Суть кэшировании: сокращение задержек доступа

Кэш состоит из двух частей: контроллер, который управляет кэш памятью, и сама кэш память. В компах используется байтовая адресация.

Чтение и записи в оперативку происходит пакетами данных, которая может состоять из нескольких машинных слов. Эти данные сохраняются в кэше, как в неком промежуточном хранилище.

Когда процессор формирует запрос на получение данных, он перехватывается контроллером, который определяет, имеется ли копия требуемых данных в кэше, если они есть, данные миную оперативную память, передаются процессору, если нет, запрос передаётся оперативной памяти. Изменённые данные записываются обратно в кэш, потом они будут выгружены обратно в оперативную память, на то место, где эти данные располагались. Обычно это происходит, когда происходит их замена другими данными, такой режим работы кэша называется обратная запись. Он работает быстрее, чем сквозная запись - изменения сразу записываются как в кэш, так и в основную память.

**Кэш строка**

Чтобы вернуть изменённый пакет данных из кэша обратно в оперативную память, нужно знать, по какому адресу он находится в оперативной памяти. Этот адрес нужно где-то хранить. Самое простое, что может прийти в голову – хранить байт данных вместе с его адресом. Это занимало бы в 5 раз больше места, например, если у нас 32-х битная система, а размер адреса – 4 байта. Чтобы так не расходовать кэш память, данные в кэше сохраняются не в виде отдельных байтов, а в виде целых блоков данных, фиксированного размера, идущих в оперативной памяти подряд. Такой блок данных называется кэш строкой или кэш линией. В современных x86 процессорах ее размер как правило составляет 64 байта. В каждой такой строке вместо адреса для каждого байта хранится 1 адрес из оперативной памяти, который соответствует адресу 1 сохраняемого в ней байта данных.

Как в одну 64 байтовую строку помещается сразу и 64 байта данных и 4 байта с адресом?

Начнем с того, какого размера может быть кэш память. Чтобы ее можно было полностью заполнить, ее размер должен быть кратен степени числа 2.

Полезный размер кэша - размер без указания служебных данных, который тоже хранится в кэш памяти. Полный размер кэша складывается из самих данных, которые записываются в него из оперативки + различные служебные данные, к примеру, 4-байтовый адрес.

Пример, говоря, что кэш размером 128 байт, состоящий из 4 строк по 32 байта, его полный размер будет составлять 32 \* 4 + 4 \* 4 = 144 байта.

**Кэш прямого отображения**

Как понять, в какую конкретно строку кэш памяти попадет определенный блок данных из оперативной памяти. Возьмем кэш на 256 байт, в котором размер каждой строки 64 байта. Получится, что в нашем кэше будет всего 4 строки. Возьмем оперативку с размером 1кб, которую условно разобьем на 4 сегмента, каждый из которых будет равен размеру кэша. В каждом таком сегменте заранее разделим данные условно по 64 байта, так как это размер кэш строки. Каждая такая строка из оперативной памяти будет записано в строку кэша. Осталось понять, в какую.

Один из популярных вариантов отображения оперативной памяти на кэш - кэш прямого отображения. Смысл его в том, что каждая строка оперативной памяти соответствует только одной строго определённой строке кэш памяти. Каждой строке кэш памяти будет соответствовать несколько строк из оперативной памяти, так как количество строк кэш памяти меньше строк в оперативной памяти.

Разделив по модулю номер каждой строки из оперативной памяти на кол-во строк кэша, получим индекс кэш строки, в которой будет помещена каждая строка из оперативки. (n mod 4)

Получится, что в первую кэш строку с индексом 0 попадут строки из оперативной памяти с индексом 0, 4, 8 и 12.

Преимущество способа кэша прямого отображения заключается в очень быстром поиске нужной кэш строки. Зная адрес конкретного байта, или конкретного слова, которое нужно получить процессору, можно узнать, в какой кэш строке оно должно находиться. Например, байт из оперативной памяти с адресом 129. Нам достаточно лишь разделить его на размер кэш строки и все это поделить по модулю на кол-во строк кэша(129 / 64 mod 4 = 2). Получится, что этот байт будет находиться в кэш строке с индексом 2.

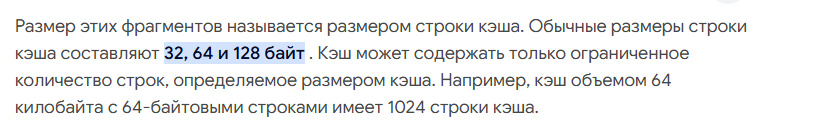
**Адрес процессора**

32-x битный адрес данных в кэш строке будет подразделяться на две основные части: смещение до нужно слова в строке, индекс самой строки, тег, который представляет собой старшую часть адреса байта. Тег также представляет собой номер сегмента оперативной памяти и он же является числом, которое будет сравниваться с тем самым адресом 1 байта, который хранится в каждой кэш строке. Сколько бит будет составлять каждая эта часть зависит от размеров кэша. Например, смещение зависит от длины кэш строки. То есть чтобы мы могли обратиться к любому из 64 байт внутри каждой кэш строки, нам потребуется 6 бит(степени 2)(2^6 = 64), если строка размером 32 байта, то это 5 бит.Но так как процссеор высчитывает данные сразу по словам, то есть в нашем примере 32-х битный процессор будет считывать из кэш памяти сразу по 4 байта, то 6 битное смещение делится еще на две части: где 4 бита отводятся для поиска нужного слова, так как их всего 16 в 64-байтовой строке, 2 бита для поиска одного из 4 байта внутри слова. Индекс строки зависит от кол-ва строк в кэше. Чтобы нам обратить к одной из 4 строк понадобится 2 бита(так как 2^2 = 4).

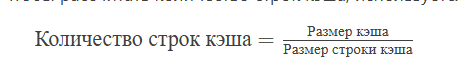
Возьмем слово, которое начинается с адреса 64. Мы знаем, что оно попадет в кэш строку с индексом 1, и что это будет самое первое слово в строке, поэтому вместо перебора всех кэш строк для поиска нужной и средней части адреса извлекается номер кэш строки и мы сразу попадаем в нужную нам строку. В кэш строке могут находится и другие данные из оперативной памяти. Чтобы понять, что в этой строке находятся именно те данные, которе мы ищем, поле тега сравнивается с адресом который записывается в каждой кэш строке, он тоже называется тег. Так как размер данных = размеру кэш строки и отсчет начинается с нуля, то этот адрес в каждой кэш строке будет кратен размеру кэш строки и размеру данных. То есть адрес первого байта – 0, адрес 65 байта – 64, и т.д Если адреса представить в двоичном виде, то можно заметить, что в каждом из них последние 6 цифр всегда будут равны 0. Количество последних нулей также зависит от размера кэш строки, то есть в 32- байтовых строках адрес будет кратен 32 – 5 нулей, 60-байтовых строках 4 нуля. Их можно вообще больше не хранить. Получается теперь ,вместо 32-битного адреса можно хранить 26 бит(было 6 нулевых бит). Поле тега в адресе составляет 24 бита. Если из оставших 26 бит в адресе кэш строки мы оставим старшие 24 бита.

Те 4 строки из оперативной памяти, которые претендуют на одну кэш строку, всегда будут находиться в разных сегментах памяти. Поэтому если мы обращаемся к слову, которое находится в 1 сегменте, нам достаточно знать лишь номер этого сегмента, чтобы понять, действительно ли в этой кэш строке содержится искомое нами слово. Число из поля тега в адресе сравнивается со старшей частью адреса, которое записано в теге кэш строки и если всё совпало, то такая ситуация называется кэш попадание.

Недостаток кэша с прямым отображением, что разные строки оперативки претендуют занять одну и ту же строку в кэш памяти. Это порождает возможность частых коллизий. Если нам нужно получить слово из строки номера 4, но в кэш строке, где оно должно находиться, уже лежит другая строка с номером 8, то это называется кэш промахом. В таком случае нам придётся идти в оперативную память и перезаписывать кэш строку нужными нам данными.



Размер строки кэша — это количество байт данных, которое может быть записано или прочитано за одну операцию в кэше. Чтобы рассчитать количество строк кэша, используется следующая формула:



В кэше прямого отображения (Direct Mapped Cache) каждая строка кэша связана с конкретным блоком памяти, и эта связь определяется адресом памяти. Для этого используется **тэг** (tag), который позволяет идентифицировать, какие данные находятся в строке кэша. Размер тэга зависит от структуры адреса памяти и организации кэша.

**Как выбирается размер тэга?**

1. **Структура адреса памяти:**  
   Адрес памяти, который используется для доступа к данным, делится на несколько частей:
   * **Индекс (Index):** Указывает, в какой строке кэша находятся данные.
   * **Тэг (Tag):** Используется для идентификации конкретного блока данных в строке кэша.
   * **Смещение (Offset):** Указывает, какой байт внутри строки кэша нужно прочитать.
2. **Формула для вычисления размера тэга:**  
   Размер тэга зависит от общего размера адреса памяти, размера строки кэша и количества строк кэша. Формула выглядит так:

Размер тэга=Размер адреса памяти−log⁡2(Количество строк кэша)−log⁡2(Размер строки кэша)Размер тэга=Размер адреса памяти−log2​(Количество строк кэша)−log2​(Размер строки кэша)

Где:

* + log⁡2(Количество строк кэша)log2​(Количество строк кэша) — количество бит, необходимых для индекса.
  + log⁡2(Размер строки кэша)log2​(Размер строки кэша) — количество бит, необходимых для смещения.

**Пример расчета размера тэга**

Допустим, у нас есть:

* Размер адреса памяти: 32 бита.
* Размер строки кэша: 64 байта (512 бит).
* Размер кэша: 64 КБ.
* Размер строки кэша: 64 байта.

1. **Рассчитаем количество строк кэша:**

Количество строк кэша=Размер кэшаРазмер строки кэша=64 КБ64 байта=1024 строкиКоличество строк кэша=Размер строки кэшаРазмер кэша​=64 байта64 КБ​=1024 строки

1. **Рассчитаем количество бит для индекса:**

log⁡2(1024)=10 битlog2​(1024)=10 бит

1. **Рассчитаем количество бит для смещения:**

log⁡2(64)=6 битlog2​(64)=6 бит

1. **Рассчитаем размер тэга:**

Размер тэга=32 бита−10 бит−6 бит=16 битРазмер тэга=32 бита−10 бит−6 бит=16 бит

Таким образом, в данном примере размер тэга составляет 16 бит.

**Объяснение структуры адреса:**

1. **Индекс (Index):**
   * Используется для выбора строки кэша.
   * В примере выше индекс занимает 10 бит, что позволяет адресовать 1024 строки кэша.
2. **Смещение (Offset):**
   * Указывает, какой байт внутри строки кэша нужно прочитать.
   * В примере выше смещение занимает 6 бит, что позволяет адресовать 64 байта внутри строки.
3. **Тэг (Tag):**
   * Используется для идентификации конкретного блока данных в строке кэша.
   * В примере выше тэг занимает 16 бит, что позволяет различать разные блоки данных, которые могут быть сохранены в одной строке кэша.

### ****2. Как работает тег?****

Тег используется для того, чтобы процессор мог определить, содержит ли строка кэша нужные данные. Процесс сравнения данных в кэше и оперативной памяти выглядит следующим образом:

1. **Процессор формирует адрес:**  
   Когда процессор хочет прочитать или записать данные, он формирует адрес в оперативной памяти. Этот адрес состоит из тега, индекса и смещения.
2. **Процессор обращается к кэшу:**
   * Процессор использует **индекс** для выбора строки кэша.
   * В выбранной строке кэша процессор сравнивает **тег** из адреса с тегом, хранящимся в строке кэша.
3. **Сравнение тегов:**
   * Если тег из адреса совпадает с тегом в строке кэша, это означает, что данные находятся в кэше (**кэш-попадание**).
   * Если тег из адреса не совпадает с тегом в строке кэша, это означает, что данные отсутствуют в кэше (**кэш-промах**).
4. **Кэш-попадание:**
   * Если тег совпал, процессор использует **смещение** для получения нужного байта из строки кэша.
   * Данные считываются из кэша, и операция завершается.
5. **Кэш-промах:**
   * Если тег не совпал, процессор обращается к оперативной памяти для получения данных.
   * Данные из оперативной памяти загружаются в строку кэша, и операция повторяется.

Чтобы облегчить эту проблему, нужно чтобы в каждом элементе кэш памяти помещалось сразу несколько строк. Поэтому придумали ещё один вид кэш памяти, называемый **наборно-ассоциативный** кэш. Весь кэш делится на несколько сегментов, называемый каналами. Каждый из которых является обычным кэшем прямого отображения. Сами сегменты являются полностью ассоциативны по отношению к оперативной памяти, то есть любая строка оперативки может быть размещена в любом сегменте кэша, но внутри каждого сегмента ей будет соответствовать строго определённая кэш строка. Количество каналов тоже кратно степени 2 (2, 4, 8 и т.д.).

Например, возьмем 4 канальный кэш, в каждом из которых будет 4 строки. Определить, какая строка из оперативной памяти в какую кэш строку попадет можно точно также, как делали до этого. Только теперь отличие в том, что она может попасть в любой из 4 каналов. Адрес к данным в кэше остается точно таким же, только теперь берется тег строки с нужным индексом в каждом из 4 каналов – этот набор еще называется сетом и одновременно сравнивается с тегом в адресе, то есть сколько будет каналов, столько будет одновременно сравниваемых тегов. Это снижает вероятность коллизий.

**Полностью ассоциативный** кэш – позволяет уменьшить число коллизий. Любая строка оперативной памяти может попасть в любую строку кэша. Низкая производительность. Чтобы найти нужную строку, придется перебрать все строки кэша, сравнивая их теги, поэтому это может быть актуально, только если кэш иметь относительно небольшой размер.

**Алгоритмы замещения:**

Чтобы свести к минимуму количество кэш промахов, кэш контроллер должен предсказывать, какие данные потребуются процессору в следующий момент времени. Постоянно подгружая их, замещая старые ненужные данные. Чтобы определить, какие данные являются ненужными, используются различные алгоритмы:

LRU (least recently used) - замещаются те данные, к которым дольше всего не было обращений. То есть ожидается, что данные будут использоваться повторно в ближайшее время.

MFU (most frequently used) - замещаются последние используемые данные. Ожидается, что данные не будут использованы повторно в ближайшее время.

LFU(least frequently used) - замещаются данные, которые использовались реже всех.

Все такие алгоритмы требуют добавления в служебные данные специальный счётчик возраста.

Какие строки должны прийти на место тех, которые мы замещаем.

Тоже есть алгоритмы:

1.Последовательные данные, и учитывать соседние с ними.

2.Предсказывают следующие запрашиваемые ячейки памяти, на основе анализов предыдущих обращений и кэш промахов.

Аппаратная и программная предвыборка данных — это два подхода к улучшению производительности системы за счет уменьшения количества промахов в кэше (cache misses). Оба метода направлены на то, чтобы заранее загрузить данные в кэш, чтобы они были доступны, когда программа их запросит.

1. Аппаратная предвыборка данных (Hardware Prefetching)

Аппаратная предвыборка реализуется на уровне процессора и выполняется автоматически без участия программиста. Процессор анализирует паттерны доступа к памяти и пытается предсказать, какие данные будут нужны в ближайшее время, чтобы загрузить их в кэш заранее.

2. Программная предвыборка данных (Software Prefetching)

Программная предвыборка реализуется на уровне кода программы. Программист явно указывает, какие данные нужно загрузить в кэш заранее, используя специальные инструкции (например, prefetch в x86).

**Многоуровневый кэш**

Кэш в современных процессорах состоит из нескольких уровней. Изначальная задача кэша ускорить работу компьютера, но чем больше размер кэша, тем больше будет требоваться времени для поиска и получения в нем данных, тем ниже будет его скорость. Нужен некий баланс между размером и скоростью. Поэтому использование нескольких маленьких, но быстрых кэшей будет лучшим вариантом, чем использовать один большой, но медленный. Поэтому кэш представляет собой целую иерархическую структуру. Как правило, большинство процессоров intel имеют 3 уровня:L1, L2, L3.

Абстрактный пример процессора:

L1 делится на 8 канальный кэш данных и 4 канальный кэш инструкций, каждое по 32 кб. Принадлежит только к конкретному ядру процессора.

L2 8 канальный, 512 кб - больше и медленнее L1.

L3 16 канальный, размером 2 мб. Самый большой и самый медленный. Разделяется между всеми ядрами процессора.

**Взаимодействие уровней.**

Пример, когда кэш состоит из двух уровне: L2 может строится относительно L1 по одной из двух архитектур:

Инклюзивное - L2 дублирует содержимое L1

Эксклюзивное, содержимое не дублируется

Инклюзивный

Строка из оперативной памяти загружается в L2. Если он заполнен, заменяется менее ценная строка, далее L2 передаёт полученные данные кэшу L1. Если тот заполнен, снова ищутся строки, которые можно перезаписать. Так данные будут присутствовать как в L1, так и в L2.

Эсклюзивный

Начинается всё точно также, но при нехватке места в L1. Вместо перезаписи в ней старой строки L1 просто обменивается ей с L2. То есть L2 передаёт в L1 новую полученную из оператива строку, а L1 на это место передаёт свою старую строку.

Инклюзивный для трёх

L3 является инклюзивным по отношению к L1 и L2. L2 и L1 являются не инклюзивными и не эксклюзивными по отношению друг к другу. Когда процессору потребуются данные, они будут искаться в L1, если нету, то в L2. Если L3 инклюзивный, то там можно не искать, так как в нём их точно нет. Если L3 был бы эсклюзивным, то искать пришлось бы.

Многоядерном сложнее. Пришлось бы производить поиск во всех трёх уровнях. В случае не нахождения дальше зависит от архитектуры. Включающий (инклюзивный) L3 означал бы, что данные в остальных ядрах можно не искать, так как их нет в L3, то в нижестоящих кэшах других ядер их тоже не будет.

Исключающий(эсклюзивный) L3 пришлось бы производить поиск.

В исключающей поиск медленее, чем в включающей, но в исключающей можно хранить данных больше.

