## 1. Как выяснить, находится ли адресуемый объект (операнд) в КЭШе? (Как понять, что объект в кэше?)

1. ЦП(Центра́льный проце́ссор) обращается к объекту

2. Из адреса объекта выделяется три поля - **БАЙТ в блоке** памяти, **ИНДЕКС** и **ПРИЗНАК**.

Индекс служит для того, чтобы определить в какой кэш-линии расположен блок памяти (или, в общем случае, группа блоков памяти)

Признак служит для того, чтобы эти блоки памяти отличать между собой.

3. Следовательно, для того, чтобы определить, находится ли нужный в данный момент центральному процессору объект в кэше, нужно **потребовать полное совпадение всех этих полей**. Если они **не совпали**, то необходимо **обращаться к памяти**, а если **совпали**, то **объект передается** из кэша в ЦП.

*// ДРУГИМИ СЛОВАМИ : Процессор, выполняя команду, запрашивает операнд по некоторому адресу в адресном пространстве. Кэш-контроллер проверяет, есть ли в кэш-памяти строка данных, соответствующая запрашиваемому адресу. В случае наличия такой строки ситуация называется кэш-попадание. Если строки в кэш-памяти нет, то происходит кэш-промах, и кэш-контроллер инициирует обращение к основной памяти ОП для переписи из нее нужной строки в кэш-память.*

**КОРОТКИЙ ОТВЕТ НА ВОПРОС**: требуется совпадение всех составляющих адреса объекта с соответствующими значениями в кэш-памяти

Информационная память кэша и оперативная память вычислительной системы разбиты на блоки одинаковой длины и каждый блок памяти отображается на одну из строк кэша. В большинстве систем размер блока - 32 или 64 байта. Адрес объекта в памяти определяет строку кэша, в которую помещается блок информации, содержащий адресуемый объект.

1. При обращении центрального процессора к адресуемому объекту из адреса объекта выделяются три поля: - номер байта в блоке, индекс и признак.

— **номер байта в блоке**, занимающий младшие разряды адреса объекта;

— **индекс, определяющий адрес блока в кэше,** в котором должен находиться адресуемый объект, и занимает средние разряды адреса объекта;

— **признак,** занимающий старшие разряды адреса и позволяющий отличить блоки оперативки; при записи блока памяти в строку КЭШа, определяемую индексом, в элемент теговой памяти строки помещается признак.

2. По индексу, находится строка кэша, в которой должен находиться нужный блок памяти.

3. Признак, выделенный из адреса объекта, сравнивается с признаком, хранящимся в элементе признаковой памяти найденной строки кэша. Если признаки совпали, то это означает что адресуемый объект в кэше - произошло кэш-попадание. . Если же нет, то запрашиваемая информация в кэше отсутствует произошел кэш-промах.т.е. сравниваются Признак объекта выделенного из адреса и признаки находящиеся в служебных полях нашего кэша.

Промах чтения в кэш-памяти вызывает загрузку строки из основной памяти. В случае промаха записи транзакция записи помешается в буфер и посылается в основную память при предоставлении шины.

При чтении блока из памяти в кэш кроме записи самого блока нужно записать признак этого блока памяти.

Если длина элемента данных превышает один байт, то возможны ситуации, когда этот элемент (частями) расположен в двух (или более) разных КЭШ-строках, тогда время на выборку такого элемента увеличится. Противодействовать этому можно, выравнивая операнды и команды по границам КЭШ-строк.

# 2. Чем определяется объем кэшированной оперативной памяти?

В общем случае, объем кэшируемой ОП определяется разрядностью признаковой памяти кэша

Нетрудно видеть, что при А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, а при

А, равном числу строк в кэше, в полностью ассоциативный

кэш.

Количество групп в кэше является величиной противоположной коэффициенту А: полностью ассоциативный кэш состоит из одной группы, а кэш прямого отображения содержит

столько групп, сколько блоков в кэше. Соответственно с ростом

ассоциативности кэша размерность признакового поля растет, а

разрядность индекса падает (до О в полностью ассоциативном

кэше). С ростом длины блока при постоянной емкости кэша Kоличество строк кэша уменьшается, и разрядность индекса также

снижается, размер поля "номер байта в блоке" растет, а объем

признакового поля не изменяется.

# 3. Отображение памяти на кэш

## КЭШ прямого отображения.

ОСНОВНАЯ ИДЕЯ прямого отображения (direct mapping) RAM на кэш-память - RAM делится на сегменты, причем размер каждого сегмента равен размеру кэша, а каждый сегмент в свою очередь делится на блоки, размер каждого блока равен размеру кэш-линии. Блоки RAM из разных сегментов, но с одинаковыми номерами в этих сегментах, всегда будут отображаться на одну и ту же кэш-линию кэша. Адрес каждого байта представляет собой сумму порядкового номера сегмента, порядкового номера кэш-линии внутри сегмента и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера сегментов, а порядковые номера кэш-линий внутри сегментов и порядковые номера байт внутри кэш-линий — повторяются.

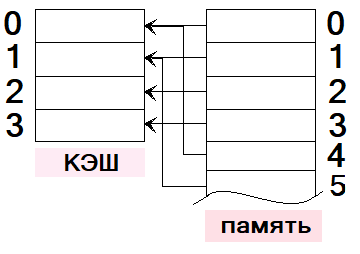
Таким образом нет необходимости хранить полный адрес кэш-линии, достаточно сохранить только старшую часть адреса.

**Каждой строке КЭШ памяти соответствуют конкретные, заранее определенные строки оперативной памяти**

ПЛЮСЫ : Требуется только **ОДНО СРАВНЕНИЕ** признаковых полей

МИНУСЫ : **КОНФЛИКТ АДРЕСОВ -**

Конкуренция **множества** блоков **ПАМЯТИ** на **одну** строку **КЭША**



//Как правило, не используется. **АЛЬТЕРНАТИВА - ПАК**.

### Полностью ассоциативный КЭШ**.**

ОСНОВНАЯ ИДЕЯ полностью ассоциативного отображения (fully associative mapping) RAM на кэш-память - RAM делится на блоки, размер которых равен размеру кэш-линий, а каждый блок RAM может сохраняться в любой кэш-линии кэша.

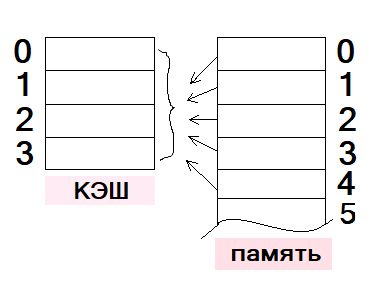
Адрес каждого байта представляет собой сумму порядкового номера кэш-линии и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера кэш-линий. Порядковые номера байт внутри кэш-линий повторяются.

**Каждая строка оперативной памяти может попадать на любую строку КЭШ**

ПЛЮСЫ : Строки **КЭША** и **ПАМЯТИ** НИКАК НЕ СВЯЗАНЫ

НЕ ВОЗНИКАЕТ КОНФЛИКТ АДРЕСОВ

МИНУСЫ : Признаковые поля **сравниваются ОЧЕНЬ МНОГО** раз

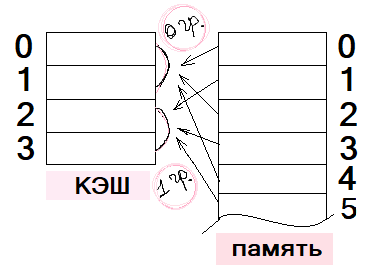


### Множественно ассоциативный КЭШ**.**

Множество строк кэша разбивается на A >= 1 строк.

Между группами - **КПО** ( кэш прямого отображения )

Внутри групп - **ПАК** ( полностью ассоциативный КЭШ )



ПЛЮСЫ : **КОНФЛИКТ** адресов **НИЖЕ**, чем у КПО

**СРАВНЕНИЙ МЕНЬШЕ**, чем у ПАК

# 4. В какой архитектуре КЭШа самый большой процент попадания?

Самый большой процент попадания в **МНОЖЕСТВЕННО-АССОЦИАТИВНЫЙ** КЭШ, так как сокращается количество коллизий блоков памяти, а поиск нужной кэш-линии внутри группы ограничивается ее размерами (и не обязательно составляет размер кэша, как это было в полностью ассоциативном кэш).

# 5. Как изменяется разновидность признака, индекса, блока с изменением емкости КЭШа, длины блока?

С ростом длины блока уменьшается признак, индекс и номер блока.

С ростом ассоциативности все поля увеличиваются.

С ростом емкости кэша - уменьшаются.

С **РОСТОМ АССОЦИАТИВНОСТИ** кэша размерность признакового поля растет, а разрядность индекса падает (до О в полностью ассоциативном кэше).

С **РОСТОМ ДЛИНЫ БЛОКА** при постоянной емкости кэша количество строк и разрядность индекса кэша уменьшается, размер поля "номер байта в блоке" растет, а объем

признакового поля не изменяется.

# 6. Разница между архитектурами CISC / RISC (Соотношение CISC и RISC архитектур)

**CISC** – ЦП (центральный процессор) с **ПОЛНЫМ** набором команд.

Форматы команд : **ПАМЯТЬ-ПАМЯТЬ, ПАМЯТЬ-РЕГИСТР,**

**РЕГИСТР-РЕГИСТР**

Архитектура процессора, которая характеризуется следующим набором свойств:

1. **Короткая программа** - длинные команды (все команды разных форматов).

*добавление в арифметический конвейер*

***длинной цепочки*** *доступа к оперативной памяти неминуемо приведет к* ***низкой эффективности*** *ero функционирования.*

1. В одной команде - несколько арифметических действий.
2. Обращение к памяти разрешено для инструкций разного рода.
3. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

**RISC** – ЦП (центральный процессор) с **СОКРАЩЕННЫМ** набором команд

Один формат : **РЕГИСТР-РЕГИСТР**

Архитектура процессора, в которой **быстродействие увеличивается** за счёт УПРОЩЕНИЯ ИНСТРУКЦИЙ, чтобы их декодирование было более простым, а **время** выполнения — **меньшим**.

1. Длинная программа - **короткие команды (**фиксированный 32-битный формат у всех команд)
2. **ПОЛНОРАЗРЯДНАЯ ОБРАБОТКА** ДАННЫХ -

Каждый арифметический конвейер за ОДИН ТАКТ формирует ОДИН 64-разрядных результат (один такт = одно слово).

1. Обращение к памяти происходит посредством команд загрузки и записи
2. **Большое количество регистров** на уровне архитектуры.
3. **Конвейерная организация** арифметических и других исполнительных устройств.

**CISC** - архитектура вычислений с полной системой команд. Выполнение команд обработки данных, сохраняемых в памяти (операции "память­-память").

**RISC** - архитектура вычислений с сокращенным набором команд. Подразумевает обработку данных, хранящихся только в регистрах. **Обе архитектуры являются воплощением SISD-архитектур.** RISC- архитектура производительнее в тех приложениях, где над каждой единицей данных выполняется большой объем вычислительной работы (инженерные и научные задачи), в противном случае часто приходится загружать данные из памяти.

# 7. Способы увеличения производительности процессов

1. (?) КОНВЕЙЕРИЗАЦИЯ - это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.
2. СУПЕРСКАЛЯРНОСТЬ.

Суперскалярные процессоры - процессоры с несколькими конвейерами.

**Основная идея**, определяющая развитие суперскалярных микропроцессоров - построение **большего количества параллельных структур**, сохранение традиционных последовательных программ.

1. НЕУПОРЯДОЧЕННОЕ выполнение.

Архитектуры с **неупорядоченным завершением и исполнением** позволяют одному конвейеру продолжать работать при “заторе” в другом.

При этом команды, **стоящие позже**, могут быть **выполнены раньше** предыдущих - ЗАСТРЯВШИХ в другом конвейере.

// “Затор” - зависание, обусловленное объективными взаимозависимостями команд в программе и способами построения суперскалярных архитектур.

**!!!!!! ВАЖНО,** чтобы процессор гарантировал, что результаты запишутся в память, а регистры модифицируются в правильной последовательности!!!!!

1. ПЕРЕИМЕНОВАНИЕ регистров.

!! Частое **использование ОДНИХ регистров** приводит к возникновению ВЗАИМОЗАВИСИМОСТЕЙ и **СНИЖЕНИЮ ПРОИЗВОДИТЕЛЬНОСТИ** !!

**CISC-**архитектуры имеют **небольшое число** архитектурных **регистров** → ЛОЖНЫЕ взаимозависимости возникают довольно **часто**.

Путь **повышения производительности** - нейтрализация возникновения ЛОЖНЫХ взаимозависимостей путем ПЕРЕИМЕНОВАНИЯ РЕГИСТРОВ:

* **КОМАНДЕ 1** требуется **регистр** → процессор динамически **ставит в соответствие** этому АРХИТЕКТУРНОМУ (логическому) регистру один из множества ФИЗИЧЕСКИХ регистров.
* **КОМАНДА 2** пытается обратиться к **этому же** регистру → процессор **ставит в соответствие** другой ФИЗИЧЕСКИЙ регистр.

Такие переименования действуют, пока **команды продвигаются по конвейерам**

1. ОБХОДЫ и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение** из регистра или памяти.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команды 1 команде 2 (которой они потребуются на более поздней стадии конвейера)

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

ОБХОДЯТСЯ ИСТИННЫЕ ЗАВИСИМОСТИ

1. ПРОГНОЗИРОВАНИЕ переходов.

Поскольку ВЕТВЛЕНИЯ (условные переходы) **разрешаются на исполнительной** ступени конвейера, **процессор не знает**, какие команды отправлять в конвейер за командой перехода.

Для этого и **используется ПРОГНОЗИРОВАНИЕ ПЕРЕХОДОВ** (предположение о пути ветвления), при котором процессор начинает :

* **ЧТЕНИЕ команд** с предсказанного адреса до того, как узнает, верным ли был прогноз.
* **ИСПОЛНЕНИЕ** по предположению “спекулятивное”.

//При этом процессор **не может модифицировать** архитектурные //**РЕГИСТРЫ** или **ПАМЯТЬ** до уточнения достоверности предсказания.

1. (?) ПРЕВРАЩЕНИЕ **CISC → в RISC**

**ГЛАВНАЯ ПРОБЛЕМА** декодирования **CISC**-команд - **определить** границы команд и **расположить** их в последовательности, которая обеспечивает эффективную ОБРАБОТКУ **ДЕШИФРАТОРОМ** (быстрое декодирование - **обязательное условие** обработки нескольких команд за такт)

**Предварительное декодирование** - распознавание сложных команд до стадии декодирования на этапе загрузки из основной памяти в кэш-память. Процесс **НЕ СНИЖАЕТ производительность** системы, но **требует** дополнительной **КЭШ**-памяти.

# 8. Стадии, фазы конвейера

**КОНВЕЙЕРИЗАЦИЯ**- это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.

Каждая СТУПЕНЬ обрабатывает свою ФАЗУ **независимо**.

В каждый момент времени конвейерный процессор работает над выполнением различных СТАДИЙ **нескольких команд**.

**КОНВЕЙЕР** - команда из 5 классических **СТАДИЙ**:

1. **ЧТЕНИЕ** КОМАНД
2. **ДЕШИФРОВАНИЕ**
3. **ЧТЕНИЕ** ОПЕРАНДОВ
4. **ИСПОЛНЕНИЕ**
5. **ЗАПИСЬ РЕЗУЛЬТАТА**

Различные **КОМАНДЫ** - неодинаковое **ВРЕМЯ** выполнения различных фаз.

Максимальная ЭФФЕКТИВНОСТЬ в случае, когда **время** выполнение ФАЗ **ОДИНАКОВО (однородный** конвейер).

К факторам, нарушающим непрерывность конвейерной реализации команд, относятся случаи, когда :

* Для выполнения команды 2 требуется завершение команды 1.
* В кэш-памяти отсутствуют некоторые данные, и требуется передать их из ОСНОВНОЙ памяти в КЭШ-память.
* При ВЕТВЛЕНИИ, по результатам проверки условий, команды, находящиеся в процессе конвейерной обработки, остаются не выполненными. Требуется снова загружать конвейер с момента чтения команды условного перехода.

## Фазы классического конвейера

В большинстве классических СISC-архитектур имеется пять ступеней обработки конвейера:

1) Выборка команды

2) Декодирование команды

3) Адресация и выборка операндов

4) Исполнение команды

5) Запись результата

# 9. Истинные и ложные зависимости

Причины **ограничения производительности** процессора :

1. **ИСТИННАЯ** взаимозависимость данных -

ВХОДНЫЕ данные **для одной** операции зависят от РЕЗУЛЬТАТА **другой** операции

(**к примеру**, в выражении **n\*k+m** НЕЛЬЗЯ выполнить сложение и умножение одновременно)

“ЧТЕНИЕ ПОСЛЕ ЗАПИСИ”

1. **ЛОЖНАЯ** взаимозависимость данных:

а) ВЗАИМОЗАВИСИМОСТЬ **ПО ВЫХОДУ:**

Команды, следующие друг за другом записывают результат в **один и тот же** РЕГИСТР

“ЗАПИСЬ ПОСЛЕ ЗАПИСИ”

б) **АНТИ** ВЗАИМОЗАВИСИМОСТЬ:

**Команда 1** может **испортить** данные, необходимые для ВХОДА **команде 2.**

“ЗАПИСЬ ПОСЛЕ ЧТЕНИЯ”

## Ложные и истинные взаимозависимости Как борятся с истинными взаимозависимостями?

**Истинная взаимозависимость данных** или взаимозависимость “чтение после записи” **-**входные данные для одной операции зависят от результата другой, (например – a\*b+c). **Анти-взаимозависимость**, или взаимозависимость “ запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой. **Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров. RISC-процессоры имеют достаточное кол-во регистров чтобы ложные взаимозависимости не появлялись часто, в то время как CISC-процессоры имеют **небольшое кол-во архитектурных регистров, что провоцирует частое возникновение ложных взаимозависимостей**. Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, то процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам. Только переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

# 10. Чем отличается модель программирования UMA от NUMA?

UMA - **Uniform Memory Access - РАВНОМЕРНЫЙ** доступ к памяти

Архитектура **С** **ОБЩЕЙ ПАМЯТЬЮ** (использует один контроллер памяти), к которой обращаются **ВСЕ ПРОЦЕССОРЫ** представленной многопроцессорной системы с помощью межсоединительной сети.

Каждый процессор имеет **РАВНОЕ** время доступа к памяти (задержка) и скорость доступа. Он может использовать одну шину / несколько шин / коммутатор.

**Пропускная способность** ОГРАНИЧЕНА.

NUMA - **Non** **Uniform Memory Access - НЕРАВНОМЕРНЫЙ** доступ к памяти

Многопроцессорная модель, в которой **каждый ПРОЦЕССОР** связан с **СОБСТВЕННОЙ** (выделенной) памятью (контроллер множественной памяти).

Все части памяти объединяются в единое АДРЕСНОЕ ПРОСТРАНСТВО.

**В отличие от UMA,** время доступа к памяти **зависит от расстояния**, на котором расположен процессор, что означает изменение времени доступа к памяти. Это позволяет получить доступ к любой ячейке памяти, используя физический адрес.

**Пропускная способность** БОЛЬШЕ, ЧЕМ У UMA.

**СС-NUМА** -системы с кэш когерентным доступом (при изменении объекта одним процессором или ядром, изменяется объект для всей системы) к неоднородной памяти. Системы этого класса строятся из узлов с SМР­ архитектурой, имеющих собственную (локальную) область общего адресного пространства и кэш "дальней" памяти (областей памяти других узлов). Узлы между собой связываются с помощью шины. По существу, данная архитектура является расширением систем с симметричной мультипроцессорностью. В отличие от классической архитектуры NUMA, при использовании кэш-когерентного доступа к неоднородной памяти все процессоры объединены в один узел, причем первый уровень иерархии памяти образует кэш-память процессоров.

**UMA** - системы с однородной памятью. Имеют общую (разделяемую) память для всех процессоров вычислителя. Архитектура предполагает наличие одной (реже двух или более) общей шины к которой подключены несколько процессоров. Доступ к любому элементу данных происходит единообразно.

**NUMA** –системас неоднородным доступом к памя­ти. Наличие индивидуальной оперативной памяти возле каждого процессора (или группы процессоров) и поддержка неоднородного доступа к памяти, обеспечивающая существенно различное время обращения к объектам в распределенной памяти. Каждый процессор имеет доступ не только к собственной ло­кальной памяти, но и к памяти других процессоров сети. Но поскольку при обращении к памяти других процессоров запросы проходят через сеть, они вы­полняются дольше, чем обращения к локальной памяти. Системы этого типа называются мультипроцессорными системами. В системе CC-NUMA **физически распределенная память объединяется в единый массив** (единое адресное пространство с частями, физически связанными специальной соединительной шиной, и аппаратно-реализованная кэш-когерентность).

**С точки зрения модели программирования:**

-Ключевое различие – диктуемая модель программирования. Задержки доступа к адресуемым объектам происходят из-за различия в способах программирования.

· SMP-, СМР-и СС-МЈМА-системы попадают в один класс компьютерных систем с общей памятью и аппаратно-поддерживаемой когерентностью данных. (+ доступ к часто используемым данным, в разделяемой памяти, за микросекунды, в то время как считывание их с диска требует миллисекунд)

· МРР-системы и вычислительные кластеры — в класс вычислительных систем с распределенной памятью и программно-реализуемой когерентностью данных (-время доступа к объектам в удаленной памяти повышается до сотни микросекунд — это в сотни раз медленнее, чем быстродействие локальной памяти)

**Наиболее выгодно программирование в SMP**, поскольку при этом **программисту не нужно заботиться о распределении данных в памяти**, так как все ее части доступны для любого процессора и доступ к ним одинаково быстр.

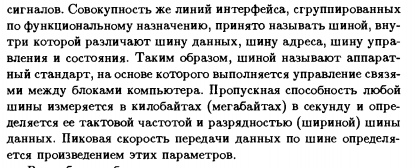
# 11. Чем определяется быстродействие истинного интерфейса?

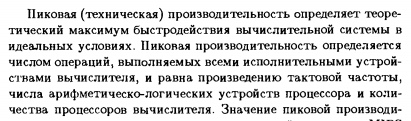
ВАЖНЕЙШИМ фактором, определяющим БЫСТРОДЕЙСТВИЕ **ВС**, является ЁМКОСТЬ КЭША (?)

ВАЖНЕЙШИМ фактором, определяющим БЫСТРОДЕЙСТВИЕ

**дискового накопителя**, является ПЛОТНОСТЬ ЗАПИСИ.

// Если **ИСТИННЫЙ** интерфейс = интерфейс самого **НИЗКОГО уровня**, тогда ответ “кэш первого уровня”





# 12. Для чего нужен обход и продвижение данных в ЦП?

Переименование регистров **не полностью исключает** возникновение ИСТИННЫХ взаимозависимостей.

Поэтому используются методы ОБХОДА и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение**.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команд.

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

# 13. В каком случае используется прогнозирование переходов и для чего это нужно?

Приемы **увеличения производительности**\* **имеют смысл,** только когда решена проблема процедурных взаимозависимостей, возникающая из-за **наличия** условных и безусловных **ПЕРЕХОДОВ** в программе.

Поскольку ВЕТВЛЕНИЯ (условные переходы) **разрешаются на исполнительной** ступени конвейера, **процессор не знает**, какие команды отправлять в конвейер за командой перехода.

Для этого и **используется ПРОГНОЗИРОВАНИЕ ПЕРЕХОДОВ** (предположение о пути ветвления), при котором процессор начинает :

* **ЧТЕНИЕ команд** с предсказанного адреса до того, как узнает, верным ли был прогноз.
* **ИСПОЛНЕНИЕ** по предположению “спекулятивное”.

//При этом процессор **не может модифицировать** архитектурные //**РЕГИСТРЫ** или **ПАМЯТЬ** до уточнения достоверности предсказания.

**ПРИЕМЫ** прогнозирования ветвлений:

* Использование буфера адреса перехода
* Буфер адресов переходов дополняется стеком возвратов, который отслеживает переходы в паре команд CALL/RETURN
* Информация о переходах добавляется к каждой строке КЭШа команд

МЕТОДЫ:

**Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.

**Динамические методы** оценивают поведение команд перехода за предшествующий период времени.

\*СУПЕРСКАЛЯРНОСТЬ, НЕУПОРЯДОЧЕННАЯ ОБРАБОТКА, ПРОДВИЖЕНИЕ данных

## Зачем нужно прогнозирование?

Предсказание ветвлений позволяет сократить время простоя конвейера за счёт предварительной загрузки и исполнения инструкций, которые должны выполниться после выполнения инструкции условного перехода. Без предсказания переходов конвейер должен дождаться выполнения инструкции условного перехода, чтобы произвести следующую выборку. Предсказатель переходов позволяет избежать траты времени, пытаясь выяснить ответвление. Ответвление выбирается по предыдущим результатам проверки условия. Предполагаемое ответвление затем загружается и частично выполняется. Если затем обнаруживается, что предсказание было выполнено неверно, отменяются результаты неверного ветвления и в конвейер загружается правильное ответвление, производя задержку. Существует 2 метода прогнозирования:

* **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов.
* **Динамические методы** оценивают поведение команд перехода за предшествующий период времени.

В Р6 используется исполнение по предположению — исполнение команд, следующих за командой условного перехода до того, как выяснится, правильно ли был предсказан переход. При этом процессор не обновляет архитектурные регистры и память до однозначного разрешения команд, выполненных по предположению. При неправильном предсказании ROB процессора Р6 отбрасывает исполненные по предположению команды, прежде чем они будут удалены.

Как происходит:

Устраняет процедурные зависимости, то есть условные и безусловные переходы. Увеличивает эффективность суперскалярности, неупорядоченной обработки и продвижения данных. Процессор использует “предложение о пути ветвления” и после этого начинает: 1. Выборку команд с предсказанного адреса перехода до того, как он узнает, верным ли был его прогноз. 2. Исполнение по предположению “спекулятивное”, при этом процессор не может модифицировать архитектурные регистры или память до однозначного разрешения перехода. Обычно используется буфер адреса перехода. Бывает предикатное выполнение команд: выполнение по всем возможным исходам условного перехода.

# ШИННЫЙ ИНТЕРФЕЙС

**Системная шина**Объединяющая центральный процессор с памятью и интерфейсом подключения внешних устройств;

**ШИНЫ ВВОДА/ВЫВОДА**

В современных вычислительных системах шину ввода-вывода называют также шиной РАСШИРЕНИЯ, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств.

**ПЕРИФЕРИЙНЫЕ ШИНЫ**

Периферийные шины являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы

**Различают шину данных, шину адреса, шину управления и состояния. Шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера**

**3.2. Шины ввода-вывода**

Она же шина расширения используется в качестве шины для подключения контроллеров внешней периферии.

**3.2.1. Шина PCI и унаследованные шины**

Шина РС1 стала стандартом подключения внешних устройств для вычислительных архитектур. Для настольных систем реализуется обычно 32-разрядная версия стандарта, работающая на частоте 33 МГц, а для серверных платформ и высокопроизводительных рабочих станций используется четыре версии: 32 бит/ЗЗ МГц, 32 бит/бб МГц, 64 бит/ЗЗ МГц и самая скоростная 64 бит/бб МГц.

Спецификация этой шины (PCI-X) увеличивает производительность (до 1 Гбайт/с. Два главных отличия PCI-X от PCI состоят в использовании межрегистрового протокола (декодирование производится за отдельный цикл, тогда как раньше оно производилось на протяжении того же цикла) и атрибутивной фазы(во время этой фазы поле длиной 36 бит сообщает более подробную информацию о транзакции , чем это предусмотрено прежней спецификацией IPC).

**3.2.2. Ускоренный графический порт AGP**

Шина обмена графической информацией. Порт AGP имеет самостоятельный доступ к памяти. Разрядность шины составляет 32 бит, рабочая частота — 66 МГц, в шине поддерживаются режимы передачи х1, х2, х4. Порт AGP предназначен для обмена данными между основной памятью и видеоадаптером.

**3.3.1. Шина EIDE**

Используется для подключения устройств долговременной памяти является расширенный интерфейс подключения к компьютерам АТ накопителей со встроенным контроллером. Данная шина имеет два канала, к каждому из которых можно подключить по два устройства.

Теоретическая пропускная способность в настоящее время составляет 16,7 Мбайт/с. Длина интерфейсного кабеля до 45 см.

Недостаток: значительную часть работы по управлению выполняет процессор.

**Из чего состоит параллельная шина?**

По способу передачи данных шины делятся на последовательные и параллельные. Последовательные шины передают данные по одному проводнику, один бит за один раз, в параллельных шинах передача данных разделена между несколькими проводниками и поэтому можно передать большее количество данных.

# 1. Классификация ВС по Флинну.

Базируется на понятии **ПОТОКА**, под которым понимается **последовательность** элементов, команд или данных, обрабатываемая процессором.



SISD

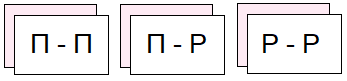
**В настоящее время имеется три воплощения SISD­ - архитектур:**

1. **CISC** - ЦП с **ПОЛНЫМ** набором команд.

Концепция проектирования процессоров, которая характеризуется следующим набором свойств:

1. Короткая программа - длинные команды.
2. Арифметические действия кодируются в одной команде.
3. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

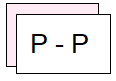
Формат команд (п- память, р - регистр) :



1. **RISC** - ЦП с СОКРАЩЕННЫМ набором команд.

Архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим.

Один формат : р - регистр



* ЗАГРУЗКА и ВЫГРУЗКА

1. **СУПЕРСКАЛЯРНЫЕ** архитектуры - больше одного КОНВЕЙЕРА
2. с **ДИНАМИЧЕСКИМ** предсказанием ветвлений
3. со **СТАТИЧЕСКИМ** предсказанием ветвлений.

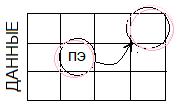
Основан на VLIW-архитектуре.

1. **VLIW -** архитектура с очень длинным (широким) командным словом **до n команд.**

Реализуется КОМПИЛЯТОРОМ.

#### SIMD. ВАРИАНТЫ РЕАЛИЗАЦИИ :

* **МАТРИЧНАЯ** архитектура команды



**ПЭ** - процессорные элементы

ВЫСОКАЯ **сложность** коммутировать результаты вычислений между ПЭ

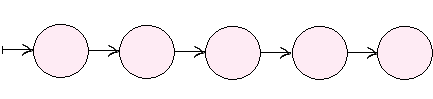
* **ВЕКТОРНО-КОНВЕЙЕРНАЯ** архитектура

ВЕКТОРИЗАЦИЯ матричных вычислений

**ОСНОВА -** длинные регистры **(множество швов)**

КОНВЕЙЕРИЗАЦИЯ :

Запускаем одну и ту же программу разными словами



#### MISD -

специальные решения, обеспечивающие **ГОТОВНОСТЬ** и **НАДЕЖНОСТЬ** ВС.

Самая простая и **распространенная система** этого класса - ЛОКАЛЬНАЯ ВЫЧИСЛИТЕЛЬНАЯ СЕТЬ персональных компьютеров, работающая **с единой базой** данных.

#### MIMD-

включает **ВСЕ УРОВНИ** параллелизма (**от конвейера** операций **до независимых** заданий и **программ**)

ВС, построенные на основе MIMD-архитектуры :

1. ВС с **СИЛЬНОЙ СВЯЗЬЮ** процессора с оперативной памятью ВС.

SMP-вычислители - **ОБЩАЯ** разделяемая память, имеющая единое адресное пространство.

При **увеличении числа процессов** становится **узким местом системы** процессов не больше нескольких десятков.

1. ВС со **СЛАБОЙ СВЯЗЬЮ** процессора с оперативной памятью ВС

МРР-вычислители -ВС с **распределенной** памятью **по узлам.**

**В каждом узле:**

время доступа к ЛОКАЛЬНОЙ памяти = **t**

время доступа к УДАЛЕННОЙ памяти = **Т**

// где Т>>t

**УЗЛЫ** объединяются между собой **КОММУТАЦИОННЫМИ** сетями регулярной структуры :

* ДЕРЕВО
* КОЛЬЦО
* ГИПЕРКУБ
* РЕШЕТКА
* ТОР

# 2. Особенности построения CISC архитектуры.

**CISC-архитектура** – это концепция проектирования процессоров, которая характеризуется следующим набором свойств:

1. **Короткая программа** - длинные команды.
2. **Пе**ренос вычислений ПРОГРАММНОГО уровня на АППАРАТНЫЙ.

**Результат**  форматы команд П-П, Р-Р, П-Р.

1. **ПРИНЦИП “80/20”:**

**“80%** кода команд используют **только 20%** ПРОСТЫХ команд **РЕГИСТР-РЕГИСТР** полного набора CISC-инструкций.”

1. Арифметические действия кодируются в **одной** команде.
2. **Небольшое число регистров**, каждый из которых выполняет строго определённую функцию.

**Любой CISC-процессор** УСТУПАЕТ **RISC-процессорам** по количеству выполняемых операций в секунду.

# 3. Особенности построения RISC-архитектуры.

**RISC-архитектура -**

Архитектура процессора, в которой **быстродействие увеличивается** за счёт УПРОЩЕНИЯ ИНСТРУКЦИЙ, чтобы их декодирование было более простым, а **время** выполнения — **меньшим**.

**ПРИНЦИПЫ RISC-архитектуры:**

1. Длинная программа - **короткие команды**
2. Любая операция **выполняется за ОДИН** такт.
3. Система команд должна содержать минимальный набор **ПРОСТЕЙШИХ КОМАНД** (инструкций) с одинаковой длиной и минимумом адресных форматов.
4. **ОБРАБОТКА** данных **только** в формате **“РЕГИСТР-РЕГИСТР”.**

Обмен “РЕГИСТР-ПАМЯТЬ” выполняется только с помощью команд “ЗАГРУЗКИ-ЗАПИСИ”

1. **Состав команд** должен быть **УДОБЕН** для компиляции операторов языков высокого уровня.

**Достижение возможного максимума производительности путем использования структурных приемов:**

ПОЛНОРАЗРЯДНАЯ ОБРАБОТКА ДАННЫХ -

Каждый арифметический **конвейер** за ОДИН **ТАКТ** формирует ОДИН **64-разрядных результат** (один такт = одно слово).

Конвейерная организация арифметических и других исполнительных устройств.

Суперскалярная структура процессора с

* комплексом средств динамического предсказания ветвлений
* большим числом регистров (min 32)
* многоуровневым КЭШем

**Набор команд** РАСШИРЯЕТСЯ за счёт **ПЕРЕРАСПРЕДЕЛЕНИЯ** площади кристалла.

# 4. КОНВЕЙЕРИЗАЦИЯ. ФАЗЫ КЛАССИЧЕСКОГО КОНВЕЙЕРА.

**КОНВЕЙЕРИЗАЦИЯ**- это параллельное выполнение всех ступеней обработки в пределах одного и того же процесса.

Каждая СТУПЕНЬ обрабатывает свою ФАЗУ **независимо**.

Различные КОМАНДЫ - различное ВРЕМЯ выполнения различных фаз.

Основные ступени обработки конвейера:

1. **ЧТЕНИЕ** КОМАНД
2. **ДЕШИФРОВАНИЕ**
3. **ЧТЕНИЕ** ОПЕРАНДОВ
4. **ИСПОЛНЕНИЕ**
5. **ЗАПИСЬ РЕЗУЛЬТАТА**

В большинстве классических СISC-архитектур имеется пять ступеней обработки конвейера:

1) Выборка команды

2) Декодирование команды

3) Адресация и выборка операндов

4) Исполнение команды

5) Запись результата

9. Методы предсказания ветвлений.  
 Существует два основных метода предсказания переходов — **статический** и **динамический**. **Статические методы** предписывают всегда выполнять или не выполнять определенные типы переходов. **Динамические методы** оценивают поведение команд перехода за предшествующий период времени

# 11. Переименование регистров.

**Переименование регистров** – нейтрализация ложных взаимозависимостей путем переименования регистров.

Причины **ограничения производительности** процессора :

1. **ИСТИННАЯ** взаимозависимость данных -

ВХОДНЫЕ данные **для одной** операции зависят от РЕЗУЛЬТАТА **другой** операции.

“ЧТЕНИЕ ПОСЛЕ ЗАПИСИ”

1. **ЛОЖНАЯ** взаимозависимость данных:

а) ВЗАИМОЗАВИСИМОСТЬ **ПО ВЫХОДУ:**

Команды, следующие друг за другом записывают результат в **один и тот же** РЕГИСТР

“ЗАПИСЬ ПОСЛЕ ЗАПИСИ”

б) **АНТИ** ВЗАИМОЗАВИСИМОСТЬ:

**Команда 1** может **испортить** данные, необходимые для ВХОДА **команде 2.**

“ЗАПИСЬ ПОСЛЕ ЧТЕНИЯ”

!! Частое **использование ОДНИХ регистров** приводит к возникновению ВЗАИМОЗАВИСИМОСТЕЙ и **СНИЖЕНИЮ ПРОИЗВОДИТЕЛЬНОСТИ** !!

**CISC-**архитектуры имеют **небольшое число** архитектурных **регистров** → ЛОЖНЫЕ взаимозависимости возникают довольно **часто**.

Путь **повышения производительности** - нейтрализация возникновения ЛОЖНЫХ взаимозависимостей путем ПЕРЕИМЕНОВАНИЯ РЕГИСТРОВ:

1. **КОМАНДЕ 1** требуется **регистр** → процессор динамически **ставит в соответствие** этому АРХИТЕКТУРНОМУ (логическому) регистру один из множества ФИЗИЧЕСКИХ регистров.
2. **КОМАНДА 2** пытается обратиться к **этому же** регистру → процессор **ставит в соответствие** другой ФИЗИЧЕСКИЙ регистр.

# 12. Обход и продвижение данных.

переименование регистров не решает проблемы ложной взаимозависимости, поэтому используется **обход данных –** результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти. Так же используется **продвижение данных** - процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

Переименование регистров **не полностью исключает** возникновение ИСТИННЫХ взаимозависимостей.

Поэтому используются методы ОБХОДА и ПРОДВИЖЕНИЯ данных.

* **ОБХОД ДАННЫ**Х :

результат работы **КОМАНДЫ 1** сразу пересылается **КОМАНДЕ 2**.

Следовательно, **исключаются** ЗАДЕРЖКИ на **МОДИФИКАЦИЮ** и повторное **чтение**.

* **ПРОДВИЖЕНИЕ ДАННЫХ :**

процессор может **выполнять** некоторые команды **ПАРАЛЛЕЛЬНО**, немедленно передавая результаты работы команд.

Следовательно, **исключается** ожидание завершения операций ЗАПИСИ и ЧТЕНИЯ используемых операндов.

# 41. КЭШ память прямой и обратной записи.

Обратная запись дает инструкции процессору записать данные только в процессорный кэш, минуя основную оперативную память. Прямой способ записи (write-through) обеспечивает запись данных одновременно в кэш память процессора и в оперативную память системы. Первый способ отложенной записи избавляет от лишней нагрузки на оперативную память компьютера, хотя очень сложен в разработке и реализации

**Как загрузить объект в кэш? Чем отличается сквозная запись от обратной?**

С помощью методов сквозной и обратной записи.

1. **Метод сквозной записи** - – информация записывается в блок кэш-памяти и в оперативную память. Однако это снижает производительность системы, т.к. при каждой записи необходимо тратить время на запись в оперативную память. 2. **Метод обратной записи** - – информация записывается в блок кэш-памяти только при вытеснении его из кэша и при наличии в нем изменений. Для этого каждому блоку кэша ставится бит изменений, который устанавливается в случае записи новых данных. При замене старого блока информации кэша на новый из оперативки проверяется состояние соответствующею бита. Если он установлен, то сначала выполняется перепись блока из кэша в память, и только после этого в кэш помешается новый блок из памяти. Такой метод обеспечивает более высокую производительность вычислительной системы, так как количество измененных блоков обычно меньше числа операций записи в память.

# 42. Раздельная и совместная (с точки зрения инструкций и данных) КЭШ память.

Когда в микропроцессорах впервые стали применять внутреннюю кэш-память, ее обычно использовали как для команд, так и для данных; Такую кэш-память при­нято называть совместной. Сравнительно недавно стало обычным разделять кэш-память на две — отдельно для команд и отдельно для данных. Такая кэш-память получила название раздельной. Смешанная кэш-память обладает тем преимуществом, что при заданной емкости ей свойственна более высокая вероятность попаданий по сравнению с разделен­ной, поскольку в ней оптимальный баланс между командами и данными устанав­ливается автоматически. Так, если в выполняемом фрагменте программы обраще­ния к памяти связаны в основном с выборкой команд, а доля обращений к данным относительно мала, кэш-память имеет тенденцию насыщаться командами, и на­оборот.

С другой стороны, при раздельной кэш-памяти выборка команд и данных мо­жет производиться одновременно, при этом исключаются возможные конфликты. Последнее обстоятельство существенно в системах, использующих конвейериза­цию команд, где процессор извлекает команды с опережением и заполняет ими буфер или конвейер.

# 43. Включающий и исключающий методы построения КЭШ памяти.

Пусть кэш имеет включающую архитектуру. Рассмотрим, каким образом происходит запись данных из оперативной памяти в такой кэш. Если в такой системе кэш ­памяти при полностью заполненном кэше L2 процессор пытается загрузить еще одну кэш ­строку, то произойдет следующее. Обнаружив, что все кэш­ строки заняты, кэш L2 избавляется от наименее ценной из них, стремясь при этом найти линейку, которая еще не была модифицирована, поскольку в противном случае ее еще придется выгружать в оперативную память.

Затем кэш L2 передает полученные из памяти данные кэшу L1. Если кэш первого уровня также заполнен, ему приходится избавляться от одной из кэш ­строк по сценарию, описанному выше.

Таким образом, загруженная порция данных присутствует и в кэше L1, и в кэше L2.

В случае кэша, построенного по исключающей архитектуре, кэш L1 никогда не уничтожает кэш ­строки при нехватке места. Даже если кэш­ строки не были модифицированы, они вытесняются в кэш L2 на то место, где находилась только что переданная кэшу L1 кэш­ строка. То есть кэши L1 и L2 как бы обмениваются друг с другом своими кэш­ строками, благодаря чему кэш ­память используется весьма эффективно.

# 44. Факторы, влияющие на выбор размера КЭШ памяти процессоров и ядер.

Оценивая размер кэш-памяти, нужно учитывать характеристики процессора и круг решаемых им задач.

Кэш-память двуядерного процессора редко превышает 3 MB. Тем более, если его тактовая частота ниже 3 Ггц. Производители прекрасно понимают, что дальнейшее увеличение размера кэша такого процессора не принесет прироста производительности, зато существенно повысит его стоимость.

Другое дело высокочастотные 4-, 6- или даже 8-миядерные процессоры. Некоторые из них (например, Intel Core i7) поддерживают технологию Hyper Threading, обеспечивающую одновременное выполнение каждым ядром двух задач. Естественно, что потенциал таких процессоров не может быть раскрыт с маленьким кэшем. Поэтому его увеличение до 15 или даже 20 MB вполне оправдано.

В целом же, эксперименты свидетельствуют, что в среднестатистическом "домашнем" процессоре влияние размера кэша на производительность находится в пределах 10 %, и его вполне можно компенсировать, например, высокой частотой.

Эффект от большого кэша наиболее ощутим при использовании архиваторов, в 3D играх, во время кодирования видео. В "не тяжелых" же приложениях разница стремится к нулю (офисные программы, интернет-серфинг, работа с фотографиями, прослушивание музыки и др.).

Многоядерные процессоры с большим кэшем необходимы на компьютерах, предназначенных для выполнения многопоточных приложений, одновременного решения нескольких сложных задач.

# 45. КЭШ прямого отображения.

Основная идея прямого отображения (direct mapping) RAM на кэш-память состоит в следующем: RAM делится на сегменты, причем размер каждого сегмента равен размеру кэша, а каждый сегмент в свою очередь делится на блоки, размер каждого блока равен размеру кэш-линии. Блоки RAM из разных сегментов, но с одинаковыми номерами в этих сегментах, всегда будут отображаться на одну и ту же кэш-линию кэша. Адрес каждого байта представляет собой сумму порядкового номера сегмента, порядкового номера кэш-линии внутри сегмента и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера сегментов, а порядковые номера кэш-линий внутри сегментов и порядковые номера байт внутри кэш-линий — повторяются.

Таким образом нет необходимости хранить полный адрес кэш-линии, достаточно сохранить только старшую часть адреса.

# 46. Полностью ассоциативный КЭШ.

Основная идея полностью ассоциативного отображения (fully associative mapping) RAM на кэш-память состоит в следующем: RAM делится на блоки, размер которых равен размеру кэш-линий, а каждый блок RAM может сохраняться в любой кэш-линии кэша.

Адрес каждого байта представляет собой сумму порядкового номера кэш-линии и порядкового номера байта внутри кэш-линии. Отсюда следует, что адреса байт различаются только старшими частями, представляющими собой порядковые номера кэш-линий. Порядковые номера байт внутри кэш-линий повторяются.

# 47. Множественно ассоциативный КЭШ.

Множество строк кэша разбивается на A >= 1 строк. Между группами такой кэш представляет собой кэш прямого отображения, а внутри групп - кэш полностью ассоциативного отображения

**Множественный ассоциативный кэш**

Отличие множественного ассоциативного кэша от кэша прямого отображения в том, что индекс адресует номер группы, и адресуемый блок может располагаться в любой строке группы. Т.е. между группами множественный ассоциативный кэш является кэшем прямого отображения, а внутри группы — полностью ассоциативным. Из-за этого растет коэффициент удачных обращений к кэшу. Кроме того, количество сравнений старших разрядов адреса основной памяти с признаком из теговой памяти, в группах строк кэша, сокращается при ”промахе“, до размера группы. Оба этих фактора положительно влияют на производительность вычислительной системы. С ростом ассоциативности, растет частота ”попаданий“ в кэш, и расходы в виде количества сравнений признаков. Для снижения расходов при поиске нужной строки в каждой группе, аппаратно реализуется параллельное сравнение признаков, от этого дорожает кэш-память и растет ассоциативность.

# 48. Механизмы обеспечения когерентности КЭШ памяти, протокол MESI.

В современных компьютерах доступ к памяти могут одновременно иметь несколько независимых процессоров (ядер, трэдов). Каждый из них имеет свои приватные кэши, в которых хранятся копии необходимых линий, а некоторые из них при этом локально модифицированы. Встает вопрос, а что если одна и та же линия одновременно понадобится нескольким процессорам. Не сложно сделать вывод, что для корректной работы системы необходимо обеспечить единое пространство памяти для всех процессоров.

Для обеспечения этого были придуманы специальные протоколы когерентности. Когерентность кэша — свойства кэш-памяти, означающее целостность данных, хранящихся в локальных кэшах, разделяемой системы. Каждая ячейка кэша имеет флаги, описывающие, как ее состояние соотносится с состоянием ячейки с таким же адресом в других процессорах системы.

При изменении состояния текущей ячейки необходимо каким-то образом сообщить об этом остальным кэшам. Например, генерируя широковещательных сообщения, доставляемые по внутренней сети многопроцессорной системы.

Было придумано множество протоколов когерентности, отличающиеся алгоритмами, количеством состояний и, как следствие скоростью работы и масштабируемостью. Большинство современных протоколов когерентности представляют вариации протокола MESI

В данной схеме каждая линию кэша может находиться в одном из четырех состояний:

* Модифицированная (M) *англ.* modified. Таким флагом может быть помечена только линия в одном кэше. Данное состояние означает, что данная линия была изменена, но до памяти эти изменения еще не дошли. Хозяин такой линии может спокойно читать и писать в нее без опроса остальных.
* Эксклюзивная (E) *англ.* exclusive. Обозначенная таким флагом линия, так же, как и M-линия может находиться только в одном кэше. Содержащиеся в ней данные полностью идентичны данным в оперативной памяти. Записывать и читать из нее можно без внешних запросов, так как она хранится только в одном кэше. После записи такая строка должна быть помечена, как модифицированная.
* Разделяемая (S) *англ.* shared. Линия может одновременно содержаться в кэшах нескольких устройств и использоваться совместно. Запросы на запись в такую линию всегда идут на общую шину, что привод к тому, что все линии с таким адресов в остальных кэшах помечаются как недействительные. При этом содержание основной памяти также обновляется. Чтение же из такой линии, не требует ни каких внешних запросов.
* Недействительная (I) *англ.* invalid. Такая линия считается невалидной и попытка прочитать приведет к кэш-промаху. Линия помечается недействительной в случае если она пусти или содержит устаревшую информацию.

# 49. Организация оперативной памяти и методы адресации элементов (CAS, RAS).

# **Организация оперативной памяти**

Оперативная память хранит информацию непосредственно доступную процессору. Максимальная ёмкость оперативной памяти определяется разрядностью адреса и разрядностью минимально адресуемой единицы. Так при 32-х разрядном адресе и байтовой адресации максимальный размер ОП составляет 232 байт, т.е. 4Гбайт. Существует несколько вариантов организации оперативной памяти.

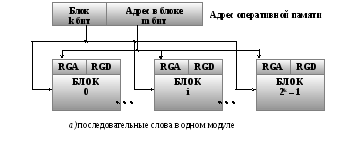
## **Блочная организация оперативной памяти**

Блочная организация оперативной памяти обеспечивает большее быстродействие, наращиваемость и большую живучесть. Классическая интерпретация адреса в этом случае предполагает, что старшие разряды определяют блок, а младшие – слово в блоке, как показано на рис.2 а).

На рис.2 б) приведен пример другого способа интерпретации адреса, когда номер блока указывается младшими разрядами адреса. Такой, более эффективный способ адресации называется *расслоением* или *чередованием адресов*(address interleaving). Если каждый блок является относительно автономным, имеет свой регистр адреса (RGA) и регистр данных (RGD), то, получив адрес и данные, он может выполнять затребованную операцию автономно, а следующее обращение по будет к другому блоку. В идеале такая память может принять столько запросов, сколько блоков. Реальная эффективность такого способа характеризуется коэффициентом расслоения – среднее количество запросов, которое может быть принято на обслуживание одновременно. Из таблицы 1 видно, что с ростом количества блоков абсолютный коэффициент расслоения растет, а относительный –снижается и для универсальных ЭВМ стремится к величине https://studfile.net/html/4018/180/html_lhEei1RBsr.nhMj/img-foOs3o.png0.4. Режим расслоения обращений позволяет использовать более медленную ОП. Например, при 8-ми блоках ОП может быть примерно в 3.5 раза медленнее источника запросов на обслуживание.

Таблица 1. Значения коэффициента расслоения.

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во блоков *N* | 2 | 4 | 8 |
| Коэф. рассл. *Кр* | 1.2 – 1.3 | 1.7 – 2.2 | 3.4 – 4.2 |
| https://studfile.net/html/4018/180/html_lhEei1RBsr.nhMj/img-W5xnCL.png | 0.6 – 0.65 | 0.425 – 0.55 | 0.425 – 0.525 |



## **Многопортовая оперативная память**

**Стандартное однопортовое ОЗУ имеет по одному входу адреса, данных и управления и в каждый момент времени обеспечивает доступ к ячейке памяти только одному устройству.**

**В отличие от стандартной ОП, в многопортовой памяти имеется несколько независимых наборов шин адреса, данных и управления,** гарантирующих одновременный и независимый доступ к ОП нескольким устройствам (естественно при блочной организации). Данное свойство позволяет существенно упростить создание многопроцессорных и многомашинных вычислительных систем, где многопортовая ОП используется совместно. В рамках одной ЭВМ подобное ЗУ может обеспечивать обмен информацией между ЦП и периферийными устройствами (например, контроллером магнитного диска) намного эффективней, чем прямой доступ к памяти. **Особенностью многопортовой памяти является наличие арбитра запросов.** Конфликты возникают при одновременном обращении через разные порты к одному и тому же блоку. Если же обращения осуществляются к разным блокам, то ОП обслуживает их одновременно.

## **Ассоциативные зу**

В рассмотренных ранее запоминающих устройствах поиск информации осуществляется по адресу ячейки. Другим способом является поиск по признаку (тегу), характеризующему хранимую информацию–ассоциативный поиск. Ассоциативный поиск может быть реализован программно в обычном адресном ЗУ. В ассоциативном ЗУ (АЗУ) ассоциативный поиск реализован аппаратно. Структура ассоциативного ЗУ приведена на рис.3.

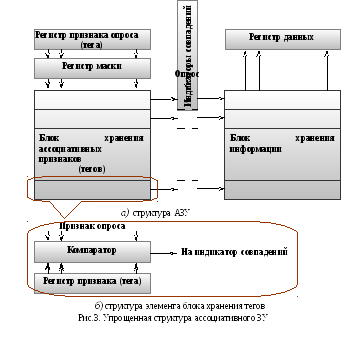
Регистр маски позволяет исключить из поиска определённые биты. На рисунке не показаны средства, обеспечивающие запись. Особенностями АЗУ являются:

* бόльшая емкость за счет блока хранения тегов;
* возможность многозначного ответа;
* возможность организовать сложный поиск.

Ёмкость АЗУ не должна превышать 2n информационных слов, где n–разрядность тега.

Многозначный ответ–это случай, когда несколько ассоциативных признаков совпадает с признаком опроса. В зависимости от области применения АЗУ подобная ситуация обрабатывается дополнительными средствами, например, с помощью регистра маски или является ошибкой, недопустимой.

Сложный поиск имеет целью решение общей задачи поиска - найти слова ассоциативные признаки (теги) которых удовлетворяют заданной комбинации логических отношений к признаку опроса (максимален, минимален, ближайший больший, ближайший меньший, больше заданного, меньше заданного, лежит внутри или вне заданных пределов). Одной из основ сложного поиска является упорядоченная выборка–выборка в порядке возрастания (или убывания) численного значения ассоциативных признаков (тегов).

Существенные отличия в архитектурах АЗУ могут быть связаны с выбранным принципом записи информации. Запись может осуществляться несколькими способами: по признаку (как чтение), по адресу, в первую свободную ячейку, запись с сортировкой по величине ассоциативного признака. В последнем случае местоположение ячейки, куда будет помещено новое слово, зависит от соотношения ассоциативных признаков вновь записываемого слова и уже хранящихся в АЗУ. Из-за относительно высокой стоимости АЗУ редко используется как самостоятельный вид памяти.

**За счет чего ускорить доступ к Оперативной памяти?** За счет Конверизации. Доступ организуется в конвейерном типе, адресуется байт, чтение байта, пока не считаетсч слово. Параллельное выполнение всех  ступеней обработки в пределах одного и того же процесса.

**Как соотносятся Risk и Sisd** риск представитель класса сисд архитектуры.

**Что произойдет с кэшем если увеличится объем кэшируемой памяти**

С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля ’’номер байта в блоке” растет, а объем признакового поля не изменяется.

**Какие поля есть в кэше (в строке кэша)?** Поля: признак, блок данных из памяти, бит грязи, бит достоверности, бит актуальности

**Бит грязи** говорит о том, была ли изменена строка или нет.

**Чем определяется быстродействие оперативной памяти?** Тип, частота, объем.

**В чем определяется быстродействие шинного интерфейса?** Разрядностью шины и тактовой чистотой

**Что такое когерентность?** Когерентность кэша — свойства кэш-памяти, означающее целостность данных, хранящихся в локальных кэшах, разделяемой системы.

**Прием для быстрого выбора команд**

Кэширование и ускорение работы памяти, которая в конвейерном виде позволяет считывать байты одного слова.

**От чего зависит пропускная способность шины**

Измеряется в килобайтах (мегабайтах) в секунду и определяется ее тактовой частотой и разрядностью (шириной) шины данных.

**От чего зависит объем кэшированной памяти?** От разрядности признака памяти кэша

**Для чего нужна векторная обработка?**

Для обработки циклов и итераций.

**На уровне какого параллелизма используется многофункциональная обработка**

На уровне операторов и команд

**Имеется кэш, емкость которого увеличилась, что изменилось**

Увеличилось кол-во строк.

**Увеличилась ассоциативность в 2 раза что изменилось?**

С ростом ассоциативности кэша разрядность признакового поля растет, а разрядность индекса падает.

**Что произойдет с кэшем если увеличится объем кэшируемой памяти**

С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля ’’номер байта в блоке” растет, а объем признакового поля не изменяется.

**Что будет происходить при увеличении длины кэшируемой строки**

Разрядность индекса уменьшится, размер поля увеличится, размер признака увеличится.

**Емкость оперативной памяти возросла, что изменилось в архитектуре кэша?** Признаковое поле возросло добавился еще один разряд.

**На базе каких устройств битовая ячейка** Транзистр и кондесатор

**Как определяется претендент на вытеснение в случае не обнаружения объекта в кэше? (бит частоты использования блока)**

При организации с прямым отображением Выбирать просто нечего: на попадание проверяется только один блок и только он может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Для замещения блоков применяются две основных стратегии: случайная и LRU. В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех (LRU – Least-Recently Used).

**Сколько в группе строк кэша?**

Столько же, чему равен коэффициент ассоциативности(А). А = 1 множественный ассоциативный кэш превращается в кэш прямого отображения, а при А, равном числу строк в кэше, - в полностью ассоциативный кэш. кэш прямого отображения содержит  столько групп, сколько блоков в кэше. Соответственно с ростом  ассоциативности кэша размерность признакового поля растет, а  разрядность индекса падает

**Достоинства и недостатки раздельного кеша?**

+выборка команд и данных может производиться одновременно, при этом исключаются возможные конфликты.

- низкая вероятность попаданий по сравнению с совместным

**Какая архитектура самая эффективная?**

Я думаю, что RISC-архитектура - Архитектура процессора, в которой быстродействие  увеличивается ​за счёт ​УПРОЩЕНИЯ ИНСТРУКЦИЙ​, чтобы их  декодирование было более простым, а ​время ​выполнения — меньшим​. Любой CISC-процессор​ УСТУПАЕТ ​RISC-процессорам​ по количеству  выполняемых операций в секунду. **Принципы выбора архитектуры.**

Из всех возможных вариантов построения системы лучшим является тот, который обеспечивается наиболее простой архитектурой.

1. Производительность компьютера определяется главным образом двумя характеристиками – тактовой частотой и разрядностью обработки. Конвейерная организация процессора “работает” на усиление значимости тактовой частоты и является необходимым атрибутом современных компьютеров.

2. Для высокопроизводительных вычислений целесообразно применять RISC-процессоры или процессоры с RISC-ядром. Системы на их основе специально ориентированы на повышение тактовой частоты обработки и “прозрачность” параллелизма архитектуры для прикладного программиста. 3. Предпочтительней использовать один быстрый однопроцессорный компьютер, чем много медленных или многопроцессорную систему с маломощными процессорами.

4. Проблему создания необходимого количества рабочих мест лучше решать с помощью многопользовательских систем на базе сервера и терминалов, чем с помощью ЛВС персональных компьютеров.

5. Вычислительные системы с массовым параллелизмом следует использовать только при полной уверенности в реально существующем параллелизме приложений и только при возможности привлечения программистов высшей квалификации.

**Отличие SMP от MPP**

SMP (symmetric multiprocessing) – симметричная многопроцессорная архитектура. Главной особенностью систем с архитектурой SMP является наличие общей физической памяти, разделяемой всеми процессорами.

MPP (massive parallel processing) – массивно-параллельная архитектура. Главная особенность такой архитектуры состоит в том, что память физически разделена.

Отсутствие общей памяти в MPP архитектуре значительно снижает скорость межпроцессорного обмена, в отличие от SMP архитектуры, в которой общая память присутствует.

Каждый процессор в MPP архитектуре может использовать только ограниченный объем банка памяти, в то время,как процессор в SMP архитектуре охватывает полный объем.

В следствии недостатков MPP архитектура трудно реализуема, что приводит к значительным затратам, реализация SMP архитектуры, напротив, больших затрат не требует.

Главным преимуществом систем с раздельной памятью является хорошая масштабируемость: в отличие от SMP-систем, в машинах с раздельной памятью каждый процессор имеет доступ только к своей локальной памяти, в связи с чем не возникает необходимости в потактовой синхронизации процессоров.

**Представители вычислителей класса MIMD**

**МIМD - архитектура** включает все уровни параллелизма от конвейера операций до независимых заданий и программ. Любая вычислительная система этого класса в частных приложениях может выступать как SISD (одиночный поток команд и данных) или SIMD (одиночный поток команд и множество потоков данных) - система.

**Прием неупорядоченной обработки команд**

Позволяет процессору “ выдавать" и “ обрабатывать“ команды также не в порядке их следования по программе. Для реализации данной обработки требуется **буфер команд** между ступенями декодирования и исполнения конвейера. Этот прием используется для повышения производительности за счет увеличения числа исполняемых команд за такт.

**Превращение CISC архитектуры в RISC**

Основа успеха микропроцессора на массовом рынке — сохранение совместимости с существующими и будущими приложениями. Для x86 это трудная задачи из-за сложной системы команд, особенностей регистров системного уровня и механизмов защиты памяти и механизмов обработки внешних событий. Решением стало преобразованием x86 команд во внутренние RISC команды.

Главная проблема декодирования CISC команд заключается в определении границ команд и в определении наиболее эффективной последовательности выполнения дешифратором. AMD решили проблему путем распознания сложных команд до стадии декодирования. После определения сложных команд создаются биты предварительного декодирования. Каждый байт в кэш-памяти имеет биты данных декодирования с адресами соответствующих RISC-подобных операций. Процесс предварительного декодирования, происходящий при выборке команд из оперативной памяти в кэш команд, практически не снижает производительности системы, но требует дополнительной кэш-памяти.

**Что делать для того, чтобы снизить время выполнения фаз конвейера?**

Кроме того, производительность может понижаться из-за:

1) Взаимозависимость по выходу или “запись после записи” возникает при выполнении двух, следующим друг за другом команд, которые записываются в 1 и тот же регистр. Процессор при этом должен гарантировать правильность модификации регистра, даже если команды, модифицирующие его, выполняются не по порядку;

2) Анти-взаимозависимость, или взаимозависимость “ запись после чтения“ возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой.

**Факторы, нарушающие непрерывность конвейерной реализации команд:** 1) Когда для выполнения следующей команды требуется результат действий, реализованных предыдущей командой, или, когда предыдущей командой определяется операнд следующей команды, следовательно, возникает задержка выполнения след. команды.

2) когда в кэш-памяти отсутствуют требуемые данные или команды, необходимо передать их в кэш-память из основной памяти;

3) когда предшествующая команда изменяет содержание следующей или когда изменяется содержимое регистра состояния программы, задающего область выполнения программы, следующая команда должна ожидать завершения предшествующей команды;

4) в случае возникновения прерывания и перехода к программе его обработки (конвейер загружается командами обработки прерывания)

5) когда операция, реализуемая машинной командой, имеет сложный характер и требует для выполнения много машинных циклов, последующая команда долго не может достичь стадии выполнения операции.

**Что такое VLIW?**

**VLIW** – архитектура с очень длинным командным словом. На этапе компиляции несвязанные операции группируются в пакеты, содержимое которых строго соответствует структуре процессора. Таким образом, в рамках данной архитектуры компилятор находит в исходном коде инструкции, выполнять которые можно параллельно, и создает машинный код, использующий этот параллелизм.

**Скалярный процессор** – процессор с единственным конвейером.

**Суперскалярный процессор** – процессор с несколькими конвейерами. Двух потоковые – dual-issue. Четырех потоковые – quad-issue.

**Кластеры**

Кластеры – разновидность MPP-систем.

Кластер состоит из двух или более узлов, удовлетворяющих следующим требованиям:

1. Каждый узел работает со своей копией ОС.

2. Каждый узел работает со своей копией приложения.

3. Узлы делят общий пул других ресурсов (в кластерах приложений). Отдельные экземпляры кластера должны знать о работе друг друга.

**Блок вычислений с плавающей точкой**

Операции с плавающей точкой имеют высокую трудоемкость, поэтому блоки вычислений с плавающей точкой реализуются в виде многоступенчатых конвейеров. SIMD-реализации обработки векторных данных.

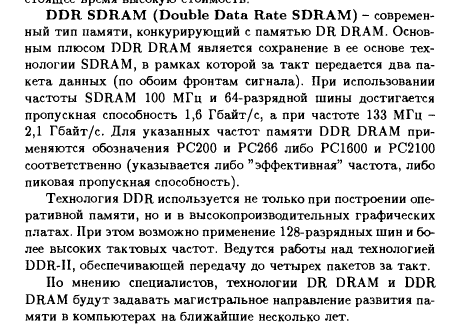
**Кэш память микропроцессора**

Реализация на кристалле раздельного кэша данных и команд. С ростом размера кристалла растет число брака.

50. Синхронный и асинхронный принципы работы оперативной памяти.

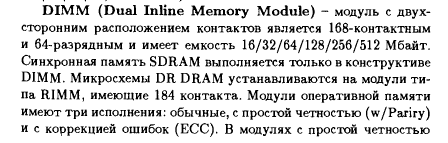
51. SDR, DDR, DDR2, DDR3 памятью. C126-131

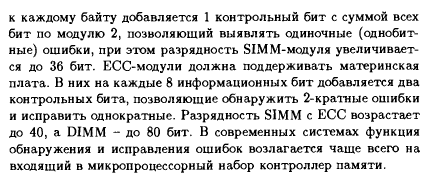
DDR:



52. Регистровая и буферизируемая память (registered DIMM, FB-DIMM).

C 131:





53. Методы обеспечения отказоустойчивости оперативной памяти.

54. Физическая организация дисковой памяти.

55. Физическая и логическая адресация дисковой памяти.

56. Методы обеспечения отказоустойчивости физической памяти, уровни RAID.

**Уровни RAID**

На уровне дисковой подсистемы предпринимаются технические меры, направленные на снижение потерь до стоимости вышедшего из строя оборудования. В основе таких мер лежит резервирование и избыточность, реализуемые в виде массива независимых резервных дисков —**RAID.**

Всего выделяют 9 уровней RAID-массивов, различающихся по скорости, надежности и стоимости изготовления.

**RAID 0** — дисковый массив без дополнительной отказоустойчивости

**RAID 1** – дисковый массив с зеркалированием данных

**RAID 2** — дисковый массив с использованием алгоритма Хэмминга для проверки/восстановления данных

**RAID 3** — дисковый массив с вычислением контрольной суммы параллельно с передачей данных

**RAID 4** — дисковый массив с независимыми дисками данных и общим диском для хранения контрольных сумм

**RAID 5** — дисковый массив с независимыми дисками данных и равномерным распределением контрольных сумм между дисками

**RAID 6** — дисковый массив с независимыми дисками данных и двумя независимыми схемами контрольных сумм, распределенными между дисками

**RAID 7** — дисковый массив с асинхронным вводом-выводом и высокой скоростью передачи данных

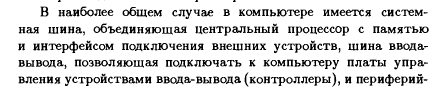
**RAID 10** — комбинация технологий RAID 1 и RAID 0 в одном дисковом массиве

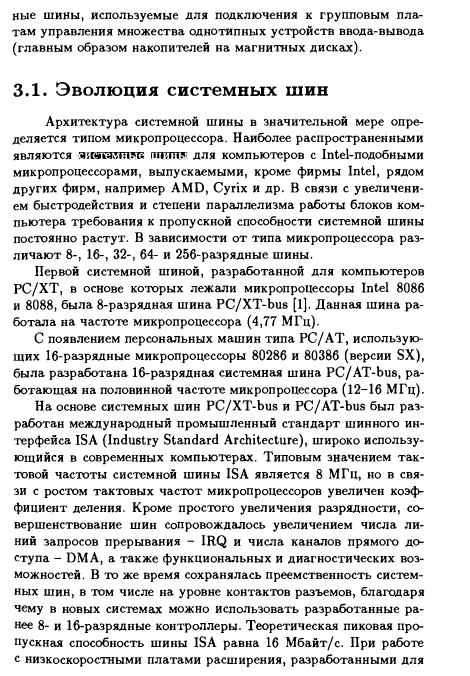
**RAID 53** — комбинация технологий RAID 0 и RAID З в одном дисковом массиве

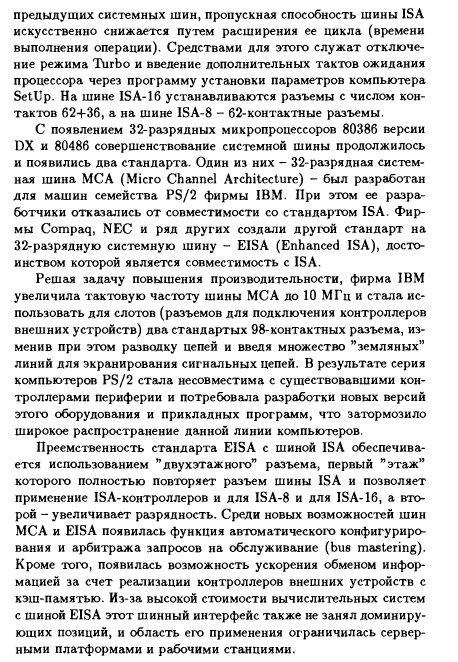
57. Классификация шин по логическим и физическим топологиям.

58. Классификация шин по функциональному назначению.

59. Системные шины. C77-80







60. Шины ввода/вывода.

**ШИНЫ ВВОДА/ВЫВОДА**

В современных вычислительных системах шину ввода-вывода называют также шиной РАСШИРЕНИЯ, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств.

61. Периферийные шины.

**ПЕРИФЕРИЙНЫЕ ШИНЫ**

Периферийные шины являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы

**Различают шину данных, шину адреса, шину управления и состояния. Шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера**

62. Виртуализация: определение и методы.

63. Аппаратная виртуализация.

64. Виртуализация средствами ОС.

65. Виртуализация средствами гипервизоров.

66. Виртуализация серверных групп и ЦОД с использованием решения vSphere от компа-нии VM-ware

5. Зависимость длины конвейера и тактовой частоты процессора.

6. Явления, оказывающие негативное влияние на конвейеризацию.

7. Суперконвейеризация, гиперковейеризация.

Суперскалярная архитектура и методы ее построения.

10. Предикативное и спекулятивное исполнение инструкций

# 13. Неупорядоченное исполнение.