1 Предыстория

1.1 Проблемы с быстродействием

Современные CPU работают значительно быстрее, чем оперативная память, которая работает на значительно меньших частотах.
Рассмотрим типичное время выполнения нескольких операций на CPU:

операция	примерное время выполнения
+ или -	1-2 такта
·	4 такта
/ или %	20-40 тактов
обращение к RAM	100-200 тактов

Например, сложение двух чисел, хранящихся в RAM, займет примерно $100 + 100 + 1 = 201$ такт.

1.2 Регистры CPU

Регистры – внутренние ячейки памяти CPU, расположенные на том же кристалле (в отличие от RAM).
Являются *статической* (SRAM) памятью.

Они работают с очень высокой скоростью, но их количество мало – порядка нескольких десятков, в зависимости от процессора. Это объясняется тем, что разместить память в непосредственной близости от CPU (для достижения максимальной скорости) технически сложно.

Регистры видны программно, и их может использовать компилятор для оптимизации программы. Например, в старых версиях языка C++ было ключевое слово `register`, которое подсказывало компилятору, что переменную по возможности следует поместить в регистр.

2 Кеширование

2.1 Концепция

В общем случае, **кеш** – промежуточный буфер с быстрым доступом, содержащий часть информации, которая может быть запрошена с высокой вероятностью. Он позволяет оптимизировать время запроса к хранящейся в нем информации, поскольку работает быстрее, чем источник информации.
Кеширование применяется как в CPU, так и в браузерах, жестких дисках, DNS и пр.

2.2 Кеш-память CPU

Кеш-память – память небольшого объема с высокой скоростью доступа, расположенная на одном кристалле с CPU. Логически находится между CPU и RAM.
Является *статической* (SRAM) памятью, как и регистры.

Шина между CPU и кешем значительно шире, чем между CPU и RAM

(соединить находящиеся далеко друг от друга компоненты, такие как CPU и RAM, сложнее, тогда так соединения на одном кристалле даются почти бесплатно).

Программно кеш-память недоступна.

На практике кеш принимает >90% обращений к памяти, что делает низкую скорость работы RAM некритичной. Однако, если промахи случаются часто, кеш замедляет работу.

3 Реализация

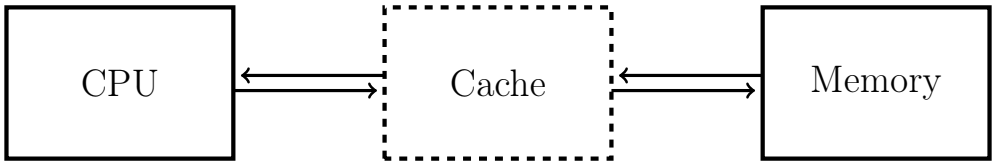
3.1 Чтение

Рассмотрим два варианта реализации.

В обоих из них CPU направляет запрос чтения из памяти по адресу; кеш проверяет, есть ли в нем такой адрес. Если есть, кеш отвечает на запрос вместо RAM.

Перед записью в кеш, если в нем нет свободного места, из него, к примеру, можно удалить случайные данные (но можно освобождать память и более интеллектуально).

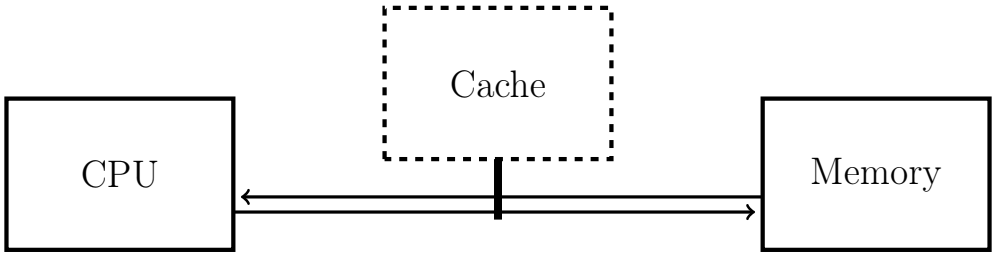
3.1.1 Look-through



В случае непопадания кеш отправляет запрос к RAM, загружает в себя данные, и затем отвечает на запрос.

Такая реализация выигрывает по скорости, когда шина между CPU и кешем быстрая.

3.1.2 Look-aside



В случае непопадания запрос дойдет до RAM, которая ответит на запрос. Кеш «прослушает» ответ на запрос и сохранит в себя.

Реализуется проще; в случае промаха отвечает на запрос быстрее.

3.2 Запись

Снова рассмотрим два способа реализации:

3.2.1 Write-through

Каждый запрос записи принимается кешем и отправляется к RAM. Как следствие, тратятся 100-200 тактов на каждый запрос, и создается большая нагрузка на шину.

Это довольно тривиальный в реализации способ, в то же время наиболее надежный (например, более устойчив к радиации).

3.2.2 Write-back

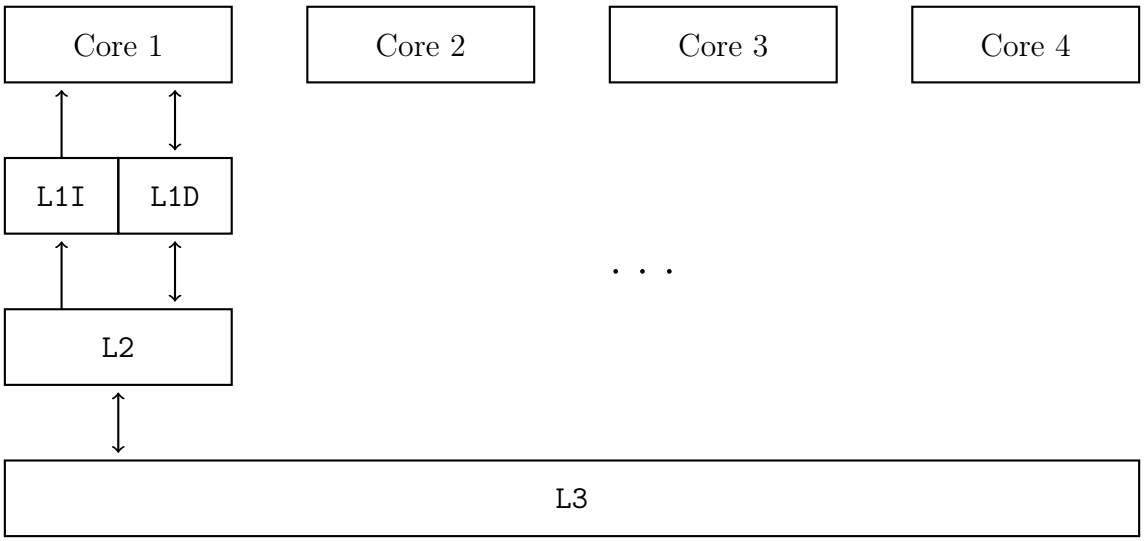
Запись происходит только в кеш. Обновление данных в RAM производится при вытеснении их из кеша.

Эта реализация более эффективна. Операции записи работают со скоростью записи в кеш, а также запись по соседним адресам требует меньше запросов к RAM. Однако, некоторые операции чтения работают внезапно долго (кеш вытесняет измененные данные и записывает в RAM).

4 Архитектура

4.1 Многоядерность

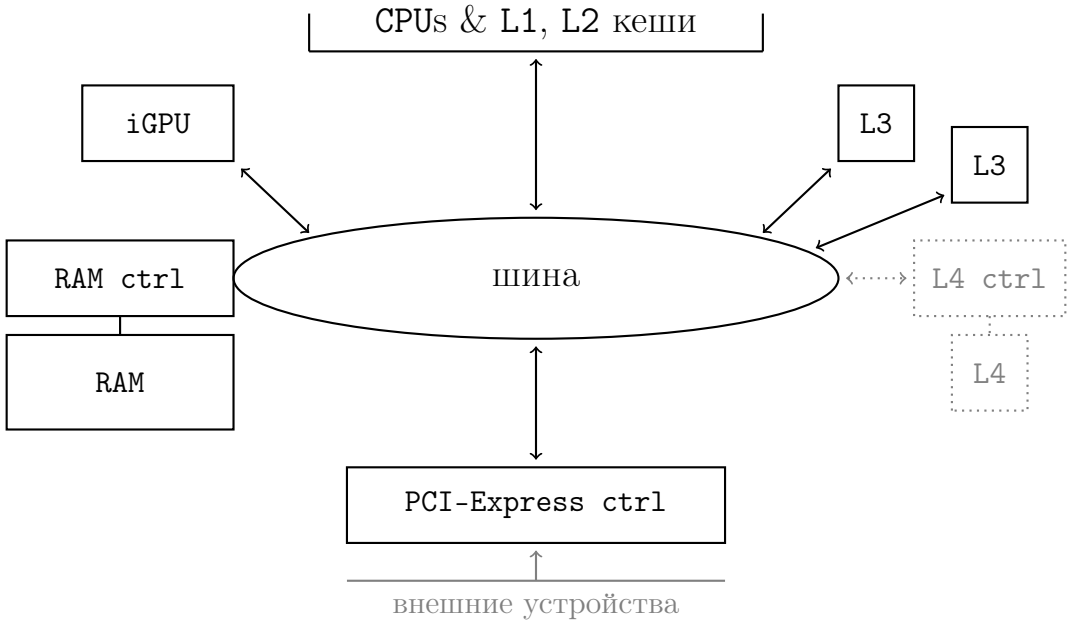
Рассмотрим упрощенную схему кешей в многоядерной системе:



В современных системах кеш первого уровня `L1` разделен на `instruction` и `data` (особенность Гарвардской архитектуры, которая попала в Фон-Неймановскую систему). При этом `instruction`-кеш поддерживает только чтение со стороны CPU. Он, в том числе, хранит распарсенные границы команд (см. билет 2.5).

4.2 Устройство

Более близкая к реальности схема **look-aside** кеша:



Система легко масштабируется – можно добавлять `L3` на шину, не меняя структуру.

С недавнего времени `RAM ctrl` стали встраивать в `CPU` в виде Integrated Memory Controller. Ранее (до 2008 г.) в процессорах Intel `RAM ctrl` был расположен на северном мосте.

Почти во всех современных системах есть `integrated GPU`. В системах с хорошими `iGPU`, например Intel Iris Pro, также присутствует `L4`-кеш, (с которым также может работать `CPU`) который находится на другом кристалле.
Кеш четвертого уровня `L4` является *динамической* памятью.

Все компоненты, кроме `L4` кеша (и, конечно, `RAM`), находятся на одном кристалле.

Рассмотрим типичные объемы и времена отклика:

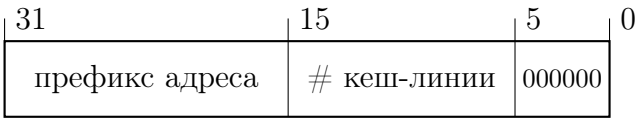
вид	объем	время отклика
`L1`	32 КБ	4 такта
`L2`	256 КБ	18 тактов
`L3`	8 МБ	50 тактов
`L4`	128 МБ	
`RAM`		100 тактов

5 Организация данных, ассоциативность

RAM разделяется на равные части – **кеш-страницы**. Их размер зависит от объема кеша и его огранизации. Каждая страница кеша разбивается на **кеш-линии**. Их размер зависит также от архитектуры процессора, обычно он составляет 64 байта.

Чтобы уметь понимать, из какой части RAM была взята данная кеш-линия, помимо самих данных нужно хранить их адрес в RAM (будем считать, что он 32-битный).

Рассмотрим адрес, по которому хранятся в RAM запрашиваемые данные:



Заметим, что поскольку байты данных внутри кеш-линии идут последовательно, у всех байтов внутри одной кеш-линии одинаковые первые $32 - 6 = 26$ бит адреса. Назовем адресом кеш-линии адрес первого ее байта. Тогда младшие 6 бит у адреса любой кеш-линии равны нулям (поскольку размер кеш-линий равен $2^6 = 64$), поэтому не участвуют в поиске и служат только для индексации байтов внутри кеш-линии.

Данные в кеше хранятся рядом с **тегом** адреса (это и есть адрес текущей кеш-линии, как на рисунке), а также служебными флагами, нужными для когерентности (см. билет № 1.5). Заметим, что при использовании 64-байтных кеш-линий можно не хранить младшие 6 разрядов тега адреса (они всегда равны 0). Каждая запись в кеше хранится в виде: `[tag] + [data] + [flags]`.

Ассоциативность – количество мест в кеше, где могут оказаться данные, хранящиеся в RAM по заданному адресу.

5.1 Полная ассоциативность

Любой кусок RAM может оказаться в любой части кеша. Такой кеш не использует кеш-страницы, только кеш-линии. RAM и кеш поделены на линии равных размеров.

Этот метод мог бы позволить достичь максимальной производительности. Его минус – сложность реализации. В частности, требуется определять, присутствуют ли запрошенные данные в кеше. Сравнение запрошенного адреса со всеми присутствующими в кеше займет слишком много времени; быстрая реализация требует использование сложных структур данных.
Поэтому такой подход применяется только для небольших кешей, <4 КБ.

5.2 Прямая ассоциативность (единичная)

Для каждого адреса данных в RAM однозначно определено их положение в кеше.

В теории работает наиболее быстро, поскольку контроллеру требуется проверить всего одну локацию в кеш-памяти. Однако на деле этот способ проигрывает в производительности – возникает много конфликтов (например, когда две постоянно используемые зоны памяти отображаются в одну позицию в кеше).

5.3 Применение в жизни

На практике часто используется ассоциативность-8.
При таком подходе кеш разбивается на 8 равных частей (пусть их размер равен $size$), тогда кеш-строка с номером idx может оказаться под номером (idx % $size$) в любой из этих частей.

Типичные ассоциативности кешей CPU:

вид кеша	ассоциативность
`L1`	4
`L2`	8
`L3`	16

Ассоциативности кеша одного уровня для `instruction` и `data` могут различаться.

FAQ

Простые вопросы

- Зачем нужен кеш?
- Когда кеш бывает вреден (замедляет работу)?
- Что такое ассоциативность?
- Сравните write-back и write-through.
- В чем плюсы и минусы инклюзивной и эксклюзивной архитектур?

Сложные вопросы

- Бывают ли алгоритмы, для которых безразлично, есть ли кеш?
  Ответ: да, алгоритм может часто запрашивать большие объемы данных из разных частей RAM, тогда кеш-попаданий не будет. Например, сжатие.
- Как CPU воспринимает кеш? Видит ли он его?
  Ответ: никак, с точки зрения CPU нет разницы, кто ответил на запрос – RAM или кеш.