11. Назначение и организация кэш-памяти

Под памятью вычислительной системы в широком смысле слова понимается целый спектр устройств, предназначенных для хранения информации. К сожалению, пока нет такого устройства, которое позволяло бы хранить **большие объемы данных**, выполнять операции с памятью с очень **высокой скоростью** и быть при этом **очень дешевым.** Перечисленные требования к устройствам памяти пока являются противоречивыми и одновременно не достигаются. Поэтому современные вычислительные системы вынуждены использовать **многоуровневую** организацию памяти, где каждый уровень имеет свои параметры.

Эти уровни в порядке уменьшения объемов, увеличения скорости доступа и увеличения стоимости хранения одного бита можно представить следующим образом:

- внешняя дисковая память;
- основная оперативная память;
- быстрая промежуточная кэш-память;
- сверхбыстрая регистровая память.

Здесь хорошо видны место кэш-памяти и ее роль: уменьшение разрыва между быстродействием процессора и быстродействием основной памяти. Все последние годы скорость работы процессора остается примерно в 10 раз выше скорости обращения к основной памяти, что, конечно же, не позволяет эффективно использовать возможности процессоров. Кэш-память предназначена для **кратковременного** хранения **наиболее часто** используемых команд и данных. Довольно часто кэш-память сама разбивается на 2 уровня: более быстрый и менее емкий 1-й уровень (десятки Кбайт) и менее быстрый, но более емкий 2-й уровень (сотни Кбайт).

Кэш-память можно представить как массив записей с возможностью **аппаратной** обработки этих записей. Каждая запись содержит:

- адрес команды или элемента данных в основной памяти;
- код команды или значение элемента данных;

- дополнительную управляющую информацию, необходимую механизму кэширования: признак действительности данных, признак модификации данных.

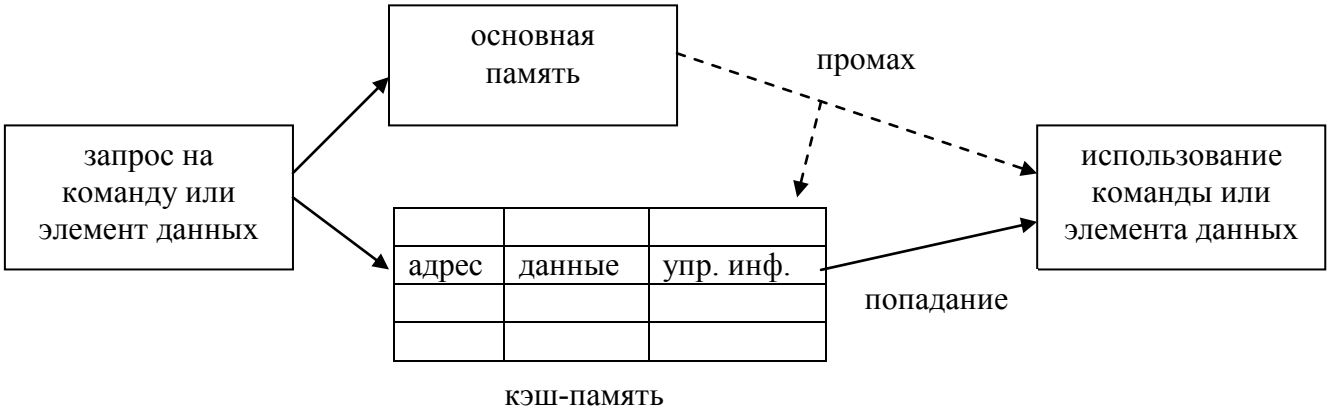
Алгоритм работы кэш-памяти отличается для случая **выборки** команд/данных из памяти и для случая **сохранения** данных в памяти. В первом случае выполняются следующие шаги:

- когда процессору необходима очередная команда или элемент данных, он обращается за ними одновременно к основной памяти и к кэш-памяти ;
- поскольку поиск в кэш-памяти выполняется значительно быстрее, то в случае наличия этих данных в кэше они выбираются именно из него, а поиск в основной памяти прекращается (эта ситуация иногда называется **кэш-попаданием**);
- если искомых данных в кэше нет **(кэш-промах),** то данные извлекаются из основной памяти, передаются на обработку процессору и одновременно заносятся в кэш.

В этом алгоритме не было бы ничего особенного, если бы не одна очень существенная деталь, во многом определяющая весь смысл кэширования: в случае промаха из ОП извлекается и помещается в кэш не только затребованный элемент, но и **целый блок соседних с ним элементов** (команд или данных). Такое поведение объясняется локальным характером размещения команд и данных. Машинные команды в основном обрабатываются последовательно друг за другом, а переходы на другие команды часто бывают в пределах относительно небольшой группы (например, цикл, многократно выполняющий несколько десятков команд). Аналогично, элементы массивов чаще всего размещаются в памяти и обрабатываются последовательно, элемент за элементом. Поэтому скопированная в кэш группа соседних команд или элементов данных позволит в дальнейшем выбирать их из быстрого кэша, а не из более медленной ОП.

Иногда такое свойство машинных программ называют **принципом локальности:** после обращения к памяти по некоторому адресу очень высока вероятность обращения в ближайшее время по соседним адресам. Отсюда можно сделать вывод, что работа кэш-памяти носит **вероятностно-статистический** характер: на несколько кэш-попаданий приходится один кэш-промах. Современные реализации кэш-памяти обеспечивают весьма хороший показатель - около 90% попаданий.

Работу алгоритма кэширования при **запросе** данных можно схематично показать следующим образом:



В начале работы системы кэш-память пустая, но по мере работы она заполняется наиболее часто используемой информацией. Такую работу кэш-памяти иногда называют “разгоном”.

Теперь рассмотрим работу кэш-памяти в случае **сохранения** информации в ОП. Прежде всего, надо отметить тот очевидный факт, что сохранять в ОП надо только измененные в процессе работы **данные.** Машинные команды в процессе своего использования изменяться не должны, и поэтому выгружать их из кэша не надо. Если кэш-память полностью заполнена командами (а так будет всегда при достаточно интенсивной работе системы) и требуется записать в нее новую группу команд, эти новые команды просто стирают старые.

Сложнее ситуация с сохранением данных, поскольку здесь возникает задача **согласования** данных в ОП и в кэш-памяти. Общий алгоритм работы

механизма кэширования при выполнении команды сохранения элемента данных в ОП выглядит следующим образом:

- при выполнении команды сохранения становится известен адрес, по которому в ОП надо сохранить элемент данных;
- этот адрес на аппаратном уровне отыскивается среди записей кэш-памяти, и если его там нет, то сохранение происходит только в ОП;
- в противном случае реализуется один из следующих подходов:
    - **немедленное** сохранение: запись производится как в ОП, так и в кэш;
    - **отложенное** сохранение: данные сохраняются только в кэше и помечаются как измененные; сохранение этих данных в ОП производится при необходимости освобождения кэша для размещения других данных или принудительно в фоновом режиме при малой загрузке системы.

Для эффективной работы механизма кэш-памяти большое значение имеет ее **организация**. Здесь существует несколько способов со своими особенностями и ограничениями. Основными способами являются следующие два:

1. Кэш-память имеет описанную ранее структуру в виде массива записей. Новая информация, извлекаемая из ОП в случае кэш-промаха, размещается в этом массиве **случайным** образом. Поиск в кэше выполняется по запрошенному адресу, но реализуется не программно (прямой перебор адресов в массиве совершенно не приемлем в силу своей низкой скорости), а аппаратно, за счет **одновременного** сравнения всех адресов в кэше с искомым. Такой поиск принято называть **ассоциативным**, так же называются запоминающие устройства, обладающие такой возможностью. Каждая ячейка памяти такого устройства состоит из двух частей: в одной хранится сама требуемая информация (запрошенная команда или элемент данных), а в другой – **уникальный ключ поиска** (это поле часто называют **тэгом**). Каждое поле тэга содержит аппаратно реализованную логику,

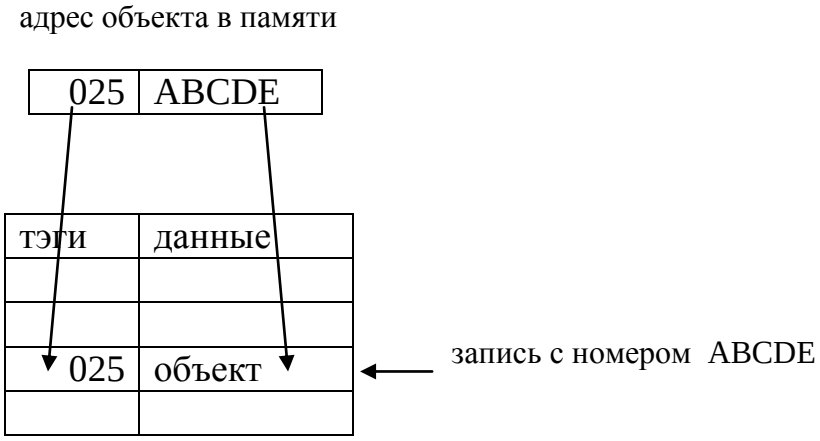
которая позволяет за **одно обращение** к памяти однозначно определить нужную ячейку. В данном случае ключом поиска является адрес затребованного объекта. Этот метод весьма дорог и поэтому используется чаще всего в кэш-памяти первого уровня, имеющей небольшой объем, но наиболее высокую скорость доступа.

2. Один и тот же адрес ОП всегда отображается в кэше на одно и то же место, и поэтому метод называется **детерминированным.** Он позволяет по затребованному адресу ОП вычислять номер ячейки в кэше с помощью простых побитовых операций. Например, если кэш-память имеет объем 1 Мбайт, то в адресе можно взять младшие 20 разрядов (2^{20} = 1 Мб), которые и определяют размещение объекта среди записей кэша. К сожалению, здесь возникает одна проблема, связанная с разными объемами основной памяти и кэш-памяти. Всю ОП объемом, например, 256 Мб нельзя без конфликтов отобразить на кэш размером в 1 Мб. На одну и ту же ячейку кэш-памяти потенциально могут претендовать разные адреса ОП, а именно – все, у которых одинаковы используемые для отображения наборы младших битов. В частности, для рассмотренного выше кэша в 1 Мб разные адреса ОП 123ABCDE, 052ABCDE, FFFABCDE претендуют на одну и ту же ячейку в кэш-памяти с номером ABCDE (младшие 20 разрядов).

Для устранения этой неоднозначности приходится в кэше вместе с содержательной информацией хранить в каждой ячейке и дополнительную информацию. Для этого опять же используется поле тэга, в которое записываются оставшиеся “без дела” старшие биты адреса (например – старшие 12 разрядов). При выборе команды или элемента данных из ОП происходит следующее:

- адрес объекта разделяется на младшую и старшую части в соответствии с объемом установленной в системе кэш-памяти;
- по младшим разрядам определяется номер записи в КЭШе;
- в основную часть этой записи заносится сам объект (команда или элемент данных);

- в дополнительную (тэговую) часть заносятся оставшиеся старшие разряды адреса.



Если при этом необходимая ячейка кэш-памяти оказывается занятой, она либо просто заменяется новыми данными (если хранящимся в ней объектом является команда или немодифицированный элемент данных), либо вытесняется в ОП.

Аналогично, при поиске объекта в кэш-памяти по младшей части адреса производятся обращение к соответствующей ячейке и сравнение хранящегося в ее тэге значения со старшей частью адреса. При совпадении этих значений фиксируется кэш-попадание, а иначе – кэш-промах.

В целом, второй способ дешевле первого, и поэтому его используют для реализации кэш-памяти второго уровня, имеющей больший объем, но меньшую скорость доступа.