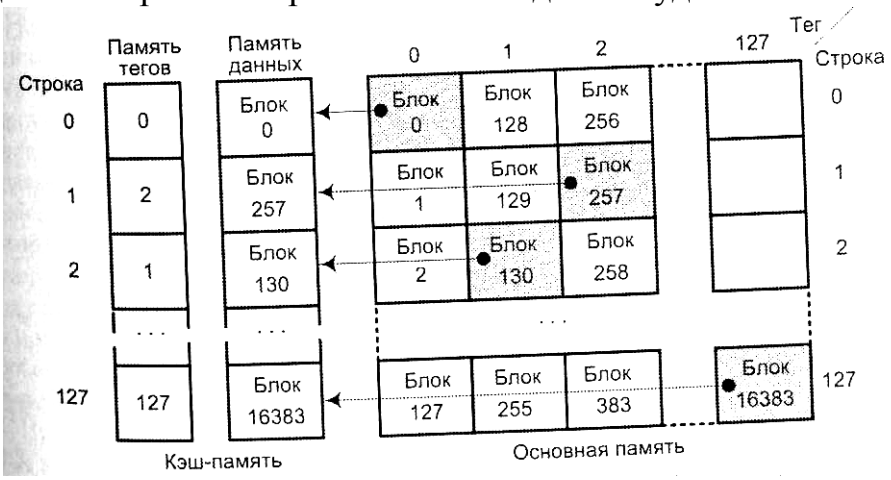

Рис. 10.3.1 Организация кэш-памяти с прямым отображением

Полностью ассоциативное отображение позволяет преодолеть недостаток прямого, разрешая загрузку любого блока ОП в любую строку кэш-памяти. Логика управления кэш-памяти выделяет в адресе ОП два поля: поле тега и поле слова. Поле тега совпадает с адресом блока основной памяти. Для проверки наличия копии блока в кэш-памяти логика управления кэша должна одновременно проверить теги всех строк на совпадение с полем тега адреса. Этому требованию наилучшим образом отвечает ассоциативная память, то есть тег должен храниться в ассоциативной памяти тегов кэша. Логика работы такой кэш-памяти иллюстрируется рис. 10.3.2.

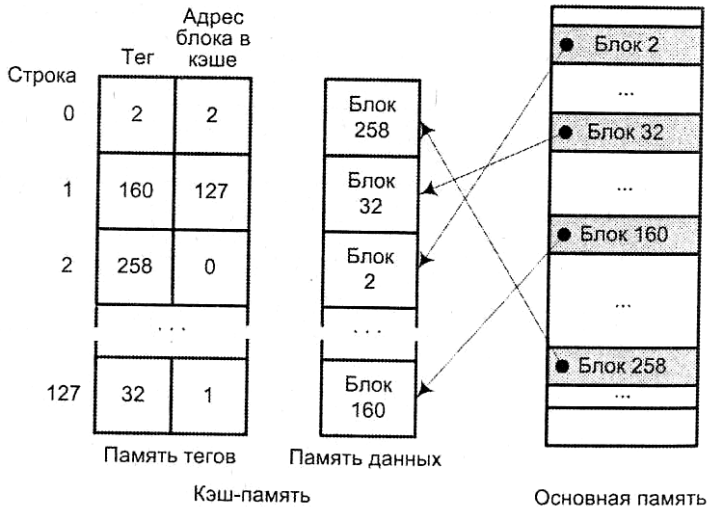

Рис. 10.3.2 Кэш-память с ассоциативным отображением

Ассоциативное отображение обеспечивает гибкость при выборе строки для вновь записываемого блока. Принципиальный недостаток этого способа — необходимость использования дорогостоящей ассоциативной памяти.

Множественно-ассоциативное отображение относится к группе методов частично-ассоциативного отображения. Оно является одним из возможных компромиссов, сочетающим достоинства прямого и ассоциативного способов отображения и, в известной мере, свободным от их недостатков. Кэш-память (как тегов, так и данных) разбивается на v подмножеств (в дальнейшем будем называть такие подмножества модулями), каждое из которых содержит k строк (принято говорить, что модуль имеет k входов). Зависимость между модулем и блоками ОП такая же, как и при прямом отображении: на строки, входящие в модуль, могут быть отображены только вполне определенные блоки основной памяти, в соответствии с соотношением $i = j \bmod v$, где j — адрес блока ОП. В то же время размещение блоков по строкам модуля — произвольное, и для поиска нужной строки в пределах модуля используется ассоциативный принцип.

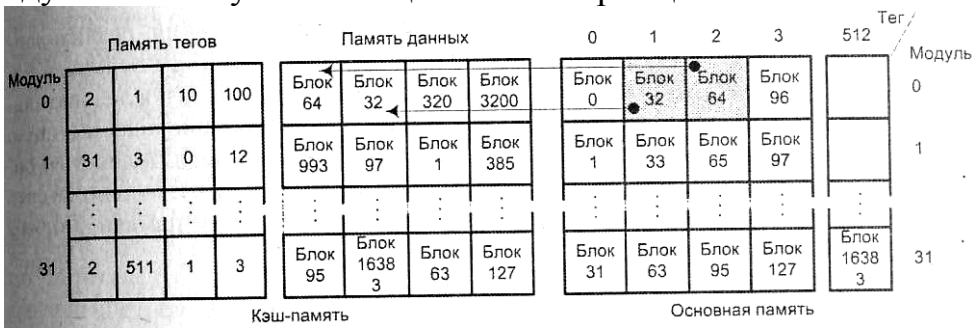

Рис. 10.3.3 Кэш-память с множественно-ассоциативным отображением

На рис. 10.3.3 показан пример четырехвходовой кэш-памяти с множественно-ассоциативным отображением. Память данных кэш-памяти разбита на 32 модуля по 4 строки в каждом. Память тегов содержит 32 ячейки, в каждой из которых может храниться 4 значения тегов (по одному на каждую строку модуля). 14-разрядный адрес блока ОП представляется в виде двух полей: 9-разрядного поля тега и 5-разрядного поля номера модуля.

Номер модуля однозначно указывает на один из модулей кэш-памяти. Он также позволяет определить номера тех блоков ОП, которые можно отображать на этот модуль. Такими являются блоки ОП, номера которых при делении на 2^5дают в остатке число, совпадающее с номером данного модуля кэш-памяти. Так, блоки 0,32,64, 96 и т. д. основной памяти отображаются на модуль с номером 0; блоки 1, 33, 65, 97 и т. д. отображаются на модуль 1 и т. д. Любой из блоков в последовательности может быть загружен в любую из четырех строк соответствующего модуля.

При такой постановке роль тега выполняют 9 старших разрядов адреса блока ОП, в которых содержится порядковый номер блока в последовательности блоков, отображаемых на один и тот же модуль кэш-памяти. Например, блок 65 в последовательности блоков, отображаемых на модуль 1, имеет порядковый номер 2 (отсчет ведется от 0).

При обращении к кэш-памяти 5-разрядный номер модуля указывает на конкретную ячейку памяти тегов (это соответствует прямому отображению). Далее производится параллельное сравнение каждого из четырех тегов, хранящих этой ячейке, с полем тега поступившего адреса, то есть поиск нужного тега среди четырех возможных осуществляется ассоциативно.

В предельных случаях, когда $v = m.$, $k = 1$, множественно-ассоциативное отображение сводится к прямому, а при $v = 1$,$k = m$ — к ассоциативному.

Наиболее общий вид организации множественно-ассоциативного отображение — использование двух строк на модуль *(v = rn/2, k* = 2). Четырехвходовая множественно-ассоциативная кэш-память *(v* = m/4, *k* = 4) дает дополнительное улучшение за сравнительно небольшую дополнительную цену. Дальнейшие увеличения числа строк в модуле существенного эффекта не привносит.

Наиболее общий вид организации множественно-ассоциативного отображение — использование двух строк на модуль *(v = rn/2, k* = 2). Четырехвходовая множественно-ассоциативная кэш-память *(v* = m/4, *k* = 4) дает дополнительное улучшение за сравнительно небольшую дополнительную цену. Дальнейшие увеличения числа строк в модуле существенного эффекта не привносит.

Следует отметить, что именно этот способ отображения наиболее широко распространен в современных микропроцессорах.

Один из первых вариантов **частично-ассоциативного отображения**, реализованный в модели 85 семейства ВМ IBM 360. Согласно данному методу, основная память разбивается на секторы, состоящие из фиксированного числа последовательных блоков. Кэш-память также разбивается на секторы, содержащие такое же число строк. Расположение блоков в секторе ОП и секторе кэш-памяти полностью совпадает. Отображение сектора на кэш-память осуществляется ассоциативно, то есть любой сектор из ОП может быть помещен в любой сектор кэш-памяти. Логику работы кэша поясняет рис, 10.3.4.

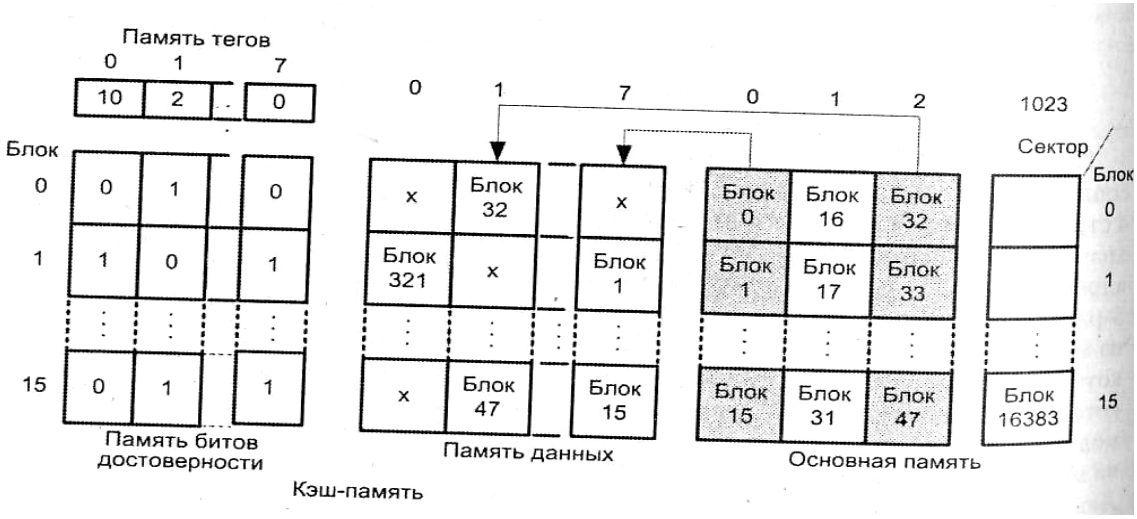

Рис. 10.3.4 Кэш-память с отображением секторов

Здесь сектор состоит из $16 = 2^4$ блоков по 16 слов, и основная память содержит $1024 = 2^{10}$ сектора. В 14-разрядном адресе блока основной памяти старшие 10 разрядов показывают номер сектора, а младшие 4 — номер блока внутри сектор В свою очередь, кэш-память состоит из $8 = 2^3$ секторов, и 7-разрядный адрес строки в кэше включает в себя адрес сектора кэша (3 старших разряда) и номер блока внутри сектора (4 младших разряда). Ввиду того что расположение блоков в секторах основной и кэш-памяти совпадает, для выбора блока внутри сектора в обоих случаях можно использовать четыре младших разряда адреса блока ОП. В результате задача преобразования адресов сводится к переходу от 10-разрядного номера сектора основной памяти к трехразрядному номеру сектора в кэш-памяти. Такое пре- образование осуществляется с помощью ассоциативной памяти тегов на 8 слов числу секторов в кэш-памяти). Каждая ячейка памяти тегов содержит 13 разрядную старших хранят тег — номер сектора ОП, а три младших разряда — номер сектора кэш-памяти, куда отображен данный сектор ОП. Для проверки того, имеется ли копия сектора в кэш-памяти, производится ассоциативный поиск по старшим разрядам памяти тегов. Если произошло совпадение, то для доступа нужному сектору в памяти данных кэш-памяти используются три младших разряда соответствующей ячейки памяти тегов.

Поскольку размер сектора велик, в рассматриваемой схеме отображения копируется не весь сектор целиком, а только требуемый его блок. Для этого к каждой строке, хранимой в кэш-памяти, добавляется один бит информации, называемый *битом достоверности,* показывающий, относится ли данный блок к текущему сектору или в нем осталась информация из сектора, хранившегося на этом месте до смены секторов.

При заполненной кэш-памяти, когда необходимо заменить один из его секторов, в ассоциативной памяти тегов делается запись о новом секторе, как при замене всего сектора полностью. Фактически же в кэш-память пересылается только тот блок сектора, к которому в данный момент обращается процессор, и в соответствующей строке памяти данных бит достоверности устанавливается в единицу. В то же время биты

достоверности остальных строк этого сектора сбрасываются в 0. Далее, по мере копирования других блоков этого сектора, их биты достоверности устанавливаются в единицу.

При такой процедуре заполнения секторов кэш-памяти, в случае обнаружения в ней нужного сектора, предварительно требуется проанализировать состояние бита достоверности соответствующей строки. Если он установлен в 1, значит, обращение к данной ячейке кэш-памяти возможно. При нулевом значении бита достоверности сначала производится копирование нужного блока в кэш-память, его бит Достоверности устанавливается в единицу, и лишь после этого разрешается доступ к указанному адресу в кэш-памяти.

В том случае, когда при замене сектора в кэш-памяти необходимо сохранить его старое содержимое в основной памяти, в ОП пересылаются только заменяемые блоки сектора.

10.4 Алгоритмы замещения информации в заполненной кэш-памяти

Когда кэш-память заполнена, занесение в нее нового блока связано с замещением содержимого одной из строк. При прямом отображении каждому блоку основной памяти соответствует только одна определенная строка в кэш-памяти, и никакой иной выбор удаляемой строки здесь невозможен. При полностью и частично ассоциативных способах отображения требуется какой-либо алгоритм замещения (выбора удаляемой из кэш-памяти строки).

Основная цель стратегии замещения — удерживать в кэш-памяти строки, которым наиболее вероятны обращения в ближайшем будущем, и заменять строки, доступ к которым произойдет в более отдаленном времени или вообще не случится. Очевидно, что оптимальным будет алгоритм, который замещает ту строку, обращение к которой в будущем произойдет позже, чем к любой другой строке кэша. К сожалению, такое предсказание практически нереализуемо, и приходится привлекать алгоритмы, уступающие оптимальному. Вне зависимости от используемого алгоритма замещения для достижения высокой скорости он должен быть реализован аппаратными средствами.

Среди множества возможных алгоритмов замещения наиболее распространенными являются четыре, рассматриваемые в порядке уменьшения их относительной эффективности.

Наиболее эффективным является алгоритм замещения на основе *наиболее давнего использования* (LRU — Least Recently Used), при котором замещается та строка кэш-памяти, к которой дольше всего не было обращения. Проводившиеся исследования показали, что алгоритм LRU, который «смотрит» назад, работает достаточно хорошо в сравнении с оптимальным алгоритмом, «смотрящим» вперед, Наиболее известны два способа аппаратурной реализации этого алгоритма. В первом из них с каждой строкой кэш-памяти ассоциируют счетчик. К содержимому всех счетчиков через определенные интервалы времени добавляется единица. При

обращении к строке ее счетчик обнуляется. Таким образом, наибольшее число будет в счетчике той строки, к которой дольше всего не было обращений, и эта строка — первый кандидат на замещение.

Второй способ реализуется с помощью очереди, куда в порядке заполнения строк кэш-памяти заносятся ссылки на эти строки. При каждом обращении к строке ссылка на нее перемещается в конец очереди. В итоге первой в очереди каждый раз оказывается ссылка на строку, к которой дольше всего не было обращений. Именно эта строка прежде всего и заменяется.

Другой возможный алгоритм замещения — алгоритм, работающий по принципу *«первый вошел, первый вышел»* (FIFO — First In First Out). Здесь заменяется строка, дольше всего находившаяся в кэш-памяти. Алгоритм легко реализуется с помощью очереди, с той лишь разницей, что после обращения к строке положение соответствующей ссылки в очереди не меняется.

Еще один алгоритм — замена *наименее часто использовавшейся* строки (Lr и Least Frequently Used). Заменяется та строка в кэш-памяти, к которой было меньше всего обращений. Принцип можно воплотить на практике, связав каждую строку со счетчиком обращений, к содержимому которого после каждого обращения добавляется единица. Главным претендентом на замещение является строка, счетчик которой содержит наименьшее число.

Простейший алгоритм — *произвольный выбор* строки для замены. Замещаемая строка выбирается случайным образом. Реализовано это может быть, например с помощью счетчика, содержимое которого увеличивается на единицу с каждым тактовым импульсом, вне зависимости от того, имело место попадание или промах. Значение в счетчике определяет заменяемую строку в полностью ассоциативной кэш-памяти или строку в пределах модуля для множественно-ассоциативной кэш-памяти. Данный алгоритм используется крайне редко. Среди известных в настоящее время систем с кэш-памятью наиболее встречаемым является алгоритм LRU.

10.5 Алгоритмы согласования содержимого кэш-памяти и основной памяти

В процессе вычислений ЦП может не только считывать имеющуюся информацию, но и записывать новую, обновляя тем самым содержимое кэш-памяти. С другой стороны, многие устройства ввода/вывода умеют напрямую обмениваться информацией с основной памятью. В обоих вариантах возникает ситуация, когда содержимое строки кэша и соответствующего блока ОП перестает совпадать. В результате на связанное с основной памятью устройство вывода может быть выдана «устаревшая» информация, поскольку все изменения в ней, сделанные процессором фиксируются только в кэш-памяти, а процессор будет использовать старое содержимое кэш-памяти вместо новых данных, загруженных в ОП из устройства ввода.

Для разрешения первой из рассмотренных ситуаций (когда процессор выполняет операцию записи) в системах с кэш-памятью предусмотрены

методы обновления основной памяти, которые можно разбить на две большие группы: *метод сквозной записи* (write through) и *метод обратной записи* (write back).

По методу сквозной записи прежде всего обновляется слово, хранящееся в основной памяти. Если в кэш-памяти существует копия этого слова, то она также обновляется. Если же в кэш-памяти отсутствует нужная копия, то либо из основной памяти в кэш-память пересылается блок, содержащий обновленное слово *(сквозная запись с отображением),* либо этого не делается *(сквозная запись без отображения).*

Главное достоинство метода сквозной записи состоит в том, что когда строка в кэш-памяти назначается для хранения другого блока, то удаляемый блок можно не возвращать в основную память, поскольку его копия там уже имеется. Метод достаточно прост в реализации. К сожалению, эффект от использования кэш-памяти (сокращение времени доступа) в отношении к операциям записи здесь отсутствует.

Определенный выигрыш дает его модификация, известная как *метод буферированной сквозной записи.* Информация сначала записывается в кэш-память и в специальный буфер, работающий по схеме FIFO. Запись в основную память производится уже из буфера, а процессор, не дожидаясь ее окончания, может сразу же продолжать свою работу. Конечно, соответствующая логика управления должна заботиться о том, чтобы своевременно «опустошать» заполненный буфер. При использовании буферизации процессор полностью освобождается от работы с ОП. Согласно методу обратной записи, слово заносится только в кэш-память. Если соответствующей строки в кэш-памяти нет, то нужный блок сначала пересылается из ОП, после чего запись все равно выполняется исключительно в кэш-память, При замещении строки ее необходимо предварительно переслать в соответствующее место основной памяти. Для метода обратной записи, в отличие от алгоритма сквозной записи, характерно то, что при каждом чтении из основной памяти осуществляются две пересылки между основной и кэш-памятью. У рассматриваемого метода есть разновидность — *метод флаговой обратной записи.* Когда в какой-то строке кэша производится изменение, устанавливается связанный с этой строкой бит изменения (флажок). При замещении строка из кэш- памяти переписывается в ОП только тогда, когда ее флажок установлен в 1. Эффективность флаговой обратной записи несколько выше.

В среднем обратная запись на 10% эффективнее сквозной записи, но для ее реализации требуются и повышенные аппаратные затраты. С другой стороны, практика показывает, что операции записи составляют небольшую долю от общего количества обращений к памяти. Обычно считают, долю операций записи величиной в диапазоне от 5 до 34%. Таким образом, различие по быстродействию между рассмотренными методами невелико.

Теперь рассмотрим ситуацию, когда в основную память из устройства ввода минуя процессор, заносится новая информация и неверной становится копия, хранящаяся в кэш-памяти. Предотвратить подобную

несогласованность позволяют два приема. В первом случае система строится так, чтобы ввод любой информации в ОП автоматически сопровождался соответствующими изменениями в кэш-памяти. Для второго подхода «прямой» доступ к основной памяти допускается только через кэш-память.