# 1. Понятие архитектуры компьютера (с примерами архитектур)

Под **архитектурой компьютера** понимается совокупность взаимосвязанных сведений о способах представления в компьютере программ и данных, о назначении, структуре и особенностях функционирования отдельных его устройств, а также об организации компьютера в целом.

**Примеры архитектур компьютера:**

По принципу разделения памяти:

- **Архитектура фон Неймана**. Она включает в себя центральный процессор (CPU), память, внешние устройства ввода/вывода и шину для передачи данных между ними. Основные принципы включают хранение программ и данных в одной памяти, последовательное выполнение команд и использование арифметическо-логического блока для выполнения операций.характерно. Используется в процессорах общего назначения(в том числе и во всех современных компьютерных процах amd и intel)

- **Архитектура Harvard:** Эта архитектура разделяет память на отдельные блоки для программ и данных, что позволяет параллельную обработку. Она имеет отдельные пути и устройства для работы с инструкциями и данными. Используется в микроконтроллерах, сигнальных процессорах. самый популярный процессор с данной архитектурой : семейство микроконтроллеров PIC

Примеры архитектур по типу применяемого процессора:

- **Архитектура RISC**. RISC-архитектура характеризуется набором простых и быстрых инструкций, которые выполняются за один такт. Она стремится к упрощению и оптимизации выполнения инструкций, уменьшая сложность процессора. Используется в мобильных устройствах, встраиваемых системах и других системах с ограниченными ресурсами, а также пк. пример:MIPS R2000.

- **Архитектура CISC**. CISC-архитектура характеризуется набором более сложных инструкций, которые могут выполнять различные операции за один такт. Она обладает богатым набором инструкций, что позволяет выполнять сложные операции более эффективно. буквально все процессоры intel и amd(современные так точно)

- **Архитектура ARM:** ARM является популярной архитектурой для мобильных и встраиваемых систем. Она основана на RISC-принципах и широко используется в процессорах с низким энергопотреблением и высокой производительностью. пример мобильные процессоры а также ноутбучные от apple

Каждая архитектура компьютера *определяет способ организации и взаимодействия компонентов компьютера* для обеспечения выполнения задач и операций.

# 2. Преобразование информации и представление данных в компьютерных системах, числовые, буквенные данные, изображения (с примерами)

При компьютерной обработке информации приходится иметь дело с текстовыми, графическими, числовыми, звуковыми и другими данными. **Для хранения данных различной природы применяются различные способы их представления в двоичном алфавите – различные способы кодировки.** Кроме того, *для одной и той же* разновидности данных также *могут использоваться различные способы кодировки*, которые отличаются друг от друга эффективностью и различными требованиями к ресурсам компьютера.

**1. Числовые данные:**  
***- Описание***: Числовые данные представляют числа и математические значения.

***- Представление:*** Последовательность двоичных цифр (битов). Для представления отрицательных чисел используются дополнительный код или обратный код.

***- Примеры***: Целые числа, вещественные числа, десятичные числа и другие математические значения представляются в виде числовых данных. Например, число 5 может быть представлено в двоичном виде как 101.

**2. Буквенные данные:**

***- Описание:*** Буквенные данные представляют символы, буквы и текст.

***- Представление:*** Буквенные данные могут быть представлены с использованием различных кодировок, таких как ASCII или тUnicode. Каждый символ или буква соответствует числовому значению, которое может быть представлено в виде числовых данных.

***- Примеры:*** Буквы алфавита, цифры, знаки пунктуации и специальные символы представляются в виде буквенных данных. Например, буква "A" может быть представлена числовым значением 65 в кодировке ASCII.

**3. Изображения:**

***- Описание:*** Изображения представляют визуальные данные, такие как фотографии, рисунки и графики.

***- Представление:*** Изображения могут быть представлены в виде пикселей, где каждый пиксель содержит информацию о цвете и яркости. Различные форматы файлов, такие как JPEG, PNG или GIF, используют разные алгоритмы сжатия и методы представления цвета для хранения и передачи изображений.

***- Примеры:*** Фотографии, иллюстрации, диаграммы и другие визуальные элементы представляются в виде изображений. Каждый пиксель изображения содержит числовое значение, которое определяет его цвет и яркость.

*В зависимости от типа данных и их специфических требований могут использоваться различные методы и форматы представления информации.*

# 3. Преобразование информации и представление данных в компьютерных системах, аудио и видеоданные, надежность кодирования, базовые принципы кодирования (с примерами)

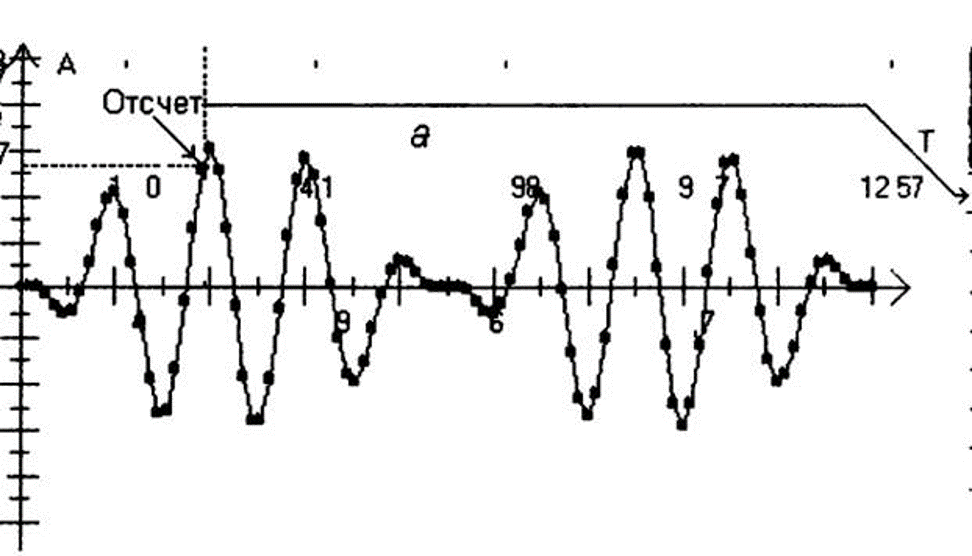
**Теорема Котельникова** гласит о том, что непрерывный сигнал с ограниченным спектром можно точно восстановить по его дискретным отсчётам, если они были взяты с частотой дискретизации, превышающего максимальную частоту сигнала минимум в два раза.

**Инфа Маркова**

**A) Звук** представляет собой довольно сложное, непрерывное колебание воздуха. Основной подход к кодированию звука, который называется **преобразованием в цифровую форму, оцифровыванием или дискретизацией,** основан на том, что:

1. ***непрерывный звуковой сигнал заменяется дискретным*** (то есть состоящим из обособленных, раздельных элементов) набором значений сигнала – отсчетов сигнала – **в некоторые последовательные моменты времени.**
2. Каждый отсчет кодируется в одном из рассмотренных ранее числовых форматов.
3. Таким образом, ***кодирование и обработка звуковых данных*** *фактически* ***сводятся к кодированию и обработке числовых данных.***

В соответствии с этим определением ***количество отсчетов сигнала в единицу времени*** называется **частотой дискретизации.** Так, на рис. 2.5 сигнал, длящийся 2 с, заменяется 100 отсчетами (жирные точки на графике амплитуды сигнала). Следовательно, в данном случае частота дискретизации равна 50 Гц. При записи звука в мультимедийных технологиях применяются частоты 8, 11, 22 и 44 кГц. Частота дискретизации 44 кГц означает, что одна секунда непрерывного звучания заменяется набором из сорока четырех тысяч отдельных отсчетов сигнала. **Чем выше частота дискретизации, тем лучше качество оцифрованного звука.** В последних разработках частота дискретизации достигает 192 кГц.

**

Каждый **отдельный отсчет** представляет собой **число**, которое затем можно представить в виде некоторого двоичного кода. Качество преобразования звука в цифровую форму определяется не только частотой дискретизации, но и количеством битов памяти, отводимых на запись кода одного отсчета. Этот параметр принято называть **разрядностью преобразования**. В настоящее время обычно используются разрядности 8, 16, 24 бит. На описанных принципах основывается формат **WAV** (от WAVeform-audio – волновая форма аудио) кодирования звука.

**Б) Кодирование видеоинформации** – еще более сложная проблема, чем кодирование звуковой информации, так как нужно позаботиться *не только о дискретизации непрерывных движений,* ***но и о синхронизации*** *изображения со звуковым сопровождением.* В настоящее время для этого используется формат, который называется **AVI** (Audio-Video Interleaved – чередующиеся аудио и видео). Отметим, что основные мультимедийные форматы AVI и WAV очень требовательны к памяти. Поэтому **на практике применяются различные способы компрессии (сжатия) звуковых и видеокодов.**

*Принцип обезличивания кода*

Для компьютера не существует разницы между кодами данных различной природы. Интерпретация (истолкование смысла) машинного кода может быть самой разной. Один и тот же код разными программами может рассматриваться и как число, и как текст, и как изображение, и как звук. Другими словами, **как именно трактуется, понимается тот или иной машинный код, определяется обрабатывающей этот код программой.**

**Принцип обезличивания машинного кода** состоит в том,

1. ***Что один и тот же код*** может восприниматься процессором компьютера ***в любом*** из используемых в аппаратных и программных средствах ***способов кодирования***.
2. Конкретная трактовка кода определяется обращающейся к коду программой.

*Надежность кодирования*

Одним из важнейших аспектов организации хранения и передачи кодов данных является обеспечение их надежности и безошибочности. ***Существует множество различных факторов***, воздействие которых на аппаратные средства компьютера приводит к искажению кодов данных. К ним относятся, **например**,

* влияние сильных электромагнитных полей от источников, находящихся вблизи запоминающих, передающих устройств и линий связи,
* влияние температурного расширения материалов, из которых состоят запоминающие устройства,
* сбои в работе аппаратуры,
* механические удар и т.д.

Для защиты данных от большинства таких факторов к настоящему времени уже разработан и внедрены достаточно надежные средства. Вместе с тем, применяя специальные методы кодирования, можно обеспечить контроль за появлением ошибки и даже восстановление исходного кода после ее обнаружения. **Применение этих методов позволяет снизить вероятность ошибки до 10^-9 и ниже.**

**Общим методом обеспечения надежности хранения и передачи данных в компьютере и по линиям связи является включение в код дополнительных контрольных разрядов (битов).** Существует много различных вариантов этого метода.

Во многих компьютерах **одним из уровней контроля** за появлением ошибок является **включение в байт дополнительного, контрольного девятого разряда, который формируется аппаратурой автоматически.** А те восемь битов, из которых, как мы до сих пор считали, состоит байт, в дальнейшем будем называть **информационными**. Значение контрольного разряда определяется так, чтобы общее количество разрядов байта (всех информационных и контрольного), которые содержат 1, было нечетным. Рассмотрим, например, байт с информационными битами 1001 11102. Количество единиц в нем нечетно, следовательно, аппаратура сформирует значение контрольного бита, равное 0. При этом общее количество единиц в девяти битах байта останется нечетным. А, например, для байта с информационными битами 1001 01102­, количество единиц в котором четное, контрольный бит окажется равным 1 – в результате образуется нужное нечетное количество единиц.

*Кроме введения контрольного разряда существуют и более развитые способы кодирования*, которые не только позволяют установить факт появления ошибок большой кратности, но и обеспечивают их исправление. К таким способам относятся коды, принцип построения которых в 1948 г. предложил **Р. Хемминг.,** обеспечивающий однозначное обнаружение и исправление ошибок в кодах данных. Он включает в себя следующие действия:

1. Контрольные биты вставляются в исходный код и нумеруются совместно с информационными битами слева направо начиная с 1.

2. Контрольные биты располагаются в позициях с номерами n = 2k, k = 0, 1, 2, 3… объединенного кода, то есть в позициях с номерами n = 1, 2, 4, 8, 16 …

3. Для каждого контрольного разряда с номером n весь код делится на группы, состоящие из 2n битов.

4. Контрольный бит с номером n контролирует в группе первые n подряд расположенных битов кода (для первой группы, включая контрольный) с пропуском следующих n битов.

5. В каждой группе контрольный бит формируется так, чтобы общее количество единиц в ней было нечетным.

*Методы построения кодов Хемминга и некоторых других, более мощных способов кодирования, используются в различных устройствах памяти компьютера. В частности, на этих методах основан аппаратных механизм контроля оперативной памяти ECC (Error Checking and Correcting – контроль и исправление ошибок), который является дополнительным средством обеспечения высокой ее помехоустойчивости.*

***Инфа GPT***

**2. Надежность кодирования:**

**- Проверочные суммы:** При передаче данных вычисляется контрольная сумма, которая представляет собой сумму всех битов данных. Получатель данных также вычисляет контрольную сумму и сравнивает ее с полученной. Если суммы не совпадают, значит, произошла ошибка.

**- Коды Хэмминга:** Это специальные коды, которые позволяют обнаруживать и исправлять ошибки в передаваемых данных. Они добавляют дополнительные биты информации, которые позволяют восстановить правильные данные в случае возникновения ошибок.

**3. Базовые принципы кодирования:**

- Для каждого типа данных существует множество форматов кодирования. **Выбор подходящего** формата зависит **от требований по использованию ресурсов, качества данных и совместимости.**

# 4. Понятие такта данных, базовое понятие синхронизации (с примерами)

**Инфа Маркова**

В выполнении действий над данными участвуют несколько устройств. Эти действия должны быть согласованы и синхронизированы друг с другом. Для синхронизации работы компьютера используется специальное устройство – **тактовый генератор,** который *через равные промежутки времени вырабатывает импульсы синхронизации – синхроимпульсы.*

Промежуток времени от начала одного импульса синхронизации до начала следующего за ним импульса называется **тактом**. Такты обладают равными длительностями с высокой степенью точности.

Выполнение процессором любых действий всегда происходит во время некоторой части такта, а за ней следует пауза, в течение которой ничего не происходит. **Наличие такой паузы является принципиальным фактором**, обусловленным физическими законами, которые управляют работой процессора.

Рассматривая выполнение процессором некоторой последовательности действий, можно считать, что любые действия, происходящие внутри такта, происходят мгновенно, в моменты времени t0, t1, t2, …, соответствующие границам тактов.

**Тактовая частота** равна **количеству тактов**, управляющих работой устройства, **в единицу времени.** Единицей измерения тактовой частоты является герц, равный одному такту в секунду.

Тактовая частота является одним из главных факторов, определяющих скорость обработки данных компьютером. Первые персональные компьютеры работали на частотах 5 – 10 МГц. В настоящее время процессоры компьютеров работают с тактовыми частотами от сотен мегагерц до 3 – 4 ГГц.

**Внутреннее состояние компьютера это:**

1. Двоичный код выполняемой программы

2 . Значения обрабатываемых дискретных данных

3. Набор дискретных значений и признаков, которые используются для управления процессом обработки.

С точки зрения введенного понятия внутреннего состояния компьютера, можно считать, что в моменты времени t0, t1, t2… происходит мгновенный переход компьютера из одного внутреннего состояния в другое – из состояния, соответствующего значениям обрабатываемых данных и признаков до выполнения действия, в состояние, соответствующее значениям данных и признаков после выполнения этого действия. Таким образом, **множество внутренних состояний компьютера также является дискретным.** В связи с этим говорят, что **компьютер** в целом является **дискретным устройством.**

Устройства, обеспечивающие работу с дискретными данными или сигналами, обладающие дискретным множеством внутренних состояний, называются **дискретными.**

**Инфа GPT**

**Такт данных (или тактовый сигнал)** - это базовый сигнал или импульс, который используется для синхронизации передачи и обработки данных в компьютерных системах. Он определяет момент времени, когда данные могут быть считаны или записаны.

**Примеры** использования **такта** данных:

**1. Шина данных:** В компьютерных системах шина данных используется для передачи информации между различными компонентами. Тактовый сигнал контролирует моменты времени, когда данные на шине становятся доступными для чтения или записи.

**2. Процессор:** Во внутренней архитектуре процессора тактовый сигнал используется для синхронизации выполнения инструкций и операций. Каждая инструкция обрабатывается в течение одного или нескольких тактовых циклов.

**3. Память:** При чтении или записи данных в память компьютера тактовый сигнал используется для синхронизации операций. Он определяет, когда данные должны быть записаны или считаны из определенных адресов памяти.

**Базовое понятие синхронизации: Синхронизация** - это процесс координации или согласования действий различных компонентов или устройств в компьютерной системе. Он обеспечивает правильную и последовательную передачу и обработку данных.

**Примеры** базовых методов **синхронизации**:

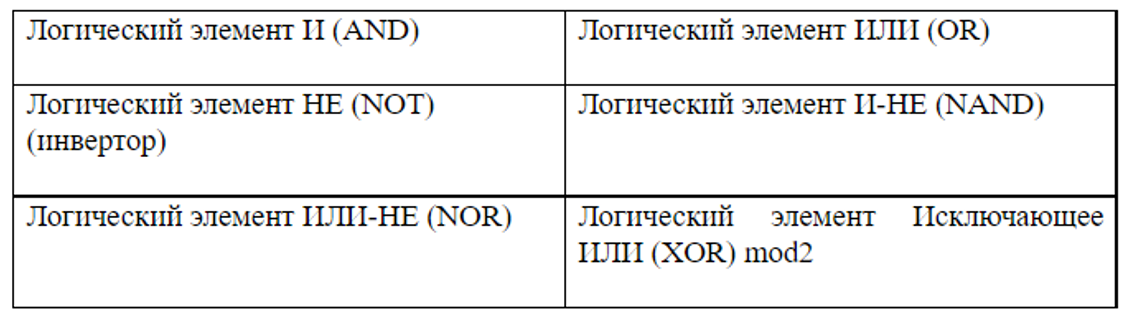
**1. Тактовый сигнал**: Как уже упоминалось, тактовый сигнал служит для синхронизации операций и передачи данных в компьютерных системах. Все компоненты и устройства синхронизируют свои действия с тактовым сигналом.

# 5. Базовые логические элементы, построение схем на базовых логических элементах, комбинаторика (с примерами)

**Из лекции**

Основными базовыми операциями, которые должен выполнять процессор компьютера, являются логические операции отрицание, дизъюнкция, конъюнкция, арифметическое сложение, сдвиг. Используемые для реализаций этих и других операций устройства принято называть вентилями (от немецкого Ventil – клапан)

Вентилем называется физическое устройство, реализующее одну из базовых логических операций: отрицание, дизъюнкцию, конъюнкцию, исключающую дизъюнкцию и т.д. Вентили, входящие в состав процессоров компьютера, называют также логическими элементами.



Все современные вентили реализуются на основе полупроводниковых устройств – **транзисторов**, характерное время срабатывания которых в настоящее время приближается к долям наносекунды (1 нс = 10-9 с).

В вентильных схемах на базе транзисторов двоичному знаку 0 соответствует низкое напряжение с уровнем от 0 до 1 В, а двоичному знаку 1 – высокое напряжение с уровнем от 2 до 5 В.

Вентили «НЕ», «НЕ И» и «НЕ ИЛИ», используемые для построения других вентилей и произвольных схем, считаются базовыми, а схемы, которые получаются с помощью всевозможных комбинаций базовых вентилей, принято называть **цифровыми логическими схемами**. Важным частным случаем цифровых схем являются **комбинационные схемы**, в которых значения, получаемые на выходах схемы, зависят только от значений, поступающих на ее входы. Такие схемы классифицируются также как **схемы без памяти**.

Для построения комбинационной схемы необходимо представить логическую функцию в виде логической формулы, провести минимизацию и получить правило соединения логических элементов. Эти вопросы были рассмотрены в других курсах.

Существуют **типовые комбинационные схемы**, например, компаратор, дешифратор, шифратор

**GPT**

Построение схем на базовых логических элементах:

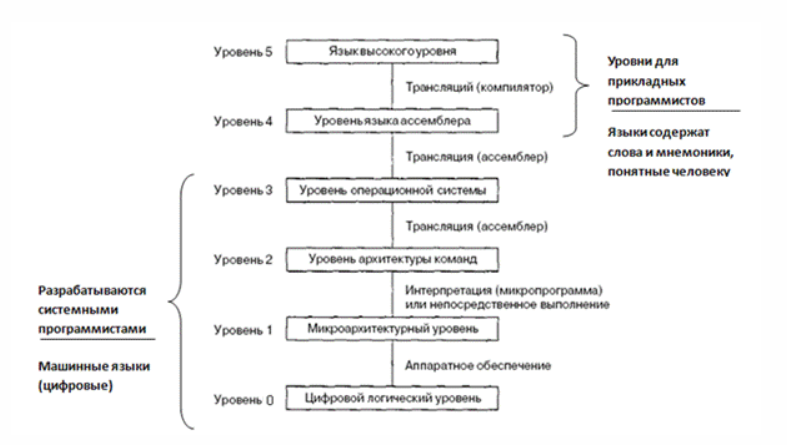
Схемы, построенные на базовых логических элементах, называются комбинационными схемами. В комбинационных схемах логические элементы соединяются в определенном порядке для получения желаемого результата.

Например, чтобы построить комбинационную схему для логической функции "ИЛИ" (OR) с двумя входами A и B, можно использовать базовые логические элементы И (AND) и НЕ (NOT). Первый И элемент принимает входные сигналы A и B, затем его выход подается на вход НЕ элемента, который инвертирует выходной сигнал. В итоге получается схема, которая выдает результат логической функции "ИЛИ" для входных сигналов A и B.

# 6. Понятие многоуровневой компьютерной организации (примеры организации уровней)

Многоуровневая компьютерная организация – подход, при котором рассматривается ряд абстракций, каждая из которых надстраивается над абстракцией более низкого уровня.

Деление на уровни является в некотором роде условным. Можно говорить о шести уровнях представления современного компьютера.



***Уровень 0 (Цифровой логический уровень):*** Объекты состоят из вентилей. Вентили образуют биты памяти, которые в свою очередь, образуют регистры.

***Уровень 1 (Микроархитектурный уровень):*** На этом уровне рассматриваются совокупности регистров, которые образуют локальную (регистровую) память и арифметико-логическое устройство (АЛУ), предназначенное для выполнения простых операций. Регистры вместе с АЛУ образуют тракт данных. Основная операция состоит в следующем: выбирается один или два регистра, АЛУ производит над ними некоторую операцию (например, сложение), результат помещается в один из регистров. В некоторых машинах тракт данных контролируется микропрограммой. На других контроль осуществляется аппаратными средствами.

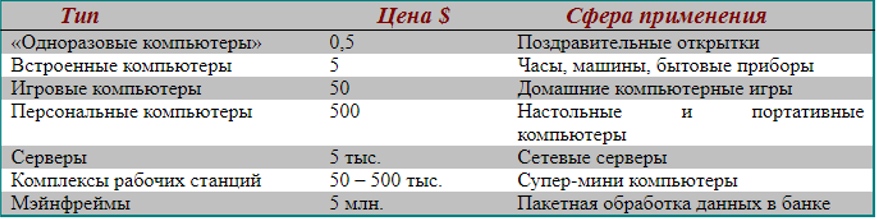
***Уровень 2 (Уровень архитектуры команд):*** Программный интерфейс между аппаратной частью компьютера (центральным процессором) и программным обеспечением (операционной системой и прикладными программами). Определяет, какие операции и функции могут выполняться процессором, какие регистры доступны для хранения данных и какие адресные режимы используются для доступа к памяти. Также определяет правила выполнения операций.

***Уровень 3 (Уровень операционной системы):*** Этот уровень включает набор команд уровня 2. Оставшаяся часть команд интерпретируется операционной системой. Особенности уровня: набор новых команд, собственная организация памяти, способность выполнять две и более программ одновременно и др.

***Уровень 4 (Уровень языка ассемблера):*** Представляет собой символьную запись языка более низкого уровня. Программа, которая выполняет трансляцию называется ассемблером.

***Уровень 5 (Уровень языка высокого уровня):*** Пятый уровень обычно состоит из языков, разработанных для прикладных программистов. Такие языки называются языками высокого уровня.

# 7. Типы компьютеров и компьютерных систем (с примерами) дописать про компьютерные системы (хз, где взять: в презентации нет)

****

1. ***Одноразовые компьютеры***

Одноразовые компьютеры обычно не предназначены для длительного использования и могут быть удобными в ситуациях, где требуется временное решение без сохранения данных или следов использования. Одноразовые виртуальные машины для тестирования программного обеспечения

***Встроенные компьютеры (микроконтроллеры)***

Они являются самостоятельными вычислительными устройствами, которые обычно встроены в другие устройства или системы для выполнения для управления ими.

Содержат процессор, память, устройства ввода-вывода.

1. ***Игровые компьютеры***

Это обычные компьютеры, в которых расширенные возможности графических и звуковых контроллеров.

1. ***Персональные компьютеры***Компьютеры, предназначенных для использования одним пользователем.
2. ***Серверы***

Серверы обычно предназначены для обслуживания относительно небольшого количества пользователей или приложений и выполнения запросов клиентов .Примеры: Dell PowerEdge,  
Некоторые серверы способны обрабатывать до миллиона транзакций в секунду.

1. ***Комплексы рабочих станций***

состоят из нескольких персональных компьютеров или рабочих станций. Эти компьютеры соединены высокоскоростной сетью и снабжены специальным ПО. Предназначены для решения «больших» задач..

1. ***Мэйнфреймы***

Это большие компьютеры, размером с комнату. Они обычно предназначены для обработки больших объемов данных и выполнения критически важных задач, таких как обработка транзакций в банковской системе. Серверы, с другой стороны, обычно являются более компактными и имеют меньшую вычислительную мощность и масштаб. Они могут использоваться для хранения и обработки данных в рамках бизнес-систем или предоставления сервисов в сети. IBM System z

# 8. Понятие микроархитектуры, тракт данных, синхронизация тракта данных

**Микроархитектура:** Над цифровым логическим уровнем находится микроархитектурный уровень. Его задача – интерпретация уровня команд в управляющие сигналы (команды) для цифровых устройств.

Микроархитектура содержит микропрограмму, располагаемую в ПЗУ. Микропрограмма должна вызывать, декодировать и выполнять команды. Поскольку аппаратное обеспечение состоит из вентилей и регистров, то микропрограммы должна выдавать команды, управляющие этими вентилями и регистрами.

**Тракт данных** **–** В общем случае содержит ряд регистров и АЛУ. Регистры и АЛУ связаны между собой шинами.

Цель синхронизации тракта данных состоит в обеспечении правильного выполнения инструкций, предотвращении конфликтов при одновременном доступе к ресурсам. Это обеспечивает надежную и предсказуемую работу процессора и устраняет возможность возникновения ошибок, связанных с параллельным выполнением инструкций.

# 9. Понятие микроархитектуры, конвейерная архитектура, кэш-память, ветвления

**Микроархитектура:** Над цифровым логическим уровнем находится микроархитектурный уровень. Его задача – интерпретация уровня команд в управляющие сигналы (команды) для цифровых устройств.

Микроархитектура содержит микропрограмму, располагаемую в ПЗУ. Микропрограмма должна вызывать, декодировать и выполнять команды. Поскольку аппаратное обеспечение состоит из вентилей и регистров, то микропрограмма должна выдавать команды, управляющие этими вентилями и регистрами.

**Конвейерная архитектура** была введена в центральный процессор с целью повышения быстродействия. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в *конвейере* продвигается на следующую стадию обработки, выполненная команда покидает *конвейер*, а новая поступает в него.

Одним из самых простых и наиболее распространенным способов повышения быстродействия процессоров является конвейеризация процесса вычислений. В любом процессоре машинная команда проходит ряд этапов обработки, например: выборку команды из памяти (ВК), вычисление адреса операнда (ВА), выборку операнда из памяти (ВО), собственно выполнение операции.



**Кэш-память:** Маленькая память с высокой скоростью обращения называется **кэш-памятью**.

**Основная идея кэш-памяти – в ней находятся слова, которые чаще всего используются. Процессор сначала обращается в кэш. Если там нет требуемого слова, он обращается в память.**

Эффективность использования кэш памяти определяется тем, насколько «удачно» она заполнена. Основная идея заполнения кэш-памяти заключается в том, что, как правило, последовательно используемые данные и команда расположены поблизости. Если определенное слово вызывается из памяти, то вместе с ним считываются и записываются в кэш соседние слова.

**Ветвления:** Современные компьютеры сильно конвейеризированы, они могут содержать десять и более стадий. Но конвейеры дают высокое быстродействие только на линейном коде, а в случае наличия ветвления возникает проблема с выбором следующей команды.

Современные машины содержат средства, позволяющие прогнозировать переходы. Первое предположение заключается в том, что переход будет назад, т.к. если переход находится в конце цикла, то переход назад более вероятен. Если переход предсказан неправильно, то необходимо отменить выполненные команды.

***Динамическое прогнозирование ветвлений.*** Один из самых простых способов – хранить специальную таблицу (в аппаратном обеспечении), в которую центральный процессор записывает условные переходы, когда они встречаются и там их можно искать, если он снова появляется. В такой таблице хранится адрес перехода и бит, который указывает, был ли сделан этот переход. Прогноз состоит в том, что программа пойдет тем же путем, что и в предыдущий раз. Если прогноз не верен, то бит меняется.

***Статическое прогнозирование ветвлений.*** Динамическое прогнозирование происходит во время работы программы и это его положительное качество. Однако реализация динамического прогнозирования ветвлений требует специализированного и достаточно дорогого аппаратного обеспечения. Статическое прогнозирование осуществляется компилятором.

Обычно статическое и динамическое прогнозирование ветвлений комбинируются в современных процессорах для достижения наилучшей производительности. Статическое прогнозирование может использоваться для предсказания ветвлений на этапе компиляции, а динамическое прогнозирование - для адаптации к изменяющимся условиям выполнения программы во время работы.

# 10. Модели памяти, регистры

**Модели памяти:** Модель памяти определяет правила и семантику доступа к памяти в компьютерных системах.

Во всех компьютерах память разделена на ячейки, которые имеют последовательные адреса. В настоящее время наиболее распространены ячейки в 8 разрядов (1 байт). Причина применения байта – ASCII-символ, который занимает 7 разрядов и бит четности. Если в будущем будет доминировать кодировка UNICODE, то ячейки, возможно, станут 16-разрядными (24 лучше, чем 23).

Байты обычно группируются в слова по 4 или 8 байт. Многие архитектуры требуют, чтобы слова были выровнены. При этом память работает более эффективно.

**Примеры:** **Последовательная консистентность** (Sequential Consistency): Эта модель представляет память как последовательную систему, где все операции чтения и записи происходят в определенном порядке. Для каждого процессора или потока все операции показываются в одном глобальном порядке, не зависящем от их физического расположения или времени выполнения. Это достаточно интуитивная модель, но может ограничивать параллелизм из-за необходимости гарантировать последовательность операций.

**Модели памяти для параллельных и суперскалярных процессоров**: Эти модели обеспечивают дополнительные гарантии согласованности памяти, чтобы учесть ситуации, когда несколько процессоров или потоков имеют доступ к общей памяти и могут выполнять операции параллельно.

**Регистры:** Регистры - это небольшие и быстрые памятные ячейки, которые находятся внутри центрального процессора (ЦП) компьютера. Они используются для хранения и манипулирования данными и инструкциями в процессе выполнения программы.

Регистры уровня команд можно разделить на **две категории**: **специальные регистры** и **регистры общего назначения**. **Специальные регистры включают счетчик команд, указатель стека, а также другие регистры с особой функцией**. **Регистры общего назначения содержат ключевые локальные переменные, и промежуточные результаты. Их основная функция – обеспечить быстрый доступ к часто используемым данным (избегать обращений в память)**.

**Флаговый регистр содержит различные биты, необходимые центральному процессору**. Самые важные из них – ***коды условий***. Они устанавливаются в каждом цикле АЛУ и отражают состояние результата предыдущей операции:

**1. N – результат отрицательный (Negative);**

**2. Z – результат равен нулю (Zero);**

**3. V – переполнение (Overflow)**

**4. C – перенос самого левого бита (Carry Out);**

**5. F – перенос бита 3 (Auxiliary carry – служебный перенос);**

**6. P – результат четный (Parity).**

**Сегментные регистры** - это особые регистры, которые используются в некоторых архитектурах процессоров для организации сегментации памяти. Сегментация памяти - это метод разделения адресного пространства на логические сегменты, каждому из которых присваивается уникальный идентификатор. Каждый сегментный регистр содержит базовый адрес начала соответствующего сегмента памяти. При обращении к памяти процессор использует значение сегментного регистра для вычисления физического адреса, с которым он будет работать.

**Записываемые в сегментный регистр слова (16 бит) называют селекторами (selector), поскольку оно выбирает или «селектирует» один из сегментов из множества возможных.**

**Регистры FPU**

Сопроцессор не имеет своей индивидуальной программы и не может осуществлять выборку команд. Если выбранная команда является командой центрального процессора, он выполняет ее обычным образом, а сопроцессор не привлекается. Они могут производиться ***параллельно*** с дальнейшими действиями центрального процессора. Этот параллелизм требует некоторой ***синхронизации*** действий ЦП и сопроцессора.

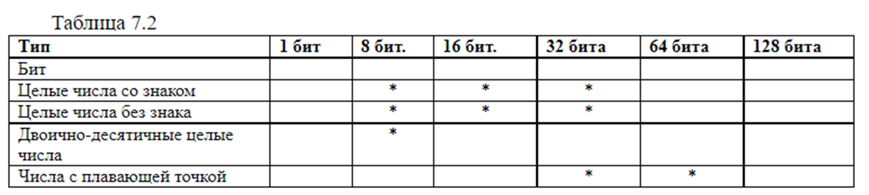
***Синхронизация по командам*.** Когда ЦП выбирает для выполнения команду сопроцессора, последний может быть занят выполнением своей предыдущей команды. Следовательно, перед каждой командой сопроцессора должна находиться специальная команда ЦП, которая только проверяет состояние сопроцессора и, если он занят, переводит ЦП в состояние ожидания (эти команды имеют мнемоники WAIT, FWAIT).

***Синхронизация данных*.** Если выполняемая сопроцессором команда записывает операнд в ячейку памяти перед последующей командой центрального процессора, которая обращается к той же ячейке, также необходима команда проверки состояния сопроцессора.

# 11. Типы данных процессоров (с примерами)

С точки зрения архитектуры ключевым вопросом является вопрос о том, осуществляется ли аппаратная поддержка конкретного типа данных. Под аппаратной поддержкой подразумевается, что одна или несколько команд ожидают данные в определенном формате и пользователь не может брать другой формат.

1. **Типы данных процессора Pentium 4**



Pentium 4 также может манипулировать 8-ми разрядными символами ASCII: существуют специальные команды для копирования и поиска цепочки символов. Эти команды используются и для цепочек, длина которых известна заранее, и для цепочек, в конце которых стоит специальный маркер. Эти операции часто используются в библиотеках операций над строками.

Операнды не обязательно должны быть выровнены в памяти, но если адреса слов кратны 4 (выравнивание по словам), то производительность повышается.

**1.2. Типы данных FPU**

Устройство **FPU определяет численные данные в 7-ми внешних форматах**, образуя при этом **три класса**:

**1. двоичные числа – целые числа (16, 32,64 разряда);**

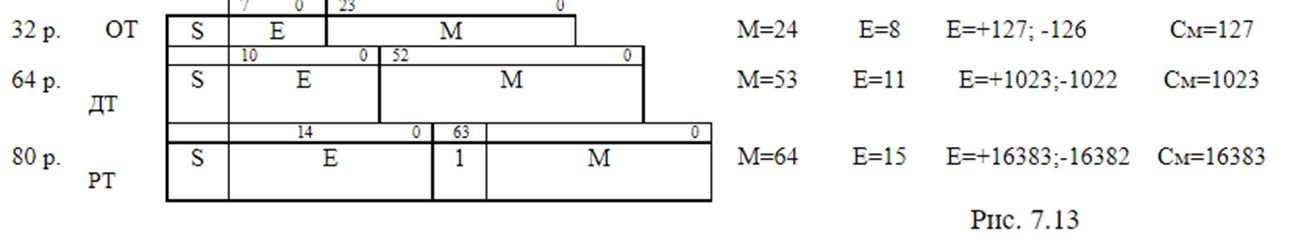
Целые числа представляются в дополнительном коде и могут занимать 16 бит (целое слово), 32 бита (короткое целое), 64 бита (длинное целое). Эти форматы существуют только в памяти, внутри FPU они автоматически преобразуются в 86-разрядный расширяемый вещественный формат.

**2. упакованные десятичные целые числа;**

Упакованные десятичные целые. Они представляются в прямом коде и упакованном формате, т.е. каждый байт содержит две десятичные цифры. Старший бит старшего разряда байта отведен для знака числа, остальные биты этого байта игнорируются. Длина этого формата –10 байт (80 бит). Опять же этот формат существует только в памяти.

**3. двоичные вещественные числа – 32, 64, 80 разрядов.**

Вещественные числа. Имеется **три формата с плавающей точкой (одинарной точности ОТ, двойной точности ДТ, расширенной точности РТ)**. Значащие числа находятся в поле мантиссы, поле порядка показывает фактическое положение десятичной точки в разрядах мантиссы, а бит знака определяет знак числа (рис. 7.13).

****

Порядок E задается в смещенной форме, он равен истинному порядку, увеличенному на значение смещения.

E = истинный порядок + смещение.

Значение числа с плавающей точкой равно:

(-1)S•2E-смещение•(F0).(F1)(F2) …(Fn), где n = 23, 52, 63.

**Напомним, что внутри FPU числа хранятся только в формате РТ. Перевод в другие форматы из них осуществляется автоматически.**

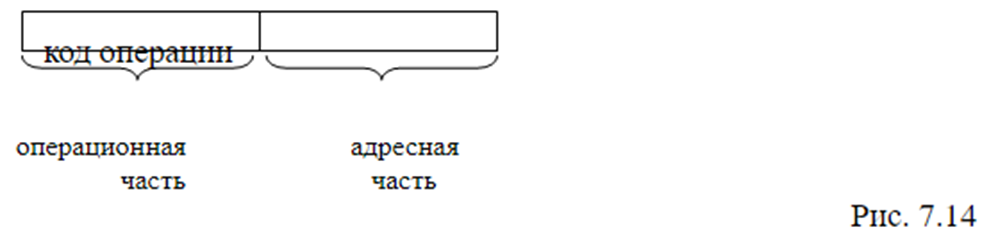
нулевым смещенным порядком и мантиссой. Нуль является знаковым (положительные или отрицательные нули).

***Бесконечность.*** Поддерживается знаковое представление бесконечности. Это значение кодируется со смещенным порядком из всех единиц и мантиссой 1.000… . Знаки бесконечностей учитываются и сравнение возможно. Бесконечности интерпретируются так:

***Нечисла.*** Обозначаются NaN (Not-a-Number). Оно имеет смещенный порядок из всех единиц, любой знак и любую мантиссу, за исключением 1.000… 0. В свою очередь могут делиться на типы. Возникают иногда как результат недействительных операций. Могут использоваться для отладки программ.

# 12. Форматы команд

**Команда** – представляет собой код, определяющий операцию и данные, участвующие в операции. Команда содержит также в явной или неявной форме информацию об адресе, по которому помещается результат операции и адрес следующей команды.



**Команда в общем случае состоит из операционной и адресной частей.**

Эти части, в свою очередь, могут состоять из нескольких полей.

**Операционная часть содержит код операции (КОП)**, который задает вид выполняемой операции. **Адресная часть команды содержит информацию об адресах операндов и результата операции, а в некоторых случаях информацию об адресе следующей команды.**

**Формат команды** - это структура, определяющая организацию и расположение различных полей в команде. Формат команды определяет, какие биты или байты в команде отведены для определенных целей, таких как код операции, адреса операндов и другая информация.

Операция, описываемая **трехадресной командой**, символически представляется в виде:

**ОП[A3] := ОП[A1] \*ОП[A2].**

**Двухадресная команда:** результат операции всегда помещается на место одного из операндов.

**ОП[A1] := ОП[A1] \*ОП[A2].**

В **одноадресной команде** подразумеваемые адреса имеют уже и результат операции, и один из операндов (рис. 7.15.)

ывает, что обязательно нужно оставлять большее количество свободных кодов операций для будущих дополнений к наборам команд.

1. **Форматы команд FPU**

**В мнемониках команд FPU приняты следующие соглашения:**

· **первая буква команды всегда F** (только команды FPU начинаются с F);

· **вторая буква I** обозначает операцию с **целым двоичным числом** из памяти, **буква B** – операцию с **десятичным числом** из памяти, **без этих букв – вещественное число**;

· **предпоследняя или последняя буква R** указывает **обратную операцию (для вычитания или деления)**;

· **последняя буква** P идентифицирует команду, заключительным действием которой является **извлечение из стека**.

# 13. Адресация в компьютерных системах (с примерами)

**Способы адресации:**

1. **неявный операнд.**

В команде не содержится в явном виде указаний об адресе операнда. **Операнд подразумевается и фактически задается кодом операции команды.** Вообще, данный метод используется нечасто, однако в архитектуре х86 имеется достаточное количество таких команд: INC, DEC.

1. **неявная адресация.**

В команде не содержится явных указаний об адресе участвующих в операции операндов, или адреса, по которому помещается результат операции, но этот адрес подразумевается. Например, **команда может содержать адреса обоих операндов, участвующих в операции, а результат помещается по адресу одного из операндов**. Сюда также можно отнести операции с использованием подразумеваемого регистра.

1. **Непосредственная адресация.**

**В команде содержится не адрес операнда, а непосредственно операнд.** Непосредственная адресация удобна при использовании различного рода констант.

1. **Прямая адресация.**При использовании прямой адресации, адрес операнда указывается явно в команде, обычно в виде числового значения или метки, которая затем преобразуется в адрес во время выполнения программы. Команда выполняет операцию над данными, которые находятся по указанному адресу.
2. **Регистровая адресация.**

**В качестве операнда используется содержимое регистров процессора**

1. **Косвенная адресация.**Косвенная адресация - это способ адресации операндов, при котором адрес операнда хранится не в самой команде, а в регистре или ячейке памяти. Вместо явного указания адреса операнда в команде, используется адрес хранящийся в указанном месте.
2. **Относительная адресация или базирование.**В относительной адресации исполнительный адрес операнда определяется путем добавления относительного смещения (или сдвига) к базовому адресу
3. **Индексная адресация.**Индексная адресация - это способ адресации операндов в командах, при котором используется комбинация базового адреса и индексного регистра. Индексный регистр содержит значение, которое будет использовано для вычисления фактического адреса операнда.
4. **Стековая адресация.**

**Стековая память реализует безадресное задание операндов.** Стек представляет собой группу последовательно пронумерованных регистров (аппаратный стек) или ячеек памяти, снабженных указателем стека (обычно регистром SP) **При выполнении операции записи в стек, слово помещается в следующую ячейку стека, а при считывании из стека последнее поступившее в него слово. Таким образом, в стеке реализуется дисциплина обслуживания «последний пришел – первый ушел» (LIFO).**

команды передачи в стек слова указатель стека увеличивается на 1, а затем слово помещается в ячейку стека, указываемую SP. При загрузке из стека сначала извлекается слово, а затем указатель уменьшается на 1.

# 14. Компьютеры параллельного действия, принципы организации, функционирование (примеры)

можно смотреть 34 вопрос

**Chat GPT:**

**Компьютеры параллельного действия** (или параллельные компьютеры) — это компьютерные системы, способные выполнять несколько вычислительных задач одновременно или параллельно. Они используют принцип параллельной обработки данных для ускорения выполнения вычислений и решения сложных задач.

**Принципы организации параллельных компьютеров могут варьироваться в зависимости от их архитектуры**. Однако, некоторые **общие принципы включают**:

1. **Задачи разделения:** Вычислительная задача разбивается на более мелкие подзадачи, которые могут быть выполнены параллельно.
2. **Распределение задач**: Подзадачи распределяются между доступными вычислительными ресурсами, такими как процессоры или ядра.
3. **Коммуникация и координация**: Вычислительные узлы должны обмениваться данными и результатами вычислений для согласования и объединения результатов.
4. **Синхронизация:** В случае необходимости, параллельные компьютеры могут использовать механизмы синхронизации для согласования выполнения задач и предотвращения конфликтов данных.

**Примеры параллельных компьютеров** включают в себя:

1. **Кластеры компьютеров**: Состоят из множества самостоятельных компьютеров (узлов), соединенных сетью. Кластеры используются для решения высокопроизводительных задач, где каждый узел может выполнять свою часть работы параллельно.
2. **Многопроцессорные системы (МПС)**: Включают в себя несколько процессоров, работающих вместе над общими задачами. Это может быть суперкомпьютер или сервер с несколькими процессорами.
3. **Графические процессоры (GPU)**: GPU являются специализированными процессорами, предназначенными для обработки графики, но также могут быть использованы для параллельных вычислений. Они широко применяются в областях, требующих интенсивной обработки данных, таких как научные исследования или глубокое обучение нейронных сетей.

# 15. Внутрипроцессный параллелизм, скалярные и VLIW процессоры

Одним из путей повышения пропускной способности процессора является выполнение большего числа операций за то же время. Другими словами, использование параллелизма. **На уровне процессора параллелизм может достигаться путем использования врожденного параллелизма, присущего инструкциям (параллелизм уровня команд), многопоточности и размещения на одной микросхеме нескольких процессорных ядер.**

**Параллелизм уровня команд** (Instruction Level Parallelism, ILP) – это потенциальное перекрытие нескольких инструкций во времени. Благодаря ILP такие инструкции могут выполняться параллельно, повышая производительность.

Низкоуровневый параллелизм достигается, в частности, выдачей (issue) нескольких команд за один тактовый цикл. **Процессоры**, в которых реализуется этот принцип, **делятся на две категории**: **суперскалярные** и **VLIW (Very Long Instruction Word – процессоры со сверхдлинным словом).** Схема суперскалярного процессора приведена на рис. 8.2. **Суперскалярные процессоры способны за один такт вызывать для выполнения несколько команд (обычно от 2 до 6), что определяется как аппаратной реализацией, так и последовательностью команд.**

Процессоры со сверхдлинным словом (VLIW) используют длинные инструкции, которые содержат несколько операций, выполняемых параллельно. Это позволяет компилятору или разработчику программы явно указывать параллельные операции, которые могут выполняться независимо друг от друга.

**Скалярный процессор** — это простейший класс микропроцессоров. [1] Скалярный процессор обрабатывает один элемент данных за одну инструкцию (SISD, Single Instruction Single Data), типичными элементами данных могут быть целые или числа с плавающей запятой. В векторных процессорах (SIMD, Single Instruction Multiple Data), в отличие от скалярных, одна инструкция работает с несколькими элементами данных.

# 16. Внутрипроцессорная многопоточность

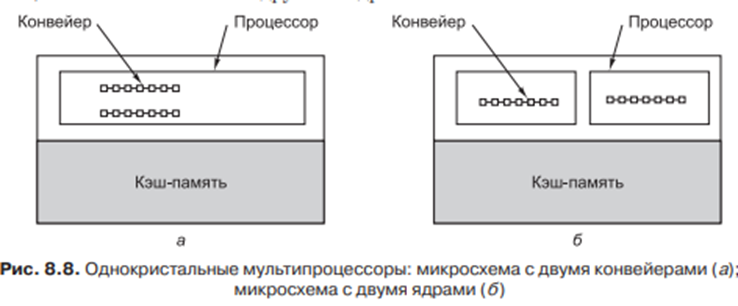
Для современных компьютеров характерно следующее. Если при запросе к памяти требуемые данные не обнаружены в кэшах первого или второго уровней, то на загрузку слова в кэш уходит длительное время, во время которого конвейер простаивает. Одна из методик решения такой проблемы называется **внутрипроцессорной многопоточностью** (on-chip multithreading). **Она позволяет процессору одновременно управлять несколькими программными потоками, тем самым маскируя простои.** Общая идея: если программный поток 1 блокируется, процессор может обеспечить полную загрузку аппаратуры, запустив программный поток 2. В отличие от ILP **внутрипроцессорная многопоточность опирается на явное указание параллелизма**. Каждый поток характеризуется состоянием (инструкции, данные, значения регистров и т. д.). Существуют следующие **способы реализации внтурипроцессорной многопоточности**: **мелкомодульная**, **крупномодульная**, **синхронная**.

**Мелкомодульная многопоточность** обеспечивает более мелкое разделение времени между потоками, что позволяет более равномерно использовать вычислительные ресурсы процессора. предполагает, что процессор быстро переключается между несколькими потоками на каждом тактовом цикле.  
обычно эффективна при наличии большого количества потоков  
  
**Крупномодульная многопоточность**В этом случае процессор задействует каждый поток на протяжении длительного времени (несколько тактов или больше) перед переключением на следующий поток.  
эффективна в ситуациях, когда доступны ограниченные параллельные ресурсы или когда переключение между потоками слишком часто может привести к нежелательным задержкам и потере производительности

В суперскалярных процессорах используется еще одна методика организации многопоточности – **синхронная многопоточность** (simultaneous multithreading). **Эта методика представляет собой усовершенствованный вариант крупномодульной многопоточности, где каждый программный поток может запускать по две команды за такт. В случае простоя с целью обеспечения полной загрузки процессора запускаются команды следующего потока.**

# 17. (с примерами)

Гомогенные однокристальные мультипроцессоры - на один кристалл можно установить два или более мощных процессоров. Поскольку такие процессоры всегда обращаются к одним и тем же модулям памяти (кэшу первого и второго уровней и основной памяти), они, считаются единым мультипроцессором.  
"Гомогенные" означает, что все процессорные ядра в мультипроцессоре имеют одинаковую архитектуру и функциональность. Они могут быть выполнены в виде нескольких экземпляров одного и того же процессорного ядра или быть идентичными по своей архитектуре.

В первом типовом решении для малых однокристальных мультипроцессоров присутствует одна микросхема и два конвейера, что позволяет удваивать скорость исполнения команд теоретически. Это достигается путем параллельного выполнения команд в двух независимых конвейерах. Во втором типовом решении в микросхеме предусмотрены два независимых ядра, каждое из которых содержит полноценный процессор. Каждое ядро представляет собой отдельный процессорный элемент, который может независимо выполнять инструкции и обращаться к ресурсам памяти. В этом случае микросхема содержит два полноценных процессора, которые работают параллельно и могут обрабатывать различные задачи независимо друг от друга.****

**Гетерогенные однокристальные мультипроцессоры**

Гетерогенные однокристальные мультипроцессоры представляют собой мультипроцессорные системы, в которых на одной микросхеме размещены процессорные ядра с различными архитектурами или функциональностями. В отличие от гомогенных мультипроцессоров, где все процессорные ядра идентичны, гетерогенные мультипроцессоры могут содержать процессоры разных типов или классов.  
Гетерогенные однокристальные мультипроцессоры могут применяться в различных областях, где требуется сочетание различных типов процессоров для эффективного выполнения разнообразных задач.  
  
Гомогенные: intel i9, amd threadripper.  
Гетерогенные: apple m1

# 18. Семейства сопроцессорных систем, мультимедиа процессоры (с примерами)

Рассмотрим варианты повышения производительности компьютера за счет введения второго, специализированного процессора. Возможны различные конструктивные реализации сопроцессоров, но все варианты объединяет подчиненная роль сопроцессора по отношению к основному процессору.

Сопроцессор - специальная интегральная схема, которая работает в содружестве с главным процессором. Обычно сопроцессор настраивается на выполнение какой-нибудь специфические функции - математической операции или графического представления. И эту операцию сопроцессор может реализовывать во много раз быстрее, чем главный процессор. Таким образом компьютер с сопроцессором работает намного проворнее.

Сопроцессор - это обычный микропроцессор, но не столь универсальный. Обычно сопроцессор разрабатывается как специальное устройство по реализации конкретно определенной функции. Так репертуар сопроцессора ограничен, он может реализовывать выделенные для него функции как никто другой.

Как и любой другой микpопpоцессоp, сопроцессор работает по тем же принципам. Он просто выполняет программы содержащие последовательность микpопpоцессоpных команд. Сопроцессор не держит под управлением основную массу цепей компьютера.

В обычном режиме микpопpоцессоp выполняет все функции компьютеpа. И лишь когда встречается задача с которой лучше справится сопроцессор, ему передаются данные и команды управления, а центральный процессор ожидает результаты.

· Сетевые сопроцессоры

· Мультимедиа-процессоры

· Криптопроцессоры

· Графические процессоры

· Математические сопроцессоры

Мультимедийные процессоры рассмотрим на примере процессора Nexiperia (Philips) – это семейство микросхем для различных тактовых частот. Он представляет собой гетерогенный однокристальный мультипроцессор и включает несколько ядер: управляющий VLIW-процессор TriMedia, а также отдельные ядра для обработки изображений, звука, видео и сетевых операций. Может использоваться как самостоятельный центральный процессор для CD-, DVD-, MP3-плееров, видеокамер и т.п., а также в качестве сопроцессора.

# 19. Сетевые сопроцессоры

В результате технологического прогресса в области сетевого оборудования передача данных в сетях сегодня происходит так быстро, что программно обрабатывать все входящие и исходящие данные становится все сложнее. В связи с этим разрабатываются специальные **сетевые процессоры, предназначенные для обработки трафика**

В сетях функционирует бесчисленное количество разнообразных промежуточных устройств, в их число входят маршрутизаторы, коммутаторы, брандмауэры, прокси-серверы, системы балансировки нагрузки. К этим промежуточным системам предъявляются самые жесткие требования – именно они обеспечивают передачу максимального числа пакетов в секунду.

Пакет перед отправкой или передачей прикладной программе может быть подвергнут обработке. Исключительно программная обработка пакетов в жестких временных рамках не всегда возможна. Прибегают к помощи аппаратных решений. Существуют следующие реализации.

Использование заказных интегральных схем (ASIC – Application-Specific Integrated Circuit). С созданием таких схем связаны проблемы: их долго проектировать и не менее долго и они очень дорогие. Это жестко запрограммированные устройства, для внесения изменений необходимо разрабатывать новую схему. Целесообразно проектировать, если гарантирован массовый сбыт.

Второй подход основан на использовании программируемых вентильных матриц (Field Programmable Gate Array, FPGA). Такая матрица представляет собой набор вентилей, из которых путем перекоммутации строится требуемая схема. Сроки выхода программируемых вентильных матриц на рынок гораздо короче, чем у специализированных интегральных схем, к тому же их можно перепрограммировать в «полевых условиях» при помощи специального программатора. Но в то же время они очень сложные, дорогие и более медленные, чем схемы ASIC, поэтому программируемые вентильные матрицы не получили широкого распространения, исключая некоторые узкоспециальные области.

**Сетевые процессоры** – устройства, позволяющие обрабатывать входящие и исходящие пакеты со скоростью их передачи. Обычно они реализуются в виде платы, которая помимо кристалла процессора содержит память и вспомогательную логику. К плате подключается одна или несколько линий.

Обычно на плате имеется как статическая (SRAM) так и синхронная динамическая (SDRAM) память. Эти виды памяти применяются для разных целей. SRAM быстрее SDRAM, но она и дороже. SRAM используется для хранения таблиц маршрутизации и ключевых структур данных. В SDRAM хранятся непосредственно пакеты. Поскольку память располагается вне кристаллов, можно гибко подойти к вопросу объема каждого вида.

По каждой сетевой линии приходят миллионы пакетов, а маршрутизатор поддерживает десятки таких линий. Такое быстродействие может обеспечить процессор с высоким уровнем параллелизма. В процессор обязательно входит несколько PPE-контроллеров (Protocol/Programmable/Packet Processing Engine – программируемая система обработки пакетов и протоколов)

В совершенных сетевых процессорах PPE-контроллеры поддерживают многопоточность,

Помимо PPE-контроллеров у всех сетевых процессоров имеется управляющий процессор для выполнения всех действий, не относящихся напрямую к обработке пакетов. Кроме того может быть несколько специализированных интегральных схем (ASIC) для выполнения критически важных операций (например, для поиска целевого адреса в таблице маршрутизации).

Все компоненты сетевого процессора взаимодействуют на мультигигабитных скоростях по одной или нескольким расположенным на кристалле параллельным шинам.

Пути повышения производительности сетевых процессоров надо искать исходя из того, что является единицей измерения производительности. Одной из метрик является количество пакетов, передаваемых в секунду, другой – количество байтов.

**Самым прямым путем является повышение тактовой частоты сетевого процессора.** Повышение частоты не означает пропорционального повышения производительности (за счет обращения в память и т.п.). Кроме того, большая частота означает проблемы с электропотреблением и отводом теплоты.

Следующий путь – увеличение числа PРE-контроллеров.

Еще один подход – увеличение числа дополнительных специализированных процессоров и интегральных схем.

# 20. Криптопроцессоры

Безопасность, а особенно сетевая безопасность — еще одна, в которой широко используются сопроцессоры. Когда между клиентом и сервером устанавливается соединение, обычно требуется их взаимная аутентификация. По установленному таким образом безопасному (шифруемому) соединению можно безопасно передавать данные и не думать о злоумышленниках, прослушивающих линию. Проблема здесь в том, что безопасность обеспечивается средствами криптографии, а эта область требует весьма объемных вычислений. В криптографии сейчас распространены два основных подхода к защите данных: шифрование с симметричным ключом и шифрование с открытым ключом. Первый основан на очень тщательном перемешивании битов (как будто сообщение помещают в некий электронный миксер). В основе второго подхода лежит умножение и возведение в степень больших чисел (1024-разрядных), что требует исключительно больших временных затрат. Многими компаниями выпущены криптографические сопроцессоры, позволяющие шифровать данные для их безопасной передачи и потом расшифровывать их. Зачастую они представляют собой карты расширения, вставляемые Сопроцессоры      623 в PCI-разъем. Благодаря специальному аппаратному обеспечению, эти процессоры могут выполнять необходимые криптографические вычисления намного быстрее, чем центральный процессор.

**Безопасный криптопроцессор** является [процессором](https://ru.frwiki.wiki/wiki/Processeur) , оптимизированным для [криптографических](https://ru.frwiki.wiki/wiki/Cryptographie) задач и обеспечивающий некоторую устойчивость к вторжению. В отличие от криптографических процессоров, которые отправляют расшифрованные данные по шине в защищенной среде, защищенный криптопроцессор не создает расшифрованные данные или расшифрованные программные инструкции в среде, где безопасность не всегда может поддерживаться.

Особенности криптопроцессоров, повышающие их безопасность по сравнению со стандартными микропроцессорами (которые могли бы выполнять большинство из вышеперечисленных функций в программном обеспечении), могут включать:

• **Обнаружение несанкционированного доступа с автоматическим уничтожением хранилища** в случае несанкционированного доступа и конструкция, которая очень затрудняет несанкционированное использование устройства, не оставляя явных следов физического взлома. Эти средства защиты могут варьироваться от наклеек с защитой от несанкционированного доступа, которые четко показывают попытки доступа к внутренним компонентам устройства, до защищенных корпусов, которые обнаруживают несанкционированные попытки открыть устройство и автоматически стирают или уничтожают конфиденциальные материалы ключей.

• Конструктивные особенности чипа, такие как з**ащитные слои для предотвращения перехвата внутренних сигналов** с помощью ионных зондов или других микроскопических устройств

• **Аппаратный криптографический ускоритель** (т.е. специализированные инструкции или логика для повышения производительности стандартных криптографических алгоритмов, таких как AES, SHA, RSA, ECC, DSA, ECDSA,)

Существует много типов защищенных криптопроцессоров:

• Фирменные, такие как процессор Apple Secure Enclave Processor (SEP), который используется в новых iPhone

**Криптографический модуль** (см. рис. 3.11) обычно представляет собой аппаратное устройство, реализующее генерацию ключей и другие криптографические функции и встроенное в более крупную систему.

HSM (см. рис. 3.12) - это тип криптографического модуля, предназначенного для автономной работы в качестве устройства и предоставления криптографических услуг через доступный извне API (обычно через сеть или USB-соединение).

Смарт–карты (см. рис. 3.13) представляют собой криптографические модули размером с кредитную карту, обычно используемые для обеспечения портативного защищенного хранилища в целях управления идентификационными данными (используется для многофакторной аутентификации), безопасной передачи конфиденциальной информации (например, ключей между аппаратными модулями безопасности) и обработки платежей (например, стандарт EMV для кредитных карт).

# 21. Мультипроцессоры и мультикомпьютеры, схемы, особенности работы (с примерами)

В любой системе параллельной обработки процессоры, выполняющие разные части одной задачи, должны взаимодействовать друг с другом, чтобы обмениваться информацией. Были предложены и реализованы две разработки: мультипроцессоры и мультикомпьютеры.

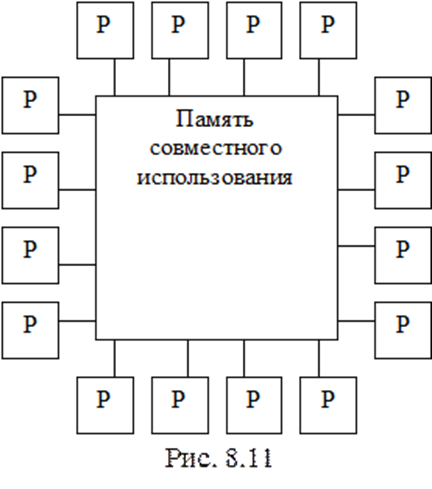
**Мультипроцессоры:**

В первом случае **все процессоры разделяют общую физическую память** как показано на рис. 8.11. Такая система называется **мультипроцессором** или **системой с совместно используемой памятью**. Мультипроцессорная модель распространяется и на программное обеспечение. **Все процессы, работающие вместе на мультипроцессоре, разделяют одно адресное пространство.**

**В такой системе два процесса могут обмениваться информацией, если один из них будет записывать в память, а второй считывать**. Благодаря такой возможности взаимодействия двух и более процессов мультипроцессорные системы очень популярны.

Поскольку все процессы, работающие в системе используют одно общее адресное пространство, то в системе функционирование одной копии операционной системы, одна таблица страниц памяти, одна таблица процессов.

В качестве **примеров мульти­про­цес­со­ров** можно назвать **Sun Enterprise 10000, Sequent NUMA-Q, SGI Origin, HP/Convex Exemplar.**

****

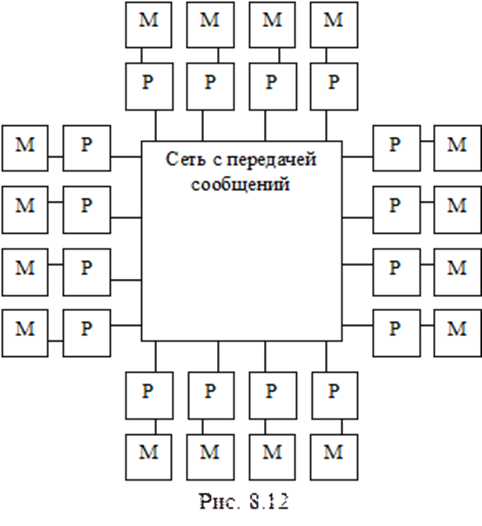
**Мультикомпьютеры:**

Во втором типе параллельной архитектуры **каждый компьютер имеет свою собственную память, доступную только этому процессору**. Такая разработка называется **мультикомпьютером** или **системой с распределенной памятью**. Она изображена на рис. 8.12. Мульти­компьютеры в большинстве случаев являются системами со слабой связью. **Ключевое отличие мульти­компь­ютера от мультипроцессора состоит в том, что каждый процессор в мульти­компь­ютере имеет свою собственную локальную память и никакой другой процессор не может получить доступ к этой памяти**. **Мультикомпьютеры содержат от­дель­ное физическое адресное про­стран­ство для каждого центрального процессора.**

Поскольку **процессоры в мульти­компьютере** не могут взаимодействовать путем совместного обращения в общую память, то здесь необходим другой механизм взаимодействия. Они **посылают друг другу сообщения, используя сеть межсоединений.**

В качестве **примеров мультикомпьютеров** можно назвать **IBM SP/2, Intel/Sandia Option Red, Wisconsin COW.**

При отсутствии памяти совместного использования в аппаратном обеспечении предполагается определенная структура программного обеспечения. **В мультикомпьютере невозможно иметь одно виртуальное адресное пространство.**

****

Если процессу 1 потребовались данные, не содержащиеся в его локальной памяти, то ему надо каким-то образом определить, какой процессор содержит эти данные и послать этому процессору сообщение с запросом копии данных. Процессор 1 блокируется до получения ответа. Когда процессор 2 получает сообщение, программное обеспечение его анализирует и отправить необходимые данные. Когда процессор 1 получает необходимые данные, программное обеспечение разблокируется и продолжает работу.

**В мультикомпьютере для взаимодействия между процессорами часто используются примитивы send и receive**. Программное обеспечение мультикомпьютера имеет более сложную структуру, чем программное обеспечение мультипроцессора. При этом основной проблемой становится правильное разделение данных и разумное их размещение. **В мультипроцессоре размещение частей данных не влияет на правильность решения задачи, хотя может повлиять на производительность. Таким образом, программировать мультикомпьютер гораздо сложнее, чем мультипроцессор.**

**С точки зрения аппаратной реализации, гораздо проще и дешевле построить большой мультикомпьютер, чем мультипроцессор с таким же количеством процессоров**. Реализация общей памяти, разделяемой несколькими сотнями процессоров, - это весьма сложная задача, а построить мультикомпьютер, содержащий 10000 процессоров и более, довольно легко.

# 22. Гибридные системы построения вычислительных систем (с примерами на различных уровнях)

Известны **попытки создания гибридных систем**, которые достаточно легко строить и относительно легко программировать. **Это привело к осознанию того, что совместную память можно реализовывать по-разному, находя компромиссы между достоинствами и недостатками каждого подхода.** Все последние исследования в области архитектур с параллельной обработкой направлены на создание гибридных форм. **Важно получить такую систему, которая расширяема, т.е. будет исправно работать при добавлении новых элементов.**

**Один из подходов** основан на том, что современные компьютерные системы состоят из ряда уровней. Это дает возможность реализовать общую память на любом из нескольких уровней, как это показано на рис.   
Этот подход предполагает создание механизма, позволяющего программам и процессам обращаться к памяти на разных уровнях, будто они используют общую память. Таким образом, процессор и программы работают с памятью, не заботясь о ее физическом расположении или наличии на определенном уровне иерархии

****

**Второй подход** – **использовать аппаратное обеспечение мультикомпь­ютера и операционную систему, которая моделирует разделенную память, обеспечивая единое виртуальное адресное пространство, разбитое на страницы.** В этом подходе каждый компьютер в мультикомпьютерной системе имеет собственную физическую память и работает автономно. Однако операционная система создает иллюзию единого виртуального адресного пространства для всех компьютеров в системе. Это достигается путем разбиения виртуального адресного пространства на страницы и сопоставления этих страниц с физической памятью на каждом компьютере. Если процессор совершает операцию считывания или записи над одной из страниц, которой у него нет, происходит прерывание операционной системы. Затем ОС находит нужную страницу и требует, чтобы процессор, обладающий нужной страницей, преобразовал ее в исходную форму и послал по сети межсоединений. Таким образом, программы могут обмениваться данными между разными компьютерами, используя единое виртуальное адресное пространство.

**Chat GPT:**

Вот **несколько примеров гибридных систем на различных уровнях:**

1. **Гибридные вычислительные кластеры:** Кластеры компьютеров, состоящие из различных типов узлов, таких как центральные процессоры (CPU) и графические процессоры (GPU), могут быть классифицированы как гибридные системы. Например, **кластер, в котором каждый узел оборудован мощным CPU и несколькими GPU, может использоваться для параллельной обработки и вычислений на графических процессорах**, чтобы достичь высокой производительности в задачах, требующих интенсивной обработки данных.
2. **Гибридные системы хранения данных:** Системы хранения данных могут использовать гибридный подход, комбинируя различные типы носителей данных для балансировки скорости доступа, емкости и стоимости. Например, **гибридные системы хранения данных могут объединять быстрые твердотельные накопители (SSD) для кэширования и обработки данных с более емкими, но медленными жесткими дисками (HDD) для долгосрочного хранения.**
3. **Гибридные облачные вычисления**: В облачных вычислениях гибридные системы объединяют локальные вычислительные ресурсы и облачную инфраструктуру. **Например, компании могут использовать локальные серверы для хранения и обработки чувствительных данных, а для более масштабируемых задач или пиковой нагрузки использовать облачные ресурсы, чтобы расширить вычислительные мощности.**
4. **Гибридные алгоритмы**: Гибридные системы могут использовать комбинацию различных алгоритмов и подходов для решения сложных задач. Например, генетические алгоритмы могут быть использованы для оптимизации параметров модели машинного обучения, где эволюционные методы могут комбинироваться с глубоким обучением для достижения лучших результатов.

# 23. Организация виртуальной памяти(Вася)

Организация виртуальной памяти является одним из основных принципов управления памятью в операционных системах. Виртуальная память представляет собой абстракцию, которая позволяет программам использовать больше памяти, чем физически доступно на компьютере.   
 Виртуальная память основана на концепции разделения памяти на страницы фиксированного размера. Каждая страница имеет свой уникальный виртуальный адрес. Когда программа обращается к адресу памяти, операционная система переводит этот виртуальный адрес в физический адрес реальной памяти, где данные фактически хранятся. Этот процесс называется преобразованием виртуальных адресов в физические. Основные преимущества организации виртуальной памяти включают:

Разделение памяти между разными процессами: Каждый процесс имеет свое собственное виртуальное адресное пространство, которое изолировано от адресных пространств других процессов. Это обеспечивает защиту и безопасность данных между процессами.

Экономия физической памяти: Виртуальная память позволяет загружать в оперативную память только ту часть программы или данных, которые действительно нужны в данный момент. Остальные части могут быть хранены на диске и загружаться по мере необходимости. Это позволяет эффективно использовать доступную память и работать с более объемными программами.

Поддержка разделенной памяти: Виртуальная память облегчает обмен данными между разными компонентами программы или между разными процессами. Разные части программы или процессы могут разделять общие страницы памяти, что упрощает коммуникацию и совместную работу.

Поддержка виртуальных адресов: Программы работают с виртуальными адресами, которые не зависят от физической конфигурации памяти компьютера. Это обеспечивает независимость программ от конкретной физической памяти и позволяет легко перемещать программы между разными системами.

Организация виртуальной памяти требует поддержки со стороны операционной системы и аппаратного обеспечения. Операционная система отвечает за управление таблицами преобразования виртуальных адресов в физические, а аппаратное обеспечение выполняет собственно преобразование адресов и управляет доступом к физической памяти.

Виртуальная память позволяет программам использовать больше адресов, чем физически доступно на компьютере. Это достигается за счет использования памяти на диске в качестве дополнительного хранилища данных.

Когда программа обращается к адресу виртуальной памяти, операционная система проверяет, находится ли соответствующая страница данных в физической памяти. Если страница отсутствует, происходит прерывание, и операционная система загружает эту страницу из виртуальной памяти на диск в физическую память. Этот процесс называется "страничным прерыванием" или "процессом подкачки".

Таким образом, виртуальная память позволяет программам использовать адресное пространство, превышающее физическую память компьютера. Это позволяет работать с более объемными программами и упрощает управление памятью, так как операционная система может загружать и выгружать страницы данных по мере необходимости.

# 24. Политика замещения страниц, фрагментация, сегментация *лк9 (Лиза)*

Если программа обращается в память за страницей, то новой странице необходимо освободить место. Возникает необходимость определения страницы, которую можно отправить во внешнюю память. Для этого используется алгоритм (политика) замещения страниц.

• **Алгоритм LRU** (Least Recently Used – алгоритм удаления наиболее давно используемых элементов). Согласно этому алгоритму удаляется та страница, которая использовалась наиболее давно.

(Где не подойдет этот алгоритм: В цикле, для реализации которого требуется n страниц, а в памяти может разместиться только n-1.)

• **Алгоритм FIFO** (First-in First-out – первым пришел, первым ушел) удаляет ту страницу, которая первая поступила в память. В такой ситуации возможно ошибочное удаление наиболее «активной» страницы.

• **В случайном (Random)** алгоритме страница для удаления выбирается случайно. Недостаток: может быть удалена даже наиболее интенсивно используемая страница.

• **Алгоритм замещения давно использовавшейся страницы (Not Recently Used)**.

Каждая страница в памяти имеет два бита: R (Referenced) и M (Modified). Бит R устанавливается, когда страница используется (читается или записывается), а бит M устанавливается, когда страница модифицируется (записывается). При первоначальной загрузке страницы в память оба бита R и M сбрасываются. Периодически (обычно с помощью аппаратного таймера) происходит сброс бита R для всех страниц. При возникновении страничного прерывания алгоритм классифицирует страницы в четыре класса на основе их битов R и M:

Класс 0: Страницы, которые не использовались и не были модифицированы (R=0, M=0).

Класс 1: Страницы, которые не использовались, но были модифицированы (R=0, M=1).

Класс 2: Страницы, которые были использованы и модифицированы (R=1, M=1).

Остальные классы: Некоторые алгоритмы могут использовать дополнительные классы для определенных состояний страниц.

Алгоритм выбирает случайную страницу для удаления из непустого класса с наименьшим номером. Например, если существуют страницы в классах 0 и 1, а страницы в классе 0 необходимо удалять чаще, чем страницы в классе 1, то будет выбрана случайная страница из класса 0. Таким образом, алгоритм замещения страницы Not Recently Used основывается на классификации страниц по их использованию и модификации, а затем выбирает страницу для удаления из наименее часто используемого класса.

**Алгоритм «Второй шанс» (Second Chance)** представляет собой модификацию алгоритма FIFO. Проверяется бит R. Если он сброшен, страница удаляется из «головы» очереди, если он установлен, то бит сбрасывается, страница помещается в «хвост» очереди и проверяется следующая страница в «голове» очереди. Недостаток: много перемещений страниц.

**Фрагментация** возникает, когда доступная память разделяется на маленькие фрагменты, что может приводить к неэффективному использованию памяти.

**Существует два основных типа фрагментации**: фрагментация **внешней** памяти и фрагментация **внутренней** памяти.

* Фрагментация внешней памяти возникает, когда доступная память разделяется на непрерывные блоки, приводящие к "дырам" между ними, что затрудняет размещение больших блоков памяти.
* Фрагментация внутренней памяти возникает, когда процесс использует только часть своего выделенного блока памяти, что может приводить к неэффективному использованию памяти.

Компилятор может иметь несколько таблиц, которые создаются в процессе компиляции:

* таблица символов, которая содержит имена и атрибуты переменных;
* исходный текст;
* таблица, содержащая все используемые целочисленные константы и константы с плавающей точкой;
* дерево, содержащее результат синтаксического анализа программы;
* стек, используемый для вызова процедур в компиляторе.

В рассмотренной ситуации, а также при работе в мультипрограммном режиме необходимо иметь возможность создания независимых адресных пространств. Такие адресные пространства называются **сегментами**.

Каждый сегмент состоит из линейной последовательности адресов от 0 до некоторого максимума.

Длина сегмента может быть любой (в допустимых пределах). Кроме того, длина сегмента может меняться в процессе выполнения программы. Чтобы определиться в пространстве необходимо указать номер сегмента и адрес внутри сегмента.

Сегментированная модель имеет ряд преимуществ при работе с памятью.

1. Если каждая процедура занимает отдельный сегмент, у которого первый адрес равен 0, то связывание процедур, 0которые компилируются отдельно сильно упрощается. Для обращения к i-му слову n-й процедуры используем адрес (n, i).
2. Если процедура в некотором сегменте изменялась и перекомпилировалась, то остальные можно не трогать.
3. Сегментация облегчает разделение общих процедур и данных между несколькими программами.
4. Разные сегменты могут иметь разные виды защиты. Например, кодовый сегмент допускает только считывание выполнение, для массивов данных – запись и считывание и т.д

Сегментацию можно организовать одним из двух способов. Это подкачка и разбиение на страницы.

При первом подходе некоторый набор сегментов находится в памяти в данный момент. Если происходит обращение к сегменту, которого нет в данный момент в памяти, этот сегмент переносится в память. Если для него нет места в памяти, один или несколько сегментов нужно

сначала записать на диск.

**Отличие сегментации от разбиения на страницы:** размер страниц фиксирован, а размер сегментов– нет. Поэтому при сегментации может возникнуть эффект внешней фрагментации.

Второй способ реализации сегментации – разделение каждого сегмента на страницы фиксированного размера и вызов страниц по требованию. Для того, чтобы разбить сегмент на страницы необходимо иметь отдельную таблицу страниц для каждого сегмента.MULTICS (Multiplexed Information and Computing Service – служба общей информации и вычислений) – это старая операционная система, которая совмещала сегментацию с разбиением на страницы. Адреса в MULTICS состоят из двух частей: номера сегмента и адреса внутри сегмента. Для каждого процесса существовал сегмент дескриптора, который содержал дескрипторы для каждого сегмента. Адрес сегмента дескриптора хранился в специальном регистре.

# 25. Уровни привилегий в защитах по привилегиям *лк9*

**Уровни привилегий**

Термин «привилегия» подразумевает права и возможности, которые обычно не разрешаются. **Процессоры х86 поддерживают 4 уровня привилегий: 0, 1, 2, 3.** Чем **меньше** номер уровня, тем он **больше привилегирован**. Уровни привилегий обычно изображаются в виде т.н. колец защиты. **При выполнении почти каждой машинной команды осуществляется проверка защиты по привилегиям.**

Процессор в защищенном режиме постоянно контролирует, что текущая программа достаточно привилегированна, чтобы

1. выполнять некоторые команды;

2. обращаться к данным других программ;

3. передавать управление внешнему (по отношению к самой программе) коду командами передачи управления типа FAR JMP или FAR CALL.

С каждым сегментом кода, данных или стека ассоциируется уровень привилегий и все, что находится внутри этого сегмента имеет этот уровень привилегий. Уровень привилегий выполняющегося в данный момент сегмента кода называется текущим уровнем привилегий (CPL) и задается полем **RPL селектора в регистре CS**. При передаче управления сегменту кода с другим уровнем при­вилегий процессор будет рабо­тать на новом уровне при­ви­легий. Часто говорят, например, «программа выполняется в кольце 0», подразумевая, что CPL процессора равен 0.

**GPT**

В архитектуре x86, используемой в большинстве современных компьютерных систем, существуют четыре основных уровня привилегий, известных как "кольца" (rings):

1. Кольцо 0 (Ring 0) - Уровень ядра: На этом самом привилегированном уровне находится ядро операционной системы. Оно имеет полный контроль над системой и обладает привилегиями для выполнения привилегированных инструкций и доступа к аппаратным ресурсам. Ядро имеет прямой доступ к памяти, управляет процессами и устройствами ввода-вывода. Кольцо 0 также известно как "режим ядра" или "привилегированный режим".

2. Кольцо 1 (Ring 1) - Уровень драйверов: Этот уровень предназначен для драйверов устройств, которые работают внутри операционной системы. Драйверы имеют некоторые привилегии, но они ограничены по сравнению с ядром. Они могут взаимодействовать с аппаратными ресурсами и выполнять специфические задачи, связанные с управлением устройствами.

3. Кольцо 2 (Ring 2) - Уровень привилегированных процессов: Этот уровень предназначен для привилегированных процессов, которые не являются ядром или драйверами устройств. Процессы на этом уровне имеют ограниченные привилегии и могут выполнять некоторые системные задачи, но не имеют полного контроля над системой.

4. Кольцо 3 (Ring 3) - Уровень пользовательских процессов: Это самый низкий уровень привилегий в архитектуре x86. На этом уровне работают обычные пользовательские приложения. Процессы на уровне 3 имеют наименьшие привилегии и могут выполнять только ограниченный набор инструкций и иметь доступ только к определенным ресурсам, предоставленным операционной системой.

Переход между уровнями привилегий осуществляется с помощью специальных механизмов и инструкций, таких как вызовы системы или прерывания. Приложения на уровне 3 могут запрашивать дополнительные привилегии у операционной системы через вызовы системы. Операционная система выполняет проверки безопасности и решает, предоставлять ли запрашиваемые привилегии или нет.

Уровни привилегий в архитектуре x86 позволяют операционной системе разграничивать доступ и контролировать выполнение кода, обеспечивая безопасность и защиту системных ресурсов.

# 26. Повышение эффективности оперативной памяти, организация статической и динамической памяти *(лк 10) (Алина)*

*Из лекций:*

***Статическая память***. Для хранения одного бита памяти требуется не менее шести транзисторов. Такая память может сохранять записанные данные неограниченно и долго без дополнительных действий (при наличии питания). Её принято обозначать SRAM (Static Random Access Memory – статическая память с произвольным доступом). Статическая память отличается высоким быстродействием, но в то же время она характеризуется довольно низкой плотностью хранения данных – в одном корпусе микросхемы можно разместить около миллиона байт. При современных требованиях к объемам ОП в компьютере статическая память считается дорогой и громоздкой. Кроме того, такая память отличается высоким энергопотреблением.

В настоящее время биты ОП реализуют с помощью конденсаторов. Считается, что конденсатор с высоким уровнем напряжения между обкладками содержит цифру 1, а с низким – цифру 0. Для обслуживания конденсатора к нему подсоединяется всего один транзистор. Такая реализация бита имеет значительно более высокую плотность хранения данных – в одном корпусе микросхемы можно поместить десятки миллионов байт. Недостатком ОП на конденсаторах являются ее относительно низкие скоростные показатели. Есть еще один негативный фактор – конденсаторы через несколько миллисекунд самопроизвольно теряют заряд. Чтобы находящиеся в ОП данные и программы при этом сохранялись, состояние конденсаторов необходимо периодически восстанавливать до исходного уровня. Для этого осуществляются периодическое считывание хранящихся данных и повторная их запись. Такой процесс называется *регенерацией памяти*. ОП, реализованную в виде периодически подзаряжаемых конденсаторов, называют ***динамической*** и обозначают DRAM (от Dynamic Random Access Memory – динамическая память с произвольным доступом).

*Какой-то левый сайт:*

В современных компьютерах SRAM используется как кэш второго уровня и имеет сравнительно небольшой объем (обычно 128…1024 Кб). В кэше она используется именно потому, что к нему предъявляются очень серьезные требования в плане надежности и производительности. Основную же память компьютера составляют микросхемы динамической памяти.

Статическую память делят на синхронную и асинхронную. Асинхронная память уже не используется в персональных компьютерах, она была вытеснена синхронной.

Применение статической памяти не ограничивается кэш-памятью в персональных компьютерах. Серверы, маршрутизаторы - вот устройства, где необходима высокоскоростная SRAM.

Память типа DRAM гораздо шире распространена в вычислительной технике благодаря двум своим достоинствам перед SRAM — дешевизне и плотности хранения данных. Эти две характеристики динамической памяти компенсируют в некоторой степени ее недостатки — невысокое быстродействие и необходимость в постоянной регенерации данных.

*GPT:*

***Повышение эффективности оперативной памяти***:

а.) используйте эффективные структуры данных;

Выбор подходящих структур данных может существенно повлиять на эффективность использования оперативной памяти. Например, использование связных списков вместо массивов может сэкономить память при работе с большими объемами данных.

б.) освобождайте память;

Важно правильно управлять памятью и освобождать неиспользуемые ресурсы. Утечки памяти могут привести к исчерпанию доступной памяти и снижению производительности. Обязательно освобождайте память после использования объектов, которые больше не нужны.

в.) оптимизируйте размер объектов;

Иногда можно оптимизировать размер объектов, чтобы сократить использование памяти. Например, можно использовать примитивные типы данных вместо объектов-оберток, если это возможно.

г.) используйте кэширование данных;

Если у вас есть данные, которые часто используются, можно рассмотреть возможность кэширования этих данных в оперативной памяти. Это может улучшить производительность и снизить нагрузку на другие ресурсы.

Организация статической и динамической памяти *(если рассказывать в контексте программирования)*:

***Статическая память*** предназначена для хранения статических переменных и объектов, которые создаются при загрузке программы и существуют до ее завершения. Эта память выделяется на этапе компиляции и освобождается автоматически при завершении программы.

***Динамическая память*** используется для хранения динамически создаваемых объектов, таких как объекты, создаваемые во время выполнения программы с использованием оператора *new*. Эта память выделяется во время выполнения программы и должна быть явно освобождена после использования с помощью оператора *delete* или сборщика мусора, если он поддерживается.

# 27. Различные реализации микросхем памяти, примеры многоуровневой организации памяти

**Микросхемы памяти**

Микросхема памяти - электронный компонент, который используется для хранения и чтения данных

Конструктивно биты как статической, так и динамической памяти объединяют в микросхемы. Группу битов микросхемы, которые считываются или записываются одновременно, принято называть **ячейкой микросхемы**.

Важной характеристикой микросхемы является **структура**, определяющая общий объем микросхемы и разрядность ее ячеек.

**Примеры реализации**

**FPM DRAM**

В FPM DRAM ускорение операций выборки и записи достигается за счет использования режима быстрого доступа к страницам памяти. При этом выбирается определенная строка памяти (страница), и все последующие доступы к ячейкам в этой строке могут быть выполнены без повторного формирования адреса строки. Таким образом, за счет отказа от формирования адреса строки для каждой ячейки из одной и той же строки, FPM DRAM сокращает время доступа к памяти и повышает производительность операций чтения и записи.

**BEDO DRAM**

Блочный режим заключается в том, что во время чтения одной ячейки вместе с ней читаются еще три, расположенных в той же строке, независимо от наличия или отсутствия запроса на них. Это позволяет увеличить скорость чтения памяти путем предварительного загрузки данных из соседних ячеек.

В режиме BEDO адрес столбца устанавливается только для первой ячейки. Далее осуществляется простой переход к следующей ячейке в строке микросхемы

**Многоуровневая организация памяти**

Кратко охарактеризуем свойства используемых в настоящее время уровней памяти компьютера.

• **Регистровая память процессора**. **Используется для промежуточного хранения данных в процессе их обработки**. Энергозависимый вид памяти. Обладает наиболее высоким быстродействием, поскольку работает на тактовых частотах процессора. Время доступа составляет десятые доли наносекунд. Объем регистров – десятки и сотни машинных слов.

* + **Кэш-память**. Используется для ускорения доступа к данным, предварительно загруженным из более медленных уровней памяти. Кэш-память бывает **нескольких уровней: L1, L2, L3** и т. д., где L1 - самый быстрый и наименьший по размеру, а последующие уровни имеют больший объем, но медленнее доступны.

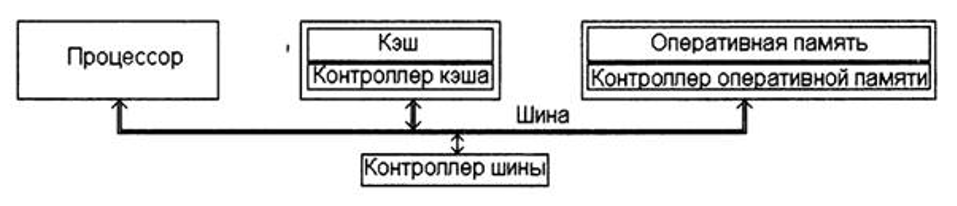
• **Оперативная память**. **Используется для хранения выполняющихся программ и необходимых им данных.** Энергозависимый вид памяти. Время доступа составляет десятки наносекунд, а объем – несколько гигабайт. Более дешевая, чем кэш-память, так как реализуется с помощью динамических микросхем. Высокая надежность хранения данных и программ.

• **Внешняя память** – **магнитные и оптические диски. Служит информационным складом.** Самый медленный уровень памяти, время доступа порядка 30 мс. Самая дешевая и самая емкая – объем современных дисков составляет десятки терабайт.

* + **Виртуальная память:** используется операционной системой для расширения доступного адресного пространства памяти. Она основана на комбинации физической памяти и файлов на внешнем носителе.

# 28. Организация кэш-памяти (Максим)

**Кэш-память -** это дополнительный уровень памяти, играющий роль буфера между оперативной памятью и процессором. Служит для сглаживания разницы в скоростях их работы. Энергозависимый вид памяти. За исключением регистрового, это самый быстрый тип памяти, реализуемый на статических микросхемах памяти. Время доступа составляет единицы наносекунд. Объем кэша может достигать нескольких мегабайт. Отличается относительно высокой стоимостью и высоким энергопотреблением. Объем кэша значительном меньше, чем объем оперативной памяти, а скорость больше, чем у микросхем динамической памяти, и примерно такая же, как у процессора. В общем случае кэш помещается между оперативной памяти и процессором.



Если при попытке записи новых данных в кэш он оказывается до конца заполненным, необходимое для записи место освобождается за счет удаления из кэша некоторой группы уже находящихся в нем данных. Удаляться могут данные, к которым было меньше всего обращений, или, в соответствии с принципом временной локализации, данные, которые дольше всего не использовались процессором. Применение описанного подхода приводит к тому, что по мере выполнения программы в кэше скапливаются наиболее часто используемые данные. Такое накопление принято называть **кэшированием.**

**Кэш прямого отображения:** Память для хранения копий делится на элементы, которые принято называть строками кэша. Строки кэша могут быть различной длины. Довольно часто используется длина 32 байта, соответствующая стандартно передаваемой из оперативной памяти в процессор группе байтов (пакету). Роль строки кэша состоит в том, что данные из оперативной памяти дублируются в кэш сразу целой строкой, которая содержит запрошенное процессором поле памяти. Таким образом, в кэш попадает не только копия требуемого поля, но и копия группы байтов, расположенных рядом с ним.Количество строк в кэше зависит от имеющегося в компьютере физического объема кэша и от длины его строк. Так, например, кэш объемом 4 Кбайт может состоять из 128 тридцатидвухбайтных или 256 шестнадцатибайтных строк. Встречаются и другие варианты.

Для получения простого и быстродействующего механизма поиска контроллер кэша рассматривает всю имеющуюся оперативную память как совокупность страниц, которые равны по объему кэшу и также состоят из строк. Простота дублирования и поиска в кэше обеспечивается тем, что копия любой строки из любой страницы всегда занимает в кэше точно такое же положение, что и оригинал на соответствующей странице оперативной памяти. Это значит, что смещение копии строки относительно начала кэш-памяти всегда равно смещению оригинала строки относительно начала ее страницы в оперативной памяти. Таким образом, страница оперативной памяти как бы полностью отображается на память данных в кэше – отсюда и название «кэш прямого отображения».

****

**Информация от ChatGPT:**

Организация кэш-памяти может варьироваться в зависимости от архитектуры процессора и используемых технологий. Однако основные принципы организации кэш-памяти обычно остаются примерно одинаковыми.

В типичной организации кэш-памяти используются три уровня: L1, L2 и L3:

-Уровень L1 (уровень первого уровня) - самый близкий к процессору и имеет наименьшую задержку доступа. Обычно L1-кэш разделен на две части: инструкционный кэш (I-cache) и данных кэш (D-cache). I-cache содержит инструкции, а D-cache содержит данные.

-Уровень L2 (уровень второго уровня) - находится немного дальше от процессора, но имеет большую ёмкость, чем L1-кэш. Он обычно также разделен на I-cache и D-cache.

-Уровень L3 (уровень третьего уровня) - находится еще дальше от процессора, но имеет еще большую емкость. L3-кэш обычно разделяется между несколькими ядрами процессора и используется для обмена данными между ними.

В кэш-памяти данные организованы в виде блоков фиксированного размера. Когда процессор запрашивает данные из оперативной памяти, они сначала ищутся в L1-кэше. Если данные найдены, это называется "кэш-попадание" (cache hit), и они считываются из кэша, что происходит очень быстро. Если данные отсутствуют в L1-кэше, происходит обращение к L2-кэшу и так далее, пока данные не будут найдены или пока не будет достигнут самый медленный уровень L3 или оперативная память. Если данные не найдены в кэше, это называется "кэш-промах" (cache miss), и процессор должен обратиться к оперативной памяти для получения данных, что занимает больше времени.

29. Шинная архитектура, циклы синхронных и асинхронных шин, многошинная архитектура (с примерами)

**GPT**

Шинная архитектура (или шинная система) - это архитектурное решение, используемое в компьютерных системах для организации и обмена данными между различными компонентами системы. В шинной архитектуре существует централизованный канал (называемый шиной), который связывает различные устройства и компоненты системы.

Основная идея шинной архитектуры состоит в том, чтобы устройства могли обмениваться информацией путем отправки данных по общему каналу - шине. Шина представляет собой физическую или логическую линию, по которой передаются сигналы или данные. Шина может быть однонаправленной или двунаправленной.

В шинной архитектуре устройства подключаются к шине с помощью интерфейсов или контроллеров шины. Каждое устройство имеет уникальный адрес на шине, чтобы его можно было идентифицировать при передаче данных. Когда устройство хочет передать данные, оно размещает данные на шине и указывает адрес получателя. Другие устройства на шине могут мониторить шину и обрабатывать данные, предназначенные для них.

Преимущества шинной архитектуры:

Гибкость и расширяемость: Шинная архитектура обеспечивает гибкость в подключении новых устройств и компонентов к системе без необходимости изменения основной структуры. Новые устройства могут быть добавлены или удалены путем подключения или отключения от шины.

Простота подключения: Устройства могут быть легко подключены к шине, поскольку они используют общий канал для обмена данными. Это упрощает процесс взаимодействия и интеграции различных компонентов системы.

Экономия ресурсов: Использование общего канала для обмена данными позволяет экономить ресурсы, поскольку не требуется создание отдельных каналов связи между каждой парой устройств.

Шинная архитектура широко применяется в компьютерных системах, включая персональные компьютеры, серверы, микроконтроллеры и другие устройства.

**Из лекции**

Основными техническими характеристиками шин являются разрядность, тактовая частота и производительность (скорость обмена, или пропускная способность). Разрядность определяется количеством проводов шины, по каждому из которых передается один бит адреса, один бит данных или один управляющий бит. Тактовая частота определяется как количество управляющих работой шины синхроимпульсов в секунду. Производительность равна количеству битов (или байтов), передаваемых по

   Шинам компьютера приходится выполнять несколько различных функций. Поэтому каждый вид работы определенным образом стандартизируется в виде отдельного **цикла шины**.

**Цикл шины -** периодически повторяемая последовательность сигналов, направленная на выполнение определенной функции шины

В частности, существуют циклы:

* чтения
* записи
* прерывания и др.

   По отношению к тактовым синхроимпульсам шины делятся на **синхронные** и **асинхронные**. У синхронных шин все действия цикла шины привязаны к определенным фазам синхроимпульса – началу, середине, концу и т. д. Они имеют строго определенную длительность, а цикл такой шины всегда занимает целое количество тактов. У асинхронных шин такая привязка отсутствует.

   Преимуществом **синхронных** шин является более простая и дешевая их реализация. Поэтому они распространены довольно широко. Вместе с тем у синхронных шин имеется ряд недостатков. К ним относится снижение общей эффективности передач, вызванное тем, что цикл шины может занимать только целое количество тактов.

**Асинхронная** организация цикла шины не содержит связанных с ожиданием пауз, требует минимальных временных издержек, а также обеспечивает каждому устройству такую длительность цикла шины, которая соответствует его возможностям, независимо от длительностей циклов для других устройств. Платой за повышение эффективности является большая стоимость асинхронных шин по сравнению с синхронными.

**Многошинная архитектура**

Одна общая шина в составе компьютера оказалась не способной обеспечить все необходимые передачи кодов и сигналов. Поэтому довольно быстро в архитектуре компьютеров появились дополнительные специализированные шины, которые связывают между собой различные группы устройств с различными скоростями обмена и различными требованиями к производительности. Архитектура компьютеров, в составе которых имеется более одной шины, называется многошинной.

Современные персональные компьютеры обычно содержат:

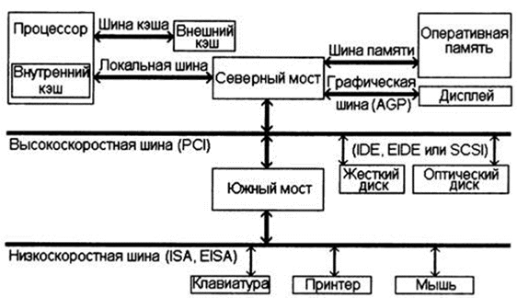
1. Шину низкоскоростных внешних устройств (клавиатура, мышь, принтер);

2. Шину высокоскоростных внешних устройств (магнитные, оптические диски);

3. Шину графического адаптера, для передачи высококачественных цветных движущихся изображений на экран дисплея;

4. Системную шину (шину памяти), связывающую процессор и ОП;

5. Шину кэша, связывающую процессор и внешний кэш.



30. Чипсет, параметры шин, шинный арбитраж (с примерами реализации)

**Чипсет** - специальный набор микросхем компьютера, который служит для согласования операций обмена и управления в сложной иерархической системе шин.

   Микросхемы чипсета содержат в себе контроллеры прерываний, контроллеры прямого доступа к памяти, контроллеры шин и т. д.

В «обязанности» чипсета, в частности, входит:

1. Обслуживание управляющих сигналов процессора;

2. Обеспечение взаимосвязи между всеми типами шин компьютера;

Характеристики чипсета определяют большинство важнейших параметров компьютера, в том числе:

1. Возможные типы и частоты центрального процессора;

2. Скоростные характеристики внешнего кэша и его допустимый объем;

3. Типы, объем и максимальное количество модулей динамической памяти;

4.Возможные частоты шин и т. д.

Тип используемого чипсета существенно влияет на производительность компьютера в целом. При одинаковых базовых компонентах компьютера – процессоре, оперативной памяти, кэше, графическом контроллере (контроллере дисплея) и жестком диске – производительность машин, собранных на разных чипсетах, может различаться на 30% и более.

   В настоящее время чипсеты персональных компьютеров функционально состоят из микросхем **северного моста**, отвечающих за соединение между собой процессора, оперативной памяти и дисплея, и микросхем **южного моста**, которые отвечают за работу с остальными внешними устройствами (дисками, клавиатурой и т. д.).

**Параметры шин**

При разработке шин проектировщики принимают во внимание ряд параметров для достижения компромисса цены и производительности.

1. **Ширина шины** соответствует ширине шин адреса и данных. Производительность системы повышается в случае более широкой шины данных, т. к. появляется возможность пересылать параллельно большее количество бит. Физическое адресное пространство системы увеличивается путем добавления новых адресных линий.
2. **Частота шины** определяет частоту работы шины и измеряется в герцах. Частота шины указывает на количество сигналов, которые передаются или принимаются на шине за секунду. Повышение частоты шины создает проблемы обратной совместимости и совместимости с медленными устройствами.
3. **Тип шин**:   
    **Специализированные** шины: Специализированные шины обычно имеют высокую пропускную способность и оптимизированы для конкретных задач.    
    Пример:  **Локальная шина** - служит для соединения процессора с чипсетом северного моста. **Шина памяти** - соединяющая с этим чипсетом северного моста оперативную память.

**Мультиплексированные** шины: Это шины, которые используются для передачи различных типов данных по одному физическому каналу, вместо выделения отдельных шин для каждого типа данных.  
 Например, **шина данных** - может использоваться для передачи данных различных типов, таких как инструкции, адреса памяти, результаты вычислений и т. д.

1. **Синхронизация**: синхронные и асинхронные шины.

**Арбитраж шин**

**Арбитр** – устройство, разрешающее конфликты доступа к шине.

**Шинный арбитраж** используется для разрешения конфликтов доступа к шине в системе с несколькими устройствами. Это процесс, при котором устройства конкурируют за доступ к шине для передачи данных.

   Виды арбитража:

*Статический* – доступ к шине осуществляется в заранее определенном порядке. Прост в реализации, неэффективен, не отражает нужд пользователей шины.

*Динамический* – осуществляется по требованию мастера. Для реализации необходимо, чтобы у каждого мастера были линии запроса шины (request) и разрешения шины (grant). Если мастеру требуется шина, устанавливается сигнал на линию request. Транзакция может быть начата только после получения разрешения по линии grant.

В зависимости от организации динамический арбитраж можно разделить:

   1. Централизованный арбитраж: существует специальный контроллер арбитража, принимающий решения о предоставлении доступа к шине

  2. Распределенный (децентрализованный): каждый компонент самостоятельно принимает решение о своем доступе к шине.

Если несколько master-устройств состязаются одновременно за доступ к шине, арбитр использует политику назначения для определения победителя. Выделяют 4 политики:

   1. В политике фиксированного приоритета каждому мастеру присваивается уникальный фиксированный приоритет. Когда несколько мастеров запрашивают шину, то получает ее мастер с наиболее высоким приоритетом.

2. В политике изменяющихся приоритетов (Rotating Priority Policy)  приоритет мастера не фиксирован. Например, приоритет мастера может быть функцией от времени ожидания назначения шины. Следовательно, чем дольше мастер ожидает, тем выше приоритет.

   3. «Справедливая» политика (Fair Policy) – политика не допускающая голодания. Она не обязательно использует приоритеты. Может использоваться время ожидания. Например запрос шины не должен ожидать более М миллисекунд.

31. Повышение эффективности процессора, конвейерные конфликты (Егор)

**Chat GPT**

**Повышение эффективности процессора** включает в себя различные методы и техники:  
 **Параллелизм**: Использование параллелизма позволяет выполнять несколько задач одновременно или разделять задачи на более мелкие подзадачи, которые могут выполняться параллельно. Это может быть достигнуто через многопоточность, многопроцессорные системы.

**Конвейер -** Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд. По очередному тактовому импульсу каждая команда в *конвейере* продвигается на следующую стадию обработки, выполненная команда покидает *конвейер*, а новая поступает в него.

Однако при работе в конвейере могут возникать конфликты, которые могут снижать эффективность процессора. Различают следующие **типы конвейерных конфликтов**:

1. **Конфликт данных**: **возникает, когда инструкция зависит от данных, которые еще не были произведены предыдущей инструкцией.** Это может происходить при наличии зависимости между чтением и записью одного и того же регистра или памяти. Для решения таких конфликтов используются методы, такие как переупорядочивание инструкций, введение задержек чтения (stall) или использование буферов для хранения промежуточных результатов.
2. **Конфликт управления:** **возникает, когда процессор не может точно предсказать следующую инструкцию ветвления (например, условный переход или вызов функции)**. Если предсказание неверно, то возникает конфликт, который может привести к перерыву работы конвейера и потере времени. Для решения этого конфликта используются методы предсказания ветвления, такие как статическое предсказание, динамическое предсказание с использованием буфера предсказаний (branch prediction buffer) или предсказание на основе истории ветвлений.
3. **Конфликт ресурсов:** **возникает, когда несколько инструкций требуют доступ к одному и тому же ресурсу одновременно, например, к одному и тому же функциональному блоку процессора.** Это может привести к задержкам и ограничениям в выполнении инструкций. Для решения такого конфликта используются методы, такие как разделение ресурсов, введение буферов или перераспределение инструкций.

32. Комплексные оценки производительности вычислительных систем (тактовой частотой, MIPS, FLOPS, LINPACK и др.) (с примерами применения систем тестирования)(Дима)

В настоящее время используется несколько способов указания мощности компьютеров:

1. Оценка с помощью тактовой частоты;

2. Оценка с помощью указания количества операций, выполняемых в единицу времени;

3. Тестирование на специально отобранных программах.

Наиболее общая характеристика - оценка производительности процессора. Однако и при этом следует учитывать специфику предполагаемого использования вычислительных систем: научные вычисления, как правило, связаны с интенсивными вычислениями над вещественными числами - коммерческие приложения более критичны к производительности работы с целочисленной арифметикой и обработкой транзакций базы данных.

**Тактовая частота** используется как характеристика процессора в тех случаях, когда требуется только приблизительная оценка мощности, например, для описания персональных компьютеров, применяющихся для решения офисных задач, развлечений и т.д. Применение тактовой частоты для оценки мощности облегчается тем, что это довольно легко измеряемый и воспринимаемый параметр. В случае многопроцессорных систем тактовая частота используется как дополнительная характеристика отдельного процессора, входящего в систему.

Использовать тактовую частоту для получения реального представления о производительности компьютера сложно, так как нужно дополнительно знать множество факторов, например, среднее количество тактов на одну команду, количество ступеней конвейера. Особенно слабое представление дает тактовая частота о производительности многопроцессорных вычислительных систем.

Одна из наиболее распространенных единиц измерения производительности процессоров **MIPS**, которая может быть вычислена как отношение числа команд в программе ко времени их выполнения. При всей кажущейся простоте данной метрики - чем больше MIPS, тем быстрее выполняется программа - ее использование связано с рядом проблем. Во-первых, для различных программ MIPS может принимать различные значения. Во-вторых, вследствие того, что процессоры имеют различный набор команд, эта метрика затрудняет сравнение процессоров. Но еще большие затруднения связаны с использованием сопроцессоров плавающей арифметики и оптимизирующих компиляторов.

Еще одна традиционная метрика, но в большей степени характеризующая использование вычислительной техники в научно-технических приложениях - M**FLOPS** (миллион арифметических операций над числами с плавающей точкой, выполненных в секунду). Эта метрика предназначена для оценки быстродействия работы только плавающей арифметики, поэтому ее использование, например, при работе компиляторов не корректно - при этом даже для высокопроизводительной системы значение MFLOPS может быть просто нулевым. Метрика MFLOPS, так же как и MIPS, зависит от конкретной архитектуры и программы, однако, в меньшей степени, она базируется на количестве выполненных операций, а не команд.

Из-за отмеченных недостатков единиц MIPS и MFlops для сравнения производительности компьютеров было предложено использовать в качестве критерия время выполнения специально подобранной эталонной программы или же связанные с этим временем показатели. Программы, на которых осуществляется тестирование, иногда называют **бенчмарками** (от bench-mark – отметка уровня).

**LINPACK** - набор программ на языке программирования Фортран, предназначенных для решения систем линейных алгебраических уравнений.

33. Классификация архитектур (Егор)

* **Разделяемая** (shared) и **распределенная** (distributed) архитектуры: Разделяемая архитектура предполагает совместное использование ресурсов, таких как память и периферийные устройства, между несколькими процессорами или ядрами. Распределенная архитектура, напротив, предполагает, что каждый процессор или ядро имеет собственные независимые ресурсы.

* **RISC** (Reduced Instruction Set Computer) и **CISC** (Complex Instruction Set Computer): RISC-архитектура предполагает использование простых и основных инструкций, сокращая сложность и увеличивая скорость выполнения инструкций. CISC-архитектура, наоборот, использует сложные инструкции, которые могут выполнять более сложные операции за одну инструкцию.

* **Von** **Neumann** и **Harvard** архитектуры: Von Neumann архитектура является классической архитектурой, где данные и инструкции хранятся в одной общей памяти и обрабатываются процессором последовательно. Harvard архитектура предусматривает разделение памяти на две части: одна для хранения данных, а другая для хранения инструкций, что позволяет параллельное выполнение и более эффективное использование ресурсов.

* **Superscalar** и **VLIW** (Very Long Instruction Word): Superscalar-архитектура позволяет параллельное выполнение нескольких инструкций за один тактовый цикл. VLIW-архитектура, напротив, позволяет компилятору сгруппировать несколько инструкций в одно длинное слово, которое затем выполняется параллельно.

34. Архитектуры компьютеров параллельного действия, классификация параллельных компьютерных систем (Тима)

**Компьютеры параллельного действия** — это компьютерные системы, способные выполнять несколько вычислительных задач одновременно или параллельно.

Примеры организации архитектуры параллельных компьютеров:

* Параллелизм на уровне процессора (Параллелизм уровня команд, многопоточность)
* Гомогенные и гетерогенные однокристальные мультипроцессоры
* Распределенные вычислительные системы(кластеры компьютеров)

**Классификация**:  
По уровню параллелизма:

* Уровень команд: Параллельные системы, которые достигают параллелизма на уровне инструкций или команд.
* Уровень данных: Параллельные системы, которые достигают параллелизма на уровне данных, обрабатывая несколько данных одновременно.
* Уровень задач: Параллельные системы, которые достигают параллелизма на уровне задач или процессов, где каждая задача может быть выполнена независимо друг от друга.

Классификация по архитектуре:

* SIMD (Single Instruction, Multiple Data): Этот тип архитектуры предполагает выполнение одной инструкции над множеством данных одновременно. Примером SIMD-архитектуры являются векторные процессоры, которые могут обрабатывать одновременно несколько элементов данных с помощью одной инструкции.
* MIMD (Multiple Instruction, Multiple Data): В этом типе архитектуры каждый процессор может выполнять независимые инструкции над своими данными. Это позволяет параллельное выполнение различных задач на разных процессорах. Примерами MIMD-архитектур являются многопроцессорные системы, кластеры и распределенные системы.

По гомогенности и гетерогенности:

* Гомогенные системы: Системы, в которых все процессоры имеют одинаковую архитектуру и характеристики.
* Гетерогенные системы: Системы, в которых процессоры имеют различные архитектуры и характеристики.

35. Семейства мультипроцессоров (Женя)

У Маркова 3 семейства, они различаются по способу реализации памяти совместного использования:

**UMA** (Uniform Memory Access)-архитектура с однородным доступом к памяти

**NUMA** (Non Uniform Memory Access)- архитектура с неоднородным доступом к памяти

**COMA** (Cache Only Memory Access)- архитектура с доступом только к кэш-памяти

**UMA** предполагает, что все процессоры в системе имеют равный доступ к общей памяти, которая обычно является разделяемой. Это означает, что каждый процессор может обращаться к любому адресу в памяти с одинаковой скоростью и задержкой. Процессоры в системе обмениваются данными через общую шину

В **NUMA**-архитектуре каждый процессор имеет свою локальную память, которая доступна с низкой задержкой и высокой пропускной способностью. Однако доступ к удаленной памяти сопряжен с более высокой задержкой и меньшей пропускной способностью из-за необходимости передачи данных через сеть или шину.

В **COMA**-архитектуре каждый узел, включая процессор и память, имеет свой собственный кэш, и данные перемещаются между узлами в виде кэш-линий. При обращении к данным, узлы сначала ищут их в своем локальном кэше, и только если данные отсутствуют, происходит обращение к более удаленным узлам, а затем к внешней памяти. Внешняя оперативная память рассматривается как распределенный кэш. В COMA доступ к данным может быть дольше, чем в NUMA, при условии что данных нет ни в своем локальном кэше, ни в локальных кэшах других узлов (приходится обращаться во внешнюю память) .

36. Мультикомпьютеры, коммуникационные взаимодействия (с примерами схем построения)(Женя)

Коммуникационные взаимодействия в мультикомпьютерах могут быть реализованы различными способами, такими как сообщения, обмен данными через сеть, распределение задач и синхронизация работы. Это позволяет компьютерам обмениваться информацией, совместно решать задачи и достигать более высокой производительности в сравнении с отдельными компьютерами.

Примеры схем построения мультикомпьютеров:

1. Система с распределенной памятью (Distributed Memory System): В этой схеме каждый компьютер имеет свою локальную память, и взаимодействие между компьютерами осуществляется через сеть. Каждый компьютер выполняет свою часть задачи и обменивается данными при необходимости. Примеры включают кластеры компьютеров, где каждый узел является отдельным компьютером со своей памятью.
2. Система с общей памятью (Shared Memory System): В этой схеме несколько компьютеров имеют общую память, к которой все они имеют доступ. Компьютеры могут обмениваться данными, читать и записывать в общую память. Примером является мультипроцессорная система, где несколько процессоров совместно используют общую память.

Коммуникационные взаимодействия в мультикомпьютерах могут осуществляться различными способами:

Сообщения: Узлы мультикомпьютера могут обмениваться сообщениями, передавая данные и команды друг другу. Это может происходить через сетевые соединения, где каждый узел имеет свой собственный адрес и может отправлять и принимать сообщения от других узлов.

Общая память: Некоторые мультикомпьютеры могут иметь общую память, к которой все узлы имеют доступ. Это позволяет им обмениваться данными и совместно использовать память для распределенных вычислений.

  Распределенные вычисления: Узлы мультикомпьютера могут выполнять вычисления параллельно, разделяя задачи между собой и совместно обрабатывая данные. Для этого они могут использовать специальные протоколы и алгоритмы для согласования и синхронизации своих действий.

37. Кластерные вычисления, принципы организации (с примерами)(Женя)

Кластерные компьютеры, также известные как системы COW (Cluster of Workstations) или NOW (Network of Workstations), состоят из нескольких компьютеров или рабочих станций, соединенных посредством сетевых плат. Они делятся на два типа: централизованные и децентрализованные. Централизованные кластеры объединяют компьютеры в одном помещении, а децентрализованные могут быть разделены по разным местам и связаны через локальную сеть. Кластерные компьютеры обладают преимуществом гибкости и низкой стоимости, поскольку состоят из доступных компонентов, их можно легко масштабировать и их цены постоянно снижаются. Примером кластерных компьютеров являются системы, разработанные компанией Google.

1. Высокая доступность: Кластерные системы обеспечивают высокую доступность приложений и данных путем дублирования и резервирования ресурсов. Если один узел выходит из строя, другие узлы могут продолжать работу без прерывания.
2. Распределение нагрузки: Задачи распределяются между узлами кластера для оптимального использования вычислительных ресурсов. Это позволяет достичь высокой производительности и сократить время выполнения задач.
3. Масштабируемость: Кластерные системы могут быть легко масштабируемыми путем добавления новых узлов в кластер. Это позволяет увеличить вычислительную мощность системы для обработки более сложных задач или увеличения нагрузки.

**Принципы организации кластерных вычислений включают:**

1. Распределение задач: Задачи распределяются между узлами кластера для параллельной обработки. Это позволяет достичь высокой производительности и ускорить выполнение задач.
2. Коммуникация: Узлы кластера должны быть связаны между собой через сеть, чтобы обеспечить обмен данными и координацию работы. Коммуникационная инфраструктура кластера должна быть высокоскоростной и эффективной.
3. Управление ресурсами: Кластер должен иметь механизмы управления ресурсами, такие как планировщик задач и мониторинг системы. Это позволяет эффективно использовать вычислительные ресурсы кластера и управлять нагрузкой.
4. Надежность и отказоустойчивость: Кластеры могут быть организованы с использованием резервирования ресурсов и механизмов восстановления после сбоев, чтобы обеспечить надежную работу системы даже в случае отказа отдельных узлов.

38. Распределенные вычисления, принципы работы (с примерами)(Женя)

 Целью системы распределенных вычислений является создание технической инфраструктуры, которая позволила бы из нескольких организаций, решающих общую задачу создать единую виртуальную организацию. Для реализации этой цели разрабатываются службы, инструменты и протоколы, что позволяет отдельным членам функционировать в рамках единой виртуальной организации. Система распределенных вычислений многомерна, с большим количеством участников – одноранговых узлов. Отличие от модели «клиент-сервер» состоит в том, что соединение не двухточечное, а многомерное.

Распределенные вычисления - это подход, при котором задача решается путем совместной работы нескольких компьютеров или узлов, объединенных в сеть. Вот некоторые принципы работы распределенных вычислений:

1. Разделение задачи: Задача разбивается на подзадачи, которые могут быть выполнены независимо друг от друга. Эти подзадачи распределяются между узлами сети, которые выполняют вычисления параллельно.

Пример: Расчет сложной математической модели разбивается на несколько частей, каждая из которых выполняется на отдельном компьютере сети.

1. Координация и обмен данными: Узлы сети взаимодействуют и обмениваются данными, необходимыми для выполнения задачи. Существуют различные протоколы и алгоритмы для координации и синхронизации работы узлов.

Пример: В системе обработки больших данных (Big Data) различные узлы сети собирают, анализируют и обмениваются большим объемом данных для выполнения сложных вычислений.

1. Отказоустойчивость: Распределенные вычисления обеспечивают надежность и отказоустойчивость путем использования резервирования и дублирования узлов. Если один узел выходит из строя, другие узлы могут продолжить работу без проблем.

Пример: В распределенных системах хранения данных, если один сервер выходит из строя, данные автоматически реплицируются на другие серверы для обеспечения доступности и надежности.

1. Масштабируемость: Распределенные вычисления могут быть масштабированы путем добавления новых узлов или ресурсов в сеть. Это позволяет увеличивать производительность и обрабатывать более сложные задачи.

Пример: Облачные вычисления предоставляют возможность масштабирования вычислительных ресурсов по требованию, позволяя компаниям адаптироваться к меняющимся потребностям и нагрузкам.

Эти принципы позволяют эффективно использовать ресурсы и повышать производительность в распределенных вычислениях, обеспечивая параллельное выполнение задач и совместную работу узлов сети.