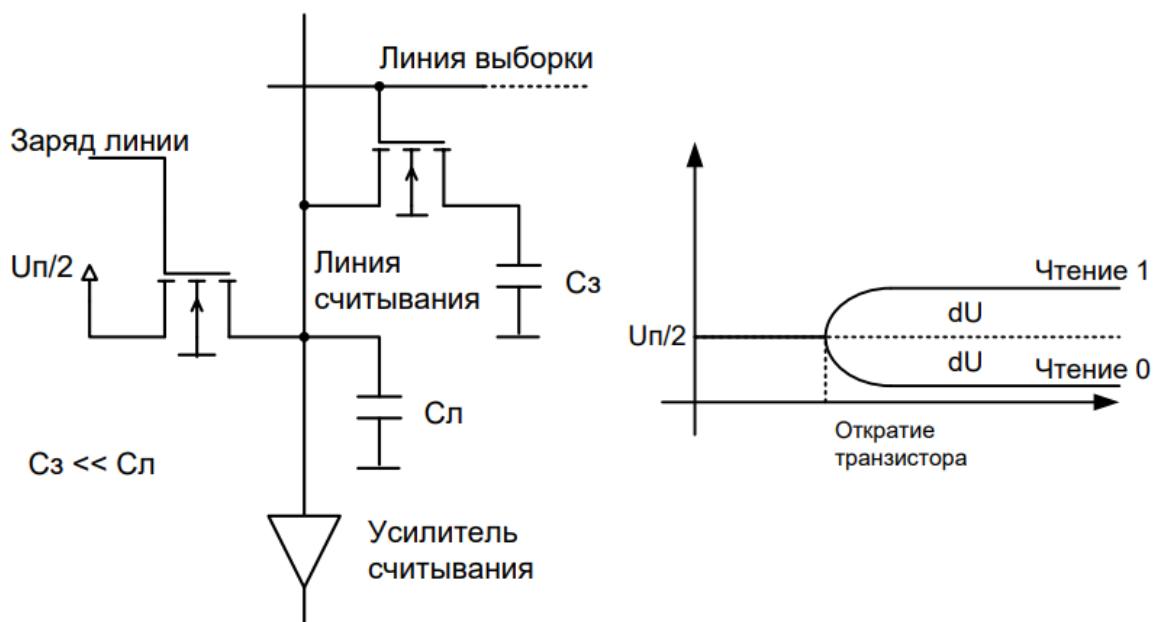


## 1. Done Процесс считывания в DRAM

Динамическая оперативная память (DRAM – Dynamic Random Access Memory) – энергозависимая полупроводниковая память с произвольным доступом. На данный момент – это основной тип оперативной памяти, используемый в современных персональных компьютерах и обеспечивающий наилучший показатель отношения цена-качество по сравнению с другими типами оперативной памяти. Однако, требования к быстродействию, энергопотреблению и надежности оперативной памяти постоянно растут, и динамическая оперативная память уже с трудом соответствует современным потребностям, так что в ближайшие годы стоит ожидать появления серийно выпускаемых конкурирующих типов оперативной памяти, таких как магниторезистивная оперативная память.

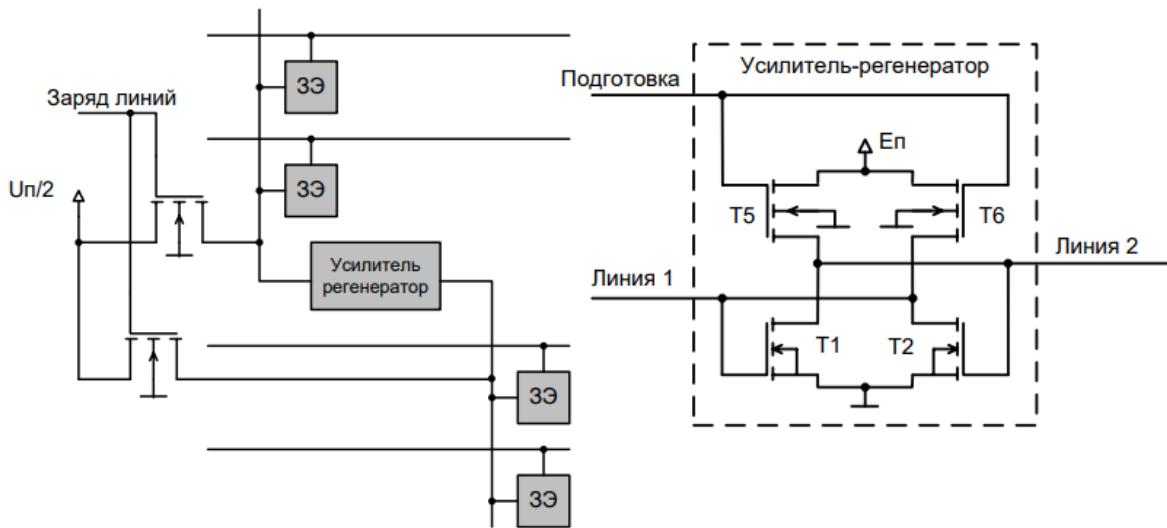
Динамическая оперативная память (DRAM – Dynamic Random Access Memory) – энергозависимая память с произвольным доступом, каждая ячейка которой состоит из одного конденсатора и нескольких транзисторов. Конденсатор хранит один бит данных, а транзисторы играют роль ключей, удерживающих заряд в конденсаторе и разрешающих доступ к конденсатору при чтении и записи данных.

## Процесс считывания в DRAM



## 2. Check!!! Принцип действия усилителя-регенератора

### Принцип действия усилителя-регенератора



*По сути, 1 в 1 с лекции*

Регенерация -- восстановление значения, потерянного в ходе чтения.

Усилитель-регенератор -- устройство, которое после усиления, должен восстанавливать информацию на конденсаторе, то есть оно и считывает, и восстанавливает одновременно.

Идеи, используемые в усилителе-регенераторе:

1. Запоминающий массив разделен на две примерно равные части. По середине запоминающего массива поставлены блоки, которые делят линии считывания записи на две примерно равные части. То есть туда как раз ставятся усилители.

Это нужно, так как мы имеем две линии и усилитель-регенератор обладает дифференциальным сигналом. То есть, оказывается, так как линия выборки только одна (которую мы включаем) и только один запоминающий элемент подключается к линии, значит только на одной линии происходит изменение на плюс или минус. То есть лишь на одной линии мы меняем потенциал, а на другой линии он остается константным. Получается, что легче усиливать дифференциальный сигнал, то есть есть

некий опорный, относительно него меняется вторая половина, то есть, разделив на две части, мы получаем дифференциальным усилитель и возможности по созданию дифференциальных усилителей намного шире.

2. Заряжать обе линии до питания пополам, то есть обе линии заряжаются до одного и того же эталонного значения. После этого мы открываем только одну из линий считывания, она соединяется только с одной ячейкой. (???) *1:00:00 -- тут чуть не уверен, что правильно понял его мысль.*

3. Самая интересная идея -- чтобы решить задачу усиления, используется структура, схожая с RS-триггером. Первоначально этот RS-триггер нужно привести в состояние равновесно-полуоткрытый, то есть обе линии (Линия 1 и Линия 2 на рисунке) должны иметь одинаковый потенциал, что противоречит идеи RS-триггера (у него два плеча должны быть в противоположных состояниях). И чтобы перейти в состояние неустойчивого равновесия, есть сигнал подготовки, который включается и некоторое время готовит этот RS-триггер к состоянию устойчивого равновесия. При подготовке транзисторы T5 и T6 (на рисунке) открываются чуть больше, чем положено в рабочем режиме, и ток потечет через них на линии, а значит линии будут заряжены от источника питания. А так как линии заряжены от источника питания, то значит мы их подзарядки. Когда подготовка прошла успешно, нужно отключить активный сигнал, то есть отключить источник активного сигнала от линий, так как при считывании они забывают своим сильным сигналом слабый сигнал запоминающей ячейки. Нужно снять сигнал подготовки почти весь, то есть надо существенно увеличить сопротивление T5 и T6 и тогда они будут выдавать очень слабый сигнал, но достаточный для поддержки работы RS-триггера для процесса усиления регенерации.

После этого мы открываем какую-нибудь из линий и потенциал на одной из линий (пусть будет Линия 1) потенциал изменяется в плюс или минус (пошел вверх или вниз), потенциал другой линии остается неизменным. Если, предположим, потенциал чуть пошел вниз, то есть уменьшается, тогда транзистор T1 начинает подзакрываться, тогда в точке над транзистором T1 соотношение сопротивления T1 и T5 таково, что сопротивление T1 становится больше, чем сопротивление T5 и потенциал

точки подтягивается к Е питания, то есть к высокому потенциалу. В ответ на это, потенциал точки, которая подтягивается, начинает расти, то есть происходит рост потенциала в Линии 2. Он сопровождается тем, что транзистор T2 начинает открываться, его сопротивление падает, оно становится меньше, чем сопротивление транзистора T6. Из-за этого потенциал точки, которая находится на выходе правого инвертора, поддерживается падающим ниже, то есть поддерживается противоположный процесс.

То есть начавшееся падение потенциала Линии 1 приводит к росту потенциала на Линии 2, что приводит к падению потенциала на Линии 1 - как два инвертора, которые помогают друг другу перейти в определенное устойчивое состояние.

Линии при этом начинают перезаряжаться в соответствии с разностью потенциалов, которую мы ввели благодаря подключению ячейки, то есть пошел процесс регенерации (то есть перезаряд линии, а при этом и конденсатора, который к этой линии подключен) и при этом и усиление (делаем большой логический перепад).

### 3. Done Контроллер динамической памяти

(КДП) обеспечивает мультиплексирование адреса системной шины, выработку управляющих сигналов CAS и RAS[3:0] (для селекции модулей ОЗУ), а также внутреннюю (по таймеру) или внешнюю (прозрачную) регенерацию.

Структурная схема контроллера (Рис. 7.) включает в себя :

- буферные схемы Буф.1,2,3 для подключения системной шины адреса и управления;
- счетчик адреса регенерации;
- мультиплексоры MUX1,2;
- схему управления с тактовым генератором, таймером и триггером регенерации, арбитром и логической схемой L для формирования управляющих сигналов.

КДП обеспечивает преобразование сигналов системной шины МПС в сигналы управления динамическим ОЗУ (см. Рис. 7.16), причем может работать в двух режимах : "16/64" (на память 16K или 64K соответственно). В режиме "16" две старшие линии адреса используются для формирования одного из сигналов RAS[0..3], в режиме "64" КДП может управлять двумя банками по 64K, причем сигнал RAS появляется на одном из выходов RAS0 или RAS1 - в зависимости от состояния линии RAS3VB0, которая в режиме "64" становится входом, определяющим номер банка ОЗУ.

Регенерация может осуществляться в двух режимах - внутреннем и внешнем. Если вход REFR остается неактивным 10..16 мкС, то формируется запрос на цикл регенерации от внутреннего таймера, причем в случае конфликта арбитр отдает предпочтение циклу памяти. Таким образом, и при регенерации по таймеру используются свободные такты шины. При внешней регенерации запрос должен быть сформирован на входе REFR.

Сигнал PCS - "Защищенный выбор кристалла" отличается от традиционного CS тем, что если PCS сформирован, то цикл ЗУ аннулировать нельзя.

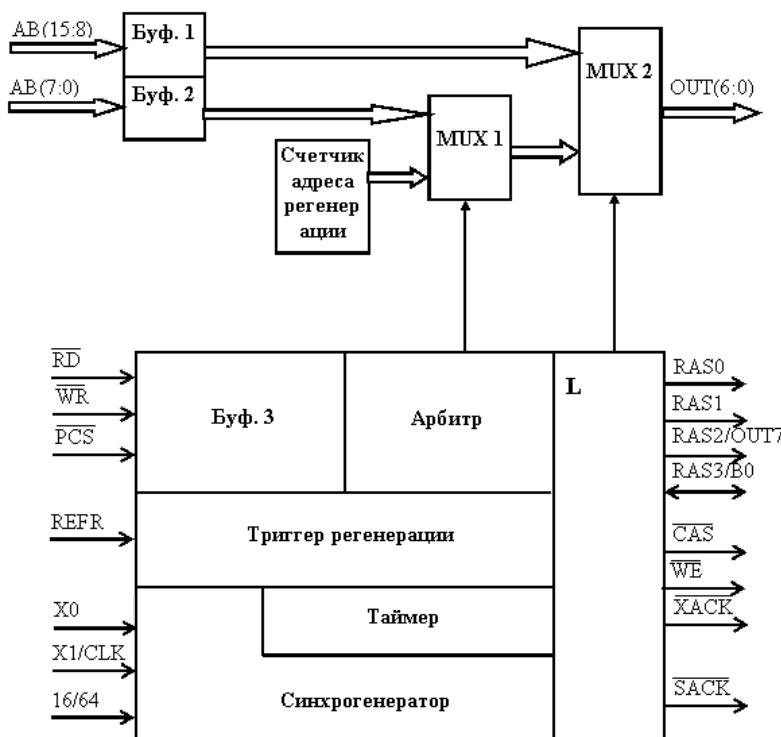
RD, WR - запросы на циклы чтения и записи соответственно.

X0, X1 - выводы для подключения кварцевого резонатора при работе с внутренним генератором. При работе с внешним генератором на вход X0 подается высокий потенциал, а на X1 - частота CLK внешнего генератора.

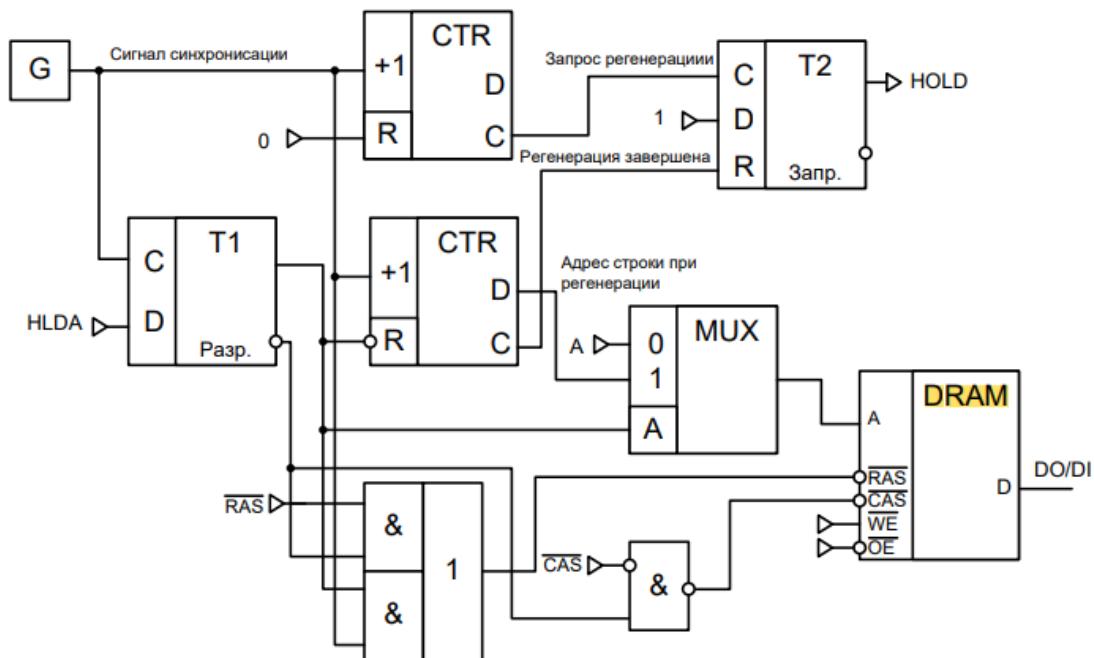
Выходной сигнал SACK\ вырабатывается КДП в начале цикла обращения к памяти. Если запрос от МП приходится на цикл регенерации, то SACK\ задерживается до начала цикла чтения/записи.

Выходной сигнал XACK\ ("Готовность данных") вырабатывается в конце цикла чтения/записи.

Сигналы SACK\ и XACK\ можно использовать для управления потенциалом на входе READY микропроцессора.

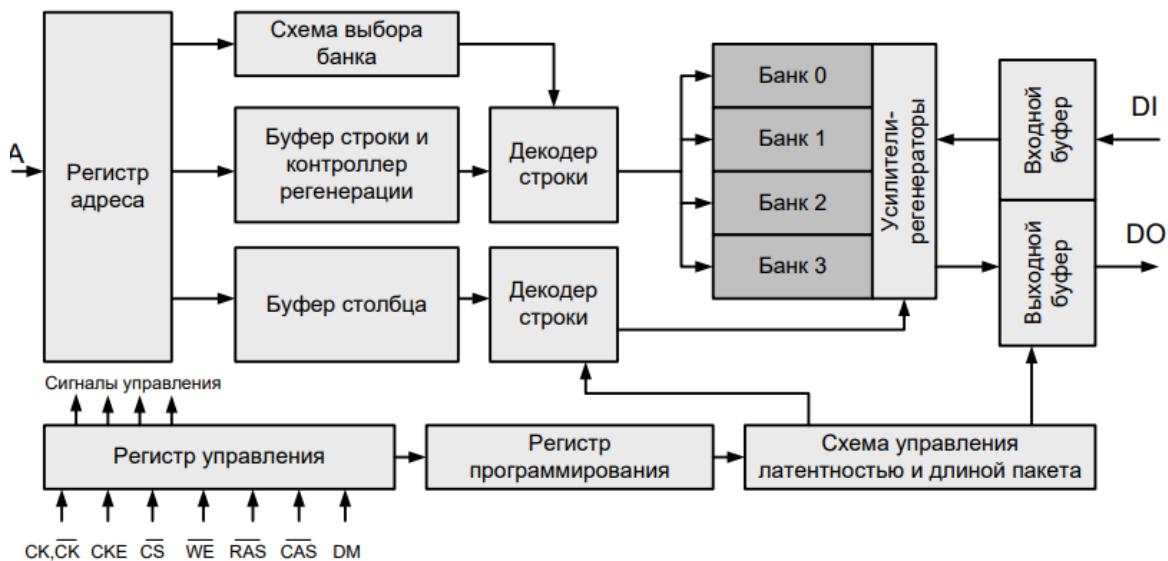


# Контроллер динамической памяти



## 4. Check Микросхема динамической памяти

### Микросхема динамической памяти



С ЛЕКЦИИ (почти слово в слово)

Лучше заниматься синхронными микросхемами, которые внутри себя имеют схему управления. Динамическая память не только обладает

схемой управления, но есть и программируется, то есть в неё можно заложить некоторые настройки. Для этого существуют специальные регистры (сейчас их 4, там биты расписаны), эти регистры имеют разрядность шины адреса, то есть туда записываются биты адреса. Это специальный режим программирования, он выполняется в начале работы микросхемы, когда включается питание и микросхема настраивается на нормальную работу. Выход микросхемы динамической памяти на рабочий режим занимает порядка 0.01 секунды. (Процедура описана в стандарте, подробнее [Лекция 20.05.2020. Динамические ЗУ.mp4](#) время: 1:13:05, если коротко, то в память нужно записать настройки, то есть каждое устройство может настроить микросхему по-своему, например, BIOS). Тут был резкий переход, где “ТАМ” не знаю :)

Там сначала был регистр OSDRAM1 (название неточное, как услышала), сейчас их много (четыре). Эти регистры влияют на схему управления, то есть определяет, какие пакеты выдаются, какие латентности внутри, с какой скоростью работает. (до этого места описывалась нижняя часть схемы, то есть все, что на одной линии с регистром управления).

Теперь про конвейерную структуру (верхняя часть схемы со входом A). Конвейерная структура -- регистр адреса, то есть регистр реальный, которые фиксирует адрес. (Оба!!! резкий переход на банки (на схеме обозначены номерами 0-3)). Внутри микросхемы есть несколько банков. Если у нас есть такие процессы, которые мы можем совместить по времени, то многобанковость -- большая польза, потому что мы можем провести одну операцию у банку 0, и он начнет выполнять эту операцию (precharge, refresh, все эти операции могут начаться), но мы не будем ждать конца этой операции, мы в этот момент обратимся в другому банку, потому к другому и т. д. То есть когда у нас много банков, мы можем несколько процессов держать “в голове”, то есть контроллер памяти знает про каждый банк, в каком он состоянии, выдает туда, соответственно, операции в зависимости от того, в каком он состоянии находится, то есть нельзя банку выдать операцию, не соответствующую его состоянию. И следит за всем, что происходит в памяти (хоть убей, не знаю, кто следит), то есть, если банк 0 выполнял precharge, то можно начать чтение, если нет, то чтение начать нельзя. Нельзя заставить банк выполнить операцию, которая не соответствует цепочке событий.

Вернемся к началу. Принимаем адреса, причем отдельно принимается адрес строки и адрес столбца. Для строки и столбца разные декодеры, суть в том, что передавать весь адрес смысла не имеет. Первое, что нужно сделать -- это выбрать строку. Выборка строки заключается в усилении и регенерации, то есть это длительный процесс. Поэтому выдавать в 2DM-массив (вроде это про массив банков) весь адрес смысла не имеет, можно, но его нужно буферизировать. Поэтому сначала принимает адрес строки и адрес банка, эти адреса фиксируются в регистре и формируют номер строки и номер банка, из которого мы выбираем строку, это происходит долго, мы усиливаем, регенерируем эту строку, в других банках происходят другие команды (про время когда в каком банке можно начинать операцию описано в стандарте подробнее в той же ссылке 1:18:40).

Выбрали строку, усилили, регенерировали, и только после этого нам нужен адрес столбца (одна строка 8 килобит). По номеру столбца из строки выбирается та часть, которая нам нужна. Номер столбца поступает с дешифратора (мультиплексора). Выбрали биты, они уже усилены и готовы к чтению. Прочесть эти биты из усилителя-регенератора (он же буфер, он же регистр), который хранит усиленные значения в непосредственной доступности, его можно выбрать и отправить в выходной буфер, если мы обращаемся к уже усиленным и регенерированным данным, то это происходит быстро. Если по новому адресу выбирать, то долго (там нужно закрыть предыдущую строку, потом перезарядить линии считывания записи, и снова открыть, усилить, регенерировать). Кучность адресов -- это, когда чаще обращаемся, в ту же самую строку, где данные уже усилены и регенерированы, и нам не приходится тратить время на смену строки. Программист этого не видит, этим управляет контроллер памяти (про него не в этом вопросе, все, что дальше это либо повторение, либо не в этот вопрос). Система памяти не может быть однородная, разделена на банки, на линейке. Чередование банков -- это плюс, но нагрузка на контроллер. Программист только догадывается об управлении в микросхеме.

## КОНЕЦ ЛЕКЦИИ

Ничего особо лучше и понятнее, не нашла. Больше про сигналы везде написано. Вот здесь кратко: [Микросхемы памяти](#). В книжке похоже, но более подробно (страницы 235 (с заголовка Основная память)-243(до последовательного режима). Книжка (надеюсь, ссылка работает, если что в

группе можно найти:  
[https://vk.com/doc166014887\\_618873554?hash=0e08d11a3f863d94e2&dl=4ec8a7ebdd655c43ce](https://vk.com/doc166014887_618873554?hash=0e08d11a3f863d94e2&dl=4ec8a7ebdd655c43ce)

В принципе, пересказать конспект лекции и норм.

## 5. Done Функциональные возможности SDRAM памяти

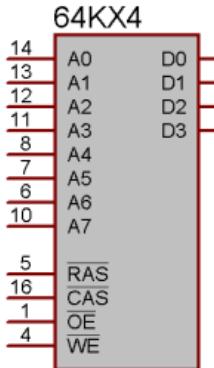
Синхронная оперативная **память (SDRAM)** — это первая технология оперативной **памяти** со случайным доступом (DRAM) разработанная для синхронизации работы **памяти** с тактами работы центрального процессора с внешней шиной данных.

Особенностью технологии SDRAM (Synchronous DRAM) является синхронная работа микросхем памяти и процессора. Тактовый генератор, задающий скорость работы микропроцессором, также управляет работой SDRAM. При этом уменьшаются временные задержки в процессе циклов ожидания, и ускоряется поиск данных. Эта синхронизация позволяет контроллеру памяти точно знать время готовности данных. Таким образом, скорость доступа увеличивается благодаря тому, что данные доступны во время каждого такта таймера.

Технология SDRAM позволяет использовать множественные банки памяти, функционирующие одновременно, дополнительно к адресации целыми блоками.

Микросхемы SDRAM имеют программируемые параметры и свои наборы команд.

Длина пакетного цикла чтения-записи может программироваться (1, 2, 4, 8, 256 элементов). Цикл может быть прерван специальной командой без утери данных. Конвейерная организация позволяет инициировать следующий цикл чтения до окончания предыдущего.



### Функциональные возможности SDRAM памяти:

- Многобанковая организация.
- Командный режим работы.
- Команды пакетного чтения/записи.
- Использование чередования банков при последовательном увеличении адресов.
- Команды пакетного чтения/записи с авто-подзарядом.
- Возможность останова чтения/записи по режиму регенерации.
- Возможность останова чтения/записи по новому запросу чтения/записи.
- Управление маскированием шины данных по сигналу DQM.
- Минимальное время (1 CLK) между последовательными командами.
- Команда PrechargeAll.
- CAS латентность 2 и 3 CLK.
- Длина пакета 1,2 и 4 слова.
- Команда само-регенерации.
- Режим энергосбережения.

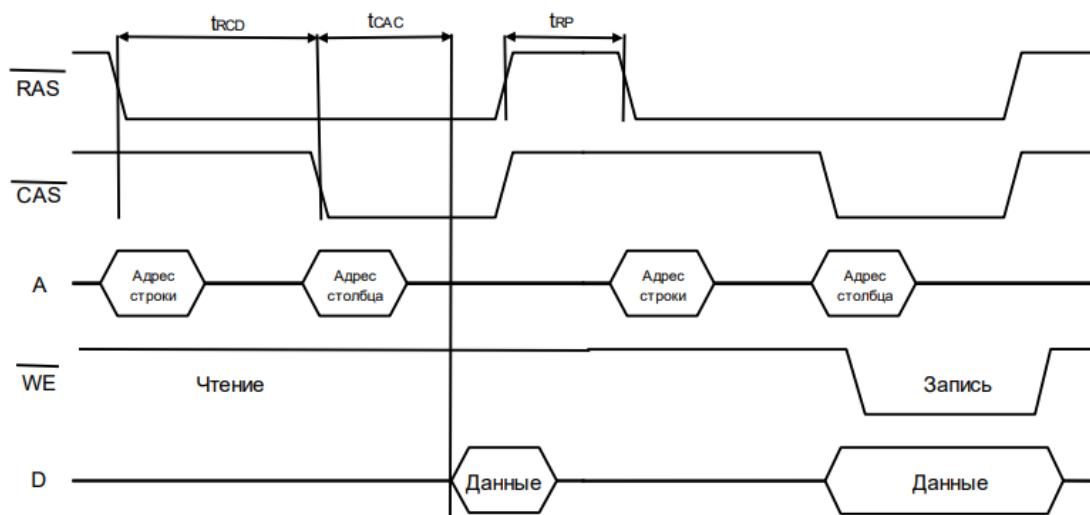
еще вариант...

SDRAM - это классификация DRAM, которая работает синхронно с тактовой частотой процессора. В начале ожидает тактового сигнала, прежде чем ответить на ввод данных (например, пользовательский интерфейс). DRAM считается асинхронным, так как немедленно реагирует на ввод данных. Но преимущество синхронной работы состоит в том, что ЦП может параллельно обрабатывать перекрывающиеся инструкции, также известные как «конвейерная обработка» - возможность получать (читать) новую инструкцию до того, как предыдущая инструкция полностью разрешена (запись).

Конвейерная обработка не влияет на время, необходимое для обработки инструкций, она позволяет одновременно выполнять больше инструкций. Обработка одной инструкции чтения и одной записи за такт приводит к более высокой общей скорости передачи/производительности ЦП. SDRAM поддерживает конвейеризацию благодаря делению памяти на отдельные участки, что и обусловило ее широкое предпочтение по сравнению с базовым DRAM.

## 6. Done Диаграмма работы DRAM памяти

### Диаграмма работы DRAM памяти



$t_{RCD}$  – RAS to CAS Delay.

$t_{RP}$  – RAS Precharge.

$t_{CAC}$  – CAS Delay.

Первый тип динамической памяти, который был изобретен

Сначала должен быть передан адрес. Он передается в виде двух частей: адрес строки и адрес столбца. Они сопровождаются сбросом RAS и CAS.

Сигнал  $\neg$ RAS определяет, что память находится в рабочем состоянии (сигнал chip-select).

Между передачей адресов есть некоторая задержка - значение  $t_{RCD}$ . За время  $t_{RCD}$  происходит усиление и регенерация информации целой строки. После этого может быть выдан адрес столбца и выбраны некоторые данные из строки.

$t_{CAC}$  - задержка после сброса CAS-а до выдачи данных. После данные появляются на шине и могут находиться там какое-то время, пока не будет сброшен сигнал chip-select (RAS). Это говорит о конце транзакции. Эти два сигнала приводят к тому, что и данные снимаются (закрываются выходные буферы) и, самое главное, сигнал  $\neg$ RAS

используется в данной микросхеме для перезаряда линии чтения-записи (время  $t_{RP}$ ).

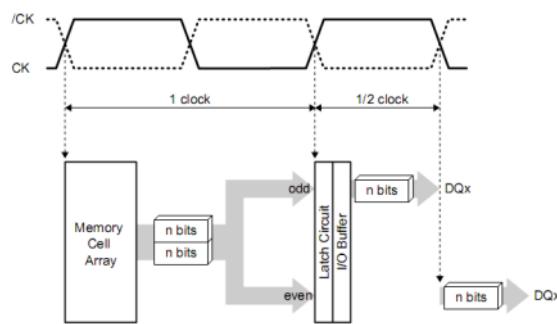
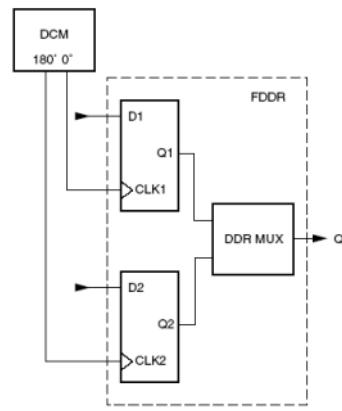
Между двумя циклами работы памяти (будь то чтение или запись) обязательно выдержать какое-то время и перезарядить линии, после этого например начиналась запись, снова сбрасывался RAS, при этом выдавался адрес строки, потом адрес столбца и поступала информация о том, что сейчас будет происходить запись (выемка с надписью "Запись"), то есть операция относится к текущей транзакции, передается вместе с адресом столбца, микросхема точно также выбирает данные, но дает возможность входные данные передать в микросхему памяти. Данные фиксируются в момент переключения CAS в 0, то есть данные буферизуемые. Для того, чтобы можно было сформировать устойчивые сигналы управления внутри, эти данные буферизируются.

## 7. Done Способы повышения производительности RAM

### Способы повышения производительности RAM

- Синхронизация.
- Конвейеризация.
- Пакетный режим обмена.
- Ускорение реверса шины.
- Чередование банков при обращении по последовательным адресам.
- Удвоение скорости.

Регистр DDR



DDR - передача по фронту и по спаду, нужно иметь на приемной стороне 2 триггера.

Один работает на синхросигнал 0 - прямой, 0 градусов сдвиг по фазе

Второй на 180 - инвертированный

Надо 2 триггера, которые принимают почти одну и ту же инфу.

Информация по фронту будет буферизованная в D1 по спаду в D2, дальше с помощью мультиплексора получаем сигнал, который мы можем выбирать, то же самое но в обратную сторону при передаче - должны с помощью мультиплексора передать сигналы, которые формируются с двух триггеров. Один формирует положительный полупериод, второй отрицательный в отношении данных. Это DDR линия. Это всегда требует повышенные требования к выходным/входным каскадам, но позволяет организовать шину в 2 раза большей пропускной способности.

Чтобы это удалось сделать, внутри памяти нужно читать и запоминающего массива сразу в 2 раза больше инфы  $n$  бит +  $n$  бит. Есть чёт/нечёт и мы должны эти биты передавать каждый в свой триггер. Эти триггеры совмещаются на шине(каждый передаёт свой полупериод), каждый передаёт свой полупериод(пол такта передаётся odd, пол такта even). Выходной каскад на картинке.

Это удвоение скорости - технология, позволяющая ускорить интерфейс памяти. Можем передавать на удвоенной частоте(при передаче по DDR чередуются данным на шине данных)

Синхронизация - позволяет организовать схему управления внутри микросхемы.

Синхронизируем схемы, они сами организуют задержки. Мы не сталкиваемся с

пенальти, которые возникают при стыковке синхр. ассинхрон. При минимальном расхождении времён необходимых для выдачи задержки и тактовой частоты - попадание на простой 1 такта. Если нужно подождать треть такта, ждём целый. Стыкуются часто плохо(синхр. ассинхр), если микросхема будет сама решать проблему синхронизации и преобр. синхр в ассинхр. интерфейс, мы избавим разработчика от трудностей. Синхронизация позволяет ускорить работу и упростить интерфейс.

Конвейеризация = память позволяет обработать сразу несколько транзакций, одна в стадии выдаче данных, вторая в стадии обращения к массиву, тем более многобанковая организация способствует конвейеризации, т.к один банк выполняет одну фазу работы с массивом, другой с другой.

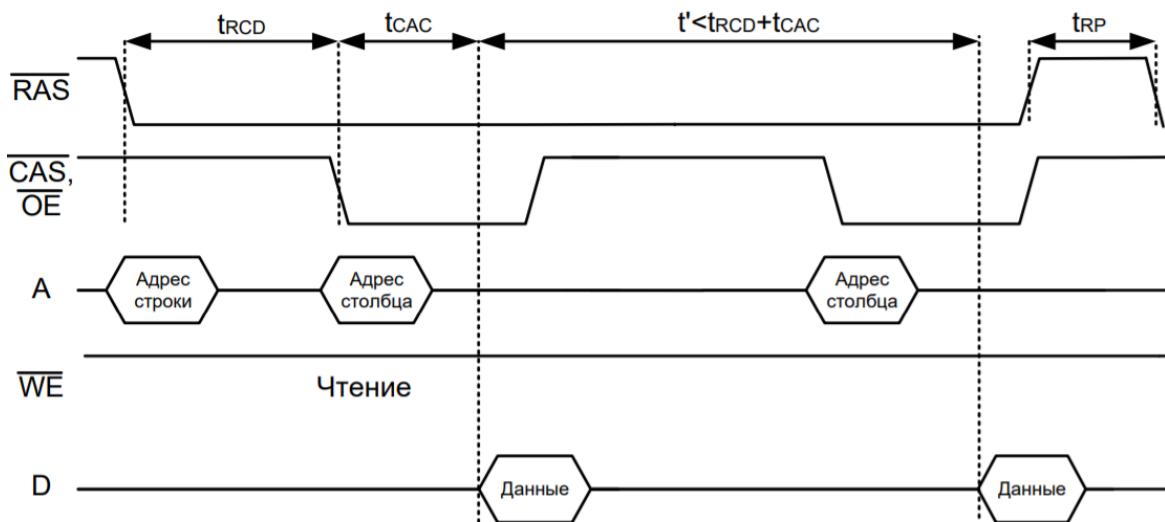
Пакетный режим - любой запрос приводит к передаче или чтению пакеты. Это упрощает управление, т.е тактов управления в расчёте на такты данных - соотношение становится удобным. Одно управление - много данных, большой пакет читаем. Хорошо укладываются диаграммы в линии, у нас нет никаких пауз из-за невозможности обрабатывать данные с той же скоростью, с которой выдаются данные.

Ускорение реверс шины - не всегда заметная особенность. Это когда сигналы (DQ, DQS), чтение выдаёт данные сама микросхемы, при записи мы выдаем микросхеме данные. Нужно переключиться. На шине должна произойти смены отправителя. Должна быть пауза между моментами владения - реверсная шина, это сокращение паузы. В электронике - схема отключается мгновенно, затухает сигнал, сокращается ток, но он не может сойти на ноль моментально, то есть ток снижается. С другой стороны, тот, кто становится драйвером шины, повышает ток напряжения. И если это сделать одновременно, возникнет некоторое короткое замыкание. Возникают сильные токи, возможен пробой, возможен перезаряд линий, препятствующих потом прохождению высокочастотных сигналов. Нужно держать линии в состоянии равноразряженности - срединном состоянии полузаряда. Она будет одновременно как разряжена, так и заряжена. Нам нужно такое состояние, а сквозные токи - плохо, т.к будет либо с высокими токами, либо разряжены. Важно в PCI шинах - делается с помощью кодирования информации, которое поддерживает заряд и потенциал шины в равновесном состоянии, не давая смещение к 0 и к 1.

Чередование банков - ping-pong - возможность совместить некот. действия над банками по времени.

## 8. Done Диаграмма работы FPM DRAM памяти

### Диаграмма работы FPM DRAM памяти



Fast Page Mode (FPM) - память, которая использовалась как RAM.

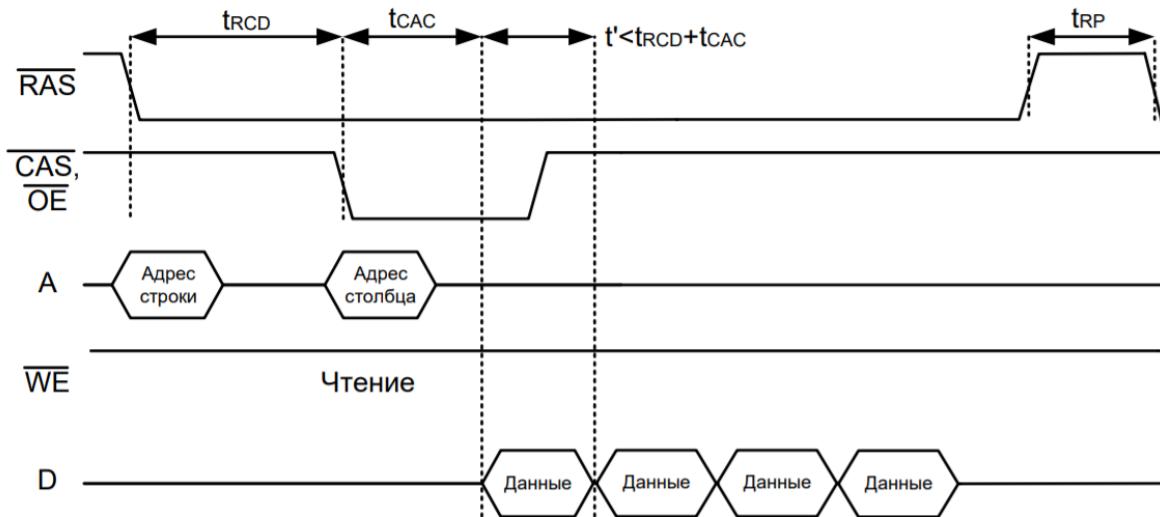
Идея быстрого страничного доступа состоит в том, что если в одном массиве открыта строка и эта строка усиленно регенерированна, то обращение к ней не требует снова закрыть ее и усиленно регенерировать. Хотя в DRAM-диаграмме это требовалось. Динамическая память позволяет повторно обратиться быстрее к уже открытой строке.

На диаграмме  $t_{RCD}$  - время открытия строки и усиленной регенерации. Далее за время  $t_{CAC}$  выдаем сигнал с адресом столбца, но не сбрасываем после этого  $\sim RAS$ , оставляем ее в состоянии 0, говоря о том, что следующая транзакция будет производиться с этой же строкой; через какое-то время когда мы обнаружили, что действительно к этой строке мы хотим обратиться, но уже по другому адресу столбца (может быть и тот же), снова сбрасываем сигнал  $\sim CAS$ , и этот сигнал позволяет выбрать другой столбец из той же самой строки.

Получается, что мы потратили на последовательную выборку данных некоторое время, меньшее чем  $t_{RCD}+t_{CAC}$  (время потраченной на открытие строки). Примерно в 3-5 раз быстрее. Весь цикл при последовательном обращении получается в несколько раз быстрее (~2-4, зависит от настроек таймингов). Итак, память при работе в рамках открытой строки отвечает на запросы примерно в 2 раза быстрее, чем когда нам приходится постоянно ее открывать при каждом запросе. Это важный критерий, говорящий о том, что память предназначена для работы по последовательным адресам, и хуже работает, если адреса случайные.

## 9. Done Диаграмма работы BEDO DRAM памяти

### Диаграмма работы BEDO DRAM памяти

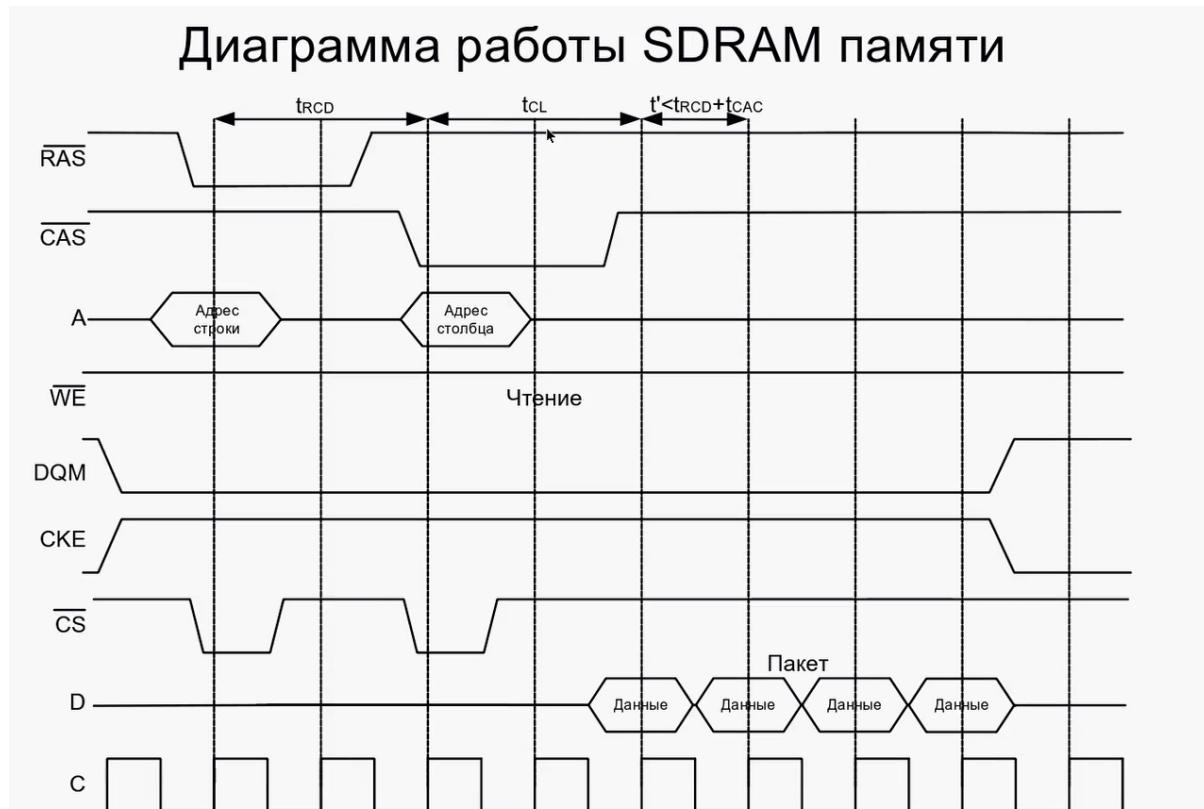


Burst Extended Data Out (BEDO) - слово burst (пакет) говорит о том, что передаются слова для пакета.

Пакетная модификация FPM DRAM. На диаграмме показано одно обращение, возможно второе обращение (данные появляются на шине без пауз). Смысл в том, что любое обращение с адресом строки и адресом столбца вызывает передачу целого пакета, т.е. память запрограммированна под то, чтобы внутри перебрать 4 адреса, т.е. выдать более широкую группу столбцов; при задаче начального столбца (на диаграмме по сигналу A), память сама перебирает младшие биты адреса столбца, увеличивая их или уменьшая (перебор цикличен, например, при передаче двух битов 10 дальше будут вычислены 11, 00, 01; также бывает реверсивным, позволяя читать адреса от старшего к младшему).

Обычно от младшего к старшему (не значит остроконечники, тупоконечники???), просто чтение памяти - настройки, которые обеспечивает BIOS, они ни на что не влияют, это не связано с расположением байт в слове, это особенности контроллера памяти. Особого смысла в этом нет, переставлять или менять порядок обхода, память работает за одно и то же время, просто меняется порядок выдачи на шину.

## 10. Диаграмма работы SDRAM памяти



Основные сигналы интерфейса SDRAM схожи с сигналами интерфейса асинхронной памяти. Главные их отличия сводятся к появлению ряда новых сигналов:

1. У памяти SDRAM присутствует синхросигнал **CLK**, по переднему фронту которого производятся все переключения в микросхеме. Кроме этого сигнала имеется также сигнал **CKE** (Clock Enable), разрешающий работу микросхемы при высоком уровне, а при низком – переводящий ее в один из режимов энергосбережения.
2. В интерфейсе SDRAM имеются сигналы выбора банка **BS0** и **BS1** (Bank Select), позволяющие адресовать конкретные обращения в один из четырех имеющихся в микросхемах SDRAM банков (массивов элементов) памяти.
3. Присутствуют сигналы **DQM** маски линий данных, позволяющие блокировать запись данных в цикле записи или переключать шину данных в состояние высокого выходного сопротивления (*z*-состояние) при чтении.
4. Имеет место специфическое использование одной из адресных линий ( $A_{10}$ ) в момент подачи сигнала **CAS#**. Значение сигнала на этой линии задает способ подзаряда строки банка.

Кроме того, SDRAM память сразу ориентирована на выполнение пакетных передач данных, причем длина пакета задается при инициализации микросхем памяти.

tRCD – задержка между сигналами RAS# и CAS# (RAS to CAS Delay): минимально допустимое время (в тактах синхроимпульсов) между подачей сигналов RAS# и CAS#;

tCL – задержка данных по отношению к сигналу CAS# (CAS Latency): минимально время (в тактах синхроимпульсов) между подачей сигнала CAS# и появлением считанных данных на шине DQ;

tCAC – время доступа (Clock Access): временной интервал (измеряется в наносекундах) от подачи сигнала синхронизации (того такта, в котором должны появиться данные чтения – второй или третий по отношению к сигналу CAS# в зависимости от значения CAS Latency) до появления данных на шине DQ;

### **Немного о самом SDRAM**

Преимущество синхронной работы состоит в том, что ЦП может параллельно обрабатывать перекрывающиеся инструкции, также известные как «конвейерная обработка» – возможность получать (читать) новую инструкцию до того, как предыдущая инструкция полностью разрешена (запись). Конвейерная обработка не влияет на время, необходимое для обработки инструкций, она позволяет одновременно выполнять больше инструкций.

Обработка одной инструкции чтения и одной записи за такт приводит к более высокой общей скорости передачи/производительности ЦП.

SDRAM поддерживает конвейеризацию благодаря делению памяти на отдельные участки, что и обусловило ее широкое предпочтение по сравнению с базовым DRAM.

### **Основные принципы на которых построена SDRAM**

- для хранения одного бита используется информация о наличии или отсутствии заряда конденсатора;
- память организована в виде матриц из емкостей и управляющей логики: со столбцами и строками;
- во время операции чтения заряд ячейки (конденсатора) расходуется, после чтения ее приходится подзаряжать;
- во время бездействия величина сохраненного заряда также уменьшается (пусть и медленнее) — ячейки памяти требуют периодической подзарядки (т.н. регенерации).

### **Дальше интернетные данные, до этого диаграммы с лекции Попова.**

В SDR SDRAM-памяти обеспечивается синхронизация всех входных и выходных сигналов с положительными фронтами импульсов тактового генератора. Весь массив памяти SDRAM-модуля разделен на два независимых банка. Такое решение позволяет совмещать выборку данных из одного банка с установкой адреса в другом банке, то есть одновременно иметь две открытые страницы. Доступ к этим страницам чередуется (bank interleaving), и соответственно устраняются задержки, что обеспечивает создание непрерывного потока данных.

В SDRAM-памяти организована пакетная обработка данных, что позволяет производить обращение по новому адресу столбца ячейки памяти на каждом тактовом

цикле. Смысл пакетной обработки заключается в том, при активированной строке задание адреса одного столбца позволяет получить доступ сразу к последовательности нескольких столбцов (пакету столбцов) без дополнительного указания их адресов. В микросхеме SDRAM имеется счетчик для наращивания адресов столбцов ячеек памяти, чтобы обеспечить к ним быстрый доступ. Количество адресуемых таким образом столбцов называется длиной пакета (Burst Length, BL).

В SDRAM-памяти ядро и буферы обмена работают в синхронном режиме на одной и той же частоте (100 или 133 МГц). Передача каждого бита из буфера происходит с каждым тактом работы ядра памяти.

Временная диаграмма работы памяти SDR SDRAM при длине пакета  $BL = 4$  и таймингах  $tRCD = 2$  и  $tCL = 2$  показана на рис. 8.

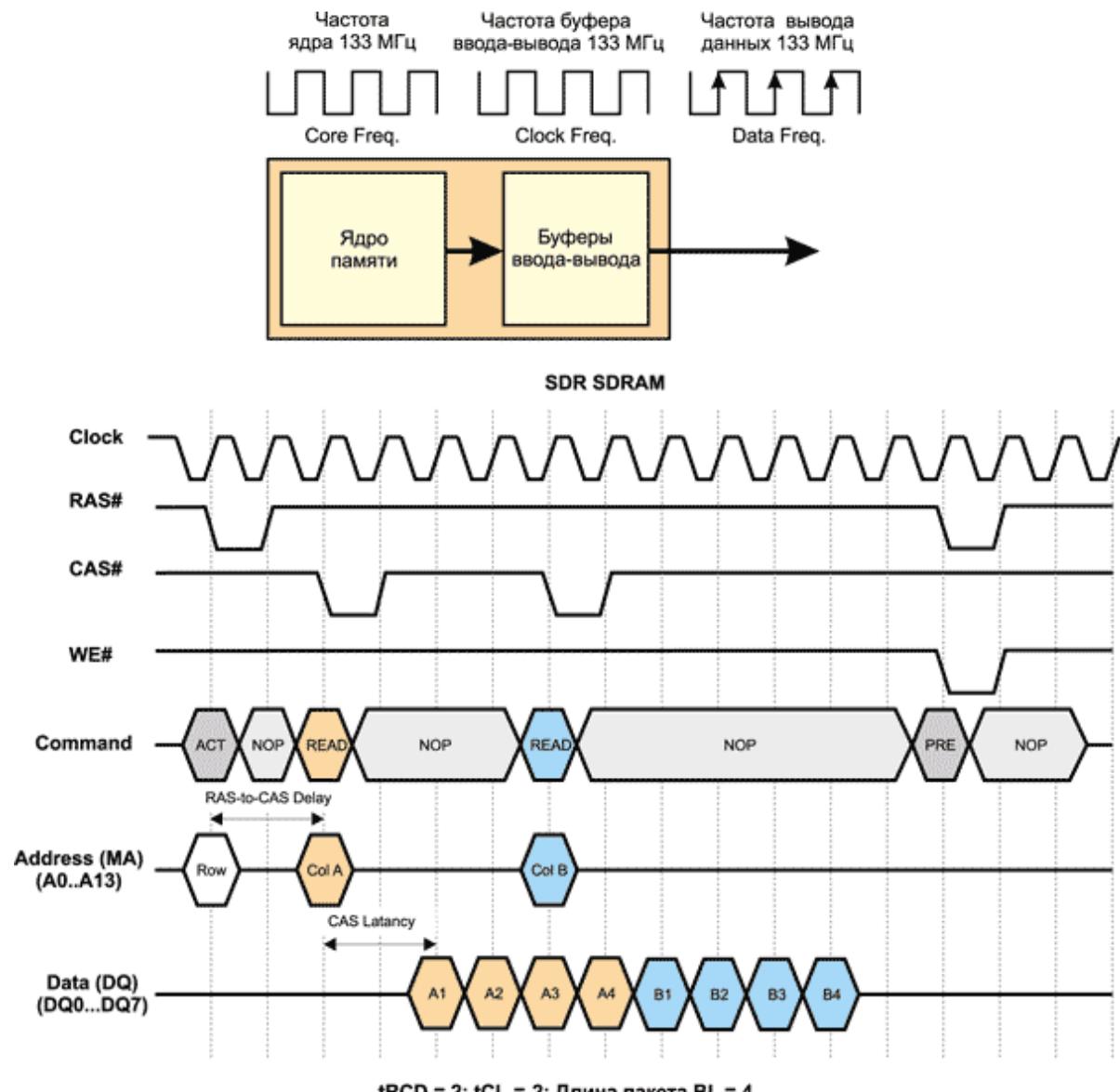
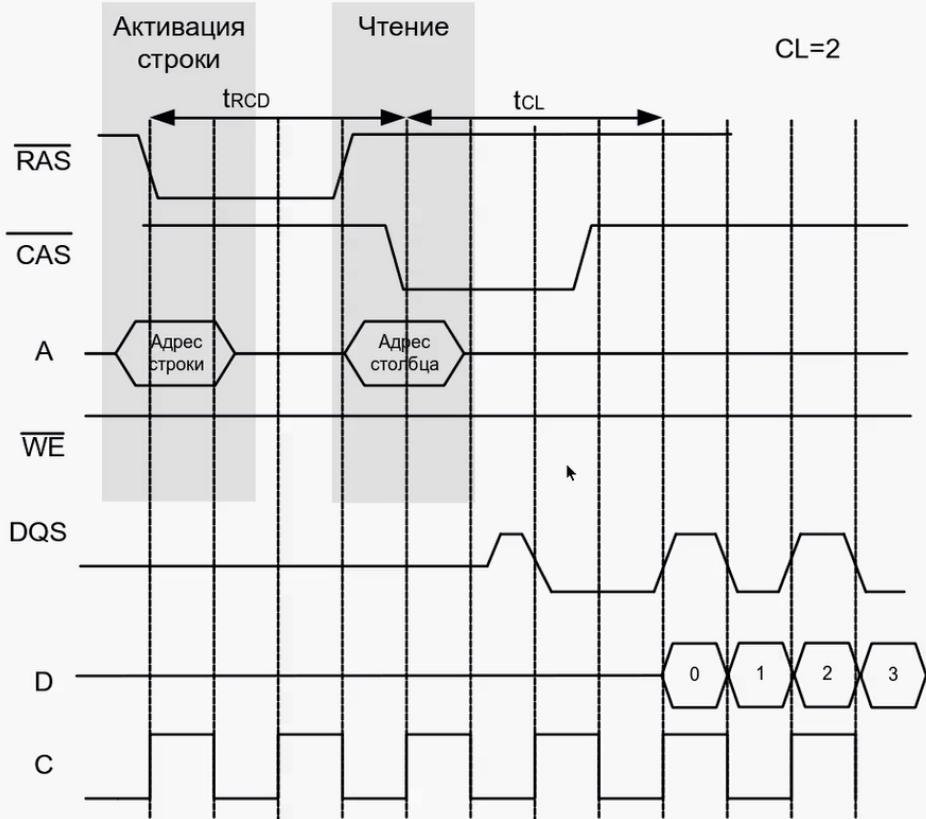


Рис. 8. Упрощенная временная диаграмма работы SDR SDRAM-памяти

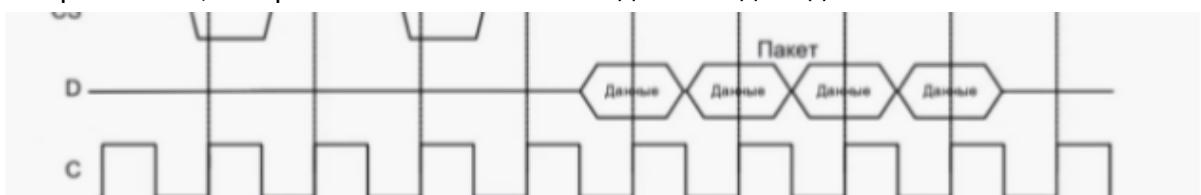
## 11. Диаграмма работы DDR SDRAM памяти

### Диаграмма работы DDR SDRAM памяти



Подход DDR состоит в том, что данные передаются и по фронту и по спаду. Изменение на каждой линии данных происходит с той же частотой, что и синхросигнал. (Это можно наблюдать внизу диаграммы)

Если посмотреть на диаграмму SDRAM(прошлый вопрос), то можно увидеть, что частота изменения данных в два раза меньше частоты изменения синхросигнала, который может меняться за один такт дважды.



Именно порядок DDR, как было сказано выше исправляет данную ситуацию. Его сигналы С и D имеют одинаковые частотные свойства. Но чтобы принять такой сигнал нужно иметь отдельные схемы, которые работают по фронтам и отдельные схемы, которые работают по спадам.

Управление остается SDRAM (синхронизируется фронтами)

## Дальше интернетные данные, до этого диаграммы с лекции Попова.

Для обозначения микросхем «обычных» синхронных динамических ОЗУ используется аббревиатура SDRAM (Synchronous DRAM — синхронная DRAM). В настоящее время наиболее распространенным типом динамической памяти персональных ВМ являются микросхемы, реализующие технологию DDR SDRAM (Double Data Rate SDRAM — SDRAM с удвоенной скоростью передачи данных). Основная особенность технологии в том, что передача данных синхронизируется как передним, так и задним фронтом тактового импульса. Таким образом, при сохранении тактовой частоты шины памяти запрошенные данные передаются вдвое быстрее — по два пакета за один тактовый период (память работает в пакетном режиме), т. е. при передачи блоков(пакетов) данных за один машинный такт передается двойная порция данных. Как следствие, получается двукратное увеличение пропускной способности шины памяти.

Следует отметить, что увеличение производительности происходит за счет оптимизации процесса адресации и чтения/записи ячеек памяти, а вот тактовая частота работы запоминающей матрицы практически не меняется. Поэтому общая производительность компьютера не увеличивается в 2 и 4 раза, а всего на десятки процентов.

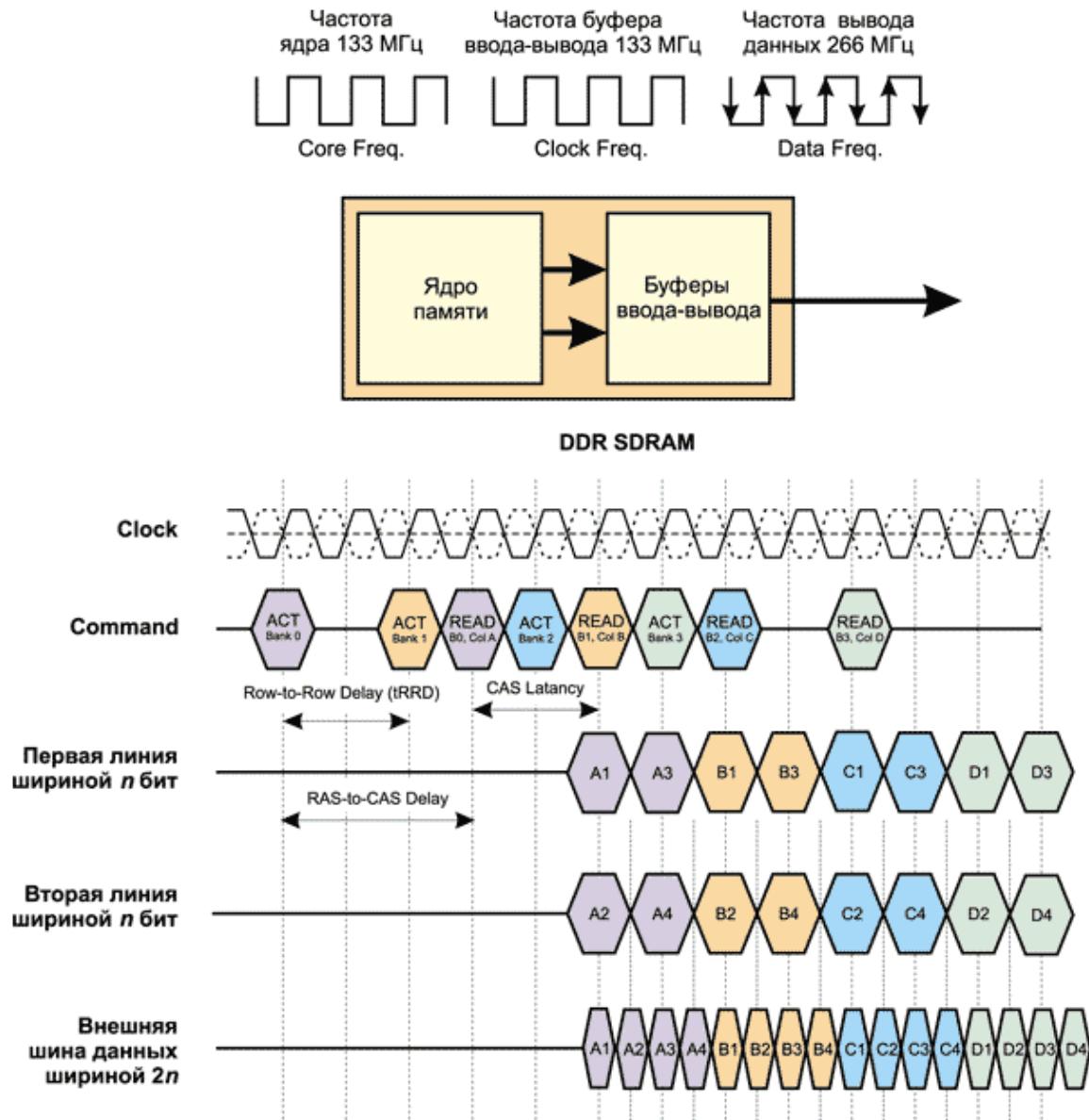


Рис. 11. Упрощенная временная диаграмма работы DDR SDRAM-памяти

Рассмотрим упрощенную схему работы DDR-памяти на примере операции чтения (рис. 11). Пусть имеется четыре банка памяти (Bank0...Bank3), длина пакета (Burst Length) равна 4, tCAS = 2 и tRCD = 3. Первоначально необходимо активировать каждый из четырех банков и получить доступ к строке в этом банке. Задержка между активацией двух банков определяется как tRRD (Row-to Row Delay) и обычно составляет 2 такта. Таким образом, через каждые два такта активизируется новый банк, а через каждые три такта после активации банка следует команда чтения данных из него.

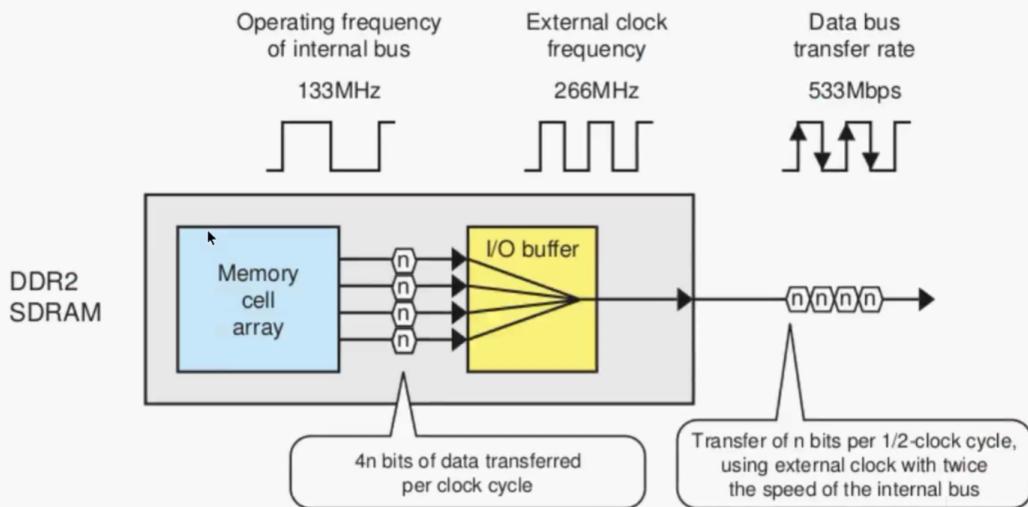
Поскольку задержка CAS Delay составляет 2 такта, то через 2 такта после команды чтения данные могут быть считаны с шины данных. Напомним, что у нас имеется две шины данных (линии) шириной  $p$  бит каждая и передача данных может происходить параллельно по каждой из этих линий. К примеру, первое слово ( $p$  бит), соответствующее первому банку (Bank0) и первому столбцу в этом банке (A1), может быть передано по первой линии, а второе слово (A2) одновременно с первым словом

— по второй линии. Далее, одновременно с передачей по первой линии слова А3, по первой линии данных может быть передано слово А4. Таким образом, по первой линии передаются данные А1, А3, В1, В3 и т.д., а по второй линии одновременно с ними — данные А2, А4, В2, В4 и т.д.

Затем эти данные передаются в мультиплексор синхронно с положительным фронтом тактового импульса и выводятся по шине шириной  $n$  бит синхронно с положительным и отрицательным фронтами.

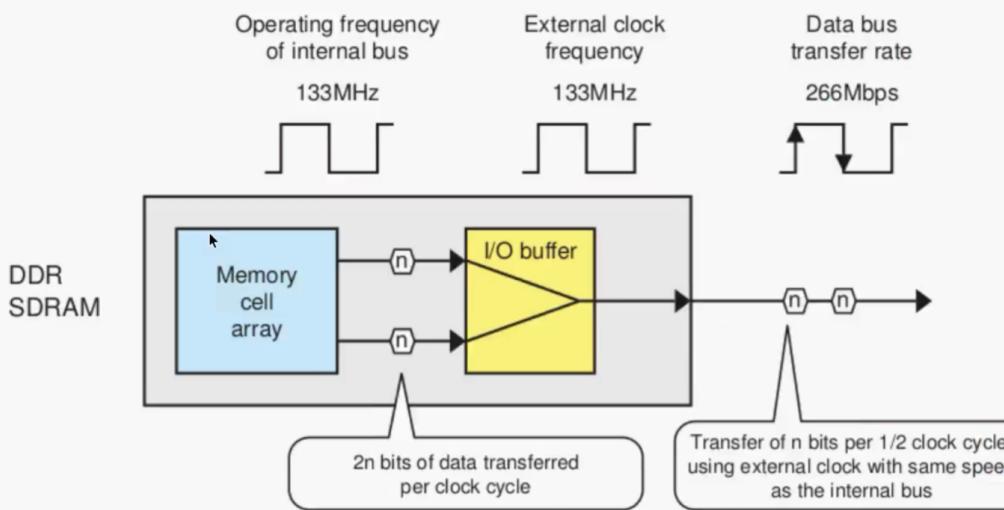
## 12. Done Сравнение DDR и DDR2

### Сравнение DDR и DDR2: DDR2 SDRAM



Document No. E0437E40 (Ver.4.0)  
Date Published September 2007 (K) Japan  
URL: <http://www.elpida.com>

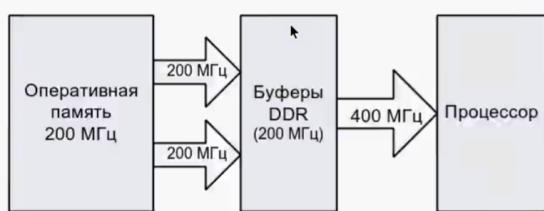
## Сравнение DDR и DDR2: DDR SDRAM



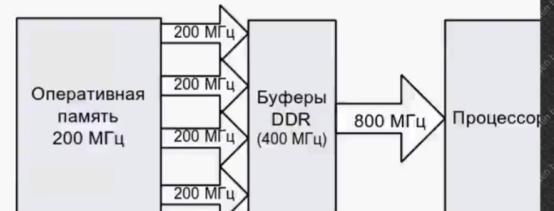
Document No. E0437E40 (Ver.4.0)  
Date Published September 2007 (K) Japan  
URL: <http://www.elpida.com>

## Сравнение DDR и DDR2

### DDR память



### DDR2 память



Основным архитектурным отличием памяти DDR2 является возможность передачи четырех блоков данных за такт вместо двух, как это было в случае DDR.

В результате выросла скорость обработки информации в два раза. Также DDR2 характеризуется более низким энергопотреблением.

Максимальная эффективная частота DDR составляет 400 МГц, а DDR2 - 800 МГц. Для DDR нужно 2,5 V, для DDR2 – 1,8 V

По внешнему виду DDR2 отличается от DDR количеством контактов, 240 против 184 у первого DDR.

(Дополнение из книжки от Масловой)

В упрощенном варианте структуру DDR SDRAM можно представить в виде, показанном на рис. 6.15.

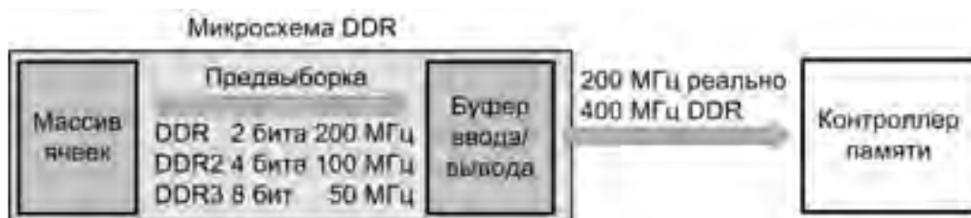


Рис. 6.15. Структура DDR SDRAM с иллюстрацией идеи n-битовой предвыборки

Чтобы обеспечить передачу данных дважды за такт, используется специальная архитектура с предвыборкой, в которой присутствует буфер ввода/вывода (буфер253 предвыборки). В каждом тактовом периоде шины из массива ячеек в этот буфер передаются два бита данных. Это называется 2n-битовой предвыборкой. В DDR2 внутренняя шина между массивом ячеек и буфером увеличена до 4 битов, а в DDR3 — до 8 битов.

Для лучшего понимания этой идеи сравним микросхемы DDR-400, DDR2-400 и условную DDR3-400 (реально такая микросхема не существует). Формально эти микросхемы работают на частоте шины памяти 200 МГц, передавая в каждом тактовом периоде два пакета данных, что внешне эквивалентно работе на тактовой частоте 400 МГц. В то же время внутри микросхемы DDR за тактовый период между массивом ячеек и буфером ввода/вывода за один тактовый период передаются 2 бита, поэтому соответствующий тракт должен работать на частоте 200 МГц (200 МГц × 2 = 400 МГц). Так как в DDR2 ширина тракта увеличена до 4 битов, то для получения той же производительности этот тракт может работать на вдвое меньшей частоте (100 МГц × 4 = 400 МГц), а в DDR3 — частота тракта может быть уменьшена еще вдвое (50 МГц × 8 = 400 МГц).

Удвоение ширины внутреннего тракта означает, что каждое новое поколение предположительно может включать микросхемы с удвоенной максимальной тактовой частотой, по сравнению с достигнутой в предыдущем поколении. Например, микросхемы DDR-400, DDR2-800 и DDR3-1600 внутренне работают на той же тактовой частоте 200 МГц.

В табл. 6.1 приведены основные характеристики поколений DDR.

Таблица 6.1. Характеристики микросхем DDR

	DDR	DDR2	DDR3
Буфер предвыборки, бит	2	4	8
Уровень сигналов, В	2,5	1,8	1,5
Частота шины	200, 266, 333, 400	400, 533, 677, 800	800, 1066, 1330, 1600

## 13. Диаграмма состояний УА DDR SDRAM

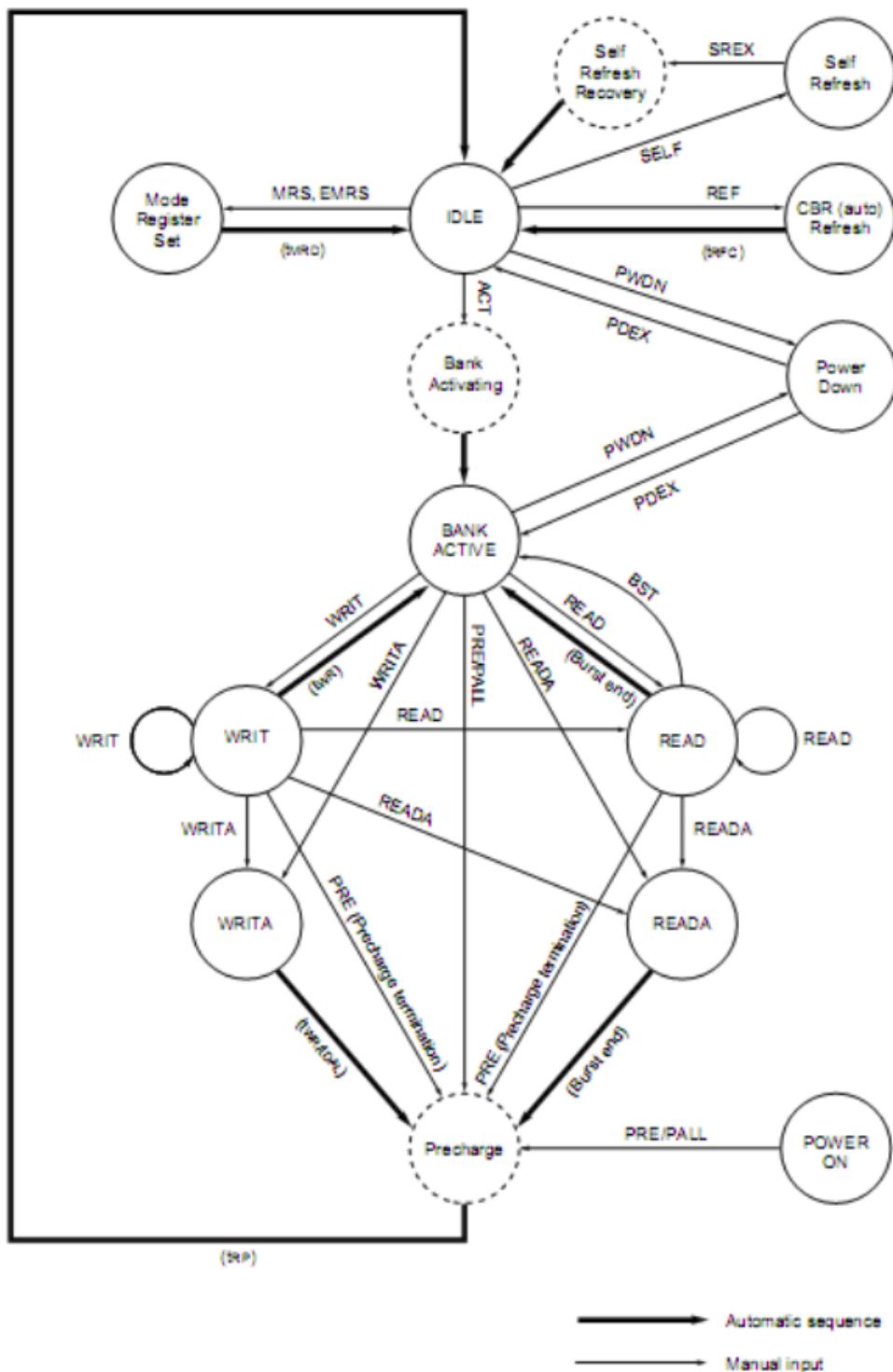
### Память DDR

Память DDR SDRAM, которая пришла на смену памяти SDR, обеспечивает вдвое большую пропускную способность. Аббревиатура DDR (Double Data Rate) в названии памяти означает удвоенную скорость передачи данных.

УА - управляющий автомат

Диаграмма переходов между состояниями автомата который находится внутри памяти и управляет работой одного банка памяти( часть из состояний на диаграмме относится ко всем банкам вместе, какие-то к конкретному банку памяти напр BANK ACTIVE) общие управляющие команды на диаграмме - те которые выше BANK ACTIVE

# Диаграмма состояний УА DDR SDRAM



power on - включается питание, должны в этом состоянии выдать некоторую цепочку инициализирующих команд, в станд. сказано что надо выдать команду PRECHARGE/PALL( команда перезарядить все линии чтения и записи)  
состояние Precharge - автомат всегда через trp перейдет в состояние IDLE

IDLE - состояние готовности к комадам когда все банки находятся в закрытом состоянии, линии чтения и записи перезаряжены и в этом состоянии мы можем сделать несколько инициализирующих действий:

1. записать Mode Register - регистр управления в который записываются cl, rl те данные для настройки интерфейса
2. можем перейти в режим Power Down те сбросить потребление (усилители выключаются) из режима пониженного энергопотребления можно потом вернуться в то состояние в котором мы находились в данном случае IDLE

Самое главное что нужно чтобы начать транзакцию - активировать банк командой Bank Activating (выборка строки те вся строка активируется, на это требуется некоторое время и это время показано как отдельное состояние из которого мы автоматически переходим в состояние Bank Active)

Bank Active - открыта определенная строка в банке(нужно выдать номер банка). Тот номер банка который мы передали вместе с номером строки тот и был активирован. К этому активному банку уже можно обращаться с командами чтения и записи чтение - WRIT - запись

WRITA - запись с автопречарджем

READ - чтение

READA - чтение с автопречарджем

автопречарж - команды за которыми автоматически следует внутренняя команда перезаряда Precharge, закрытия строки и возврата в IDLE. Идея - мы можем читать и писать в открытую строку много раз как только мы подходим к концу этой строки и знаем что следующий запрос будет к другой строке этот процесс можно сделать одной этой командой. Экономия команд управления.

На диаграмме видно что после чтения запись требует выдачи на шину данных этих данных и требуется пауза в управлении чтобы это сделать. А наоборот это не требуется.

Команды refresh:

self refresh - саморегенерация, команда запускающая внутренний контроллер регенерации находящийся внутри микросхемы, он будет перебирать адреса один за другим до тех пор пока мы его не остановим командой SRE - self refresh exit. счетчик перебирающий адреса остановится на том значении на котором мы его прервали и в следующий раз продолжит с этого момента. Это способ с помощью которого мы можем заполнять паузы в чтении и записи регенерацией.

CBR after refresh - команда с тем же самым счетчиком что и предыдущая, используется однократно, команда занимает 7 тактов, команда требует чтобы банк находился в состоянии precharge. Команда для того чтобы одну очередную строку регенерировать потратив на это 7 тактов и затем освободим память.

Идеология refresh - первая - берет и делает полный рефреш всего массива и за это время вся память регенерируется. Запускаем self refresh и ждем время за которое он обработает всю информацию в массиве

И второй вариант - вставлять self refresh в какие-то паузы например между чтением(плохо тем что чтение может быть интенсивным и такая стратегия приведет к тому что мы не выполним рефреш всего массива в эти паузы) поэтому более разумна 3-я стратегия, которая делит 64 миллисекунды на одинаковые интервалы порядка 7.2 микросекунд и раз в это время нужно выполнить однократный рефреш. Таким образом гарантируется что весь массив памяти рефрешится.

## 14. Тайминг памяти

### Тайминги памяти

Кроме максимальной пропускной способности, память характеризуется латентностью. Причем во многих случаях латентность памяти оказывает большее влияние на производительность всей системы в целом, нежели тактовая частота работы памяти.

латентность - задержка между поступлением команды и ее реализацией.

Латентность памяти определяется ее таймингами, то есть задержками, измеряемыми в количествах тактов, между отдельными командами. Принято различать несколько разных таймингов памяти, соответствующих задержкам между различным коммандами.

### Тайминг памяти: tCL-tRCD-tRP-tRAS

- CAS Latency (tCL) - задержка в тактах между подачей сигнала CAS и непосредственно выдачей данных из соответствующей ячейки. Одна из важнейших характеристик любого модуля памяти;
- RAS to CAS Delay (tRCD) - количество тактов шины памяти, которые должны пройти после подачи сигнала RAS до того, как можно будет подать сигнал CAS;
- Row Precharge (tRP) - время закрытия страницы памяти в пределах одного банка, тратящееся на его перезарядку;
- Activate to Precharge (tRAS) - время активности строба. Минимальное количество циклов между командой активации (RAS) и командой подзарядки (Precharge), которой заканчивается работа с этой строкой, или закрытия одного и того же банка.

Примеры таймингов памяти DDR: 2-2-2-5; 2.5-3-3-7

Примеры таймингов памяти DDR2: 3-3-3-9, 4-4-4-12 и 5-5-5-15

### RAS-to-CAS Delay (tRCD)

Первоначально происходит активация нужной строки памяти (команда ACTIVE), для чего сигнал RAS переводится в низкий уровень и происходит считывание адреса строки и адреса банка. Далее следует команда записи (WRITE) или чтения (READ) данных, для чего сигнал CAS переводится в низкий уровень и в надлежащий уровень устанавливается сигнал WE. При установке CAS в низкий уровень после прихода положительного фронта тактирующего импульса происходит выборка адреса столбца, наличествующего в данный момент на шине адреса, и открывается доступ к нужному столбцу матрицы памяти. Однако команда чтения или записи не может следовать непосредственно за командой активации – требуется, чтобы между этими командами, то есть между импульсами RAS и CAS, существовал некий промежуток времени – RAS-to-CAS Delay (задержка сигнала CAS относительно сигнала RAS). Эту задержку, измеряемую в тактах системной шины, принято обозначать tRCD. Учитывая, что импульс RAS эквивалентен выполнению команды активации строки (ACTIVE), а импульс CAS – команде READ, под задержкой RAS-to-CAS Delay можно понимать время между командами ACTIVE и READ (рис. 3).

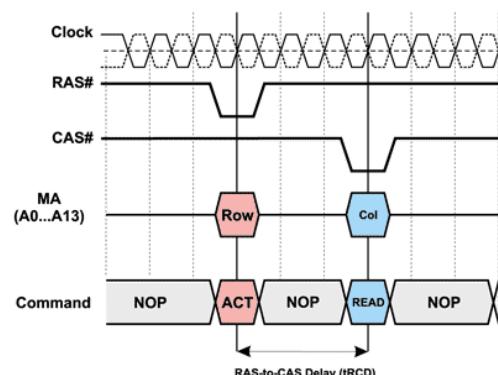


Рис. 3. Задержка RAS-to-CAS Delay (tRCD)

### CAS Latency (tCL)

От команды чтения (записи) данных и до выдачи первого элемента данных на шину (записи данных в ячейку памяти) проходит промежуток времени, который называется CAS Latency (рис. 4). Эта задержка измеряется в тактах системной шины и обозначается tCL. Каждый последующий элемент данных появляется на шине данных в очередном такте.

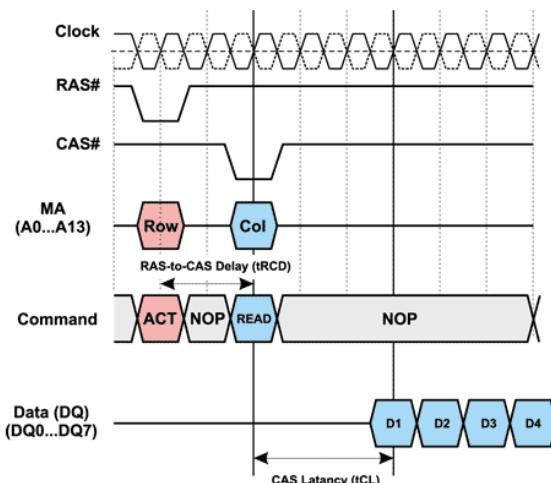


Рис. 4. Задержка CAS Latency (tCL)

### Active-to-precharge delay (tRAS)

Еще один тип задержки, называемый Active-to-precharge delay, – это минимальный промежуток времени, который должен пройти с момента подачи команды активации строки до команды PRECHARGE (рис. 5). Эта задержка обозначается tRAS и измеряется в тактах системной шины.

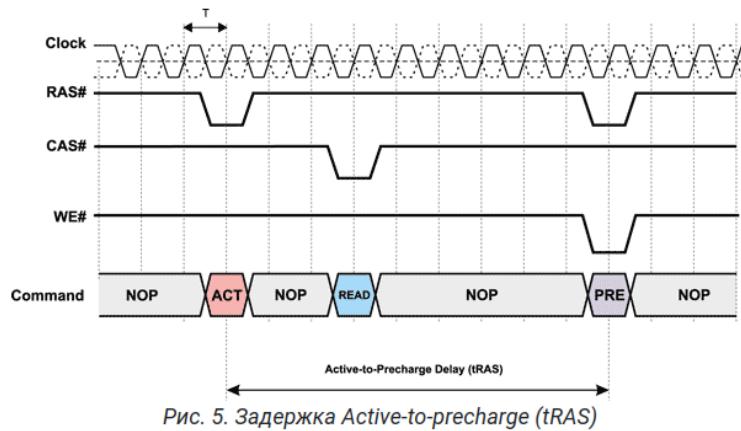


Рис. 5. Задержка Active-to-precharge (tRAS)

### RAS Precharge (tRP)

Завершение цикла обращения к банку памяти осуществляется подачей команды PRECHARGE, приводящей к закрытию строки памяти. От команды PRECHARGE и до поступления новой команды активации строки памяти должен пройти промежуток времени (tRP), называемый RAS Precharge (рис. 6).

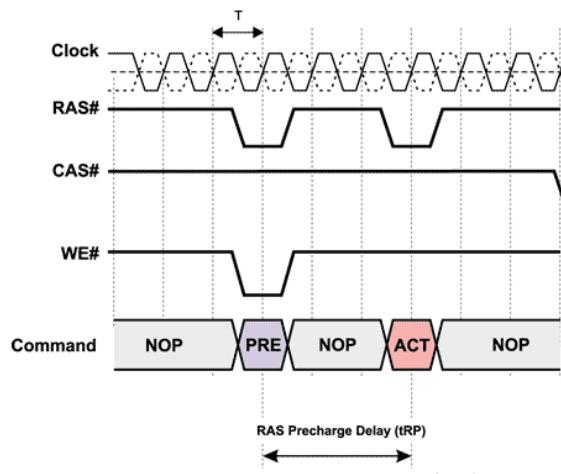


Рис. 6. Задержка RAS Precharge (tRP)

### Command Rate

Следующий тип задержки, о котором необходимо сказать, – это скорость выполнения команд (Command Rate), представляющая собой задержку в тактах системной шины между командой CS# выбора чипа и командой активации строки (рис. 7). Как правило, задержка Command Rate составляет один или два такта (1T или 2T).

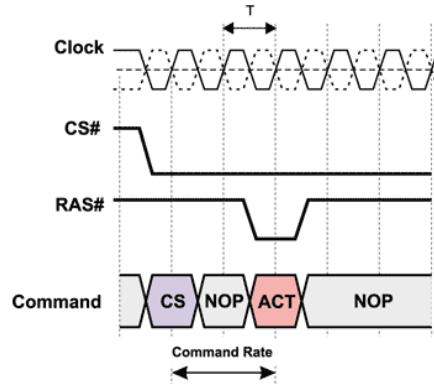


Рис. 7. Задержка Command Rate

## Соотношения между таймингами памяти

для каждого типа памяти значения различных задержек выбираются из допустимых значений. Кроме того, между разными таймингами должны соблюдаться определенные соотношения.

В простейшем случае для чтения данных из памяти необходимо выполнить последовательность следующих операций:

- активировать строку в банке памяти (команда ACTIVE);
- подать команду чтения данных (команда READ);
- считать данные, поступающие на внешнюю шину данных;
- закрыть активированную строку (команда PRECHARGE);
- активировать строку в банке памяти (команда ACTIVE).

Временной промежуток между активацией строки и командой чтения определяется как tRCD, а временной промежуток между командой чтения и появлением данных на шине — как tCL. Временной промежуток между началом считывания данных и закрытием активной строки (tRAS) зависит от длины передаваемого пакета, причем должно выполняться соотношение  $tRAS > tRCD + tCL$ .

Минимальное значение tRAS должно быть больше суммы tRCD и tCL на столько, на сколько велика длительность третьей операции, определяемая длиной передаваемого пакета. В качестве примера рассмотрим память типа SDR с величинами задержек tCL = 2 и tRCD = 2 ( $tRAS > 4$ ). При длине пакета BL = 2 необходимо затратить не менее 2 тактов для передачи всего пакета, поэтому минимальное значение tRAS должно быть равным 6.

## Запись таймингов памяти

Описанные задержки — RAS-to-CAS Delay (tRCD), CAS Latency (tCL), RAS Precharge (tRP), Active-to-precharge delay (tRAS) и Command Rate — определяют тайминги памяти, обычно записываемые в виде последовательности tCL-tRCD-tRP-tRAS-Command Rate. К примеру, для модуля DDR400 (PC3200) тайминги могут быть следующими: 2-3-4-5-(1T). Это означает, что для данного модуля CAS Latency (tCL) составляет 2 такта, RAS to CAS Delay (tRCD) — 3 такта, RAS Precharge (tRP) — 4 такта, ACTIVE-to- precharge delay (tRAS) — 5 тактов и Command Rate — 1 такт.

## 15. Done Классификация ПЗУ

### Классификация ПЗУ

#### 1) По типу исполнения:

- Массив данных совмещен с устройством выборки (считывающим устройством), в этом случае массив данных называется прошивкой:
  - микросхема ПЗУ;
  - один из внутренних ресурсов однокристальной микро ЭВМ (микроконтроллера), как правило FlashROM.
- Массив данных существует самостоятельно:
  - компакт-диск;
  - перфокарта;
  - перфолента;
  - монтажные «1» и монтажные «0».

#### 2) По разновидностям микросхем ПЗУ:

- По технологии изготовления кристалла:
  - ROM — (англ. Read-Only Memory, постоянное запоминающее устройство), масочное ПЗУ, изготавливается фабричным методом. В дальнейшем нет возможности изменить записанные данные.
  - PROM — (англ. Programmable Read-Only Memory, программируемое ПЗУ (ППЗУ)) — ПЗУ, однократно «прошиваемое» пользователем.
  - EEPROM — (англ. Erasable Programmable Read-Only Memory, перепрограммируемое ПЗУ (ПППЗУ)).
  - EEPROM — (англ. Electrically Erasable Programmable Read-Only Memory, электрически стираемое перепрограммируемое ПЗУ). Память такого типа может стираться и заполняться данными несколько десятков тысяч раз. Используется в твердотельных накопителях. Одной из разновидностей EEPROM является флеш-память (англ. Flash Memory).
  - ПЗУ на магнитных доменах, например K1602РЦ5, имело сложное устройство выборки и хранило довольно большой объем данных в виде намагниченных областей кристалла, при этом не имея движущихся частей (см. Компьютерная память). Обеспечивалось неограниченное количество циклов перезаписи. 12
  - NVRAM, Non-volatile memory — «неразрушающаяся» память, строго говоря, не является ПЗУ. Это ОЗУ небольшого объема, совмещенное с первичным источником электропитания. В СССР такие устройства часто назывались «Dallas» по имени фирмы, выпустившей их на рынок. В

NVRAM современных ЭВМ батарейка уже конструктивно не связана с ОЗУ и может быть заменена.

- • По виду доступа:
  - с параллельным доступом (Parallel mode или Random access): такое ПЗУ может быть доступно в системе в адресном пространстве ОЗУ. Например, K573РФ5; о с последовательным доступом: такие ПЗУ часто используются для однократной загрузки констант или прошивки в процессор или ПЛИС, используются для хранения настроек каналов телевизора, и др. Например, 93С46, AT17LV512A.
  - 13 • По способу программирования микросхем (записи в них прошивки):
  - непрограммируемые ПЗУ; о ПЗУ, программируемые только с помощью специального устройства — программатора ПЗУ (как однократно, так и многократно прошиваемые). Использование программатора необходимо, в частности, для подачи нестандартных и довольно высоких напряжений (до +/- 27 В) на специальные выводы.
  - внутрисхемно (пере)программируемые ПЗУ (ISP, in-system programming) — такие микросхемы имеют внутри генератор всех необходимых высоких напряжений, и могут быть перепрошиты без программатора и даже без выпайки из печатной платы, программным способом.

## 16. Done Структура ПЗУ (ROM)

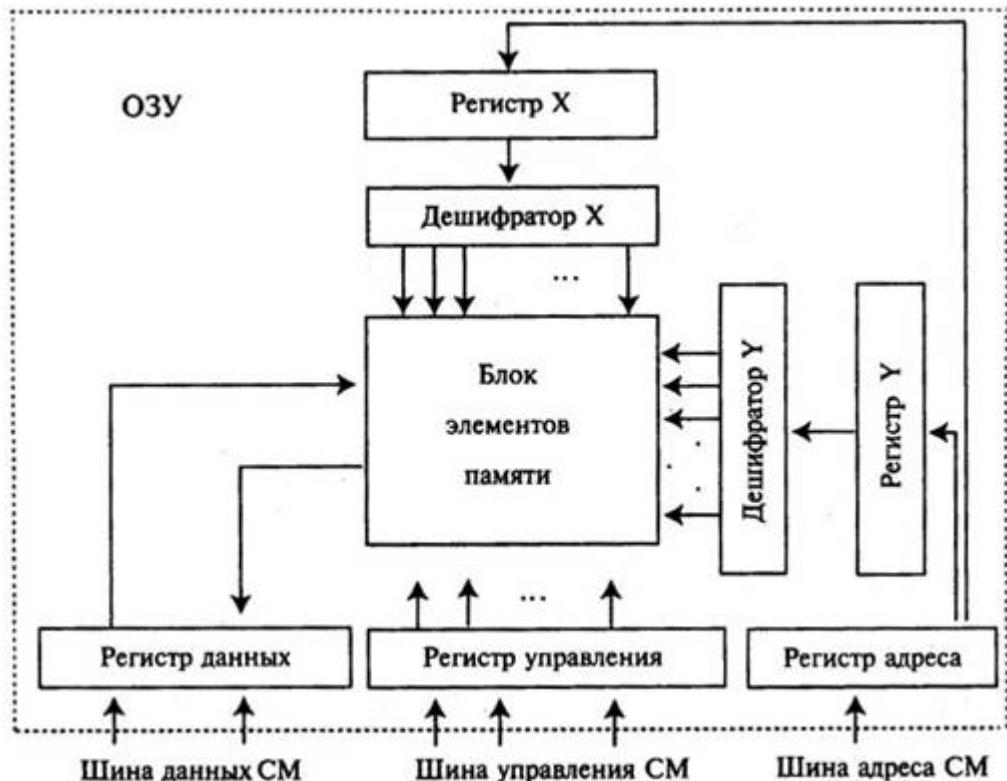
Постоянное запоминающее устройство (ПЗУ) — это основной блок памяти любой компьютерной системы вместе с ОЗУ, но в отличие от ОЗУ в ПЗУ двоичная информация хранится постоянно.

Основной составной частью микросхемы является массив элементов памяти (ЭП), объединенных в матрицу накопителя.

Каждый элемент памяти может хранить 1 бит информации и имеет свой адрес. ЗУ, позволяющие обращаться по адресу к любому ЭП в произвольном порядке, называются запоминающими устройствами с произвольным доступом.

При матричной организации памяти реализуется координатный принцип адресации ЭП, в связи с чем адрес делится на две части (две координаты) - X и Y. На пересечении этих координат находится элемент памяти, чья информация должна быть прочитана или изменена.

ОЗУ связано с остальным микропроцессорным комплектом ЭВМ через системную магистраль (рис.1).

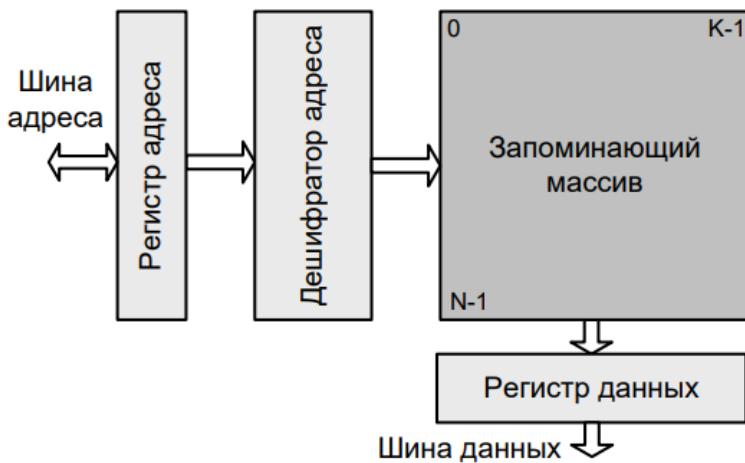


По шине управления передается сигнал, определяющий, какую операцию необходимо выполнить.

По шине данных передается информация, записываемая в память или считываемая из нее.

По шине адреса передается адрес участвующих в обмене элементов памяти (поскольку данные передаются машинными словами, а один ЭП может воспринять только один бит информации, блок элементов памяти состоит из  $n$  матриц ЭП, где  $n$  - количество разрядов в машинном слове). Максимальная емкость памяти определяется количеством линий в шине адреса системной магистрали.

## Структура ПЗУ (ROM)



### 17. ?Элементная база МПЗУ

(Ковалец К)

#### По лекциям:

В него нельзя записать, можно только прочитать. Там задержки крайне малы. На таких ПЗУ реализован микрокод процессоров, т.е. мы успеваем не только извлечь команду, но даже её исполнить. Информация для операции сложения хранится в МПЗУ.

Пример:

Есть горизонтальные линии - это линии выборки, идущие из дешифраторов, есть вертикальные линии - это линии считывания, по которым мы поймём, что у нас на этой линии, если диод есть, то это единица, потому что высокий уровень сигнала на линии выборки приведёт к тому, что этот потенциал через открытый PN-переход будет повторён на трансе (трансе? плохо слышно слово) 0.7 вольт (он сказал ноль семь вольт, 0.7 или 0-7 хз), как обычно на линии. Т.е. ток потечёт таким образом, через диод, дальше через сопротивление на дембель (вроде дембель сказал).

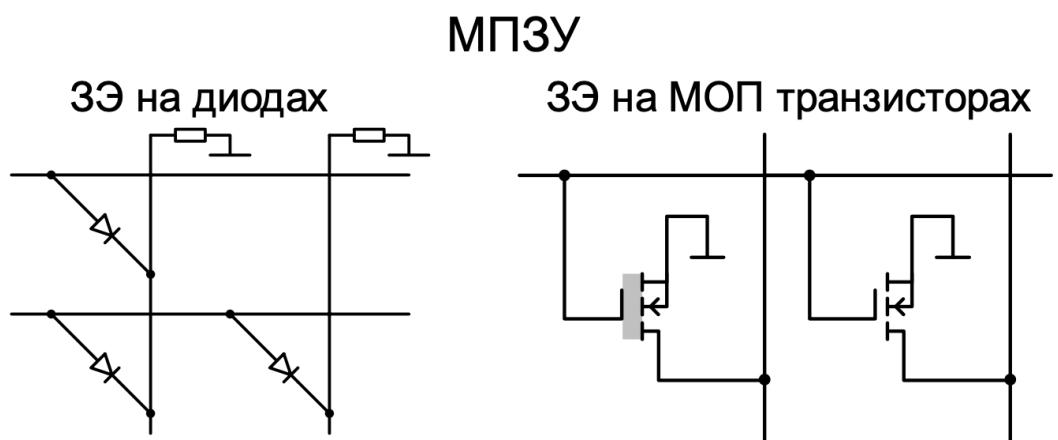
Диоды плохо реализуются в интегральной технологии, PN-переход никому не нужен. Нужен либо PNP, т.е. биполярный, либо полевой транзистор.

Тип транзисторов с индуцированным каналом, рисуется с прерывистой чертой, потому что канал между стоком и истоком это верхний и нижний проводник, средний называется помощник. Между ними создаётся канал, т.е. падает сопротивление и течёт ток только тогда, когда имеет высокий уровень, т.е. выше порогового значения. Так вот, есть два транзистора, у одного под затвором тонкий слой диэлектрика, а у другого толстый. Получается, чем толще слой диэлектрика, тем более затвор меньше влияет на канал, а именно полем

втягиваются носители заряда, которые обеспечивают канал. При одном и том же потенциале на линии выборки один транзистор останется открытым, другой закрытым. (Дальше Попов начал что-то кому-то показывать и говорить про какую-то линию и всё ему стало очевидно и понятно)

#### По книге:

Группу ПЗУ, программируемых при изготовлении, образуют так называемые масочные устройства, и именно к ним принято применять аббревиатуру ПЗУ. Занесение информации в масочные ПЗУ составляет часть производственного процесса и заключается в подключении или не подключении запоминающего элемента к разрядной линии считывания. В зависимости от этого из ЗЭ будет всегда извлекаться 1 или 0. В роли перемычки выступает транзистор, расположенный на пересечении адресной и разрядной линий. Какие именно ЗЭ должны быть подключены к выходной линии, определяет маска, «закрывающая» определенные участки кристалла.



## 18. ?Элементная база ППЗУ

(Ковалец К)

#### По лекциям:

На заводе хорошо и получаются самые быстродействующие устройства, задержка минимальна, потому что они определяются сопротивлениями устройств.. (дальше какой-то бубнеж)

С завода мы получаем микросхему с записанными значениями, мы можем один раз прописать и после этого она будет надежно храниться и обладать хорошими свойствами по быстродействию. Это называется программируемое ПЗУ.

Первый вариант ППЗУ с плавкими перемычками. Перемычка это тонкая алюминиевая часть проводника утолщена, и благодаря импульсу тока, который мы пропускаем в этом направлении, происходит локальный нагрев. При таком

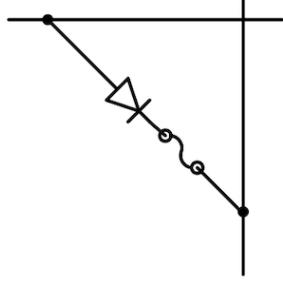
нагреве алюминий диффузируется в кремний, т.е. он распространяется из-за нагрева по толще кремния, и сопротивление этого куска резко возрастает, пережигая (?) перемычку. Второй вариант ППЗУ с пережигаемым р-п переходом. Два диода встречных включены, если так, то ток не течёт. Но стоит нам пробить диод выпирающий, он будет пробит, через него ток потечёт. Диоды не очень удобно делать кремниевые технологии, он занимает много места. Обычно там полевой транзистор делают или биполярный. Диоды можно делать с разным напряжением пробоя, и поэтому подбирают так, чтобы импульсом тока пробить этот диод, он уже не восстановился. В чём недостаток? В том, что здесь возникает нагрев локальный и надо, чтобы он не повлиял на соседние элементы. Можно не с первого раза получить нужную нам прошивку, можно что-то недопережечь и иногда повторно пережигают.

#### **По книге:**

Создание масок для ПЗУ оправдано при производстве большого числа копий. Если требуется относительно небольшое количество микросхем с данной информацией, разумной альтернативой являются однократно программируемые ПЗУ. Такие микросхемы обозначают аббревиатурой PROM (Programmable ROM — программируемые ПЗУ). Информация в них может быть записана только однократно. Первыми PROM стали микросхемы памяти на базе плавких предохранителей. В исходной микросхеме во всех узлах адресные линии соединены с разрядными. Занесение информации в PROM производится электрически, путем пережигания отдельных перемычек, и может быть выполнено поставщиком или потребителем спустя какое-то время после изготовления микросхемы. Подобные ПЗУ выпускались в рамках серий К556 и К1556. Позже появились ИМС, где в перемычку входили два диода, соединенные навстречу. В процессе программирования удалить перемычку можно было с помощью электрического пробоя одного из этих диодов. В любом варианте для записи информации требуется специальное оборудование — программаторы. Основными недостатками данного вида ПЗУ были большой процент брака и необходимость специальной термической тренировки после программирования, без которой надежность хранения данных была невысокой.

## ППЗУ

ППЗУ с плавкими  
перемычками

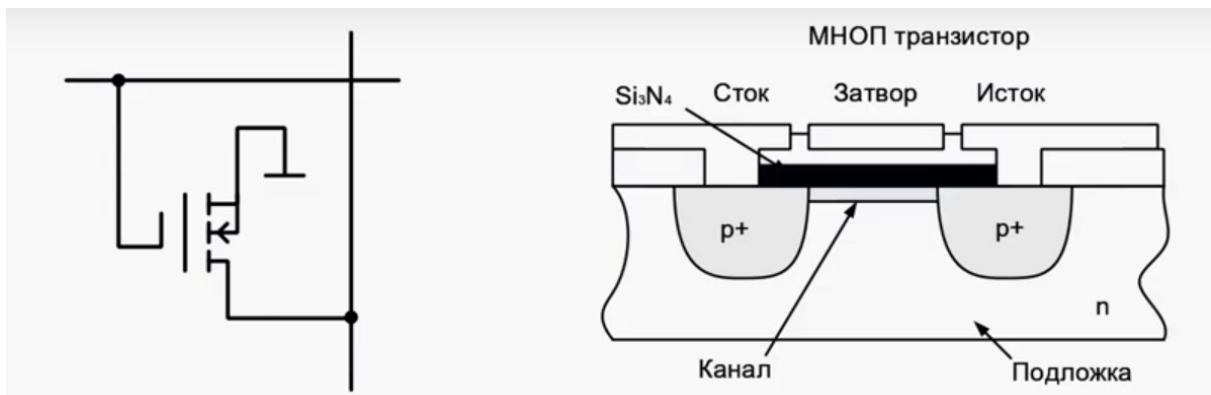


ППЗУ с пережигаемым р-  
n переходом



### 19. Check Элементная база РПЗУ-УФ и ОПРРПЗУ-УФ

С лекции: появился транзистор с ультрафиолетовым стиранием, который назывался МНОП (металл нитрид оксид полупроводник) под затвором находилась композиция 2 материалов: оксид кремния и нитрид кремния. Эти 2 материала имеют разную кристаллическую решетку. В месте соприкосновения возникает дефектная область, которая является потенциальной ямой. Если электрон попадает в эту яму, то он может находиться там миллионы лет, то есть нам нужно внести заряд, и он не покинет область, если не приложить ему обратный потенциал для перехода оттуда или не передать какую-то энергию. Энергию вносят из канала, а рассасывают его ультрафиолетовыми лучами, то есть с помощью облучения УФ лампой заряд удается оттуда удалить, но ультрафиолет плохо влияет на все эти материалы, они разрушаются. Из-за этого у такой ультрафиолетовой флешки было порядка 10 циклов записи



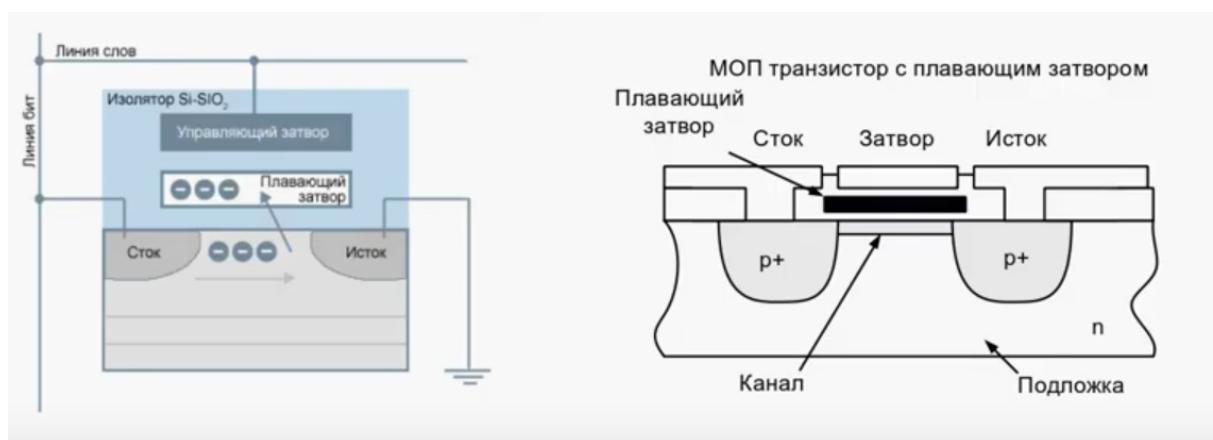
более подробно про МНОП транзистор:

<http://dfe.petrsu.ru/koi/posob/microcpru/pam1.html>

лекция: <https://youtu.be/dyc0YwvphQI?t=2648>

## 20. Check Элементная база РПЗУ-ЭС

Репрограммируемое ПЗУ с электрическим стиранием. Сейчас используются транзисторы с плавающим затвором. Идея работы схожа с МНОП. Плавающий затвор формируется с использованием затворного диэлектрика. Создается изолированный кусок проводника, куда мы переносим заряды благодаря полю, то есть электроны стягиваем в плавающий затвор, который целиком изолирован от всех областей. При обычном стечении обстоятельств и обычном потенциале этот заряд никуда не денется. Он попадает в некую потенциальную яму. Чтобы заряд рассосался, нужно приложить обратную разность потенциалов. Заряд стягивается из канала и покидает плавающий затвор снова в канал. Флешки умеют внутри себя генерировать сигналы высокого напряжения (12-17В) для стирания и записи. Процесс стирания может повредить подзатворный диэлектрик, количество циклов перезаписывания порядка нескольких миллионов.



лекция: <https://youtu.be/dyc0YwvphQI?t=2869>

## 21. Доделать Структура NAND FLASH

Флэш-память NAND характеризуется малой стоимостью из расчета на 1 бит, а также высокой скоростью операций стирания и записи. Адресная шина мультиплексируется с шиной данных, что упрощает схему. В настоящее время это основная технология хранения данных для огромного числа устройств, начиная с твердотельных накопителей и заканчивая SD-картами с eMMC-накопителями. Флэш-память NAND выбирают в тех случаях, когда требуется обеспечить высокую плотность записи и быстродействие.

Память этого типа состоит из набора блоков, каждый из которых разделен на физические страницы. Каждая страница содержит область данных и свободную область, которая используется под данные управления NAND-памятью. Все чаще в памяти NAND используется несколько слоев с независимым обращением. Эти слои параллельно программируются или стираются.

Три основные особенности флэшпамяти NAND

- Для исправления некорректно хранящихся данных необходим код коррекции ошибок (ECC).
- Запоминающие устройства этого типа содержат небольшое количество поврежденных блоков, число которых может увеличиться в течение срока службы памяти.
- Если устройство эксплуатируется в жестких условиях, требуется оптимизировать (выровнять) его износ, чтобы продлить срок службы

## 22. доделать и проверить Структура NOR FLASH

Флэш-память NOR, которая появилась до NAND, по-прежнему пользуется спросом на рынке встраиваемых систем благодаря своей простоте и надежности. В случае применения памяти NOR код исправления ошибок не должен обеспечить 100 тыс. циклов стирания/записи. При изготовлении этой памяти с гораздо более простой структурой блоки не повреждаются. Следует заметить, что более высокая надежность достигается за счет намного более длительного времени стирания и записи. Однако использование NOR, как и флэш-памяти всех остальных типов, требует ясного понимания особенностей этой технологии.

Существуют два основных вида флэшпамяти NOR: параллельная, у которой шины адресов и данных напрямую подключены к микроконтроллеру, и последовательная, которая использует только SPI- (или схожий) интерфейс. Как правило, объем последовательной памяти NOR составляет <1–128 Мбайт. У параллельной NOR объем намного больше.

В приложениях с большой нагрузкой на память NOR требуется управляющий уровень (часто FTL), который обеспечивает оптимизацию износа и равномерное использование устройства. Этот уровень также управляет поврежденными блоками. Цена памяти NOR может превышать цену альтернативных решений, а ее время стирания и записи относительно больше. К коду исправления ошибок никаких требований не предъявляется, но для его эффективного использования необходимо из логического адреса получать информацию о физическом адресе, чтобы обеспечить оптимизацию износа.

## 23. Check Сравнение NAND и NOR FLASH

	<b>NOR</b>	<b>NAND</b>
--	------------	-------------

Доступ к данным	<p>К данным можно обращаться в случайному порядке, так же как к SRAM.</p> <p>Операциями с флеш-памятью могут быть:</p> <p><b>Процедура чтения:</b> чтение содержимого флеш-памяти.</p> <p><b>Процедура стирания:</b> стирание - это процесс установки всех битов флеш-памяти в 1. Стирание в микросхемах NOR происходит в терминах блоков (называемых областями стирания).</p> <p><b>Процедура записи:</b> запись - это процесс изменения во флеш-памяти 1 в 0. После того, как бит стал 0, он не может быть записан, пока блок не будет стёрт, что означает установку всех битов в блоке в 1.</p>	<p>В микросхемах NAND флеш пространство разделено на блоки, которые также разделены на страницы. Каждая страница разделена на обычные данные и дополнительные (out-of-band) данные. Дополнительные данные используются для того, чтобы хранить метаданные, такие как ECC (Error-Correction Code, Код для коррекции ошибок) данные и информацию о неисправном блоке.</p> <p>NAND флеш, как и NOR флеш, имеет три основные операции: <b>чтение, стирание и запись</b>. Однако, в отличие от NOR, к которым можно обращаться в произвольном порядке, чтение и запись в NAND флеш выполняются в терминах страниц, тогда как стирание происходит в терминах блоков.</p>
Подключение к плате	Подключается как обычное устройство SRAM к шинам адреса и	Существуют несколько способов подключения NAND флеш к CPU, зависящих от производителя. Для доступа к NAND выполняется соединение выводов данных и

	данных процессора.	команд к обычно 8 выводам ввода-вывода на микросхеме флеш-памяти.
Выполнение кода	Код может быть выполнен прямо из NOR, поскольку она подключена непосредственно к шинам адреса и данных.	Если код находится в NAND флеш, для запуска он нуждается в копировании в память.
Производительность	Флеш-память NOR характеризуется медленным стиранием, медленной записью и быстрым чтением.	Флеш-память NAND характеризуется быстрым стиранием, быстрой записью и быстрым чтением.
Неисправные блоки	Неисправные блоки в микросхемах NOR флеш не ожидаются, поскольку они были разработаны, чтобы хранить системные данные.	Эти микросхемы были разработаны в основном как устройства хранения медиа-данных с более низкой ценой, поэтому стоит ожидать, что они имеют неисправные блоки. Обычно такие микросхемы флеш-памяти поставляются с помеченными неисправными участками. Также сектора NAND флеш больше страдают от проблемы переключения битов, когда бит становится перевернутым во время записи; это обнаруживается алгоритмами коррекции ошибок, называемых ECC/EDC, которые выполняются либо в оборудовании, либо в программном обеспечении.

Применение	<p>В основном используются для выполнения кода. На NOR флеш могут находиться загрузчики, потому что код из такой флеш-памяти может быть выполнен напрямую. Такая флеш-память довольно дорогая и она обеспечивает меньшие плотности памяти и имеет относительно более короткую продолжительность жизни (приблизительно 100 000 циклов стирания).</p>	<p>Используются главным образом в качестве устройств хранения для встраиваемых систем, таких как приставки к телевизорам и MP3-плееры. Если вы планируете использовать плату только с NAND, вам придётся добавить дополнительное ПЗУ загрузки. Они предлагают высокие плотности при более низких ценах и имеют более длительный срок службы (около 10 в 6-ой степени циклов стирания).</p>
------------	---	--

Дополнительная таблица сравнения:

Feature	NOR Flash		NAND Flash	
	General	S70GL02GT	General	S34ML04G2
<b>Capacity</b>	8MB – 256MB	256MB	256MB – 2GB	256MB
<b>Cost per bit</b>	Higher	$6.57 \times 10^{-9}$ USD/bit for 1ku	Lower	$2.533 \times 10^{-9}$ USD/bit for 1ku
<b>Random Read speed</b>	Faster	120ns	Slower	30μS
<b>Write speed</b>	Slower		Faster	
<b>Erase speed</b>	Slower	520ms	Faster	3.5ms
<b>Power on current</b>	Higher	160mA (max)	Lower	50mA (max)
<b>Standby current</b>	Lower	200μA (max)	Higher	1mA (max)
<b>Bit-flipping</b>	Less common		More common	
<b>Bad blocks while shipping</b>	0%		Up to 2%	
<b>Bad block development</b>	Less frequent		More frequent	
<b>Bad block handling</b>	Not mandatory		Mandatory	
<b>Data Retention</b>	Very high	20 years for 1K program-erase cycles	Lower	10 years (typ)
<b>Program-erase cycles</b>	Lower	100,000	Higher	100,000
<b>Preferred Application</b>	Code storage & execution		Data storage	

## 24. Доделать и проверить ПЗУ типа NVRAM

Сокращая от **энергонезависимой оперативной памяти**, **NVRAM** - это память, которая сохраняет свои сохраненные данные независимо от того, включено или выключено питание благодаря наличию встроенной литиевой батареи для резервного питания (обычно составляет от 32 КБ до 128 КБ). Интегрированная схема контроля и переключения на резервный источник питания гарантирует работоспособность данного типа ПЗУ и сохранение данных в течение десяти лет при полном отсутствии внешнего питания. Но, иногда, происходит нехватка “места” для NVRAM, что может повлечь за собой невозможность сохранения всех данных или вовсе содержимое NVRAM повредится, что приведет к сбою в работе.

Сегодня хорошим примером NVRAM является флэш-память, подобная той, которая используется в накопителе Jump. NVRAM также присутствует в мониторе вашего компьютера, принтерах, автомобилях, смарт-картах и других устройствах, которые требуют запомненных настроек.

В настоящий момент, для всех компьютеров доступны разные типы NVRAM. RTC / NVRAM на материнской плате вашего компьютера - это NVRAM с батарейным питанием, который использует батарею CMOS для зарядки и сохранения системных настроек, таких как дата и время.

Другим хорошим примером NVRAM в компьютерах является EEPROM, который используется для BIOS во многих компьютерах. Даже ваш жесткий диск и другие устройства хранения считаются энергонезависимой памятью

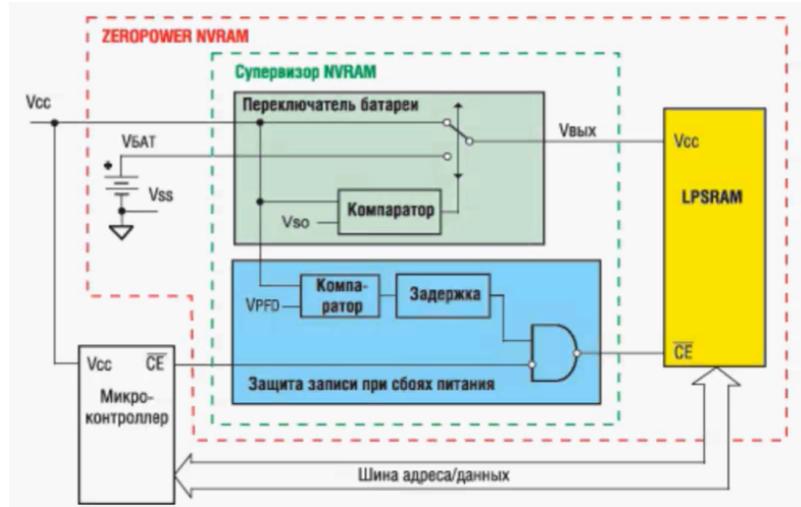
### **преимущества**

- NVRAM не имеет движущихся частей и почти всегда быстрее, чем энергозависимая память для чтения и записи.
- С меньшим количеством движущихся частей NVRAM требует гораздо меньше энергии.

### **Недостатки**

- NVRAM, для которого требуется батарея, которую в конечном итоге потребуется заменить.
- По мере того как информация перезаписывается во флэш-память, она ухудшается и в конечном итоге перестает работать.

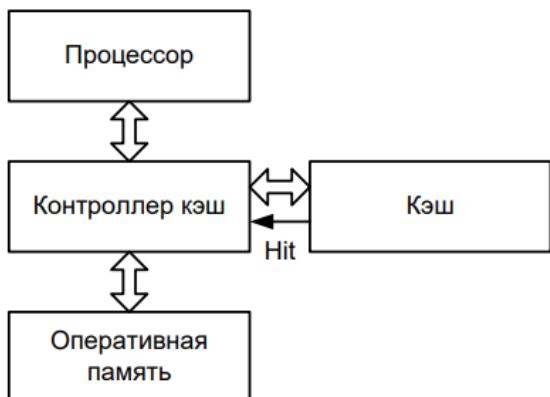
### Схема работы:



## 25. доделать Принципы построения кэш-памяти

### Принципы построения кэш-памяти

Кэш-память – ассоциативное ЗУ, позволяющее сгладить разрыв в производительности процессора и оперативной памяти. Выборка из кэш-памяти осуществляется по физическому адресу ОП.



Эффективность кэш-памяти зависит от:

- Емкости кэш-памяти.
- Размера строки.
- Способа отображения ОП в кэш.
- Алгоритма замещения информации в кэш.
- Алгоритма согласования ОП и кэш.
- Числа уровней кэш.

Кэш-память – это сверхбыстрая память используемая процессором, для временного хранения данных, которые наиболее часто используются. Вот так, вкратце, можно описать данный тип памяти.

Кэш-память построена на триггерах, которые, в свою очередь, состоят из транзисторов. Группа транзисторов занимает гораздо больше места, нежели те же самые конденсаторы, из которых состоит оперативная память. Это тянет за собой множество трудностей в производстве, а также ограничения в объёмах. Именно поэтому кэш память является очень дорогой памятью, при этом обладая ничтожными объёмами. Но из такой структуры, вытекает главное преимущество такой памяти – скорость. Так как триггеры не нуждаются в регенерации, а время задержки вентиля, на которых они собраны, невелико, то время переключения триггера из одного состояния в другое происходит очень быстро. Это и позволяет кэш-памяти работать на таких же частотах, что и современные процессоры

Также, немаловажным фактором является размещение кэш-памяти. Размещена она, на самом кристалле процессора, что значительно уменьшает время доступа к ней. Ранее, кэш память некоторых уровней, размещалась за пределами кристалла процессора, на специальной микросхеме SRAM где-то на просторах материнской платы. Сейчас же, практически у всех процессоров, кэш-память размещена на кристалле процессора.

## 26. Done Емкость и размер линейки 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.

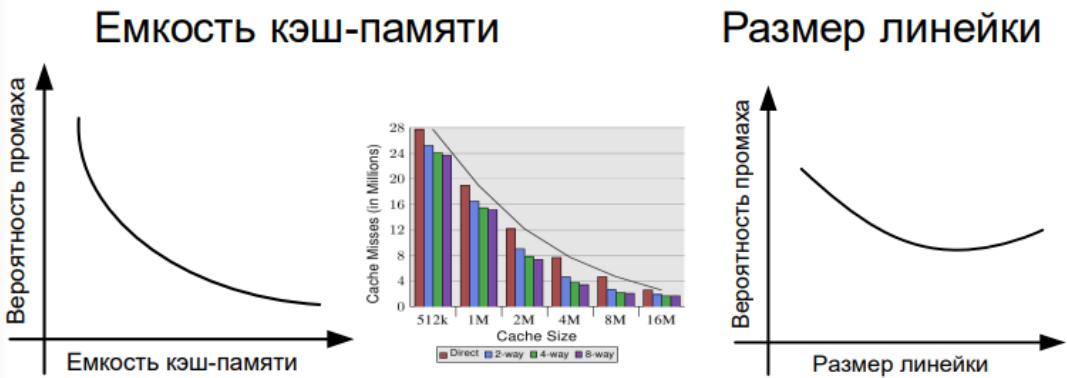
Несмотря на очевидные различия, просматривается и общая тенденция: по мере увеличения емкости кэш-памяти вероятность промахов сначала существенно снижается, но при достижении определенного значения эффект сглаживается и становится несущественным. Установлено, что для большинства задач близкой к оптимальной является кэш-память емкостью от 1 до 512 кб.

Еще одним важным фактором, влияющим на эффективность использования кэшпамяти, 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 принципу локальности, попадает все больше данных, которые понадобятся в ближайшее время. Однако вероятность промахов начинает расти, когда размер строки становится излишне большим (рис. 5.25, б). 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.

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.



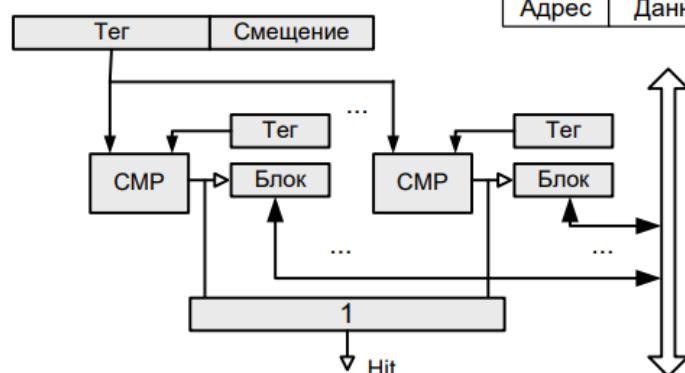
Способы отображения ОП в кэш:

- Произвольная загрузка.
- Прямое размещение.
- Наборно-ассоциативный способ отображения.

27. Done Способы отображения ОП в кэш: произвольная загрузка

**Произвольная** загрузка  
(Fully associated cache memory, FACM).

Адрес строки FACM  
определяется из условия  
формирования наиболее  
представительной выборки



В данном способе между местом хранения в оперативной памяти (ОП) и местом хранения в кэш-памяти нет никаких ограничений (это отражено на схеме с таблицами Адрес-Данные) - разрешается загрузка любого блока ОП в любой блок кэш-памяти. Связь обеспечивает кэш-контроллер на основе своего алгоритма. Он может подсчитывать, сколько обращений было к каждой из линеек кэш-памяти, и решать какие данные оставить, какие вытеснить. Нужно найти менее используемые и вытеснить их.

К схеме с тегом и смещением: в этих условиях тегом служит полный адрес блока в ОП. Для проверки наличия копии блока в кэш-памяти необходимо сравнить теги всех блоков кэш-памяти на совпадение со старшими разрядами адреса, поступившего из ЦП.

Преимущество: хорошая представительность выборки.

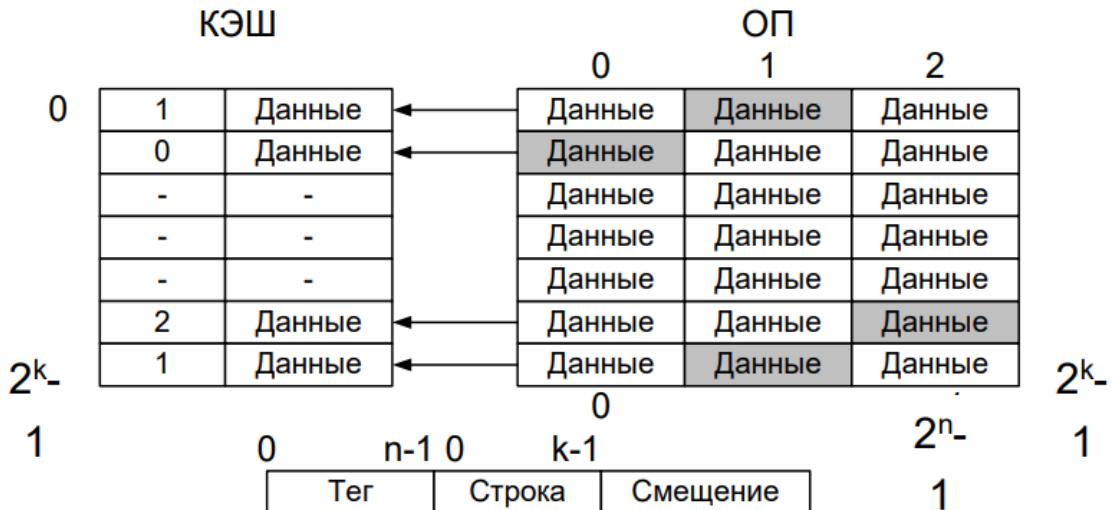
Недостаток: аппаратно сложный способ, так как число компараторов должно соответствовать числу линеек в кэш-памяти.

Применимость: сейчас практически не используется. Используется для маленьких кэшей в задачах, требующих переупорядоченного ответа. Например, при соответствии транзакций: посылают на шину транзакции в одной последовательности, а отвечают в другой последовательности.

## 28. Done Способы отображения ОП в кэш: прямое размещение

Прямое размещение.

Адрес строки однозначно определяется по тегу ( $i = t \bmod k$ ).



В данном способе четкое соответствие места в ОП (физический адрес) и ячейки кэша. ОП делим на части по размеру блока кэш-памяти - множество блоков ОП условно представляется в виде матрицы, в которой количество строк равно числу блоков в кэш-памяти.

К схеме: В блок кэш-памяти с номером  $i$  может быть помещен любой блок ОП, но только из  $i$ -й строки. Номер строки  $i$  и столбца  $j$  матрицы, на пересечении которых располагается блок основной памяти с адресом  $t$ , определяются выражениями  $i = t \bmod k$ ,  $j = t \div k$ ,  $k$  - число блоков в кэш-памяти. В качестве указателя того, какой из блоков  $i$ -й строки ОП отображен или должен быть отображен на  $i$ -й блок КП (тега), используется номер столбца строки, где расположен отображенный (отображаемый) блок ОП.

При обращении совпадение означает, что в кэш-памяти находится копия затребованного блока ОП, и процессор адресуется к кэш-памяти. Несовпадение порождает копирование затребованного блока в кэш-память

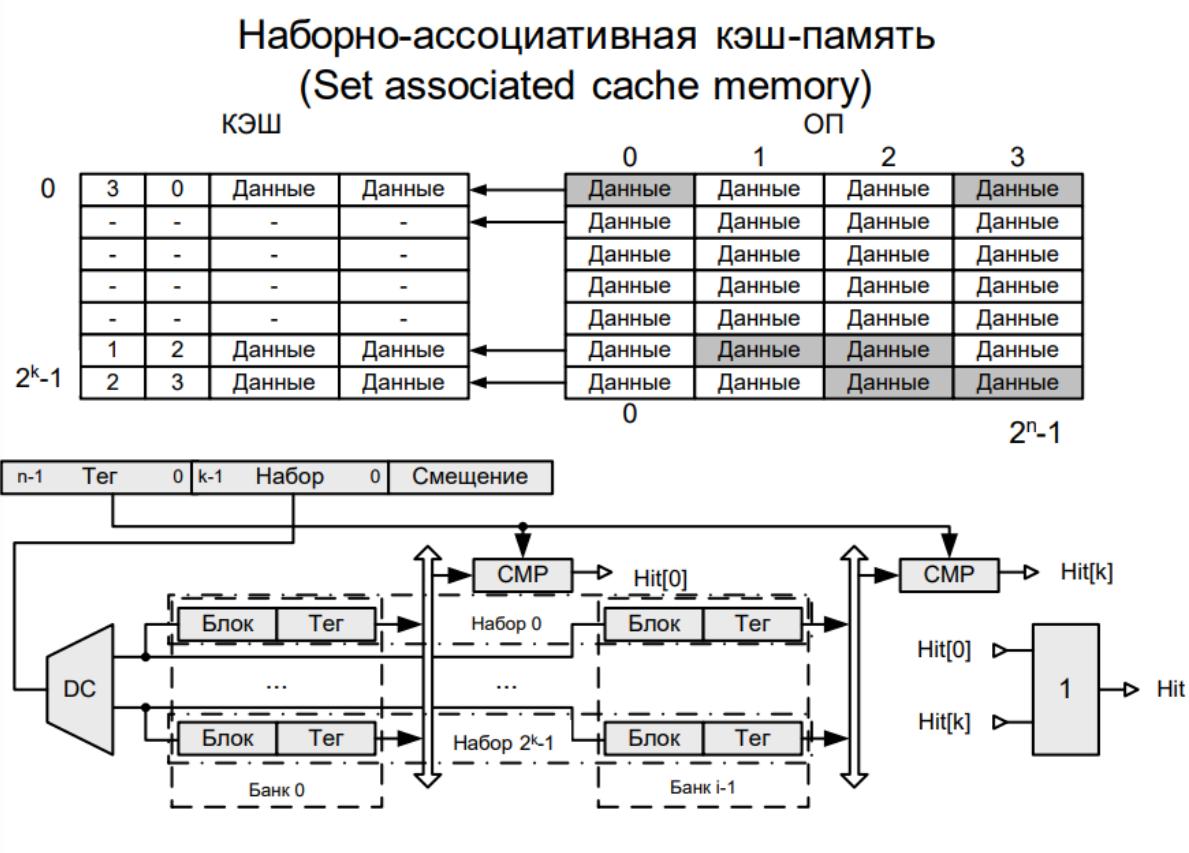
с одновременной записью в память тегов номера колонки, из которой был скопирован блок.

Конфликт максимально частый: при обращении к блокам данных в одной кэш-линейке. Жесткое закрепление за определенными блоками ОП одного блока в КП. 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.

Преимущество: аппаратно простой способ, так как нужен один компаратор, потому что нужно сравнить тег с тегом в запросе.

Недостаток: наихудшая представительность выборки (частые конфликты).

## 29. Done Способы отображения ОП в кэш: наборно-ассоциативный способ отображения



Идея следующая : взять и объединить несколько кэшей с прямым размещением, дав вариативность, в какой из них можно записать.

Банк, который указан на схеме, -- кэш с прямым размещением. Задается несколько таких банков параллельно, их количество, обычно, равно 4ем или 8ми (по сути это и есть степень ассоциативности). Фактически оказывается, что можно разместить данные в одном банке, а конфликтующие данные -- в следующем банке. Но можно это раскинуть и по 4ем банкам, тогда количество "конфликтов" будет сокращено в 4 раза и тд.

Отличительная особенность данного способа в использовании нескольких банков, но строка в этих банках четко определяется адресом.

Для любой ячейки памяти выделяется набор, в котором можно более-менее свободно выбрать одну из имеющихся кэш-линеек. То есть в двух ассоциативной памяти -- 2 линейки, в 4ех -- 4 линейки.

Наборно-ассоциативный способ отображения -- самый часто используемый способ из-за вариативности того, что можно его изменять -- делать больше наборов, больше банков, то есть один и тот же объем кэш памяти меняется и по вертикали, и по горизонтали.

9 минут из следующей лекции:

Желательно иметь кэш-память с гибкой структурой, с большим количеством вариантов размещения. Данный способ сочетает два предыдущих варианта и объединяет их достоинства:

- 1) Аппаратная простота - один компаратор на банк. Банк является кэшем прямого размещения. Берем несколько таких кэшей, объединяем их вместе в набор. Получается один дешифратор на все банки.
- 2) Ассоциативность обеспечивается за счет возможности выбора любого набора из них для хранения данных. По вертикали - прямое размещение, по горизонтали - произвольное размещение в рамках количества банков.

Каждая ячейка может быть размещена только в одном наборе, но в этом наборе есть вариативность размещения.

Как отслеживать то, какую ячейку хотим вытеснить, когда набор уже заполнен, а необходимо что-то добавить?

Приходится к этому набору добавлять некоторую информацию о состоянии каждой линейки. Т.е. необходимо хранить служебную информацию. Например:

- 1) Информация о востребованности линейки, чтобы иметь возможность построить алгоритм замещения.
- 2) Информация о модификации данных. Так как кэш-память может по-разному себя вести к данным, поступающим из процессора, ее можно по-разному построить. Можно сохранять данные внутри кэш-памяти, может передать их дальше или стереть.

### 30. Done Алгоритмы замещения и способы согласование ОП и кэш

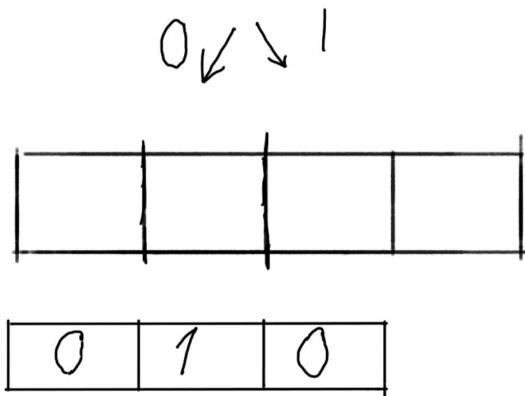
**Алгоритмы замещения** - позволяют найти наименее используемый, наиболее легкий блок памяти, чтобы заместить его и удерживать в кэш памяти те блоки, обращения к которым наиболее вероятны в ближайшем будущем.

Существуют следующие алгоритмы замещения:

- 1) Замещение немодифицированных данных - если среди линеек одного набора находится такая линейка, которую процессор не модифицировал, а только получил из памяти и считал, то такую линейку можно стереть. Это не повлечет за собой последствий, но даст ускорение при замещении. В этом случае любая операция замещения приводится к операции чтения из памяти.
- 2) Рандомизированный алгоритм - в этом методе замещение происходит независимо от использования того или иного набора линеек. В нём может применяться смена чисел табличным образом. По некой таблице составляется бесконечная последовательность псевдослучайных чисел. Обычно используется таблица чисел до 8, расположенных в матрице. В матрице по заданному числу в каждый раз получаем новое число. Недостаток этого алгоритма заключается в том, что он не предполагает учета того, к каким данным за чтением обращается процессор, не учитывает частоту использования.
- 3) Замещение наименее используемого (Least Recently Used, LRU) - этот алгоритм применяется наиболее широко. При реализации точного LRU приходится хранить информацию о том, насколько часто используется каждая линейка набора в двусвязном списке. Обращение к какой-либо линейке набора - это

показатель того, насколько эта линейка нужна. Такая линейка помещается в самый конец очереди и будет вытеснена последней. Таким образом, в начало очереди на замещение становится линейка, обращений к которой не было больше всего. Именно она будет замещена в первую очередь. В этом алгоритме задействуется двусвязный список в аппаратном исполнении.

Также можно реализовать псевдо-LRU алгоритм. В таком алгоритме используется дополнительный блок памяти, в битах которого указываются пути, по которым происходило обращение.



Рассмотрим следующую ситуацию:

Есть набор, состоящий из 4-х линеек. Мы должны постоянно сохранять информацию о том, куда обращались. Для этого используем дополнительный блок памяти из 3 битов.

Пусть обращение за чтением приводит к поиску в наборе (сравнение с тегами). Если предыдущее обращение было выполнено в левую половину (две младшие линейки набора), то это кодируется нулем в младшем бите дополнительного блока (если обращение было в правую часть, то произошло бы кодирование единицей). Центральный бит дополнительного блока отвечает за то, в какую правую или левую четверть левой половины было обращение (0 - левая четверть, 1 - правая). Если имело место обращение к левой четверти правой половины набора, то в старший бит дополнительного блока записывается 0. Таким образом, с помощью трёх битов можно хранить информацию о пути в левую или правую часть и в левую или правую четверть. Если в прошлый раз обращение было в правую половину, то наименее используемой становится левая половина (с четвертями - аналогично). Таким кодом из трех разрядов можно закодировать каждую из линеек.

### Согласование ОП и кэш:

Согласование ОП и кэша определяет, как ведёт себя оперативная память в ответ на модификации в процессоре.

-Метод сквозной записи (Write True) - вся информация записывается в оперативной памяти и всегда поддерживается максимальное сегментное

состояние. При любом изменении значения аннулируются и память читается обратно. Кэш память постоянно придерживается режима записи в оперативную память. Любая запись значения из процессора доходит до оперативной памяти. При работе с данными, которыми владеет только процессор, использование этого метода приводит к сильным задержкам, так как постоянно происходит запись в память, которая занимает время. Тем не менее, такой режим позволяет быстро уведомить через память всю систему об изменениях.

-Метод сквозной записи с буферизацией (Write Combining) - модифицированные данные и сохраняются в кэш памяти, и поступают на выгрузку в оперативную память. Разница с первым методом заключается в том, что процессор первым знает о модификациях и имеет локальную копию данных, которая никуда не отправляется. Запись в оперативную память и уведомление всей системе произойдет позже.

-Метод обратной записи (Write Back) - сначала модифицируется значение в кэш, а запись в оперативную память происходит только по необходимости. Таким образом, в этом методе работает предположение об "эксклюзивности" данных. Данные хранятся в кэш памяти и будут загружены в оперативную память только в том случае, если потребуется выгрузить их оттуда.

На практике в основном используются методы Write True или Write Back.

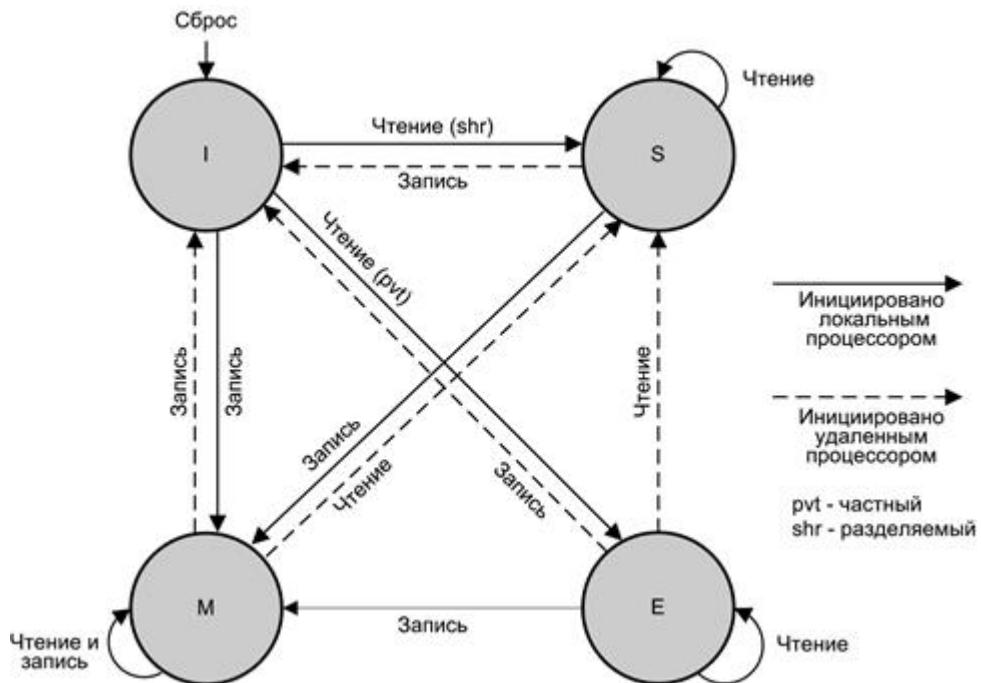
### 31. Done Протокол MESI

Протокол MESI широко используется в коммерческих микропроцессорных системах, например, на базе микропроцессоров Pentium и PowerPC.

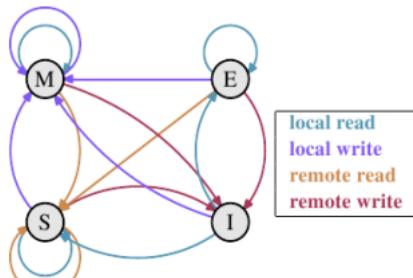
Протокол был разработан для кэш-памяти с обратной записью. Одной из основных задач протокола MESI является откладывание на максимально возможный срок операции обратной записи кэшированных данных в основную память системы. Это позволяет улучшить производительность системы за счет минимизации ненужных пересылок информации между кэшами и основной памятью.

Протокол MESI приписывает каждой кэш-строке одно из четырех состояний, которые контролируются двумя битами состояния MESI в теге данной строки. Состояние кэш-строки может быть изменено как процессором, для которого эта кэш-память является локальной, так и другими процессорами мультипроцессорной системы. Управление состоянием кэш-строк может быть возложено и на внешние логические устройства. Одна из версий протокола предусматривает использование ранее рассмотренной схемы однократной записи.

Каждая строка согласно протоколу MESI может быть в одном из четырех возможных состояний (в дальнейшем будем ссылаться на эти состояния с помощью букв М, Е, С и I):



### Протокол MESI



**Modified**  
**Exclusive**  
**Shared**  
**Invalid**

- Признак несогласованных данных.
- Признак согласованных данных.
- Признак согласованных данных в ВС.
- Признак отсутствия данных.

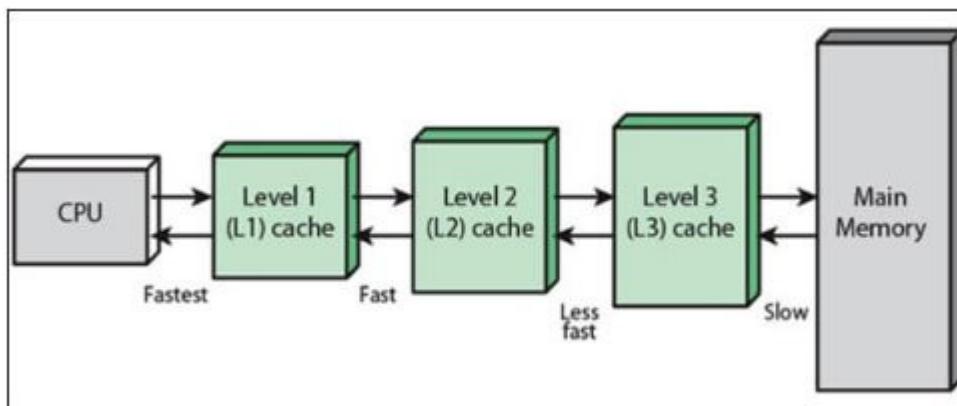
\* - <http://lnn.net/Articles/252125/>

## 32. Done Разделение кэш-памяти

Кэш процессора разделен на три основных уровня: L1, L2 и L3. Отличаются они скоростью доступа и размером.

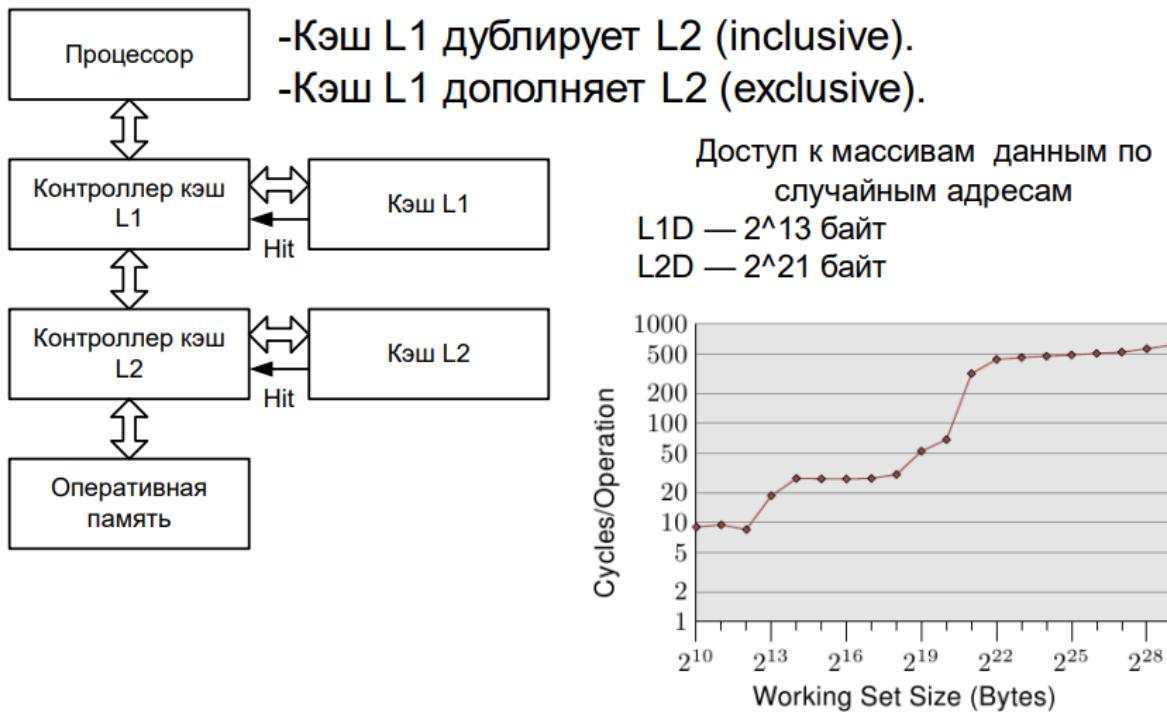
- **Кэш L1** (уровень 1) — это самая быстрая память которая присутствует в компьютере. С точки зрения приоритета, L1 содержит данные и команды, которые понадобятся в первую очередь. Размер обычно достигает 256 КБ, хотя некоторые топовые процессоры (типа Intel Xeon) могут иметь более 1 МБ.

- Кэш L2 (уровень 2) медленнее, но больше по размеру. Объем в диапазоне от 256 КБ до 8 МБ. Содержит данные, которые также могут скоро потребоваться, но не уместились в L1. Память первых двух уровней встроена прямо в ядро процессора. То есть, у каждого ядра она своя.
- Кэш L3 (уровень 3) — самая медленная из них, но и самая большая. Размер может достигать 62 МБ. Физически располагается внутри кристалла процессора, что позволяет обращаться к её содержимому намного быстрее, чем к ячейкам оперативной памяти.



Когда процессор ищет данные для выполнения операции, он последовательно начинает просматривать все уровни, начиная с L1 и заканчивая L3. Если поиск завершился неудачей, то приходиться обращаться к оперативной памяти, а это вызывает задержку в работе. Поэтому, чем объемней кэш, тем больше вероятность нахождения в нем нужных данных, а значит меньше задержек.

## Разделение кэш-памяти



### 33. Доделать Виртуальная память: назначение и преимущества

**Виртуальная память** – распространенная стратегия распределения памяти, используемая во всех современных операционных системах, основанная на идеи расширения физической памяти путем размещения расширенной памяти на диске и использования таблиц страниц (или сегментов) для трансляции адресов.

ТУТ ЛЕГЧЕ ВСТАВИТЬ ЛЕКЦИЮ РЯЗАНОВОЙ

Виртуальная память (из лекций Попова)

Механизм виртуализации адресного пространства позволяет:

- Увеличить объем адресуемой памяти.
- Использовать физическую память различного объема.
- Возложить на аппаратную составляющую механизмы доступа к ВЗУ
- Сгладить разрыв в производительности ОП и ВЗУ.
- Ускоряет доступ к данным по последовательным адресам.
- Способствует реализации защиты памяти.

Виртуальные системы строятся по трем принципам:

- Системы с блоками различного размера (сегментная организация).
- Системы с блоками одинакового размера (страничная организация).
- Смешанные системы (сегментно-страничная организация).

## 34. Страницчная организация виртуальной памяти

Виртуальная память является подкачкой (дополнением) оперативной памяти. Она присутствует практически во всех операционных системах.

Механизм виртуализации адресного пространства позволяет:

- Увеличить объем адресуемой памяти
- Использовать физическую память различного объема
- Возложить на аппаратную составляющую механизмы доступа к ВЗУ
- Сгладить разрыв в производительности ОП и ВЗУ
- Ускоряет доступ к данным по последовательным адресам
- Способствует реализации защиты памяти

Виртуальные системы строятся по трем принципам:

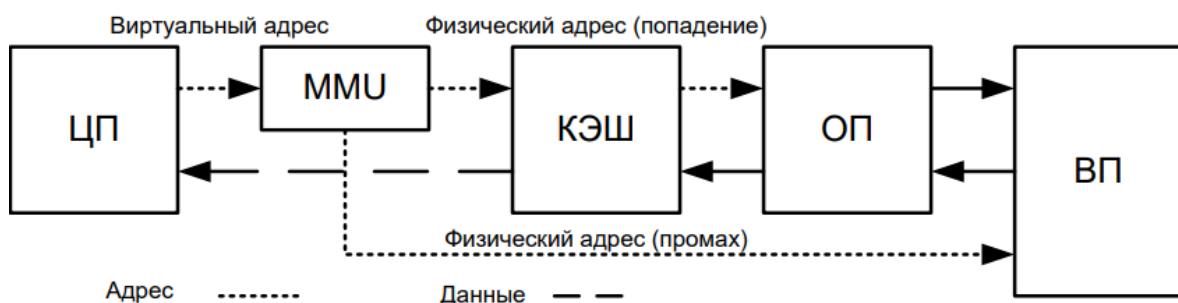
- Системы с блоками одинакового размера (страницчная организация)
- Системы с блоками различного размера (сегментная организация, опирается на механизм сегментации)
- Смешанные системы (сегментно-страницчная организация) – используются оба механизма вместе, что дает некий дополнительный выигрыш

Программа, когда попадает в память, разделяется на равные блоки – страницы. Суть страницочной организации в том, что все виртуализированное адресное пространство разделено на одинаковые блоки. Преобразование логического адреса в физический осуществляется с помощью таблицы страниц. Это преобразование отведено под обработку специального блока – MMU aka Memory Manage Unit (Устройство Управления Памятью), который определяет, находится ли страница в физической памяти. Данный блок принадлежит процессору (находится внутри него).

## Страницчная организация

Программа отображается в память равными блоками – страницами. Преобразование логического адреса в физический осуществляется с помощью таблицы страниц.

Преобразование логического адреса в физический реализуется в устройстве управления памятью (Memory Manage Unit), который определяет, находится ли страница в физической памяти (попадение).



Лучший случай (90% всех случаев):

ЦП получает команду, в которой числится запрос к памяти (она содержит адрес). Его задача передать этот адрес (виртуальный) на загрузку. Этот адрес попадает в ММУ. Во внутреннем кэше ММУ есть информация о преобразовании данного адреса, и он выполняет это преобразование (виртуального адреса в физический). Сформированный физ. адрес передается в КЭШ, дальше он идет в ОП, откуда мы его извлекаем и через КЭШ передаем данные в ЦП.

Худший случай (каждое 512 обращение – промах):

ЦП виртуальный адрес попадает в ММУ. ММУ, получив этот адрес, должно понять, где он находится (во внутреннем кэше нет информации о преобразовании этого адреса), для этого ММУ пользуется таблицей страниц (местоположения страниц + атрибуты). ММУ требует загрузки из ОП части таблицы страниц.

\* КЭШ хранит результаты ранее проделанных преобразований виртуального адреса в физический, поэтому чаще всего MMU не приходится обращаться в ОП, но иногда все же приходится и у нас как раз этот случай.

Минуя КЭШ память MMU обращается в ОП за таблицей страниц, теперь MMU понимает, где находится та страница, к которой относится запрашиваемый адрес.

То есть в худшем случае эта информация за пределами физической памяти, получаем промах, который вызывает прерывание Page Fault (страница отсутствует и за ней надо обращаться во ВН)

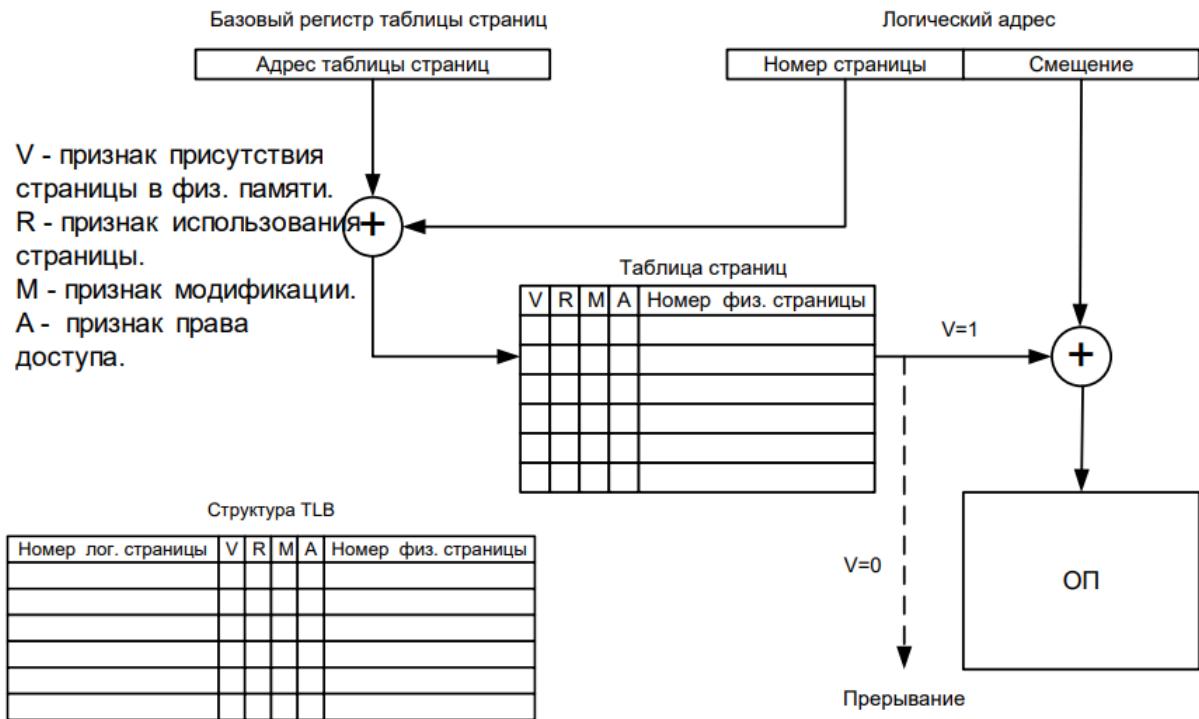
Берем кусок ВП (страницу), загружаем ее в ОП, как следствие меняется таблица страниц, поэтому то, что в ОП, выгружаем в ВП (две стрелки туда обратно – это DMA (aka Direct Memory Access, прямой обмен между ОП и ВП, без ЦП).

После завершения обработчика прерывания говорим MMU, что информация, которая вызвала прерывание, появилась в физической памяти и можно повторить чтение таблицы страниц (потому что старая инфа неактуальная, произошли некие изменения).

MMU загружает из ОП новую версию таблицы страниц и определяет, что нужная страница теперь в физической памяти. MMU формирует физический адрес, дает его в КЭШ, но КЭШ не может у себя найти этот адрес, т.к. он новый и адрес идет дальше в ОП, откуда мы его извлекаем и через КЭШ данные идут в ЦП.

Схема страничного преобразования:

## Схема страничного преобразования



Архитектура ЭВМ

ИУ6

86

В MMU поступает логический адрес, он делится на 2 части – номер страницы и смещение. Номер страницы складывается с начальным адресом таблицы страниц, и в результате мы получаем строку в этой таблице, по которой можем прочесть нужную информацию о том, где хранится данная страница (физический адрес).

Физический адрес складывается со смещением (смещение внутри страницы – это всегда физ. адрес) и после этого можно обращаться в ОП для получения данных.

Флаги:

**V** – признак присутствия страницы в физической памяти (самый главный флаг, т.к. строка может как быть в физической памяти, так и отсутствовать)

**V = 1** – строка присутствует в физ. памяти, **V = 0** – прерывание Page Fault

**R** – признак использования страницы, **M** – признак модификации; эти флаги нужны для замещения информации

А – признак прав доступа

TLB (Translation Lookaside Buffer aka Буфер Быстрого Страницочного Преобразования) – КЭШ память внутри MMU, которая хранит по номерам логических страниц то же, что и таблица страниц. После выполнения логического адреса в физический результат сохраняется во внутренней памяти TLB.

Достоинства страницочной организации – нет фрагментации (сколько бы ни было свободных страниц и какие бы приложения в каком объеме нам не надо было бы разместить, если в памяти есть хотя бы необходимое количество страниц и где бы они не находились, мы всегда сможем его разместить)

Недостаток – мы не можем локализовать информацию необходимую для работы конкретного приложения (страницы приложения хаотично разбросаны по таблице)

### 35. Сегментная организация виртуальной памяти

Виртуальная память является подкачкой (дополнением) оперативной памяти. Она присутствует практически во всех операционных системах.

Механизм виртуализации адресного пространства позволяет:

- Увеличить объем адресуемой памяти
- Использовать физическую память различного объема
- Возложить на аппаратную составляющую механизмы доступа к ВЗУ
- Сгладить разрыв в производительности ОП и ВЗУ
- Ускоряет доступ к данным по последовательным адресам
- Способствует реализации защиты памяти

Виртуальные системы строятся по трем принципам:

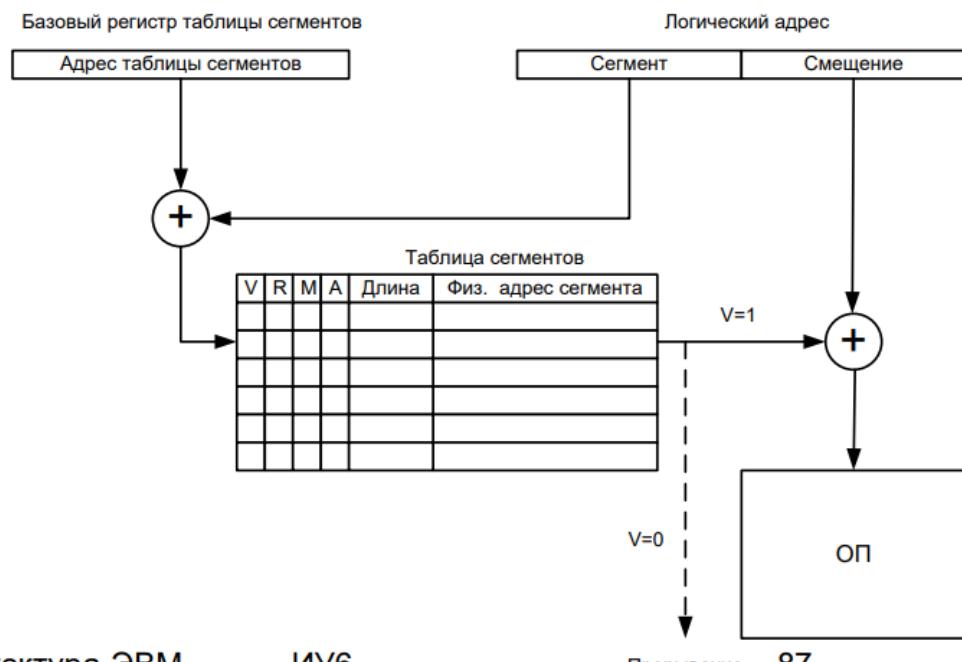
- Системы с блоками различного размера (сегментная организация)
- Системы с блоками одинакового размера (страничная организация)
- Смешанные системы (сегментно-страничная организация)

Сегментная организация опирается на естественное отображение программы в сегменты (кода/стека/данных).

Программа отображается в память блоками различного размера – сегментами. Преобразование логического адреса в физический осуществляется с помощью таблицы сегментов. Это преобразование отведено под обработку специального блока – MMU aka Memory Manage Unit (Устройство Управления Памятью).

## Сегментная организация

Программа отображается в память блоками различного размера – сегментами. Преобразование логического адреса в физический осуществляется с помощью таблицы сегментов.



Логический адрес делится на две части – сегмент и смещение. Номер сегмента складываем с начальным адресом таблицы сегментов и в результате получаем строку в таблице сегментов, флаг V и остальные

флаги. Получив физический адрес сегмента, складываем его со смещением и можем обратиться в ОП для чтения/записи данных.

\* Тут в отличии от страничной организации в таблице сегментов есть еще длина сегмента (мы получим ошибку, если выйдем за границы сегмента)

Флаги:

V – признак присутствия сегмента в физической памяти (самый главный флаг, т.к. строка может как быть в физической памяти, так и отсутствовать)

V = 1 – строка присутствует в физ. памяти, V = 0 – прерывание Page Fault

R – признак использования сегмента, M – признак модификации; эти флаги нужны для замещения информации

A – признак прав доступа

Достоинство сегментной организации – локализация информации о сегментах

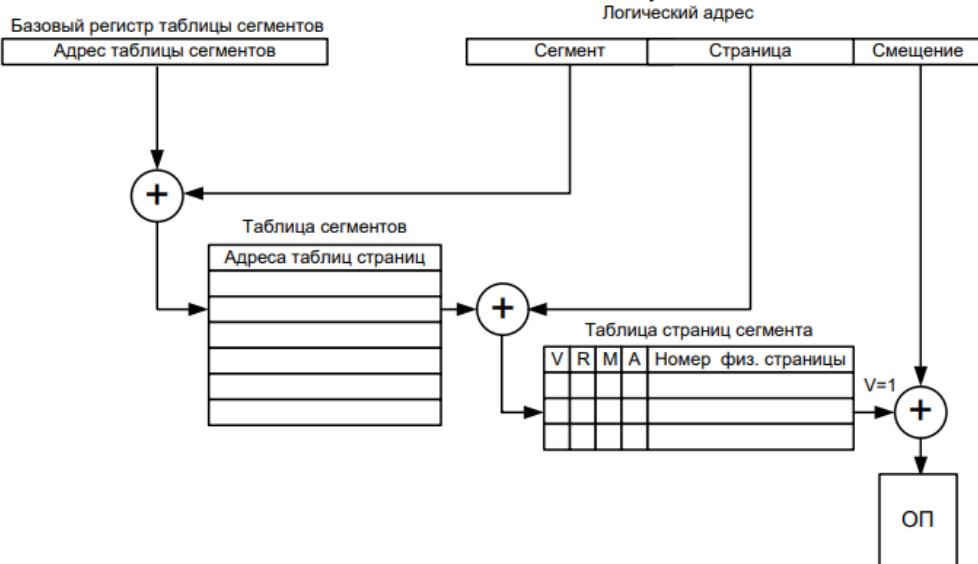
Недостаток – большая фрагментация (чтобы разместить сегмент в памяти, нужен непрерывный кусок памяти => память быстро заполнится такими кусками и несмотря на то, что свободное место будет, мы не сможем его занять)

### 36. Done Сегментно-страничная организация виртуальной памяти

## Сегментно-страничная организация памяти

Программа отображается в память блоками различного размера – сегментами, каждый из которых целое число страниц.

Преобразование логического адреса в физический осуществляется с помощью таблицы сегментов и таблицы страниц сегмента.



Возникает такая потребность из-за того, что ни один из 2х предыдущих случаев не имеет свойств: фрагментации(страничная) и локализации(сегментная).

Для этого сочетаем 2 способа с недостатками и получаем третий с достоинствами.

Если имеем память, то можем всю её задействовать.

Приложение формируется компилятором из сегментов. Сегмент должен быть разделен на страницы. Сегмент может состоять из любого количества страниц и про конкретный сегмент мы должны знать, какие страницы в него входят и где они находятся. Выходит, что у нас получится 2 уровня таблиц. Таблицы 1 уровня - таблицы сегментов, пусть она хранит сегменты всех приложений(общая - все загруженные сегменты), либо таблицу сегментов конкретного приложения(локальную). К таблице сегмента прибавляем номер сегмента, получаем смещение сегмента. Но в этой таблице хранится адреса вторичной таблицы, которая уже хранит страницы этого сегмента. Т.е для каждого сегмента строится вторая таблица - локальная дескрипторная таблица - в которой хранятся записи страниц этого сегмента(сегмент, его размер выравненный до 4 КБ в большую сторону: если 5КБ то будет 8, если 9, то 12КБ и тп) и такое кол-во страниц выделяется.

Однако останется небольшой "хвост", какая-то последняя страница у сегмента не будет занята - ничего страшного, но нам нужно этот сегмент как-то вычислять для безопасности. Получается, если мы загружаем приложение, то его кол-во сегментов небольшое и как только мы обратимся к какой-то локальной таблице страниц, мы загрузим страницы этого сегмента вместе - пакетный режим памяти позволит лучше запомнить нашу память, информацию о страницах конкретного приложения.

И проблема с падением производительности будет меньше, из-за того, что локализована информация о размещении физ. страниц.

Фрагментация - т.к сегменты отображаются в какое-то кол-во страниц -> если есть какое-то место для этих страниц разделенной физ. памяти на куски, мы можем это сделать и нам неважно иметь непрерывную память.

Помимо этого, делают большее кол-во блоков(называют регионы), помимо сегментно-страничного, бывают регионы из сегментов и страниц - возникает ещё одна таблица. Зачем? Представим виртуализацию - окружение. Выделяется память под вирт. машины, которая находится в разных регионах, мы их не будем путать, будем обращаться к ним как к единому целому, т.к они будут находиться в разных регионах. Используется в VM, адресных пространствах контейнеров, монолитных приложениях СУБД(Oracle), т.к разумно выделять в отдельные регионы, в которых они будут функционировать и смена контекста, которая вызывает провал производительности, будет реже, нежели в случае с сегментно-страничной организацией + защита памяти.

### 37. Done но можно и дописать Общие принципы построения современных ЭВМ.

Основным принципом построения всех современных ЭВМ является программное управление. В основе его лежит представление алгоритма решения любой задачи в виде программы вычислений. Стандартом для построения практических всех ЭВМ стал способ, описанный Дж. фон Нейманом в 1945 г. при построении еще первых образцов ЭВМ. Суть его заключается в следующем.

Все вычисления, предписанные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов-команд. Каждая команда содержит указания на конкретную выполняемую операцию, место нахождения operandов (адреса operandов) и ряд служебных признаков. Operandы - переменные, значения которых участвуют в операциях преобразования данных. Список (массив) всех переменных (входных данных, промежуточных значений и результатов вычислений) является еще одним неотъемлемым элементом любой программы.

Для доступа к программам, командам и operandам используются их адреса. В качестве адресов выступают номера ячеек памяти ЭВМ, предназначенных для хранения объектов. Различные типы объектов, размещенные в памяти ЭВМ, идентифицируются по контексту.

Последовательность битов в формате, имеющая определенный смысл, называется полем. Например, в каждой команде программы различают поле кода операций, поле адресов operandов. Применительно к

числовой информации выделяют знаковые разряды, поле значащих разрядов чисел, старшие и младшие разряды.

Последовательность, состоящая из определенного принятого для данной ЭВМ числа байтов, называется *словом*.

## Структура современных ЭВМ

- Центральное процессорное устройство (ЦПУ).
  - Арифметико-логическое устройство (АЛУ)
  - Устройство управления (УУ)
  - Регистры общего назначения (РОН)
- Основная память
- Система ввода-вывода
- Управление внешними устройствами
- Внешняя память
- Система передачи информации
- Система синхронизации
- Система прерываний
- Система прямого доступа к памяти
- Система подвода питания/земли и система энергосбережения
- Система контроля и повышения отказоустойчивости

# IX. Принципы построения и архитектура ЭВМ

## Общие принципы построения современных ЭВМ

### Принципы Фон-Неймана

- Двоичное кодирование информации
- Программное управление
- Адресность памяти
- Однородность памяти

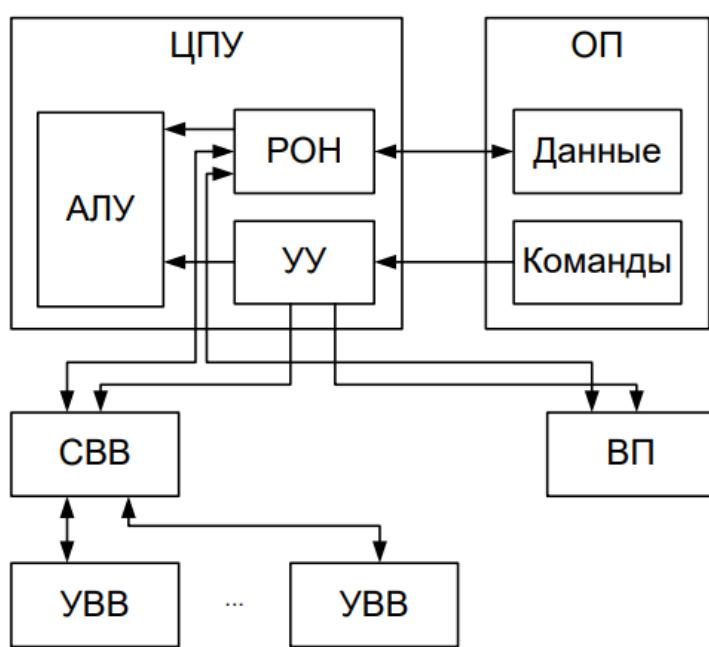
ОКОД, SISD



-Гарвардская архитектура  
(ОП для хранения команд и ОП для хранения данных)  
Принстонская архитектура  
(ОП для хранения команд и данных)

### 38. Done ЭВМ с непосредственными связями и магистральной структурой. Основные тенденции развития ЭВМ

#### ЭВМ с непосредственными связями

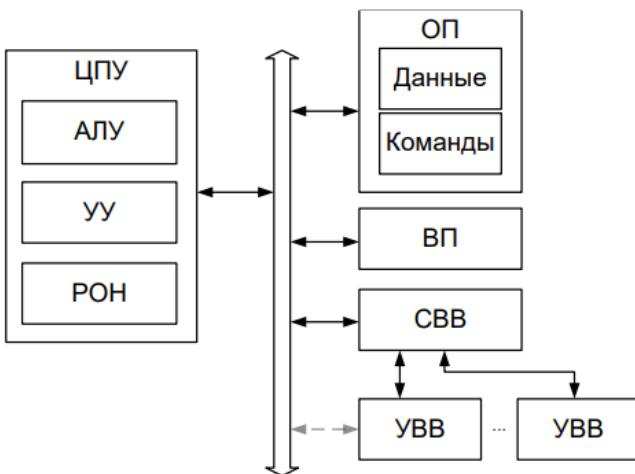


- (+) При построении оптимальных линий связи вычислительная машина обладает максимальным быстродействием.
- (-) Ограничение на количество выводов микросхем не позволяет организовать широкие шины.
- (-) Реконфигурация системы требует изменения характеристик линий связи.

#### Особенности структуры

Каждое из устройств связано собственным интерфейсом с ядром ЭВМ, имеет собственную разрядность шин: ШД (шина данных) ША (шина адреса) и ШУ (шина управления), т. е. Плюсы: структура наиболее оптимальна и недорога.

## ЭВМ с магистральной структурой



(+) Общая шина позволяет легко реконфигурировать систему.

(-) Шина является узким местом.

- Шина, используемая всеми устройствами системы для передачи данных называется системной.
- Для разгрузки системной шины используют иерархию шин.
- По назначению, разделяют шины адреса, шины данных и шины управления.

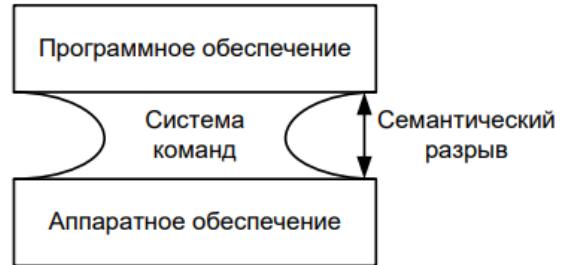
Магистрально-модульная организация вычислительной системы предполагает выделение общего универсального канала (магистрали связи между элементами системы – модулями и определения общих правил взаимодействия).

В центре ВС – центральный процессор (ЦП), управляющий информационной связью между устройствами, подключенными к магистрали (внешние устройства (ВУ) и память).

Магистраль, называемая также Общей шиной (ОШ), представляет собой множество проводов. По одной группе проводов (шина данных) передается обрабатываемая информация, по другой (шина адреса) – адреса памяти или внешних устройств, к которым обращается процессор. Есть еще третья часть магистрали – шина управления, по ней передаются управляющие сигналы (например, сигнал готовности устройства к работе, сигнал запуска операции в устройстве и др.).

## Основные тенденции развития ЭВМ

- Повышение степени интеграции элементной базы
  - Увеличение набора команд
  - Увеличение степени аппаратной поддержки.
- Обратная совместимость
- Наличие семантического разрыва



### Проблема семантического разрыва

Технология программирования непрерывно развивается, что позволяет увеличивать функциональность программ и сокращать время их разработки. Создание проблемно-ориентированных языков высокого уровня усугубляет принципиальное отличие языка машинных команд, реализуемого компьютером, от языков, используемых при написании программ. Данная проблема носит название "семантического разрыва" и выражается в неоправданном падении производительности вычислительной системы.

из инета:

#### 5.7.1. Повышение тактовой частоты

Для повышения тактовой частоты при выбранных материалах используются:

- более совершенный технологический процесс с меньшими проектными нормами;
- увеличение числа слоев металлизации;
- более совершенная схемотехника меньшей каскадности и с более совершенными транзисторами,
- более плотная компоновка функциональных блоков кристалла.

#### 5.7.2. Увеличение объема и пропускной способности подсистемы памяти

Возможные решения по увеличению пропускной способности подсистемы памяти включают:

- создание кэш-памяти одного или нескольких уровней,
- увеличение пропускной способности интерфейсов между процессором и кэш-памятью и конфликтующей с этим увеличением пропускной способности между процессором и основной памятью.

### 5.7.3. Увеличение количества параллельно работающих вычислительных устройств

Каждое семейство микропроцессоров демонстрирует в следующем поколении увеличение числа функциональных исполнительных устройств и улучшение их характеристик, как временных (сокращение числа ступеней конвейера и уменьшение длительности каждой ступени), так и функциональных (введение MMX-расширений системы команд и т.д.).

### 5.7.4. Системы на одном кристалле и новые технологии

### 5.7.5. Нанотехнологии

## 39. Done RISC, CISC, VLIW архитектура

### CISC

Первоначально почти все производители первых микропроцессоров использовали архитектуру с расширенным набором команд – CISC (Complex Instruction Set Computer). Причина этого в том, что разработчики пытались уменьшить так называемый семантический разрыв между тем, что компьютеры способны делать, и тем, что требуют языки программирования высокого уровня, пытаясь заменить одной инструкцией многочисленные машинные коды. Также в то время на рынке коммерческих вычислений доминировали мини-компьютеры PDP компании DEC и мейнфреймы компании IBM, которые были основаны на архитектуре CISC. Среди микропроцессоров типичными представителями данной архитектуры стали процессоры компании Intel.

Для архитектуры CISC характерно:

- 1) малое количество регистров общего назначения;
- 2) большое количество различных машинных команд, каждая из которых выполняется за несколько тактов процессора;
- 3) различные форматы команд с разной длиной;
- 4) преобладание двухадресной системы команд;

5) развитой механизм адресации operandов.

### RISC

Повышение производительности CISC-микропроцессоров из-за особенностей архитектуры приводило к росту количества транзисторов, в результате чего кристаллы становились всё более сложными и дорогостоящими в производстве. Вопросы закрывались конструктивно-технологическими решениями, но в конечном итоге по экономическим соображениям уже не давали адекватного роста производительности. CISC-архитектура в первозданном своём виде достигла потолка производительности. Применение конвейера для повышения производительности требовало использования простых и быстрых команд. Необходимость дальнейшего роста производительности привела к использованию архитектуры RISC (Reduced Instruction Set Computer), что означает «компьютер с сокращённым набором команд»

Сравнение архитектур CISC и RISC

	CISC	RISC
Набор команд	Расширенный	Сокращённый
Время выполнения команд	Различное	Команда за такт
Программный код	Короткий	Длинный
Использование конвейера	Плохое	Отличное
Реализация процессора	Очень сложная	Простая
Модели процессоров	IBM 360/370, DEC PDP/VAX, Intel, AMD, Motorola	SPARC, PowerPC, ARM, MIPS, МЦСТ R

### Основные особенности RISC-процессоров

- Сокращённый набор команд.
- Большинство команд выполняется за один такт.
- Большое количество регистров общего назначения.
- Наличие жёстких многоступенчатых конвейеров.
- Все команды имеют простой формат, и используются немногие способы адресации
- Наличие вместительной раздельной кэш-памяти
- Использование оптимизирующих компиляторов, которые анализируют исходный код и частично меняют порядок следования команд

### VLIW

Архитектура VLIW (Very Long Instruction Word – сверхдлинное командное слово) основывается на явно выраженнем параллелизме вычислений, заложенном в систему команд процессора, – EPIC (Explicitly Parallel Instruction Computing – вычисления с явным параллелизмом команд). Основной принцип организации этой архитектуры сводится к тому, чтобы перенести нагрузку с периода исполнения в период компиляции. Суперскалярный процессор в ходе исполнения переупорядочивает команды, подменяет регистры, распределяет функциональные блоки и выполняет множество других функций, что ведёт к максимальной загрузке аппаратных ресурсов. В архитектуре VLIW эти задачи заранее решает компилятор, который располагает полной и достоверной информацией о регистрах процессора и генерирует оптимальный код, в котором нет конфликтов между регистрами. Кроме того, компилятор следит за загрузкой функциональных блоков и не запускает команды, в которых предполагается обращение к занятым функциональным блокам.

Итогом такого подхода, с одной стороны, являются следующие преимущества архитектуры

1. Прежде всего, это более тщательное планирование выполнения программы и оптимизация кода, что даёт лучшее заполнение исполнительных устройств и выполнение большего количества операций за такт.
2. В связи с переносом функциональности на компилятор в процессоре меньше места тратится на функциональные блоки, отвечающие за управление, и больше площади остаётся непосредственно на вычислительные ресурсы, такие как регистры, исполнительные устройства и кэш-память.
3. Это, в свою очередь, приводит к упрощению конструкции процессора и технологического процесса его производства, уменьшению количества транзисторов и снижению тепловыделения.

Сравнение CISC, RISC и VLIW архитектур СК

Характеристика	CISC	RISC	VLIW
Длина команды	Различная	Одинарная	Одинарная
Расположение полей в командах	Различное	Однотипное	Однотипное
Количество регистров	Малое. Регистры специализированные	Большое. Регистры универсальные	Большое. Регистры универсальные
Доступ к памяти	Кодируется в команде. Выполняется по микрокоманде	Выполняется по специальной команде	Выполняется по специальной команде
Длительность выполнения команд	Различная	Одинарная (для большинства команд)	Различная

#### 40. Done Назначение и обобщенная структура процессорного устройства. Микропроцессор. Классификация микропроцессорных СБИС

Процессор или процессорное ядро - устройство ЭВМ, непосредственно осуществляющее процесс переработки информации и управления им в соответствии с заданным алгоритмом, который, как правило, представлен программой.

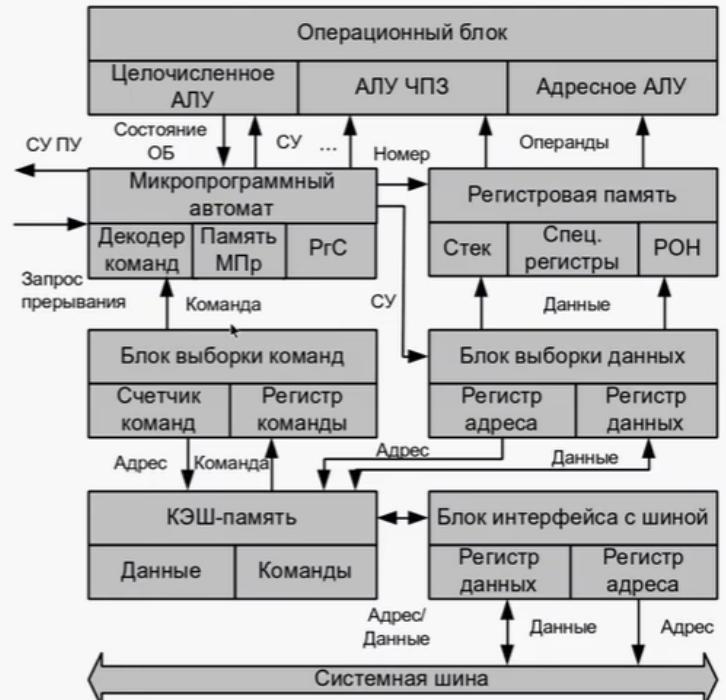
ЭВМ может содержать несколько процессоров. Процессор, управляющий вычислительным процессом, называется центральным.

Процессор осуществляет управление взаимодействием всех устройств ЭВМ.

Процессор более широкое понятие. Внутри процессора может присутствовать многоядерность. Сейчас все процессоры являются микро.

### Обобщенная структура универсального процессорного устройства

- Архитектурные особенности:**
- Конвейерное исполнение команд.
  - Внутренняя КЭШ-память.
  - Целочисленное АЛУ.
  - Устройство выполнения операций над числами с плавающей запятой.
  - Обработка прерываний от ПУ.
  - Поддержка мультипроцессорной обработки.



### Описание структуры.

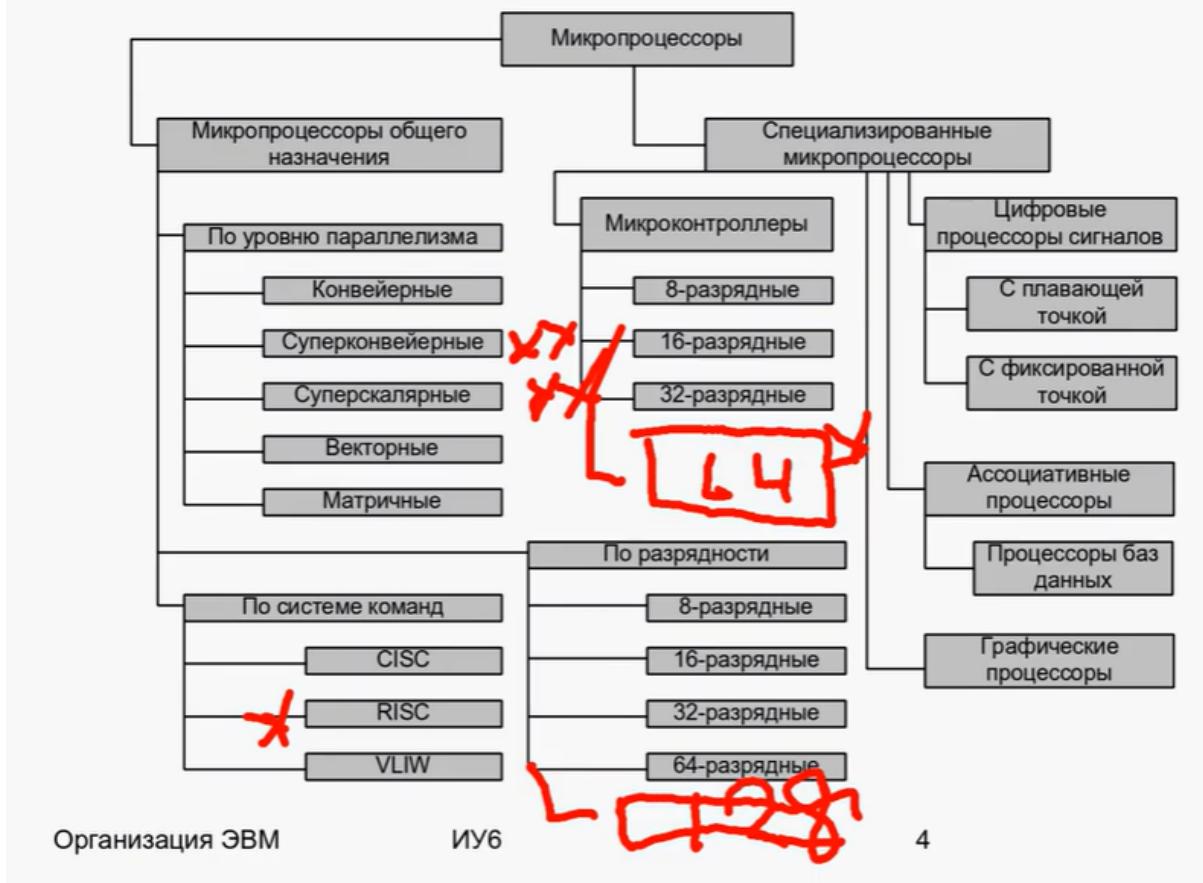
Процессор не имеет внутренней памяти для хранения команд и данных. Должен быть сформирован адрес команды, и адрес должен дойти до шины. Нужно сформировать в регистре адреса адрес обращения на загрузку и ждать ответа памяти. Из памяти обратно присыпаются данные, они записываются в регистр данных. Далее, эти данные должны поступить во внутреннюю ассоциативную КЭШ-память. Данные должны как можно быстрее загрузиться в РОН. А вот команды нужно последовательно разложить на микрокоманды и исполнить. Поэтому конвейер команд более длинный. В функцию блока выборки команд входит вычисление адреса команды. В счетчик команд заносится адрес. Счетчик команд в следующем такте отправляется вниз(в КЭШ), чтобы загрузить команды. Если команд нет в КЭШ-памяти, то запрос транслируется дальше в блок интерфейса. Когда блок выборки команд получает ответ от КЭШ-памяти, он получает команду или группу команд(т.е. блок из памяти может содержать больше 1 команды). Блок команд передает команду на декодирование. В 1 инструкцию может быть включено от 1 до 3 микрокоманд.

На основе инструкции условно сформировать 3 микрокоманды(в разных случаях по разному). Чтобы сформировать эту цепочку действий нам нужен микропрограммный автомат: он выдает последовательность микрокоманд в ответ на входящую команду. Он анализирует флаги, которые используются в арифметических действиях, перед началом. И формирует микрокоманды на основе флагов и входной команды. Из микропрограммного автомата посыпается сигнал управления на блок выборки данных, и формируется регистр адреса на загрузку. Мы подготавливаем транзакцию на чтение или на запись. Она стартует через КЭШ-память. Сначала пытаемся найти в КЭШ, а потом отправляемся в блок интерфейса и это все там передается в память на шину. Блок выборки данных, получив данные от КЭШ-памяти отправляет их в регистр. И в регистровой памяти подготавливаются операнды для исполнения. Операнды отправляются в операционный блок на исполнение. Операционный блок выполняет то, что нужно. Получили результат. И обратный процесс сохранения результата: операнды идут обратно. И эти результаты либо сохраняются в регистрах, либо идут дальше через регистровую память. Потом в КЭШ-память, потом, если надо и в шинный интерфейс, и в память.

Микропроцессор - функционально законченное устройство, представляющее собой вариант процессора или нескольких процессорных ядер современной ЭВМ и реализованное в виде одной или нескольких СБИС.

Микропроцессорный комплект представляет собой совокупность микропроцессора и специализированных ИС, совместимых по временным, электрическим и конструктивным параметрам, совместное использование которых позволяет реализовать основные функции ЭВМ.

## Классификация процессорных устройств



### 41. Done Форматы команд. Типы команд.

Несмотря на различие в системах команд разных ЭВМ, некоторые основные типы операций могут быть найдены в любой из них. Для описания этих типов примем следующую классификацию:

- команды пересылки данных;
- команды арифметической и логической обработки;
- команды работы со строками;
- команды SIMD;
- команды преобразования;
- команды ввода/вывода;
- команды управления потоком команд.

**Форматом** команды называется заранее обговоренная структура полей в её кодах, позволяющая ЭВМ распознавать составные части кода.

Главным элементом кода команды является **код операции** (КОП), что определяет, какие действия будут выполнены по данной команде. Под него выделяется  $N$  старших разрядов формата.

Распределение полей в формате команды может изменяться при смене способа адресации. Длина команды зависит от числа адресных полей. По числу адресов команды делятся на:

- безадресные
- одно-, двух-, трехадресные

По функциональному назначению в системе команд ЭВМ различают следующие группы:

- команды передачи данных (обмен входами между регистрами процессора, процессора и оперативной памятью, процессора и периферийными установками).
- Команды обработки данных (команды сложения, умножения, сдвига, сравнения-).
- Команды передачи управления (команды безусловного и условного перехода).
- Команды дополнительные (типа RESET, TEST,-).

## Форматы команд.

Операционная часть	Адресная часть
--------------------	----------------

1. Четырехадресная команда.

КОП	1 операнд	2 операнд	результат	Адр след ком.
-----	-----------	-----------	-----------	---------------

2. Трехадресная команда

КОП	1 операнд	2 операнд	результат
-----	-----------	-----------	-----------

3. Двухадресная команда.

КОП	1 операнд	2 оп-д/результат	Характерна для CISC-архитектуры
-----	-----------	------------------	---------------------------------

4. Аккумуляторная архитектура

КОП	1 операнд	Второй операнд хранится в аккумуляторе. Данный формат команд характерен для RISC-архитектур.
-----	-----------	---

5. Нульоперандная команда.

КОП
-----

## 42. Done Стековая архитектура системы команд (основное вроде есть, мб что-то дополнить можно) by Maslova

Введение про архитектуры системы команд.

Системой команд вычислительной машины (ВМ) называют полный перечень команд, которые способна выполнять данная ВМ. Под архитектурой системы команд понимают средства вычислительной машины, которые видны и доступны программисту. АСК можно рассматривать как линию согласования нужд разработчиков программного обеспечения с возможностями создателей аппаратуры вычислительной машины (рис. 2.1). В конечном итоге цель тех и других — реализация вычислений наиболее эффективным образом, то есть за минимальное время, и здесь важнейшую роль играет правильный выбор архитектуры системы команд.

## С ЛЕКЦИИ (29.09.20, 00:01:03)

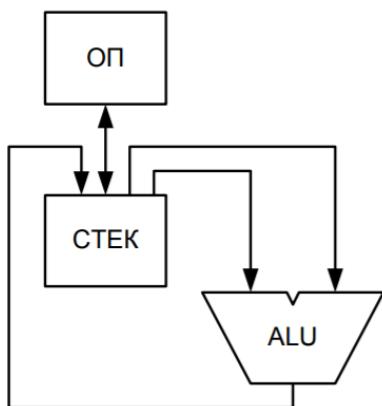
Все системы команд созданы для организации базового набора функционала (арифметического). Отличаются они только способом реализации (разная работа с памятью, разные способы адресации, размещения операндов и тп)

ТУТ УЖЕ ПРО СТЕКОВУЮ

Презентация 4 из 5-ого сема слайды 12

### Стековая архитектура СК

(+) При размещении операндов в стековой памяти (LIFO) архитектура команд упрощается (большое количество действий выполняется аппаратно)



#### Операции:

- занесение в стек (PUSH);
- извлечение из стека (POP);
- выполнение действий на стеком (извлечение операндов из вершины стека, выполнение действий, помещение результата в вершину стека)

Для выполнение арифметических операций их преобразуют к постфиксной форме (Польской записи).

Пример:  $a = a + b * (c - d)$ ; Постфиксная форма: abcd-\*+;

Действия: PUSH a; PUSH b; PUSH c; PUSH d; SUB; MUL; ADD; POP a.

- (-) Отсутствие прямого доступа к памяти ограничивает область применения.
- (-) Сложность организации параллельной обработки.

С параллельностью можно что-то сотворить (но лекция в слишком прохом качестве :( грусть-печаль)

Конец слов с лекции (там оооочень неразборчиво все)

Суть:

1. Операнды перед обработкой помещаются в две верхних ячейки стековой памяти.
2. Результат также заносится в стек (автоматически)

3. Математические выражения записываются в обратной польской нотации (подробнее в источнике ниже, на рисунке показательный пример, как это происходит; вычисляется выражение  $(a+b)(c+d)-e$

<i>push a</i>	<i>push b</i>	<i>add</i>	<i>push c</i>	<i>push d</i>	<i>add</i>	<i>mul</i>	<i>push e</i>	<i>sub</i>
<i>a</i>	<i>b</i>	<i>a+b</i>	<i>c</i>	<i>d</i>	<i>c+d</i>	$(a+b)*(c+d)$	<i>e</i>	$(a+b)*(c+d)-e$
	<i>a</i>		<i>a+b</i>	<i>c</i>	<i>a+b</i>		$(a+b)*(c+d)$	
				<i>a+b</i>				

Рис. 2.5. Последовательность вычисления выражения  $ab+cd+^*e-$  на вычислительной машине со стековой архитектурой

4. Таким образом, *push X* -- запись в стек из памяти или из АЛУ (чтение из ячейки, занесение в регистр данных, проталкивание в стек).
5. *pop X* -- сохранение вершины стека по адресу X
6. Для выполнения операции на вход АЛУ подается инфа из двух верхних ячеек стека (операнды удаляются из стека), а результат заносится в стек (или сразу же записывается в память)
7. Верхние ячейки стековой памяти быстродействующие => их размещают в процессоре, другие могут размещать либо в основной памяти, либо даже на магнитном диске.

Плюсы и минусы:

- + Сокращение адресной части команд
- + Компактный код
- + Простое декодирование команд
- Нет произвольного доступа к памяти
- Компилятору трудно создать эффективный код
- + Но создание компиляторов упрощается

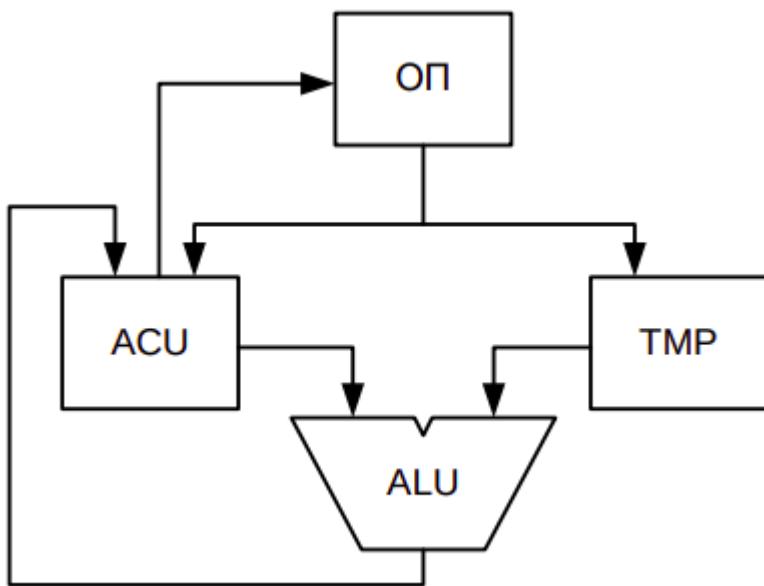
- Сложности при повышении производительности.

Слова не из лекции взяты отсюда: [ГЛАВА 2 Архитектура системы команд](#)

### 43. Done Аккумуляторная архитектура системы команд

Аккумуляторная архитектура использует выделенный специальный регистр - **аккумулятор**. Он используется для хранения одного из операндов арифметической или логической операции. Результат вычислений тоже заносится в аккумулятор.

В классической аккумуляторной архитектуре только один аккумулятор, его адрес понятен и известен, поэтому для команд обработки нужно указывать местоположение только второго операнда. Но перед этим первый operand нужно загрузить в аккумулятор. Второй operand загружается в теневой регистр TMP, после чего ALU может выполнять команду. Результат вычислений заносится в аккумулятор, откуда можно выгрузить в память полученное значение.



Соответствующие архитектуре операции:

1. Занесение в аккумулятор (**load**)
2. Извлечение из аккумулятора (**store**)
3. выполнение действий над операндами (извлечение первого операнда из аккумулятора, извлечение второго операнда из ОП и помещение во временный теневой регистр TMP, выполнение действий, помещение результата в аккумулятор).

У этого подхода есть несколько преимуществ:

1. Требуется указывать адрес только второго операнда, следовательно, команды становятся меньше, и при этом декомпозиция становится проще.

Пример: нужно вычислить  $a = a + b * (c - d)$

Тогда в соответствии с математическими правилами сначала вычисляется  $(c - d)$ , затем умножение, и затем сложение. Поэтому последовательность действий будет такая:

```
LOAD c;  
SUB d;  
MUL b;  
ADD a;  
STORE a;
```

2. Ускоряются длинные вычисления. Например:  $a/b*c/d$ . Так как нам постоянно использовать результат вычисления на предыдущем шаге, то промежуточные результаты не будут выгружаться из аккумулятора в оперативную память, отсюда и выигрыш.

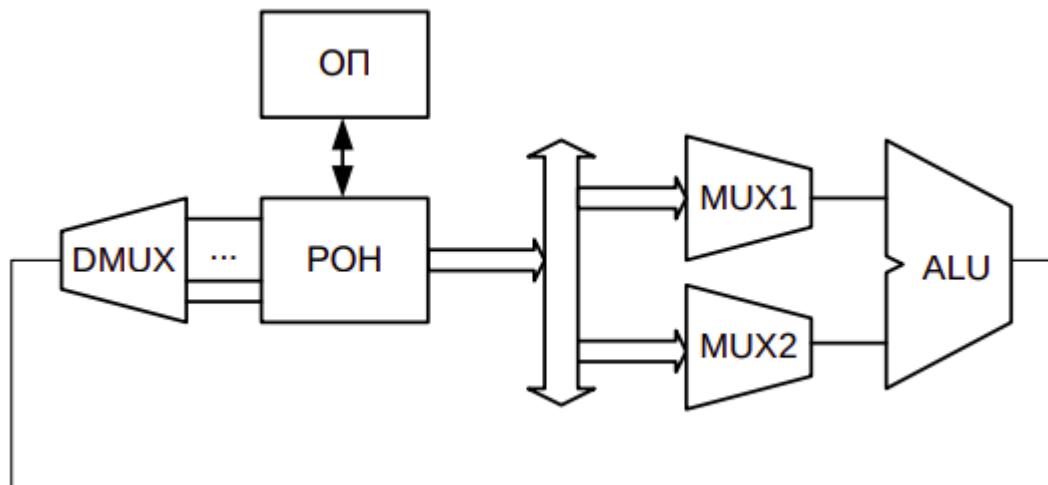
Но есть несколько недостатков такой архитектуры:

1. Если результат временно не требуется, то его нужно выгружать в оперативную память, а затем загружать снова. Это увеличивает затрату системных ресурсов.
2. Так как регистр всего один, то к оперативной памяти запросов очень много, поэтому скорость замедляется.

Аккумуляторная архитектура использовалась в первых микропроцессорах. Буква "A" от "Accumulator" затем стала использоваться в названии регистра AX (и затем EAX, RAX).

## Done Регистровая архитектура системы команд

В машинах с регистровой архитектурой в процессоре находится массив однотипных регистров общего назначения (РОН). К каждому регистру можно обратиться, указав его номер. Соответственно, в команде необходимо указать номера регистров, которые хранят операнды, а также номер регистра результата.



Часто в адресную часть команд обработки допускается указать номера двух и даже трёх регистров, так как для представления конкретного регистра требуется не более 5-6 разрядов, если их количество невелико.

В регистровой архитектуре допускается расположение операндов и в регистрах, и в основной памяти, поэтому выделяют три формата команд обработки:

### 1. Регистр-регистр.

Операнды хранятся только в регистрах. Результат тоже помещается в регистр

Достоинства заключаются в простоте реализации, фиксированной длине команд, возможности выполнения всех команд за фиксированное число тактов, простоте формирования объектного кода, простоте параллельной обработки.

Недостатки заключаются большой длине объектного кода, при этом в коротких командах часть разрядов не используются из-за фиксированной длины команд.

### 2. Регистр-память

Один из операндов находится в регистре, другой - в основной памяти. Результат чаще всего заносится в один из операндов.

Плюсы: компактность объектного кода, данные могут быть доступны без загрузки в регистры, простота кодирования команд.

Минусы: потеря одного из операндов при записи результата, длинное поле адреса памяти сокращает место под номер регистра -> возможное максимальное число регистров уменьшается.

### 3. Память-память

Оба операнда находятся в основной памяти, результат тоже записывается в память.

Преимущества подхода состоят в компактности кода и отсутствии необходимости использования регистров.

Но к недостаткам относится максимальная длина операндов. Из-за наличия коротких и длинных команд трудно оптимизировать тракты передачи данных и декодеры инструкций. Низкое быстродействие из-за обращения к основной памяти.

Выполнение операции в АЛУ при такой архитектуре включает в себя:

1. Определение местоположения первого операнда
2. Выбор регистра первого операнда или считывание операнда из памяти
3. Определение местоположения второго операнда.
4. Выбор регистра второго операнда или считывание операнда из памяти
5. Подача на вход АЛУ операндов и выполнение операции
6. Определение местоположения результата.

7. Выбор регистра результата или ячейки памяти и занесение результата операции из АЛУ.

**Достоинства** регистровой архитектуры:

1. Компактность получаемого кода
2. Высокая скорость вычислений за счёт замены обращений к основной памяти на обращения к быстрым регистрам.

**Недостаток:**

1. Более длинные команды по сравнению с аккумуляторной архитектурой.

Сейчас этот вид архитектуры системы команд является преобладающим.

#### 44. Способы адресации: непосредственная, прямая, регистровая, неявная, косвенная, косвенная регистровая

##### Способы адресации

Адресная часть		
Способ адресации (СА)	Адрес/операнд	

##### Непосредственная адресация

КОП	СА	Непосредственный операнд
-----	----	--------------------------

Вместо адреса команда содержит непосредственно операнд.

(+) команда выполняется быстро

(-) непосредственный операнд может не войти в команду

##### Прямая адресация

КОП	СА	Адрес операнда			
		<table border="1"><tr><td>ОП</td></tr><tr><td>Операнд</td></tr><tr><td></td></tr></table>	ОП	Операнд	
ОП					
Операнд					

Адрес в команде является адресом операнда

(+) если операнд находится в памяти, то это самый быстрый способ указать на него

(-) заранее определенный адрес влияет на переносимость программы.

(-) Адрес занимает много места

### Неявная адресация

КОП	СА
-----	----

Операнд подразумевается (следует из КОП).

- (+) Команда занимает мало места
- (-) только такие команды нельзя использовать для построение всей системы команд.

### Регистровая адресация

Адрес в команде указывает не на ячейку ОП, а на регистр.

- (+) Быстрее прямой адресации
- (-) Количество регистров ограничено

### Косвенная адресация

КОП	СА	Адрес операнда
-----	----	----------------

Адрес в команде указывает на ячейку памяти, в которой находится адрес операнда.

- (+) удобна для обработки структурных типов данных.
- (-) приходится осуществлять много обращений к ОП.

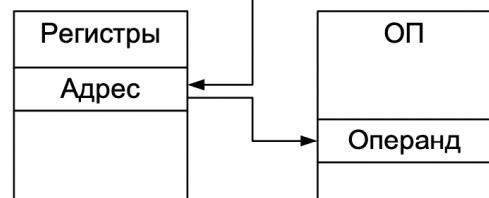


### Косвенная регистровая адресация

КОП	СА	Адрес операнда
-----	----	----------------

В команде содержится номер регистра, в котором содержится адрес операнда.

- (+) быстрее косвенной адресации
- (-) для перемещения по ОП нужно менять содержимое регистра



*Способ адресации* — это способ формирования исполнительного адреса операнда по адресному коду команды.

*Адресный код команды* ( $A_k$ ) — это двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

*Исполнительным адресом операнда* ( $A_{исп}$ ) называется двоичный код номера ячейки памяти, служащей источником или приемником операнда. Этот код подается на адресные входы запоминающего устройства (ЗУ) и по нему происходит фактическое обращение к указанной ячейке. Если операнд хранится не в основной памяти, а в регистре процессора, его исполнительным адресом будет номер регистра.

1) Непосредственная адресация:

При *непосредственной адресации* (НА) в адресном поле команды вместо адреса содержится непосредственно сам операнд. Этот способ может применяться при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры.

При записи в регистр, имеющий разрядность, превышающую длину непосредственного операнда, операнд размещается в младшей части регистра, а оставшиеся свободными позиции заполняются значением знакового бита операнда.

Помимо того, что в адресном поле могут быть указаны только константы, еще одним недостатком данного способа адресации является то, что размер непосредственного операнда ограничен длиной адресного поля команды, которое в большинстве случаев меньше длины машинного слова.

Непосредственная адресация сокращает время выполнения команды, так как не требуется обращение к памяти за операндом. Кроме того, экономится память, поскольку отпадает необходимость в ячейке для хранения операнда.

## 2) Прямая адресация:

При прямой или абсолютной адресации (ПА) адресный код прямо указывает номер ячейки памяти, к которой производится обращение, то есть адресный код совпадает с исполнительным адресом.

При всей простоте использования способ имеет существенный недостаток — ограниченный размер адресного пространства, так как для обращения к памяти большой емкости нужно «длинное» адресное поле. Однако более существенным недовершенством можно считать то, что адрес, указанный в команде, не может быть изменен в процессе вычислений (во всяком случае, такое изменение не рекомендуется). Это ограничивает возможности по произвольному размещению программы (и данных) в памяти.

## 3) Косвенная адресация:

Одним из путей преодоления проблем, свойственных прямой адресации, может служить прием, когда с помощью ограниченного адресного поля команды указывается адрес ячейки, в свою очередь, содержащей полноразрядный адрес операнда. Этот способ известен как **косвенная адресация** (КА).

При косвенной адресации содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять. Это позволяет проводить вычисления, когда адреса operandов заранее неизвестны и появляются лишь в процессе решения задачи. Дополнительно такой прием упрощает обработку массивов и списков, а также передачу параметров подпрограммам.

Недостатком косвенной адресации является необходимость в двухкратном обращении к памяти: сначала для извлечения адреса операнда, а затем для обращения к операнду. Сверх того, действует лишняя ячейка памяти для хранения исполнительного адреса операнда.

## 4) Регистровая адресация:

**Регистровая адресация** (РА) напоминает прямую адресацию. Различие состоит в том, что адресное поле команды указывает не на ячейку памяти, а на регистр процессора. Обычно размер адресного поля в данном случае составляет три или четыре бита, что позволяет указать соответственно на один из 8 или 16 регистров общего назначения (РОН).

Двумя основными преимуществами регистровой адресации являются: короткое адресное поле в команде и исключение обращений к памяти. Малое число РОН позволяет сократить длину адресного поля команды. К сожалению, возможности по использованию регистровой адресации ограничены малым числом РОН в составе процессора.

##### 5) Косвенная регистровая адресация:

*Косвенная регистровая адресация* (КРА) представляет собой косвенную адресацию, где исполнительный адрес операнда хранится не в ячейке основной памяти, а в регистре процессора. Соответственно, адресное поле команды указывает не на ячейку памяти, а на регистр.

Достоинства и ограничения косвенной регистровой адресации те же, что и у обычной косвенной адресации, но благодаря тому что косвенный адрес хранится не в памяти, а в регистре, для доступа к операнду требуется на одно обращение к памяти меньше.

##### 6) Неявная адресация:

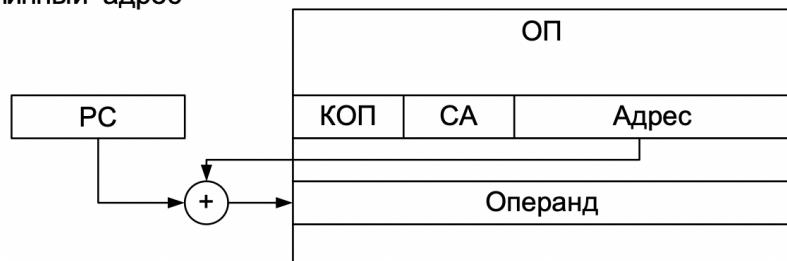
При неявной адресации в команде вообще отсутствует адресная информация, адресного поля либо просто нет, либо оно содержит не все необходимые адреса — отсутствующий адрес подразумевается кодом операции. Так, при исключении из команды адреса результата подразумевается, что результат помещается на место второго операнда. Неявная адресация применяется достаточно широко, поскольку позволяет сократить длину команды.

## 45. ?Способы адресации со смещением: относительная, базовая регистровая, индексная, автоинкрементная и автодекрементная, индексная с масштабированием

### Относительная адресация

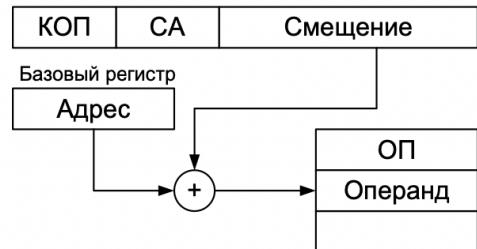
Адрес вычисляется относительно счётчика команд

- (+) Код переносим, команды занимают мало места
- (-) Может понадобиться длинный адрес

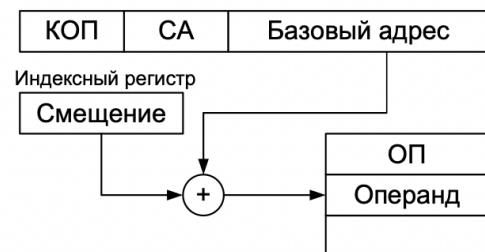


### Базовая регистровая адресация

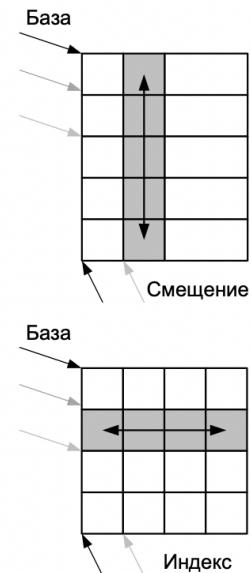
Адрес в команде представляет собой смещение, которое складывается со значением в базовом регистре для получения адреса операнда  
(+ Удобна для работы со структурами данных, размещаемых динамически.  
(-) Переносимость меньше, чем у относительной адресации



### Индексная регистровая адресация



В поле адреса команды содержится базовый адрес, складываемый со значением смещения в индексном регистре.  
(+ Удобна для работы со структурами данных, размещаемых динамически.  
(-) Переносимость меньше, чем у относительной адресации



### Автоинкрементная/автодекрементная адресация

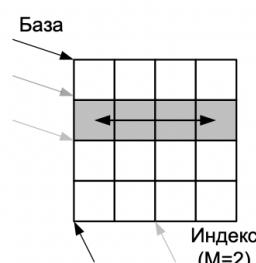
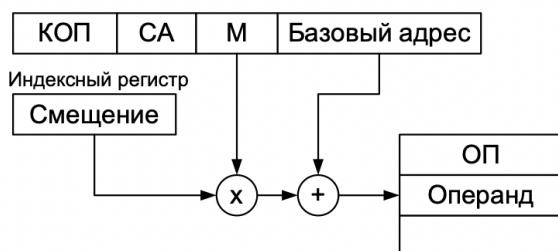
Разновидность регистровой индексной или базовой адресации. До или после выполнения команды значение базового или индексного регистра увеличивается/уменьшается на единицу или масштаб.

- (+) Способ адресации удобен для команд обработки строк.  
(-) Автоматическое изменение часто требуется выполнять на величину, большую единицы.

### Индексная адресация с масштабированием

Индексный регистр умножается на масштаб M и суммируется с базовым адресом из команды.

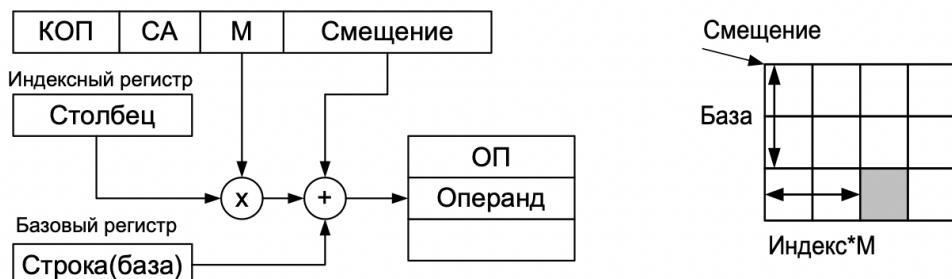
- (+) Удобен для модификации адреса на величину M.  
(-) Вычисление адреса замедляется, т.к. требуется выполнять умножение.



## Базовая индексная адресация с масштабированием

Адрес определяется по формуле Адрес=Индекс\*Масштаб+База+Смещение.

- (+) Базовая индексная адресация с масштабированием часто используется при обращении к системным таблицам, находящимся в ОП (таблица дескрипторов, таблицы страниц, таблица векторов прерываний и т.д.)  
(-) Ограничено на величину M (M=1,2,4,8).



При адресации со смещением исполнительный адрес формируется в результате суммирования содержимого адресного поля команды с содержимым одного или нескольких регистров процессора

*Адресный код команды ( $A_k$ )* — это двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

*Исполнительным адресом операнда ( $A_{исп}$ )* называется двоичный код номера ячейки памяти, служащей источником или приемником операнда. Этот код подается на адресные входы запоминающего устройства (ЗУ) и по нему происходит фактическое обращение к указанной ячейке. Если операнд хранится не в основной памяти, а в регистре процессора, его исполнительным адресом будет номер регистра.

### 1) Относительная адресация:

При *относительной адресации* для получения исполнительного адреса операнда содержимое поля  $A_k$  команды складывается с содержимым счетчика команд. Таким образом, адресный код в команде представляет собой смещение относительно адреса текущей команды. Следует отметить, что в момент вычисления исполнительного адреса операнда в счетчике команд может уже быть сформирован адрес следующей команды, что нужно учитывать при выборе величины смещения. Обычно поле  $A_k$  трактуется как двоичное число в дополнительном коде.

Адресация относительно счетчика команд базируется на свойстве локальности, выражаящемся в том, что большая часть обращений происходит к ячейкам, расположенным в непосредственной близости от выполняемой команды. Это позволяет экономить на длине адресной части команды, поскольку разрядность поля  $A_k$  может быть небольшой. Главное достоинство данного способа адресации состоит в том, что он делает программу перемещаемой в памяти: независимо от текущего расположения программы в адресном

пространстве взаимное положение команды и операнда остается неизменным, поэтому адресация операнда остается корректной.

## 2) Базовая регистровая адресация:

В случае *базовой регистровой адресации* (БРА) регистр, называемый базовым, содержит полноразрядный адрес, а поле  $A_k$  — смещение относительно этого адреса. Ссылка на базовый регистр может быть явной или неявной.

Базовую регистровую адресацию обычно используют для доступа к элементам массива, положение которого в памяти в процессе вычислений может меняться. В базовый регистр заносится начальный адрес массива, а адрес элемента массива указывается в поле  $A_k$  команды в виде смещения относительно начального адреса массива. Достоинство данного способа адресации в том, что смещение имеет меньшую длину, чем полный адрес, и это позволяет сократить длину адресного поля команды. Короткое смещение расширяется до полной длины исполнительного адреса путем добавления слева битов, совпадающих со значением знакового разряда смещения.

## 3) Индексная адресация:

При *индексной адресации* (ИА) поле  $A_k$  содержит адрес ячейки памяти, а регистр (указанный явно или неявно) — смещение относительно этого адреса. Как видно, этот способ адресации похож на базовую регистровую адресацию. Поскольку при индексной адресации в поле  $A_k$  находится полноразрядный адрес ячейки памяти, играющий роль базы, длина этого поля больше, чем при базовой регистровой адресации. Тем не менее вычисление исполнительного адреса операнда производится идентично.

Индексная адресация предоставляет удобный механизм для организации итеративных вычислений. Пусть, например, имеется массив чисел, расположенных в памяти последовательно, начиная с адреса  $N$ , и мы хотим увеличить на единицу все элементы данного массива. Для этого требуется извлечь каждое число из памяти, прибавить к нему 1 и вернуть обратно, а последовательность исполнительных адресов будет следующей:  $N, N + 1, N + 2$  и т. д., вплоть до последней ячейки, занимаемой рассматриваемым массивом. Значение  $N$  берется из поля  $A_k$  команды, а в выбранный регистр, называемый *индексным регистром*, сначала заносится 0. После каждой операции содержимое индексного регистра увеличивается на 1.

Так как это довольно типичный случай, в большинстве ВМ увеличение или уменьшение содержимого индексного регистра до или после обращения к нему осуществляется автоматически как часть машинного цикла. Такой прием называется *автоиндексированием*. Если для индексной адресации используются специально выделенные регистры, автоиндексирование может производиться неявно и автоматически. При задействовании для хранения индексов регистров общего назначения необходимость автоиндексирования должна указываться в команде специальным битом.

## 4) Автоинкрементная адресация:

Автоиндексирование с увеличением содержимого индексного регистра носит название *автоинкрементной адресации* и может быть описано следующим образом:

$$A_{исп} = A_k + (R), R \leftarrow (R) + 1 \text{ или } R \leftarrow (R) + 1, A_{исп} = A_k + (R),$$

где  $(R)$  — содержимое индексного регистра с адресом  $R$ . В первом варианте увеличение содержимого индексного регистра происходит после формирования исполнительного адреса, и

этот способ называется *постинкрементным автоиндексированием*. Во втором случае сначала производится увеличение содержимого индексного регистра, и уже новое значение используется для формирования исполнительного адреса. Тогда говорят о *преинкрементном автоиндексировании*.

5) Автодекрементная адресация:

Автоиндексирование с уменьшением содержимого индексного регистра носит на-звание *автодекрементной адресации* и может быть описано так:

$$A_{исп} = A_k + (R), R \leftarrow (R) - 1 \text{ или } R \leftarrow (R) - 1, A_{исп} = A_k + (R).$$

Здесь также возможны два варианта, отличающиеся последовательностью выполнения операций уменьшения содержимого индексного регистра и вычисления исполнительного адреса: *постдекрементное автоиндексирование* и *преддекрементное автоиндексирование*.

6) Индексная адресация с масштабированием:

Интересным и весьма полезным является еще один вариант индексной адресации — *индексная адресация с масштабированием и смещением*: содержимое индексного регистра умножается на масштабный коэффициент и суммируется с  $A_k$ . Масштабный коэффициент может принимать значения 1, 2, 4 или 8, для чего в адресной части команды выделяется дополнительное поле. Описанный способ адресации реализован, например, в микропроцессорах фирмы Intel.

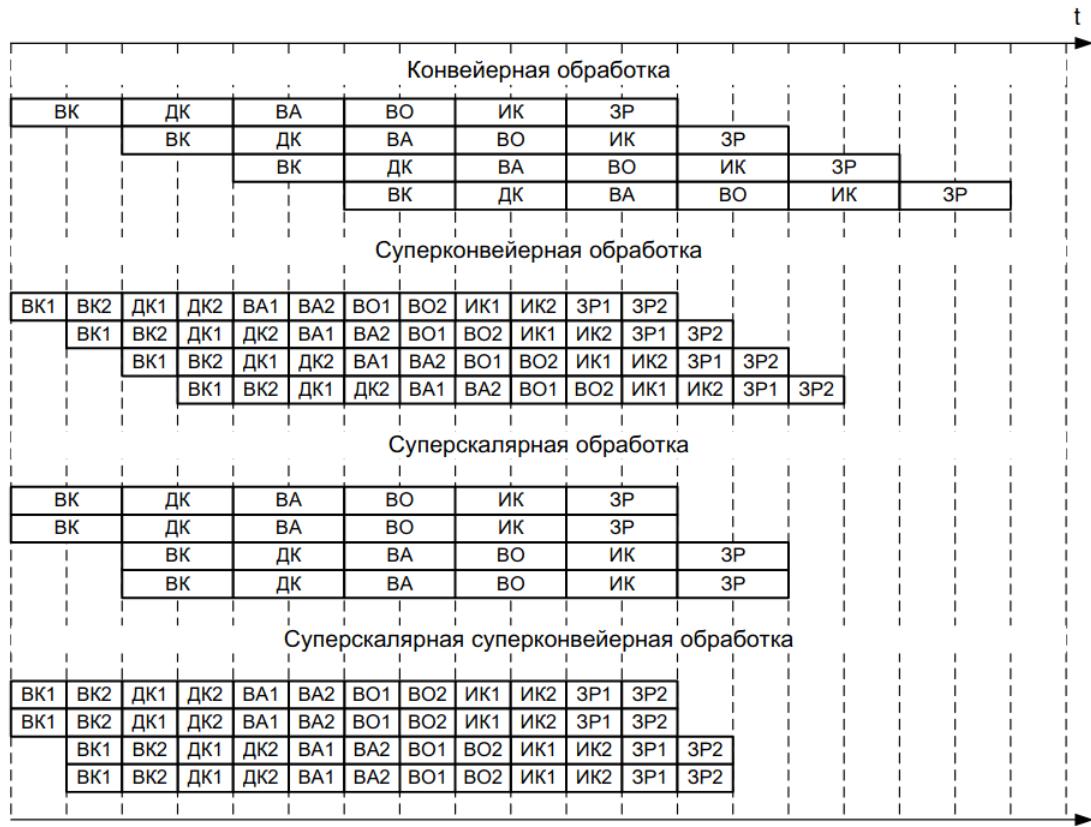
Следует особо отметить, что система команд многих ВМ предоставляет возможность различным образом сочетать базовую и индексную адресации в качестве дополнительных способов адресации.

## 46. Архитектура конвейерного суперскалярного процессора.

### Проблема условных переходов

По этому вопросу у Попова есть метода, все копи + паст с нее  
Не уверена, что этого достаточно, но это все что есть

Суперскалярная обработка основывается на способности процессора выполнять более одной простой (скаллярной) операции за один такт, что достигается благодаря включению в его состав нескольких параллельно работающих исполнительных устройств. На рис. 1, в видно, что дублирование всех блоков конвейерного процессора позволяет завершить обработку четырех команд на два такта быстрее. Реализация этого принципа совместно с внедрением суперскалярной суперконвейерной обработки (рис. 1, г) дает наилучший результат.



Предсказание направления ветвления, реализуемое буфером меток перехода и блоком вычисления адреса следующей команды входит в функции подсистемы вычисления адресов процессоров Р6.

Блок вычисления адреса следующей команды (Prediction Unit — PU) служит для определения адреса очередной команды и выдачи его в блок FIU. Как уже отмечалось, конвейерная обработка предполагает непрерывное поступление команд, исполнение которых происходит лишь после декодирования и подготовки операндов. Очевидно, что ожидание исполнения команд ветвления существенно замедлило бы обработку, поэтому в процессорах активно используются способы предсказания направления ветвления.

Ускоренное по сравнению с иными командами декодирование команд перехода обеспечивается благодаря предварительному сканированию блока, считанного из кэш-памяти команд L1I блока. Указанное действие позволяет минимизировать потери времени. Далее при обнаружении команды безусловного перехода (команды вызова подпрограмм, возврата из процедур и прерываний, прямых переходов и т. п.) быстро вычисляется логический адрес очередной команды.

Более существенные задержки в работе конвейера вызывают команды условного перехода, что связано с невозможностью вычисления условия перехода до окончания выполнения всех предшествующих команд.

Частичное решение этой проблемы достигается применением различных способов предсказания - статический и динамический, см. вопрос 47.

Декодированные команды переходов и информация о результате предсказания совместно с другими командами поступают на следующие стадии конвейера. После вычисления всех использованных в таких командах операндов (например, флагового регистра) команда перехода поступает на специальное устройство арифметики переходов, которое проверяет правильность предсказания. В случае правильного предсказания дополнительных действий не требуется. При ложном предсказании устройство арифметики переходов инициирует сброс конвейера, заключающийся в выполнении следующих действий:

- восстановление состояния физических регистров на момент последней актуальной команды перехода (такое действие в современных процессорах сводится к стиранию содержимого нескольких регистров);
- изменение статистической информации о неправильно предсказанном переходе;
- запрос очередной команды правильной ветви исходной программы.

## **47. Архитектура конвейерного суперскалярного процессора. Статическое и динамическое предсказание переходов**

Наличие команд условного перехода и механизм предсказания приводят к необходимости сброса конвейера в случае неправильно выполненного предсказания ветвления.

Наибольшую задержку в работе конвейера вызывают команды условных переходов, что связано с невозможностью вычисления условия перехода до окончания выполнения всех предшествующих команд. Частичное решение этой проблемы достигается применением различных способов предсказания.

1. **Статический** - основан на анализе предварительно собранных сведений. Такая информация внедряется в программный код на этапе компиляции и может быть получена, например, на основе заблаговременно выявленной неравномерности исходов условных переходов программы.
2. **Динамический** - основано на использовании предыстории переходов, содержащей информацию об исходах выполнения команд условного перехода, поступавших в процессор ранее. Кроме того, применяются известные статистические сведения. Например, переход в сторону убывания адресов (переход назад) целесообразно предсказывать «совершенным», в то время как переход в сторону

увеличения адресов (переход вперед), наоборот, целесообразно не предсказывать.

Кроме того, в зависимости от сложности реализации алгоритмы динамического предсказания позволяют выявить в программе шаблонные кодовые конструкции: циклы, попереемые переходы, маловероятные переходы и др. Статистическая информация накапливается в специальной ассоциативной памяти — буфере меток перехода (Branch Target Buffer — BTB), обращение к которой выполняется по целевому программному адресу.