1.Что такое компьютерная система? Отличие информационной и управляющей системы? Почему большинство современных компьютерных систем считаются системами с преобладающей программной составляющей? Примеры.

Компьютерная система — любое устройство или группа взаимосвязанных или смежных устройств, одно или более из которых, действуя в соответствии с программой, осуществляет автоматизированную обработку данных.

Примеры компьютерной системы это автомобиль (беспилотный автомобиль), станок

Информационные компьютерные системы, основная задача которых получить набор данных на вход, преобразовать/накопить его, и выдать в изменено/обработанном виде

Управляющие компьютерные системы, основная задача которых взаимодействовать с реальным физическим миром с целью контроля или управления за ним.

Системы с преобладающей программной составляющей - это системы, в которых разработка и интеграция программного обеспечения являются приоритетными аспектами. К ним относятся компьютерные системы, начиная от отдельных программных приложений, информационных систем, встроенных систем, программных продуктовых линеек и семейств продуктов и систем систем.

{#fig:HW-SW-поддержка-стоимость}

Приходя данные -> обработка -> ответ

2. Понятие системы. Варианты рассмотрения систем. Модульность. Жизненный цикл. Операционное окружение и обеспечивающие системы. Идентичность системы. Заинтересованные стороны (stakeholders).

Система - это конструкция из взаимодействующих компонентов(полупроводники, конденсаторы, резисторы, триггеры и т.д.), включающая в себя множество информационных компонентов(программы, данные)

Система может рассматриваться как продукт или как услуги, которые она предоставляет.

Вспомогательная (обеспечивающая) система - система, которая дополняет интересующую систему на этапах ее жизненного цикла, но не обязательно вносит непосредственный вклад в ее функционирование во время эксплуатации.

Модульность - выбор гранулярности компонентов, уровня абстракции и вычислительных платформ, который имеет решающее значение при разработке компьютерной системы. То есть вся внутренняя организация системы состоит из взаимосвязанных модулей, которые в совокупности определяют целевую систему.

Стадии ЖЦ:

Этап концепции: выполняется для оценки новых возможностей бизнеса и разработки предварительных требований к системе и осуществимого проектного решения.
Этап разработки: выполняется для создания интересующей системы, которая отвечает требованиям заказчика и может быть произведена, протестирована, оценена, эксплуатируется, поддерживается и выводится из эксплуатации. Для
Этап производства: выполняется для производства или изготовления продукта, тестирования продукта и производства соответствующих вспомогательных и обеспечивающих систем по мере необходимости. Для
Этап использования: выполняется для эксплуатации продукта, предоставления услуг в намеченных условиях и обеспечения постоянной операционной эффективности.
Этап поддержки: выполняется для предоставления услуг логистики, технического обслуживания и поддержки, которые обеспечивают непрерывную работу интересующей системы и устойчивое обслуживание.
Этап вывода из эксплуатации: выполняется для обеспечения удаления интересующей системы и соответствующих эксплуатационных и вспомогательных услуг, а также для эксплуатации и поддержки самой выводимой из эксплуатации системы.

Операционная среда — это совокупность инструментов, методов их интеграции и приёмов работы с ними, позволяющая пользователю решать любые задачи в инструментальной области и большинство задач в прикладных областях.

Stakeholder - физическое и юридическое лицо, которое может накладывать ограничения на вашу систему т.е. например, пользователь, который определяет дизайн, разработчик, определяющий какие именно технологии будут использоваться, директор, заинтересованный в актуальности и спросе на рынке.

Идентичность системы

Представим себе насосную станцию, в которой стоит два насоса. Каждый насос имеет собственный агрегат: у первого серийный номер 1.1.1 и у второго 1.2.1. Первый насос ломается и мы заменяем в нем агрегат с другим номером 1.1.2. В результате, мы получаем, что насосная станция и насосы в ней не изменились, это все те же конструкции, но уже используется другой агрегат с другим номером 1.1.2. В этом видимо и заключается идентичность системы: то есть с внешней стороны станция не изменилась, так как она функционирует так же, как и раньше, но если посмотреть внутрь,, то там стоит теперь другой агрегат и это абсолютно другая по составу система.

3.Цели и задачи архитектурного проектирования компьютерных систем. Понятие архитектуры. Различные трактовки и их практическая значимость.

Архитектура - это набор проектных решений, которые, если они будут приняты неправильно, могут привести к краху проекта.

Ключевыми особенностями архитектуры системы являются ее многочисленные представления. Например, UML, который включает в себя множество типов диаграмм для описания системы для различных целей.

Архитектура - логическая и физическая структура компонентов системы и их взаимосвязей, сформированная стратегическими и тактическими проектными решениями, применяемыми в процессе разработки.

Логическая структура содержит в себе концепции, созданные в концептуальной модели, и устанавливает существование и значение ключевых абстракций и механизмов, которые будут определять архитектуру и общий дизайн системы.

Физическая структура описывает конкретный программный и аппаратный состав реализации системы. Очевидно, что физическая модель зависит от конкретной технологии.

Использование архитектуры в разработке программного обеспечения и систем приводит к следующим последствиям:

б) Что влияет на создание архитектуры компьютера?

• **команда и компетентность**, например, используемый стек технологий, если по нему нет специалистов, то это значительно повышает риски при создании проекта;

• **время выхода на рынок**: если внедрение системы займет несколько лет, существует высокая вероятность того, что по истечении этих сроков она станет бесполезной;

• **обслуживание, поддержка**, развертывание и обновление системы; • и много другого.

в) Для того чтобы добавить функциональные возможности в существующую систему без архитектурной документации НУЖНО:

1) Добавить новую функцию в систему методом “brutal force”. Обычно это можно быстро сделать за счет обеспечения согласованности системы, но если это долговечный продукт, это значительно увеличит затраты на техническое обслуживание;

либо

2) Проанализируйте архитектуру системы (если возможно) и интегрируйте новую функцию. Такой подход позволяет элегантно расширить существующую вычислительную систему, не внося в нее концептуальных несоответствий.

Система имеет архитектуру вне зависимости от ее описания. В качестве примера, вам необходимо добавить новые функциональные возможности в существующую систему без архитектурной документации. Существует два способа выполнения этих задач:

- Добавьте новую функцию в систему методом "грубой силы". Обычно это можно быстро сделать за счет обеспечения согласованности системы, но если это долговечный продукт, это значительно увеличит затраты на техническое обслуживание.
- Проанализируйте архитектуру системы (если возможно) и интегрируйте новую функцию. Такой подход позволяет элегантно расширить существующую вычислительную систему, не внося в нее концептуальных несоответствий.

Методы системного проектирования и архитектурного проектирования нацелены на то, чтобы помочь людям решить все необходимые для достижения успеха вопросы.

4.Реле как базис компьютерной системы. Область применений и принципы построения систем на базе реле. Примеры релейных схем.

Реле́ (фр. *relais*) — коммутационный аппарат, который при воздействии на него внешних физических явлений скачкообразно принимает конечное число значений выходной величины[1].

Назначение реле заключается в автоматизации замыкания или размыкания электрической цепи.

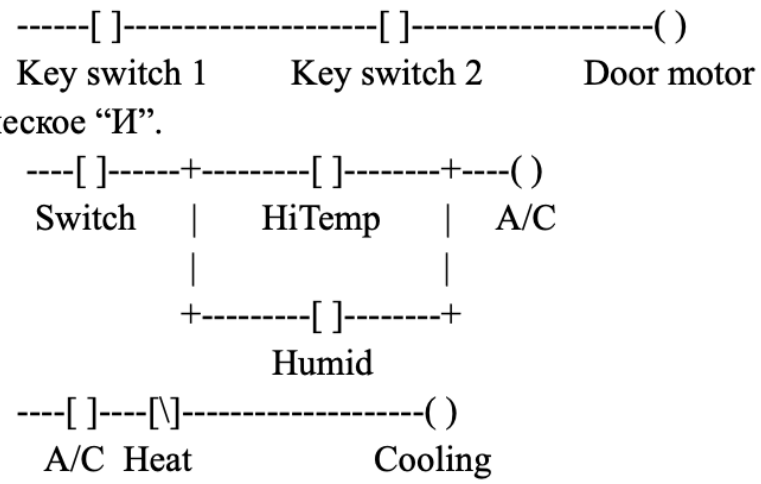
1. В основе вычислительной платформы “релейных схем” лежит относительно простое устройство – реле.

2. вход и выход, между ними металлический ключ (нормально разомкнутый или нормально замкнутый);

3. магнитная катушка, при подаче тока на которую ключ автоматически замыкается или размыкается, при снятии тока – ключ возвращается в нормальное состояние автоматически.

• Релейные схемы управления используются по сей день
• реле бывают не только электрические, но и механические (включая пневматические), тепловые, оптические, акустические и магнитные, что позволяет применять их в тех областях, в которых электрические компоненты не могут работать;

Пример простых релейных схем, выполненных в нотации языка IEC 61131.



Управление холодильником, логическое “И” и “ИЛИ”

1. Реализует функцию: A/C = Switch AND (HiTemp OR Humid).
2. Реализует функцию: Охлаждение = A/C И (НЕ Тепло).

5.Что такое комбинационная схема? Состояние и параллелизм в комбинационных схемах и схемах с регистрами. Особенности поведения систем на базе комбинационных схем от программных систем.

Комбинационная схема – схема составленная из набора логических элементов, (реализует таблицу истинности).

Основные свойства любой комбинационной схемы:

• возможность установления стабильного состояния при корректном входе;

• параллельная работа элементов комбинационной схемы;

• накопление ошибки в физическом процессе, что может привести к ошибке на логическом уровне.

Синхронные схемы позволяют нам делать следующее:

• Конвейеризация вычислений (количество стадий конвейера равно количеству обрабатываемых в такт значений);

• Управлять тактовой частотой схемы, так как мы всегда можем разделить комбинационную схему триггером на часть до и после, при этом если

1) комбинационные схемы делятся ровно пополам, то у нас нет избыточных задержек;

2) триггер имеет свою длительность срабатывания.

Особенности “условного оператора”

В булевых компьютерах ветвление реализовано в два подхода:

• через состояние (по сути реализован в современных процессорах, когда состояние регистров определяет следующий шаг вычислительного процесса);

• через спекулятивные вычисления и выбор результата (мультиплексор).

Основы работы логических компьютеров:

• все процессы выполняются параллельно, последовательность обработки данных – логическая конструкция которая отсутствует внутри

• Система постоянно продолжает функционировать

• Происходит передача сигнала – физический аналоговый процесс, а значит он не может прекратиться, если мы не определили какой сигнал пойдет дальше, это значит что дальше пойдет случайный сигнал.

6.Понятия Hardware и Software, их свойства. Сравнение с понятиями программного и аппаратного обеспечения. Особенности. Причины разделения.

Вообще, hardware и software это аппаратное и программное обеспечение

Hardware - Аппаратное обеспечение

В целом - все что не является программами и данными. То есть - оборудование для питания, аккумуляторы, внешние устройства, логическое устройства. Все, что является частью системы или сети

Software - Программное обеспечение

Все программы для хранения, обработки и поиска информации, работы с файлами и другими ресурсами.

Кроме того есть понятие **Компьютерная программа** - взаимодействие и комбинация компьютерных инструкций и данных. Единственная цель - дать возможность аппаратному обеспечению выполнять вычисления.

Система определяется своей ролью в надсистеме, а не только своим наполнением. При обсуждении HW и SF мы говорим не столько о том что написано выше, а о том, что, как и как дорого мы можем изменить.

И тогда мы подходим к жизненному циклу всей комп. системы:

- производство;
- сборка и комплектация --все зависит от того, как, в каком порядке мы соберем нашу систему. От этого зависит получение системы с конкретными необходимыми возможностями.
    - макетные платы;
    - платы расширения:
        - расширение портов ввода/вывода, включая обработку данных;
        - предоставления специализированных вычислителей под конкретные задачи;
        - фиксация алгоритмов на уровне аппаратуры;
- [ре-]конфигурирование -- настройка функционирования аппаратных средств, управление режимом работы аппаратных средств;
    - джамперы, перемычки, дип-переключатели;
    - конфигурация аппаратных узлов для реализации требуемой функциональности;
    - конфигурирование данными;
- программирование -- определение компьютерной программы, определяющей реализуемую компьютерной системой функциональность;
- настройка / пользовательское программирование -- определение настроек программных продуктов.

Разделение в первую очередь условное. Нельзя говорить о полноте или корректности.

Особенности аппаратной составляющей

- У аппаратной части компьютерной системы есть свой срок службы по истечению которого она лишается гарантии, риск внезапного выхода их строя резко возрастает, она требует обслуживания.
- Замена аппаратной составляющей затруднена: сложностью и дороговизной долгого хранения деталей для ремонта или замены. Иногда изменение аппаратной составляющей подразумевает физический контакт с устройством, что не всегда физически возможно
- Воспроизводство аппаратной составляющей сопряжено с устареванием элементной базы. Устаревшая элементная база вытесняется из производства новой, как следствие она довольно быстро переходит в разряд штучного, который:
    - сперва производится несколькими компаниями за очень большие деньги;
    - потом не производится совсем (периодически можно видеть как крупные компании ищут то или иное оборудование по барахолкам, есть и компании которые на этом специализируются).

Изменение ПО:

- надо применить патч к программной документации
- повторно собрать проект
- обновить программное обеспечение на компьютерах и перезапустить
- легкость изменения ПО является одной из причин огромного количества уязвимостей компьютерных систем.

Hardware - то что тяжело/долго/дорого поменять;
Software - то что легко/быстро/дешево поменять.

Разделение зависит от свойств обеспечивающей системы. То есть от того, как мы смотрим на проблему

“команды и коды не стареют”, чего к сожалению нельзя сказать о командах разработчиков. Как следствие – держатель компьютерной системы рано или поздно в любом случае будет терять над ней контроль.

7.Состав программной системы в соответствии с OMG (Object management group) Essence. Роль данных и аппаратного обеспечения.

OMG (*Object Management Group*) — рабочая группа, занимающийся разработкой и продвижением объектно-ориентированных технологий и стандартов. Это некоммерческое объединение, разрабатывающее стандарты для создания интероперабельных, то есть платформо-независимых, приложений на уровне предприятия.

Они определяют основы разработки программного обеспечения и общую точку зрения.
В состав входит:

- **Дисциплина** -- это что-то типа модели предметной области. Они определяют важные понятия, которые общие для любого, который работает в этой предметной области (системная инженерия, программная инженерия).
- **Практика** -- это описание того, как справиться с каким-то отдельным аспектом инженерного проекта как в собственно инженерной части, так и в части организации работы команды, включая стратегирование. Практика даёт имя группе элементов, которая составляет руководство по достижению указанной цели. Практика может быть определена и как составленная из других практик.
- **Метод** -- это сочетание дисциплины и набора практик для достижения какой-то цели. Бывают методы разработки, методы сопровождения, методы интеграции систем, и т.д.
- **Библиотека** -- это именованный контейнер, в котором содержится коллекция групп элементов (и только групп -- т.е. дисциплин, практик, методов и т.д.) для какой-то определённой цели.

Роль аппаратного обеспечения заключается в том, что он него зависит, насколько быстро будет реагировать система на попадающие в нее данные и насколько быстро ,она будет их просчитывать. От данных, также зависит скорость работы, например от их количества, их значений, и расположения в котором они поступают в систему, от всего этого зависит скорость работы многих алгоритмов

8.Закон Мура и закон Деннарда. Их роль в развитии компьютерной техники.

Закон Мура: количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца.

он сдерживается следующими факторами:

• **закон Амдала**. Данный закон характеризует рост производительности системы относительного того, какой процент задач может быть выполнен параллельно и количества доступных вычислителей (см. картинку); Нет смысла чрезмерно наращивать количество параллельно идущих вычислений, если программа не будет их все задействовать или минимальное время будет зависеть от самого медленного процесса.

• объективная сложность параллельного программирования. Программистам сейчас помогает абстракция, благодаря которой они работают с простыми ситуациями
• доставка данных, так как для работы необходимо не только получить входные данные, но также и выдать полученные результаты (возможно и промежуточные);
• ограничения связанные с физическими процессами (к примеру, когда не остается атомов для того, чтобы сложить затвор транзистора). СЮда же тепловыделение и питание;

Последние 50 лет направление и темп развития электроники определялись законом Мура.

Стоит выделить 3 типа влияния закона Мура на мир:

- борьба за первенство в создании более быстрого и более мощного процессора;
- разработка архитектуры вычислительных мощностей: обновление технологических алгоритмов происходит постоянно и регулярно (раз в 2 года);
- прогнозирование рынка.

Начиная с 2010 года, следование закону Мура перестало быть выгодным для разработчиков. На его соблюдение нужно тратить множество ресурсов: материалы, оборудование, увеличение штата и проч. На 2019 год закон Мура не работает эффективно, эра кремниевых транзисторов завершится предположительно до 2030 года.

К сожалению, как и закон Мура, данный закон имеет свою область применимости и срок годности, так как наталкивается на:

• Ограничения современных технологий (сложность и стоимость производства).
• Физические ограничения на возможности уменьшения транзисторов (размеры атомов и эффекты связанные с уменьшением транзисторов).

Закона масштабирования Деннарда –смысл в том, что уменьшая размеры транзистора и повышая тактовую частоту процессора, мы можем легко повышать его производительность.

Правило Деннарда закрепило уменьшение ширины проводника (сейчас это называют техпроцессом) в качестве главного показателя прогресса. Но закон масштабирования Деннарда перестал действовать примерно в 2006 году. Количество транзисторов в

чипах продолжает увеличиваться, но этот факт **не дает значительного прироста** к производительности устройств.

Например, представители TSMC (производитель полупроводников) говорят, что переход с 7-нм техпроцесса на 5-нм увеличит тактовую частоту процессора всего на 15%.

Причиной замедления роста частоты, **являются утечки токов**, которые Деннард не учитывал в конце 70-х. При уменьшении размеров транзистора и повышении частоты ток начинает сильнее нагревать микросхему, что может вывести ее из строя. Поэтому производителям приходится балансировать выделяемую процессором мощность. В результате с 2006 года частота массовых чипов установилась на отметке в 4–5 ГГц.

9.Понятие модели вычислений. Примеры моделей вычислений и их роль в разработке компьютерных систем. Модель-ориентированная инженерия

Модель вычислений (дальше МВ) - это модель, которая описывает, как вычисляется результат математической функции с учетом входных данных.

Модель описывает, как организованы единицы вычислений, памяти и связи.

С помощью данной модели можно измерить вычислительную сложность алгоритма.

Задачи МВ:

1) решает какие вычислительные процессы могут исполняться при помощи доступной нам компьютерной системы.
2) определяет ограничения, в которых должен быть реализован вычислитель (Нарушение ограничений модели вычислений приводит к росту сложности системы и появлению “дыр” в безопасности или предсказуемости)

Модели вычислений можно разделить на три категории:

1) Последовательные модели (позволяют описать последовательные процессы)

– Конечные автоматы

– Нажимные автоматы

– Машины произвольного доступа – Машины Тьюринга.

2) Функциональные модели (вычислительныие процессы представляются в символьной форме):

– Лямбда-исчисление

– Общие рекурсивные функции

– Комбинаторная логика

– Системы переписывания абстрактных текстов.

3) Параллельные модели, (используется, когда процесс взаимодействует с другими процессами):

– Клеточный автомат

– Технологические сети Кана – Сети Петри

– Синхронный Поток Данных – Сети взаимодействия

– Модель-актер.

Модельно-ориентированная инженерия (**MDE**) - это методология разработки программного обеспечения, которая фокусируется на создании и использовании моделей предметной области.

Подход MDE предназначен для:

1) повышения производительности за счет максимальной совместимости между системами (путем повторного использования стандартизованных моделей)
2) упрощения процесса проектирования (с помощью моделей повторяющихся шаблонов проектирования)
3) содействия коммуникации между отдельными людьми и группами, работающими над системой (посредством стандартизации терминологии и передовых практик , используемых в области приложения)

10.Понятие информационного процессора. Машина Тьюринга. Свойства универсального компьютера.

Информационный процессор - система , которая принимает информацию в одной форме и преобразует ее в другую форму, например к статистике через алгоритмический процесс.
Системы обработки информации состоит из четырех основных подсистем:

- ввод
- процессор
- хранение
- вывод

Процессор - электронный блок либо интегральная схема, исполняющая машинные инструкции, главная часть аппаратного обеспечения компьютера или программируемого логического контроллера.

Машина Тьюринга — модель абстрактного вычислителя, предложенная британским математиком Аланом Тьюрингом в 1936 году. Эта модель позволила Тьюрингу доказать два утверждения. Первое — проблема останова неразрешима, т.е. не существует такой машины Тьюринга, которая способна определить, что другая произвольная машина Тьюринга на её ленте зациклится или прекратит работу. Второе — не существует такой машины Тьюринга, которая способна определить, что другая произвольная машина Тьюринга на ее ленте когда-нибудь напечатает заданный символ.

Автомат машины Тьюринга в процессе своей работы может выполнять следующие действия**:**

- Записывать символ внешнего алфавита в ячейку (в том числе и пустой), заменяя находившийся в ней (в том числе и пустой).
- Передвигаться на одну ячейку влево или вправо.
- Менять свое внутреннее состояние.

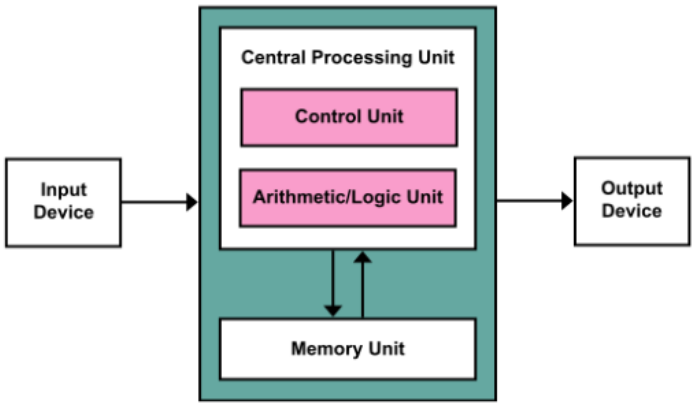
Логика работы:

Машина Тьюринга состоит из бесконечной в обе стороны ленты, разделенной на ячейки, и автомата (головки), которая управляется программой.

Программы для машин Тьюринга записываются в виде таблицы, где первые столбец и строка содержат буквы внешнего алфавита и возможные внутренние состояния автомата (внутренний алфавит). Содержимое таблицы представляет собой команды для машины Тьюринга. Буква, которую считывает головка в ячейке (над которой она находится в данный момент), и внутреннее состояние головки определяют, какую команду нужно выполнить. Команда определяется пересечением символов внешнего и внутреннего алфавитов в таблице.

Универсальный компьютер — компьютер, предназначенный для решения широкого класса задач. Компьютеры этого класса имеют разветвленную и алгоритмически полную систему операций, иерархическую структуру запоминающих устройств и развитую систему устройств ввода-вывода данных.

11.Архитектура фон Неймана. Принципы. Свойства. Особенности и ограничения. Применение на практике.



Смотрим на картинку и рассказываем что видим.
Машина Неймана - некоторая эволюция машины Тьюринга. Память, использующая произвольный доступ, заменят ленту в машине тьюринга. Система команд в устройстве управления преобразовалась в более привычную форму.

Данная архитектура основана на следующих принципах:

• Память компьютера используется для хранения данных и программ.Все кодируются в двоичном формате. Поэтому в некоторых случаях вы можете выполнять с командами те же действия, что и с данными.

• Преимущество двоичной системы состоит в том, что устройства могут быть сделаны довольно простыми, а арифметические и логические операции в двоичной системе также довольно просты.

• Возможность условного ответвления во время выполнения программы. Команды выполняются последовательно, но программы могут реализовать возможность перехода к любой части кода.

• Управление программным обеспечением. Команды выполняются последовательно, одна за другой. Создание машины с программой, хранящейся в памяти, было началом того, что мы сегодня называем программированием. (Наиболее условным является пункт о последовательном исполнении команд в современных процессорах);

• Ячейки памяти компьютера имеют пронумерованные адреса. В любое время вы можете получить доступ к любой ячейке памяти по ее адресу. Этот принцип позволил использовать переменные в программировании.

– Понятие адреса не сводится к целому числу в реальных процессорах.
– Память в современных процессорах не является пассивным элементом

Сегодня чистая архитектура фон Неймана не используется на практике. Но его элементы присутствуют в подавляющем большинстве процессоров и средств разработки.

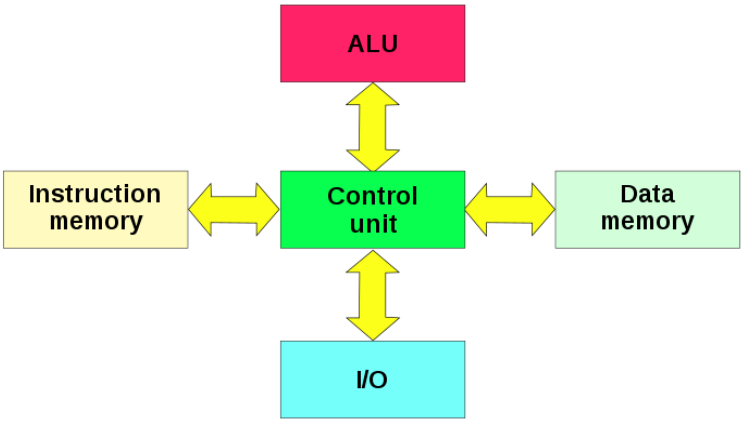
(Про это говорим по желанию, если кажется что мало. Это к особенностям Неймановской архитектуры)
Система команд процессора – абстрактная модель процессора, формирующая интерфейс взаимодействия между программным обеспечением и процессором, затрагивающая:

- типы данных;
- систему регистров;
- методы адресации;
- модели памяти;
- инструкции;
- способы обработки прерываний и исключений;
- методов ввода и вывода.

Система команд процессора описывает поведенческий аспект работы процессора и напрямую не касается вопросов производительности, энергопотребления и временных задержек. Сейчас это не очень круто, и думают над тем чтобы пересмотреть такой подход

12.Гарвардская архитектура. Принципы. Свойства. Особенности и ограничения. Применение на практике.

Гарвардская архитектура – архитектура, в которой память команд и память данных физически разделены, имеют собственные наборы шин для взаимодействия. Она широко применяется во внутренней структуре современных высокопроизводительных микропроцессоров, где используется отдельная кэш-память для хранения команд и данных.



Принципы архитектуры:

- Одновременный доступ к памяти (многопортовая память является редкостью).
- Разные длины машинного слова и адреса для данных и программ.
    - Оптимизация под решаемую задачу.
    - Данные и память программ всегда перемешаны (например, непосредственная адресация и указатели на функции).
- Два физических канала между процессором и памятью.

Свойства архитектуры:

Существует множество реализаций данной архитектуры:

- **Модифицированная гарвардская архитектура.** Доступ к памяти реализуется через независимые кеши для данных и программ, за счет чего, с точки зрения внутренней организации процессора доступ реализован независимо, при этом канал между процессором и памятью один.
- **Архитектура "Память инструкций как данные"** (Instruction-memory-as-data) – реализуется возможность читать и писать данные в память программ. Позволяет генерировать и запускать машинный код.
- **Архитектура "Данные как память для инструкци**й" (Data-memory-as-instruction) – реализует возможность запуска инструкций из памяти данных. Также позволяет генерировать и запускать машинный код, при этом параллельный доступ в некоторых экземплярах реализуется за счет возможности параллельной работы с разными сегментами памяти.

Особенности и ограничения:

- Разрядность процессора: 8 бит.
- Организация памяти: гарвардская, с отдельными блоками памяти для команд и данных.
- Внешние устройства отображаются в адресное пространство данных. Работа с ними выполняется по запросу.
- Регистры:
    - PC -- счетчик команд

    - IR -- регистр инструкций
    - AR -- регистр адреса операнда
    - C -- флаг переноса/займа
    - Z -- флаг нуля
- Команды выполняются за 2 или 3 такта (в зависимости от типа команды):

    1)Выборка команды

    2)Выборка операндов

    3)Выполнение команды.

- Подсистема обработки прерываний и команды вызова подпрограмм отсутствует.

13.Механизм микроопераций и его роль в развитии компьютерных систем. Особенности и ограничения. Применение на практике.

Изначально процессоры были реализованы полностью на аппаратном уровне.

Ассемблер - был основным инструментом программирования.

Но так как хотелось увеличить производительность и сделать процесс программирования более удобным. Был придуман метод введения абстракций, но это приводило к высоким накладным расходам (использование подпрограмм);

Поэтому было придуманно увеличение системы команд с целью естественной реализации всех необходимых инструкций.

В увеличение системы команд входило добавление новых арифметических операций, создание большого количества типовых команд (команды, которые используют большое количество аргументов, сохраняют результат не в регистр, а сразу в память и т.п.)

Таким образом для множества команд нет необходимости добавлять новую логику в процессор (шины данных, регистры, сигнальные линии), необходимо только специфическим образом использовать уже имеющиеся.

Инструкции прибывают из памяти, обычно из высокоскоростного кэша.

Далее они идут в декодер, который разбивает каждую инструкцию на одну или несколько микроопераций.

Но это приводит к необходимости "программирования" системы команд процессоров. Это привело к понятию микрокода – программы, реализующей набор инструкций процессора.

14.Что такое CISC? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана.

Аббревиатура CISC обозначает Complex Instruction Set Computer - тип процессорной архитектуры, которая характеризуется следующим набором свойств:

- нефиксированное значение длины команды;
- арифметические действия кодируются в одной команде;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

Описание:

Архитектура CISC в значительной степени стала возможна благодаря возможностям микропрограммного управления. Для каждой инструкции из набора создается подпрограмма, которая состоит из простых инструкций и хранится в специальной памяти внутри микропроцессора. Таким образом, процессор содержит небольшой набор простых инструкций. На их основе можно создать множество сложных инструкций из набора команд с помощью добавления подпрограмм в микрокод.

Микропрограмма позволяет относительно легко формировать большое количество команд со сложным поведением. Поднять уровень машинных инструкций ближе к уровню языков программирования высокого уровня. К примеру:

- относительно легко поддержать все интересующие варианты адресаций для всех типов команд;
- реализовать операции с произвольным набором аргументов, к примеру инструкцию для расчета многочленов в рамках одной инструкции;
- реализовать операции работающие в потоковом режиме.

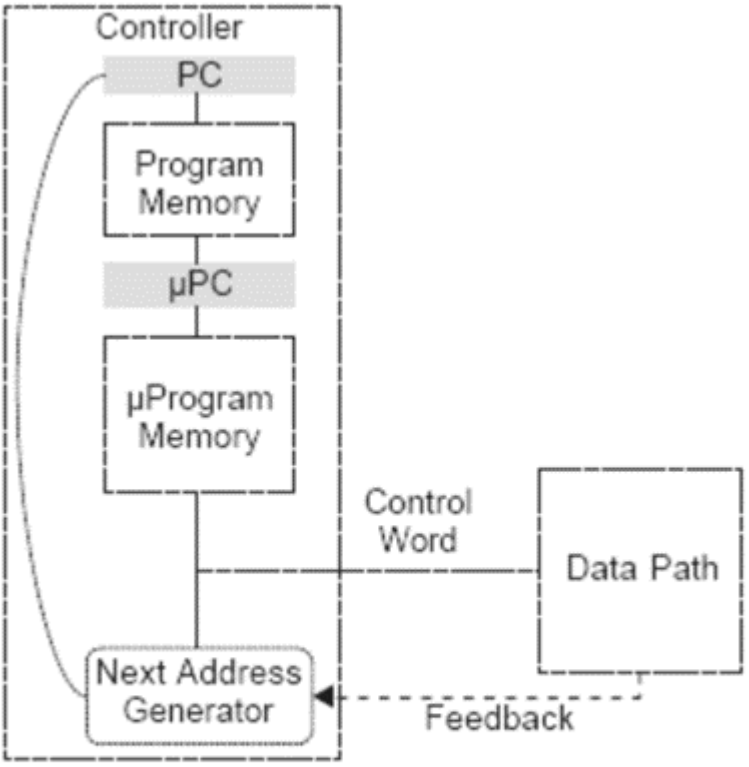
Достоинства:

- удобство использования ассемблера как инструмента программирования;
- сокращение затрат на доступ к памяти команд, так как гибкость позволяет сократить количество инструкций;
- возможность обновления микропрограммного управления позволяет обновить "прошивку" процессора, и тем самым улучшить его функциональность уже после производства аппаратуры;

Недостатки:

- Необходимо хранить микрокод непосредственно в процессоре (необходим мгновенный доступ к ней), где хранение данных весьма дорогой процесс, по сравнению с внешней памятью. Чем сложнее система команд, тем больше необходимо памяти.
- Развитый ассемблер требует большого объёма знаний о процессоре от разработчика
- Наличие большого количества программного обеспечения (пусть и в виде микрокода) сопряжено со всеми сложностями программирования. В данном случае это означает что программное обеспечение нуждается в дорогостоящей отладке и оптимизации, что значительно повышает стоимость разработки.
- Большое разнообразие команд делает их относительно уникальными с точки зрения формата, размера команды, длительности исполнения и количества доступов к памяти. Как следствие это усложняет:
    - реализацию оптимизаций в рамках процессора (конвейерное исполнения, суперскалярность и т.п.);
    - поддержка со стороны инструментальных средств (компиляторы, дизассемблеры, отладчики и т.п.)

Применение: В рабочих станциях, серверах среднего звена и персональных компьютерах используются процессоры с CISC.



CISC
Complex Instructions Possible
1 Instruction = n μ-Instructions

15.Что такое RISC? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана.

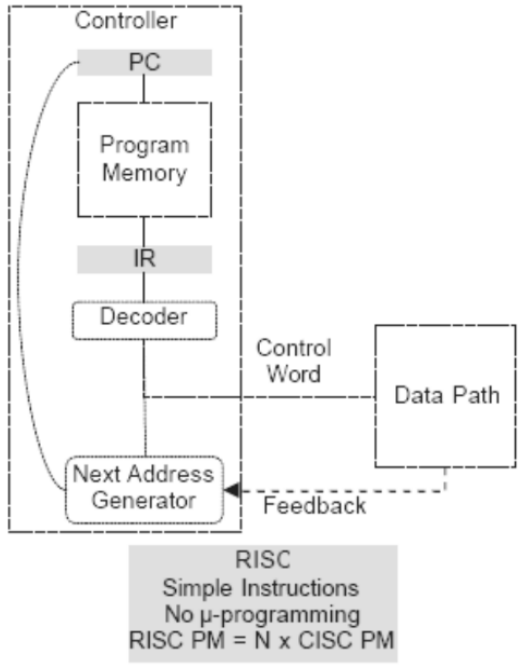


Рисунок нахуй не нужен, но вдруг спросит что-то

RISC — Reduced Instruction Set Computer
Ключевой технологией для RISC процессоров стали **языки высокого уровня**, которые позволили разработчикам ПО уйти от написания исходного кода на уровне ассемблера. Появилось удобство написания кода и встал вопрос о генерации этого самого кода и переводе его в язык машинных команд

Основная идея RISC процессора заключается в том, чтобы сделать минимальный набор унифицированных команд ориентированных на быстрое исполнение, а "сложное поведение" реализовывать их последовательностями. Что это дает:

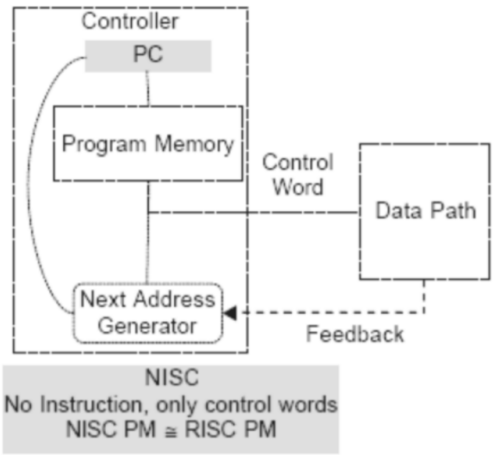
- освободить в процессоре площадь от элементов сложных декодеров команд, что позволило расширить регистровые файлы;
- освободить большое количество памяти микрокоманд для кеша команд, что отчасти компенсировало необходимость более частой выборки команд;
- добиться малого количества быстро исполняемых команд;

16. Что такое NISC? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры ф hion Неймана.

NISC - no instruction set computing

Почему бы не сделать так, чтобы наш компилятор вместо того чтобы генерировать код какой-то системы команд - мог генерировать напрямую управляющие сигналы для нашего чипа (то есть грубо говоря тот самый микрокод который мы писали)

идея = в принципе, если у нас есть хороший компилятор, то мы можем генерировать управляющий сигнал для нашего чипа тем самым можем получить 100% от той аппаратуры , с которой работаем



Память может быть реализована как сдвиговый регистр:

- полный контроль со стороны компилятора за тем, как работает процессор (+)
- трудно написать такой компилятор (-)
- упрощается строение процессора (+)

Процессор разделяется на две части:

- часть обработки данных
- часть управления

Широко применяется во встраиваемых системах.

Отличие от Фон-Неймана: Нет системы команд, поэтому машинный код имеет прямой доступ к сигналам и элементам процессора.

17.Что такое стековый процессор? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана.

Стек — это одна из простых структур данных, которая занимается сбором элементов с особыми функциями.

- push, который производит элемент в коллекцию
- pop, который истек последний добавленный элемент из коллекции.

Эта простая структура позволяет выполнять задачи процедур, включая поиск, определение контекстов и многие другие важные программные вещи.

Лучшим предположением является организация вычислительного процесса через развитие языка программирования Forth.

Сегодня язык программирования Forth в основном используется в качестве встроенного языка для пользовательской настройки (например, проект Open Firmware), поскольку он реализован в реализации. Кроме того, некоторые проекты все еще используют его в качестве стандартного языка программирования.

Плюсы и минусы:

+ К преимуществам стековых машин относят компактность кода, простоту компиляции и интерпретации кода, а также компактность состояния процессора.
- В случае стека фиксированного размера (который у Пенского в теории и на картинке ниже), мы имеем проблемы с компилятором и/или операционной системой, которые обязаны думать о том, что произойдет, если стек переполнится или опустеет и как отработать соответствующее прерывание. Если обработка прерывания отсутствует, мы имеем дело с микроконтроллером, обреченным на то, чтобы быть программируемым вручную.
- Аппаратно реализованный стек подразумевает строгую последовательность происходящих событий, а, следовательно, и невозможность использовать неявный параллелизм программ.
- Программирование для регистровых машин более «технологично». Имея всего лишь несколько регистров общего назначения, «вручную» легко можно создать код, который будет обращаться к памяти лишь тогда, когда этого действительно нельзя избежать.

Такой тип процессоров оказал широкое влияние на микроэлектронику в СССР. Стековые процессоры имеют гораздо меньшую требовательность к памяти и низкое энергопотребление, по сравнению, например с RISC процессором. Однако они не

такие мощные. Их было бы логично применять в маленьких встроенных системах, где не нужна очень высокая производительность
В архитектуре фон неймана предполагается наличие регистров. Здесь же все взаимодействие с алу осуществляется через стек.

18. Что такое VLIW? Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки. Отличия от архитектуры фон Неймана.

Что такое VLIW?

VLIW — Very Long Instruction Word. То есть архитектура процессоров, характеризующаяся возможностью объединения нескольких простых команд в так называемую связку. Входящие в нее команды должны быть независимы друг от друга и выполняться параллельно. Таким образом, из нескольких независимых машинных команд транслятор формирует одно «очень длинное командное слово». Откуда и пошло название архитектуры.

Роль в развитии компьютерных систем.

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения команд возлагается на «разумный» компилятор. Такой компилятор вначале анализирует исходную программу. Цель анализа: обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы между командами не возникали конфликты. На следующем этапе компилятор пытается объединить такие команды в пакеты (связки), каждый из которых рассматривается как одна сверхдлинная команда.

Де факто VLIW является логическим продолжением RISC архитектуры. Можно сказать что данная архитектура стала логической связью между прошлым и будущим архитектур. На сегодня VLIW процессора широко применяются для решения специализированных вычислительных задач,

Применение на практике.

1. AMD GPU также использовали модифицированную VLIW архитектуру до 2012 года, после стали использоваться RISC
2. На данный момент VLIW архитектура используется в процессорах Qualcomm Snapdragon. Можно сказать это компания которая производит мощнейшие процессоры для мобильных устройств.
3. Также VLIW архитектуру берут Российские процессоры Эльбрус

Достоинства и недостатки.

Плюсы

Выполняемый VLIW параллелизм операций позволяет:

- повысить уровень параллелизма программ по сравнению с суперскалярными(параллелизм на уровне инструкций за счёт включения в состав его вычислительного ядра нескольких одинаковых функциональных узлов.)процессорами, так как компилятор не ограничен относительно небольшим количеством инструкций, а имеет доступ ко всему программному обеспечению;
- упростить процессор, так как он более не требует в своем составе сложного диспетчера;
- снизить энергопотребление.

Минусы

- Низкая плотность исходного кода. Вместо того, чтобы компактно представлять сложные параллельные процессы система команд VLIW стала часто представлять простой последовательный смешанный код, оставляя часть инструкции пустыми, но при этом затрачивая энергию на их передачу, обработку и хранение.
- Ширина команды VLIW процессора (архитектура системы команд) накладывает ограничение на микроархитектуру процессора, что затрудняет независимую разработку процессора и компилятора. Также это затрудняет бинарную совместимость.
- Эффективный код требует большого количества оптимизаций и спекулятивных вычислений, что не всегда удается реализовать в рамках инструментария, а также требует аппаратной поддержки со стороны процессора.
- Высокопроизводительная работа VLIW процессора требует активного использования спекулятивных вычислений.

Отличия от архитектуры фон Неймана.

Архитектура фон Неймана является последовательной архитектурой. То есть в чистой архитектуре фон Неймана процессор одномоментно может либо читать инструкцию, либо читать/записывать единицу данных из/в памяти. То и другое не может происходить одновременно, поскольку инструкции и данные используют одну и ту же системную шину.

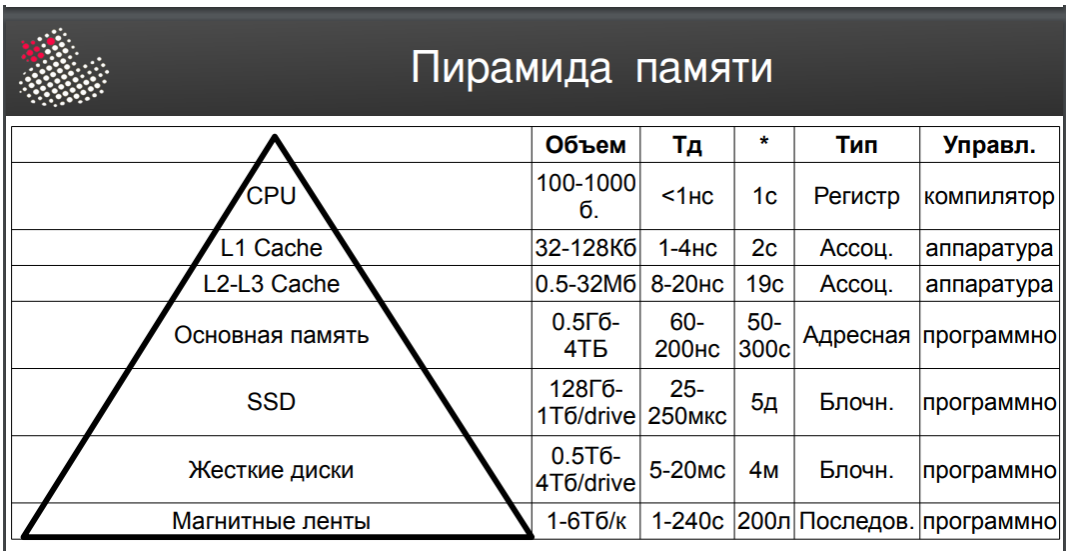
В то время как VLIW имеет способность выполнения нескольких машинных инструкций за один такт процессора путем увеличения числа исполнительных устройств.

19.Иерархия памяти. Виды памяти. Особенности использования на практике. Устройство памяти с произвольным доступом.

Память в компьютерных системах решает две основные задач

- хранение исходных данных, необходимых для работы системы (программное обеспечение, настройки и начальные значения);
- хранение и обновление данных, которые необходимы для функционированию компьютерной системы (видео поток, данные для расчетов, текстовый документ).

С ростом скорости памяти растёт её стоимость



	Объем	Тд	*	Тип	Управл.
CPU	100-1000 б.	<1нс	1с	Регистр	компилятор
L1 Cache	32-128Кб	1-4нс	2с	Ассоц.	аппаратура
L2-L3 Cache	0.5-32Мб	8-20нс	19с	Ассоц.	аппаратура
Основная память	0.5Гб-4ТБ	60-200нс	50-300с	Адресная	программно
SSD	128Гб-1Тб/drive	25-250мкс	5д	Блочн.	программно
Жесткие диски	0.5Тб-4Тб/drive	5-20мс	4м	Блочн.	программно
Магнитные ленты	1-6Тб/к	1-240с	200л	Последов.	программно

Виды памяти представлены в пирамиде памяти.

Но, память также подразделяется по типу доступа:

1) с произвольным доступом, (при котором память предоставляет данные по любому адресу с одинаковой задержкой относительно прошлого запроса)
2) с последовательным доступом, (данные читаются последовательно)
3) гибридный варианты: доступ к магнитной ленте реализуется произвольно, а доступ к данным в рамках ленты - последовательно.

Особенности использования на практике:

Последовательный доступ удобно использовать с дисками, т к головка крутится по кругу. (при чем на диске можно использовать произвольный доступ, но очень долго)

Стоит использовать тот или иной вид памяти для различных задач.

Устройство памяти с произвольным доступом:
Произвольный доступ к данным памяти реализуется за счет особой организации хранилища данных. Обычно это массив ячеек памяти. Такое хранение повышает плотность хранения данных и упрощает её масштабирование.

(В вопросе этого нет, но в доке написано)

1. Read Only Memory (ROM)

Память только для чтения. Может реализовываться как путем физического размещения / неразмещения транзисторов (тогда реализуется в рамках производства), так и путем однократного программирования, к примеру, за счет

"пережигания перемычек" (см. PROM).

2. Static Random Access Memory (SRAM)

Статическая память с произвольным доступом. Особенность такой памяти - хранение данных при помощи состояния группы транзисторов, что с одной стороны обеспечивает:

- быстрый доступ к данным на чтение и запись;
- долговременное хранение данных (значение будет храниться произвольное время, если не отключать питание с ячейки).

В тоже время для реализации требуется довольно большое количество транзисторов, а значит плотность ячеек памяти будет не очень высока.

3. Dynamic Random Access Memory (DRAM)

DRAM - тип компьютерной **памяти**, отличающийся использованием полупроводниковых материалов, энергозависимостью и возможностью доступа к данным, хранящимся в произвольных ячейках **памяти**.

В то же время, данный тип памяти требует наличия всего одного транзистора и конденсатора, что радикально увеличивает плотность ячеек памяти по сравнению со SRAM.

20.Механизм кеширования в компьютерных системах. Основные свойства кэш памяти. Иерархия кэш памяти. Условия эффективной работы кэш памяти.

Механизм кеширования в компьютерных системах.

Посмотрим конкретно на примере процессора:

—Мы хотим **прочитать** данные и запрашиваем эти данные из кэша:

- **Если искомый тег не найден фиксируется *кэш промах*** *(cache miss)*, данные запрашиваются из основной памяти в кэш, после чего передаются в процессор. Это сложно и плохо потому что:
    - если вся память кеша уже занята (типовая ситуация), значит необходимо принять решение о том, какие данные будут вытеснены / замещены (evict);
    - доступ к основной памяти может быть заблокирован другим процессом или внутренними процессами памяти (если это DRAM).
- **Если искомый тег найден, то фиксируется *кэш попадание*** *(cache hit)* и данные из кеша передаются прямо в процессор.

Эффективность работы кешей характеризуются процентом обращений к кэшу, коэффициентом попаданий в кэш.

Для выбора вытесняемой строки кэша используется специальный алгоритм. Основной **проблемой алгоритма является предсказание**, какая строка вероятнее всего не потребуется для последующих операций. Качественные предсказания сложны, и **аппаратные кэши используют простые правила**, такие, как **LRU** (least recently used). Кроме того есть **пометка** некоторых **областей памяти** как **не кэшируемых**. **Промахи для такой памяти не создают копии данных в кэше**.

—При доступе к данным **на запись** процесс сложнее, так как подразумевает перенос внесенных изменений из кеша в основную память. Выделяются следующие варианты:

- *Немедленная запись* (сквозная, write-through). Каждое изменение вызывает **синхронное обновление данных в основной памяти**, что делает доступ на запись медленным, причем зачастую медленнее чем если бы кеша не существовало (из-за лишнего шага). В тоже время, такой подход позволяет поддерживать основную память в наиболее актуальном состоянии.
- *Отложенная запись* (обратная запись, write-back). **Обновление происходит в случае вытеснения записи данных**, периодически или по запросу, что позволяет группировать записи в память.
    - Промах в кэше с отложенной записью может потребовать два обращения к основной памяти: первое для записи вытесняемых данных из кэша, второе для чтения необходимого элемента данных.
    - Обратная запись может приводить к рассоглоасованному состоянию кэша и основной памяти. Для самого процессора эта рассогласованность не заметна, но перед обращением к памяти другого ведущего системной

шины (контроллера DMA, bus-master-устройства шины PCI) кэш должен быть записан в память принудительно.

Не обязательно - Гибридные варианты. **Кэш может быть со сквозной записью**, но для уменьшения количества транзакций на шине записи могут временно помещаться в очередь и объединяться друг с другом.

21. Ассоциативность кэш памяти. Детальное описание принципов работы кэш памяти с разным уровнем ассоциативности.

Ассоциативность кэша - характеризует то, как и какие области памяти могут быть отображены на кэш линии. Отображает её логическую сегментацию, которая вызвана тем, что последовательный перебор всех строк кэша в поисках необходимых данных потребовал бы десятков тактов и свел бы на нет весь выигрыш от использования встроенной в ЦП памяти.

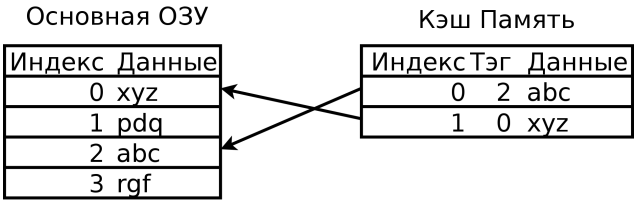
Полностью ассоциативный кэш (fully associative cache) - любая строка памяти ОЗУ может быть отображена в любую строку кэша. Такой способ позволяет реализовать наиболее эффективный кэш с точки зрения работы, но в то же время требует либо большого количества логики, либо длительного времени работы.

Кэш с прямым отображением (direct mapping cache) - данная строка ОЗУ может быть отображена в единственную строку кэша, но в каждую строку кэша может быть отображено много возможных строк ОЗУ.

Множественно-ассоциативный кэш - кэш-память делится на несколько "банков", каждый из которых функционирует как *кэш с прямым отображением*[КЕД1] , таким образом строка памяти может быть отображена не в единственную возможную запись кэша (как было бы в случае прямого отображения), а столько раз, сколько доступно банков (в один из них); выбор банка осуществляется на основе алгоритма вытеснения (LRU – вспоминаем ОСИ).

Как это работает

Каждая строка — группа ячеек памяти содержит данные, организованные в *кэш-линии*. Размер каждой кэш-линии



может различаться в разных процессорах, но для большинства x86-процессоров он составляет 64 байта. Размер кэш-линии обычно больше размера данных, к которому возможен доступ из одной машинной команды (типичные размеры от 1 до 16 байт). Каждая группа данных в памяти размером в 1 кэш-линию имеет порядковый номер. Для основной памяти этот номер является адресом памяти с выброшенными младшими битами. В кэше каждой кэш-линии дополнительно ставится в соответствие **тег**, который является адресом продублированных в этой кэш-линии данных в основной памяти.

При доступе процессора в память сначала производится проверка, хранит ли кэш запрашиваемые из памяти данные. Для этого производится сравнение адреса запроса со значениями всех тегов кэша, в которых эти данные могут храниться. Случай совпадения с тегом какой-либо кэш-линии называется *попаданием в кэш*, обратный же случай называется *кэш-промахом*. Попадание в кэш позволяет процессору немедленно произвести чтение или запись данных в кэш-линии с совпавшим тегом. Отношение количества попаданий в кэш к общему количеству запросов к памяти называют рейтингом попаданий, оно является мерой эффективности кэша для выбранного алгоритма или программы.

22. Поддержка операций ввода-вывода в фон Неймановских процессорах: система команд, механизм прерываний, механизм прямого доступа к памяти.

В данном контексте **ввод-вывод** - это передача информации между процессором (CPU) и внешними устройствами (диски, мышки, экраны, GPIO и т.п.). Можно выделить три подхода:

Программно-управляемый ввод-вывод – все операции реализуются процессором, включая постоянное наблюдение за готовностью внешнего устройства к передаче данных. Требует огромных временных затрат на взаимодействие с внешним устройством.

Процессор в цикле (polling) опрашивает состояние внешнего устройства и по наступлению необходимости начинает с ним взаимодействовать.

Ввод-вывод по прерыванию. Снимает с процессора задачу постоянного наблюдения за внешним устройством и позволяет это реализовать по внешнему событию, к примеру готовность устройства к передаче данных или наступление заданного события (срабатывание таймера, нажатие на клавишу и т.п.). Непосредственно ввод-вывод реализуется по-прежнему процессором.

Получив прерывание, процессор незамедлительно прерывает текущий поток управления и передает управление по вектору прерывания, сохраняя состояние прерванного потока, а после завершения, возвращается к нему, восстанавливая состояние исходного.

Работа с вводом-выводом при программном управлении реализуется следующим образом: процессор в цикле (опрос) опрашивает состояние внешнего устройства и по мере необходимости начинает взаимодействовать с ним. Например, наше устройство - Slave шины SPI, реализованные через порты общего назначения (GPIO):

- регистрируем сигнал CS (Chip Select), который говорит о том, что вот-вот начнется передача данных, а значит, что опрос должен начать производиться чаще;
- регистрируем изменение сигнала SCLK (сигнал синхронизации), при чем частота опроса должна быть на 2 раза выше частоты передачи данных (в теории по теореме Котельников) или больше (на практике), но и не в порядок (регистрация электрического дребезга);
- и далее, в зависимости от настроек SPI, фиксируем биты на линии данных (Master Out Slave In, MOSI или Controller Out Peripheral In, COPI) в память побито на положительный или отрицательный фронт SCLK, а также выставляем наши данные на линию (Master In Slave Out, MISO или Controller In Peripheral Out, CI
- завершаем работу с интерфейсом по установлению сигнала CS.

Обработка прерываний с точки зрения процессора довольно сложный процесс, который включает в себя:

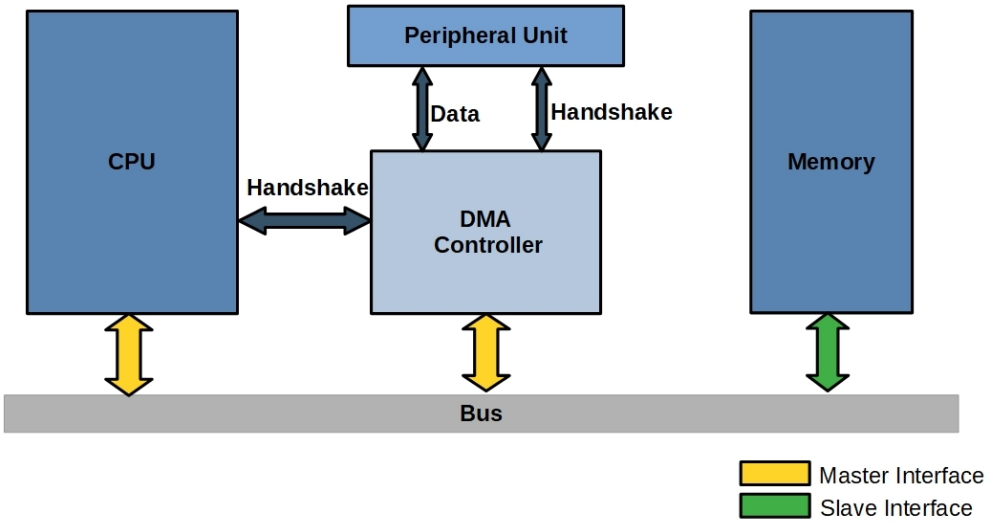
- Принятие решения о необходимости обработки прерывания. Прерывания делятся на:
    - Маскируемые прерывания, для которых разработчику доступны регистры, конфигурация которых позволяет их игнорировать. Например: отключение прерываний в рамках выполнения критической секции кода.
    - Немаскируемые прерывания, которые нельзя проигнорировать. Например: ошибка доступа к основной памяти или обработки сторожевого таймера (WatchDog таймер).
- Приоритизация прерываний. В случае, если во время обработки прерывания приходит друг другу, они делятся на:
    - относительные прерывания (может быть обработано немного позже);
    - абсолютно (необходимо прервать текущее прерывание и немедленно перейти к обработке нового).
- Прерывания могут срабатывать по различным видов событий, а именно:
    - по фронту (положительный -- изменение сигнала с 0 на 1 и отрицательное наоборот);
    - по уровню сигнала, в таком случае после обработки прерывания необходимо «сбросить» прерывание, чтобы избежания его циклической

обработки. Позволяют объединять множество прерываний через логические или (что порождает неидентифицируемые (поддельные прерывания) прерывания, что, конечно, плохо, но может быть целесообразно);
    - по сообщению (сообщение сигнализируется), реализуется очередь сообщений о наступлении прерываний);
    - по дверному звонку (дверной звонок), сигнализирует о нападению прерывания, в то время как информация о нем сохраняется в условном месте.

Данные задачи обычно решаются на аппаратном уровне (контроллер прерываний).

Channel I/O и прямой доступ к памяти (Direct Memory Access -- DMA). В случае, если необходимо передать большой объём данных, использование для этого процессора является нецелесообразным по причине его откровенной избыточности для этих задач.

В таком случае могут применяться процессоры ввода-вывода и контроллеры прямого доступа к памяти, которые позволяют процессору задекларировать необходимость передачи данных (откуда, куда, сколько), после чего её сможет реализовать специализированное устройство, которое собственно и обеспечит передачу данных и уведомит процессор о результатах через систему прерываний.



Выделяют два основных принципа работы:

- Сторонняя сторона, где работа с DMA управляется полностью процессором, и любая передача данных должна инициализироваться им.
- Мастеринг шины, где работа с DMA управляется, в том числе и со стороны ввода-вывода устройств, что позволяет инициализировать передачу данных без участия процессора.

Режимы функционирования DMA по отношению к памяти, поскольку в подавляющем большинстве случаев одновременная работа контроллера прямого доступа с памятью и процессором не возможна:

- Пакетный режим (Burst Mode). Передача данных осуществляется единой операцией, которая не может быть прервана процессором. В случае, если в процессе пакетной передачи процессор обращается к памяти, то он задерживается.
- Циклический режим (режим кражи циклов). Работа с памятью для процессора и устройств ввода-вывода имеет циклический ны, где для каждого вида операций выделяется свой временной слот. Это позволяет избежать непредсказуемой задержки в работе процессора, но может снизить общую производительность системы.
- Прозрачный режим (Transparent Mode). Передача данных осуществляется в те моменты времени, когда процессор не взаимодействует с памятью. В этом случае приоритет отдается работе процессора, что в большинстве случаев оправдано, поскольку его скорость работы обычно превышает скорость ввода-вывода устройств, но вносит задержки в передачу данных. В то же время кеширование системы обеспечивают наличие времени для передачи данных.

Выбор режима работы в значительной степени определяется той прикладной задачей, которая решается процессором, и тем, в каком процессе разработчик в большей степени устраивает задержки.

23. Операции ввода-вывода с точки зрения параллелизма уровня задач. Механизм прерываний. Как реализуется параллелизм?

Операция IO происходит, через контроллер прерываний.
Прерывание — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором.
Контроллер прерываний (англ. Programmable Interrupt **Controller**, PIC) — микросхема или встроенный блок процессора, отвечающий за возможность последовательной обработки запросов на прерывание от разных устройств. При регистрации прерывания, PIC оповещает процессор об этом, соответственно процессор начинает обработку прерывания, после обработки возвращается к дальнейшему исполнению команд.
асинхронные, или внешние (аппаратные) — события, которые исходят от внешних аппаратных устройств (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ) — устройства сообщают, что они требуют внимания со стороны ОС;

синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;

Механизм прерываний.
В основе архитектуры фон Неймана лежат принципы последовательного выполнения команд и возможности условного перехода. Именно они позволили сделать программирование данного типа процессоров относительно простым и гибким процессом.
Обратной стороной данного эффекта стала невозможность изменить поток управления извне его, процессор будет "переваривать" последовательность команд без перерыва столько, сколько посчитает необходимым.

К сожалению, данное поведение не всегда является удобным, так как:

- не позволяет эффективно работать с внешними событиями и устройствами ввода-вывода
- не позволяет обрабатывать нештатные ситуации иначе, чем при помощи кодов ошибок или флагов
- не позволяет независимо выполнять несколько потоков команд

Решает эту проблему механизм прерываний, который позволяет сигнализировать процессору о том, что текущий поток управления должен:

- быть незамедлительно прерван,
- его состояние сохранено,
- а управление передано по указанному вектору прерывания (в зависимости от источника прерывания может быть необходимым вызвать разные его реализации).
- По завершению выполнения обработчика прерывания управление должно быть возвращено исходному потоку управления с восстановлением его состояния.

24. Параллелизм уровня битов в рамках комбинационных схем и Фон-Неймановских процессоров

Параллелизм уровня битов (Bit-level Parallelism) достигается за счет увеличения “ширины” комбинационных схем, то есть за счет увеличения количества входных сигналов схемы.

В контексте фон Неймановского процессора, ширина машинного слова – фрагмент данных фиксированного размера, обрабатываемый как единое целое с помощью набора команд или аппаратного обеспечения процессора

Пример оптимизации за счёт параллелизма уровня битов для сложения бинарных чисел:

- машинное слово: 8 бит, данные: 16 бит, в таком случае сложение производится несколько шагов:
    1. сложение младших битов, фиксация бита переполнения и сохранение результата в память;
    2. сложение старших битов, добавление бита переполнения (если бы установлен) и сохранение результата в память;
- машинное слово: 16 бит, данные: 16 бит, сложение осуществляется в рамках большой комбинационной схемы за один такт (примечание: комбинационная схема в 16 бит будет работать на меньшей частоте чем в 8 бит, но в данном случае это разница несопоставима с необходимостью нескольких тактов).

Параллелизм уровня битов имеет серьезные ограничения так как “простые типы данных” не имеет практического смысла наращивать (к примеру, int64 более чем достаточно для подавляющего числа задач, а значит переход на int128 приведёт к тому, что старшие биты будут в основном “греть воздух”, а в тех редких случаях где они действительно нужны не факт, что их будет достаточно).

Использовать же широкое машинное слово для составных данных также затруднено в случае процессоров общего назначения из-за их многообразия (составные данные зависят от прикладной задачи) и необходимости их поддержки на уровне системы команд (см. CISC vs RISC), без которой это теряет смысл.

25. Параллелизм уровня инструкций. Принцип конвейерного и суперскалярного исполнения инструкций.

Параллелизм уровня инструкций

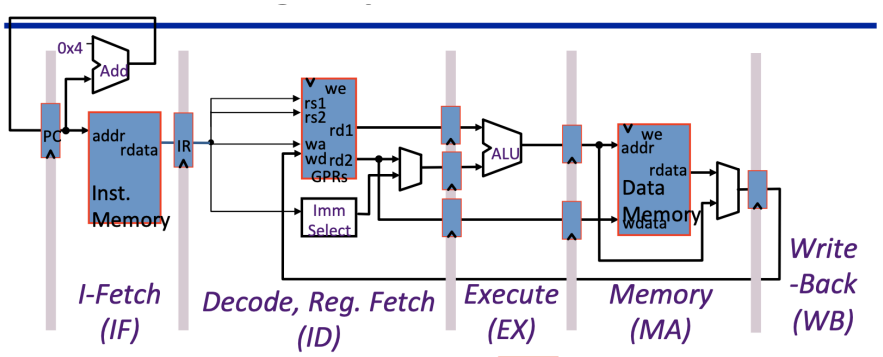
Суть в том, что мы представляем весь алгоритм в виде последовательно идущих команд. Из этого как бы следует что эти команды не могут быть выполнены параллельно. Но этой параллельности мы можем добиться по другому:

- выполнение основной задачи команды (к примеру, деления) может требовать большого количества времени, а значит в процессоре будут присутствовать простаивающие элементы, которые можно загрузить другими задачами командами;

- Из этого следует решение - Разделить команду на составляющие и пустить так сказать по конвейеру - в каждый момент времени определенная часть инструкции будет выполняться только у одной команды.
- Кроме того, нужно помнить, что изменение порядка выполнения команд, зачастую, не изменяет их результата. Это и позволяет так делать

Конвейеризированное исполнение команд

Вспоминаем конвейер на производстве. Каждый пост производства выполняет свою задачу параллельно другим постам. Никто ни с кем не пересекается, все крутА. С таким подходом к работе с процессом, можно легко повысить производительность, увеличивая количество этих постов. С точки зрения цифровой схемотехники, данный способ эксплуатирует возможность параллельной и независимой работы комбинационных схем разделенных регистрами, сдвиг конвейера реализуется по глобальному тактовому сигналу.



time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
instruction1	IF_1	ID_1	EX_1	MA_1	WB_1				
instruction2		IF_2	ID_2	EX_2	MA_2	WB_2			
instruction3			IF_3	ID_3	EX_3	MA_3	WB_3		
instruction4				IF_4	ID_4	EX_4	MA_4	WB_4	
instruction5					IF_5	ID_5	EX_5	MA_5	WB_5

Картинка прикольная, но тут только нижняя часть важна с изображением конвейера.

Недостатки конвейера:

- снижение скорости выполнения отдельной команды (за счёт плохой балансировки стадий и дополнительных регистров)
- не все операции могут пройти стадию конвейера за один машинный цикл (загрузка ресурсов)
- необходимость разрешения конфликтов
- непредсказуемое время исполнения
- противоречие с фон Неймановской архитектурой (принцип единой памяти)

В целом еще раз

Каждая команда так или иначе состоит из нескольких основных этапов. Так как каждый из таких этапов выполняет определенную логику, работает с определенными регистрами, разные этапы конвейера можно выполнить параллельно. К примеру, когда процессор ищет в памяти аргумент для инструкции, АЛУ свободен, в этот момента на нем можно произвести операцию. Поэтому внутри процессора команды отрабатываются на так называемом конвейере. Стадии конвейера связаны между собой, а также выровнены по времени исполнения.

Суперскалярные архитектуры

Термин "суперскалярный" произошёл от понятия "скалярной величины" -- величины, которая может быть представлена числом (целочисленным или с плавающей точкой). Отсюда:

- скалярная операция -- операция над числом (если один операнд) или числами (несколько операндов);
- векторная операция -- операция, для которой в качестве операнда и результата могут выступать массивы, за счёт чего нет необходимости на каждый набор увеличивать счётчик, декодировать команду и т.п.

Суперскалярность — архитектура вычислительного ядра, использующая несколько декодеров команд, которые могут нагружать работой множество исполнительных блоков. Планирование исполнения потока команд является динамическим и осуществляется самим вычислительным ядром.

К примеру, Мы используем много АЛУ разных типов. И все они одновременно задействованы. **Еще важно сказать** что суперскалярные процессоры соответствуют простым процессорам, все происходит без ведома пользователя.

26. Конвейеризированное исполнение команд. Стадии конвейера. Виды конфликтов и их примеры.

Конвейеризировнное исполнение команд - способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени).

Для организации конвейерной работы процессора необходимо:

1) Выделить стадии выполнения команд;
2) Организовать внутренние структуры процессора таким образом, чтобы:
    - У процессора был входной (поступают команды) и выходной конец (готовые команды "покидают" конвейер процессора);
    - каждый сегмент процессора должен быть связан с последующим через регистровый интерфейс)
    - Все сегменты процессора должны управляться от одного тактового сигнала;
3) Настроить управление конвейером таким образом, чтобы загружать в него команды каждый такт;
4) Разрешить конфликты и противоречия, вызванные частично параллельным исполнением команд.

Стадии конвейера (обычного конвейера RISC процессора)

1. Instruction Fetch (Извлечение инструкций). Процессор считывает инструкцию по адресу в памяти, значение которого присутствует в счетчике программ.
2. Instruction Decode (Декодирование инструкций). Команда декодируется и осуществляет доступ к регистровому файлу для получения значений из регистров, используемых в инструкции.
3. Instruction Execute (Выполнение инструкции). На этом этапе выполняются операции. (Single-cycle latency): Сложение, вычитание, сравнение и логические операции, (Two-cycle latency): Получение данных из памяти и их загрузка в регистры с учетом формирования физического адреса в памяти выполняемого АЛУ, (Many cycle latency): Целочисленное умножение и деление и все операции с плавающей запятой.)
4. Memory Access (Доступ к памяти). На этом этапе операнды памяти считываются и записываются в виде/в память, присутствующую в инструкции.
5. Write Back (Написать ответ). На этом этапе вычисленное/полученное значение записывается обратно в регистр, присутствующий в инструкциях.

Есть 3 вида конфликтов при реализации конвейерных вычислений:

1. Структурные конфликты

(возникают из-за конфликтов ресурсов, когда аппаратура не может поддерживать все возможные комбинации одновременно выполняемых команд (идет обращение к памяти в то время, когда другая инструкция достигла обращения к памяти для взятия операндов).

Решение данной проблемы - приостановить конвейер на один такт (вставка пузырька)

2. Конфликты по данным

(возникают, когда при совмещении команд в конвейере, оказывается что есть зависимость команды от результатов предыдущей)

Решения:

вставка пузыря в конвейер - это пропуск нескольких тактов процессора;
- изменение порядка выполнения команд с учётом конвейерного исполнения на уровне компиляции или процессора
- проброс операндов между стадиями процессора - для выполнения операций

3. Конфликты по управлению

(возникает если при выполнении условного перехода, может измениться счетчик команд и если не принять нужных мер то произойдет остановка конвейера на много тактов, пока не будет вычислено условие перехода (переход определится).

Решение - предсказания будущих переходов.

Преимущества и недостатки конвейера

Преимущества: повышение производительности и уровня утилизации вычислительных ресурсов.

Недостатки: снижение скорости исполнения отдельных команд из за:

- не все операции могут пройти стадию конвейера за один машинный цикл (загрузка ресурсов);
- непредсказуемое время исполнения;
- противоречие с фон Неймановской архитектурой (принцип единой памяти);
- уязвимости связанные с доступом через "косвенные каналы".

27. Параллелизм уровня задач. Закон Амдала. Способы реализации. Механизмы синхронизации.

Параллелизм уровня задач подразумевает параллельное выполнения нескольких потоков команд, которые определены разработчиком программного обеспечения. Другими словами: он не является прозрачным для программиста, а значит не может быть рассмотрен без учета средств программирования.

Как происходит переключение между потоками команд, влияние этого на разработку ПО и синхронизацию потоков исполнения? - это центральный вопрос.

Кооперативная многозадачность (Cooperative multitasking)

Тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам.

При простом переключении активная программа получает все процессорное время, а фоновые приложения полностью замораживаются. При кооперативной многозадачности приложение может захватить фактически столько процессорного времени, сколько оно считает нужным. Все приложения делят процессорное время, периодически передавая управление следующей задаче.

Вытесняющая многозадачность (Preemptive multitasking)

Вид многозадачности, в котором операционная система сама передает управление от одной выполняемой программы другой в случае завершения операций ввода-вывода, возникновения событий в аппаратуре компьютера, истечения таймеров и квантов времени, или же поступлений тех или иных сигналов от одной программы к другой.

В этом виде многозадачности процессор может быть переключен с исполнения одной программы на исполнение другой без всякого пожелания первой программы и буквально между любыми двумя инструкциями в её коде. Распределение процессорного времени осуществляется планировщиком процессов. К тому же каждой задаче может быть назначен пользователем или самой операционной системой определенный приоритет, что обеспечивает гибкое управление распределением процессорного времени между задачами (например, можно снизить приоритет ресурсоемкой программе, снизив тем самым скорость её работы, но повысив производительность фоновых процессов). Этот вид многозадачности обеспечивает более быстрый отклик на действия пользователя.

- поддержка со стороны процессора механизмами сохранения состояния процессора в память с целью последующего его возобновления;
- поддержка механизмов прерывания, позволяющих осуществить переключение процессов независимо от их желания, а также защиты отдельных операций или их последовательностей от прерывания;
- поддержка со стороны операционной системы, а именно: планировщик, который должен определять когда один поток команд будет прерван и какой поток команд его заместит.

Закон Амдала: *ускорение программы с помощью параллельных вычислений на нескольких процессорах ограничено размером последовательной части программы*. Например, если можно распараллелить 95% программы, то теоретически максимальное ускорение составит максимум 20×, невзирая на то, сколько процессоров используется.

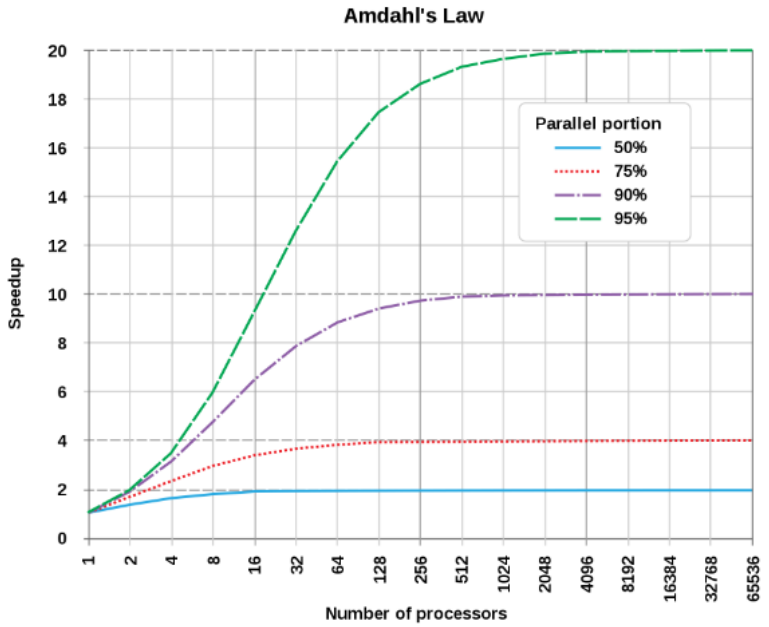


Figure 18: Amdahl's law

28. Проблема детерминизма в современном параллельном программировании. Примеры. Механизмы обеспечения детерминизма.

Детерминированный алгоритм представляет собой алгоритм который при конкретном вводе всегда будет производить один и тот же вывод

Используя последовательные программы, выполнение кода идет по счастливому пути предсказуемости и детерминизма. И наоборот, многопоточное программирование требует приверженности и усилий для достижения правильности.

Основное требование для создания программного обеспечения это детерминизм, так как часто подразумевается, что компьютерные программы вернут одинаковые результаты от одного запуска к другому. Но это свойство трудно решить параллельно. Внешние обстоятельства, такие как планировщик операционной системы или когерентность кеша, могут влиять на время выполнения и, следовательно, порядок доступа для двух или более потоков и изменять одну и ту же ячейку памяти. Этот вариант времени может повлиять на результат программы.

Пример:

Два потока находятся в состоянии гонки.

Пусть есть x = 1.

Также есть Поток 1 и Поток 2.

Первый поток: x + 5

Второй поток: x * 3

Запускаем программу..

// В зависимости от того, какой поток первым займет переменную, такой результат и получится. Если не использовать специальные средства, результат может отличаться от запуска к запуску программы.

// Дополнительно стоит помнить, что запуск одних и тех же программ на системах разных архитектур также может привести к разному результату.

Механизмы обеспечения:

1. Программными способами: запретить потокам кэшировать переменную, создавать очередь из потоков и ТД.
2. Использование детерминированных алгоритмов

В лекциях вроде как указывается, что есть и аппаратные механизмы:

1. Потоковые машины - эффективная реализация параллельных вычислений, реализуются на графах и сетях Петри.
- Особенностью **потоковых машин** является то, что последовательность вычислений задается не последовательностью команд, что характерно для неймановской архитектуры, а по мере готовности данных для выполнения команд. Данные загружаются в операционное устройство, если оно свободно, и для определенной команды имеются все необходимые данные.

4.Таксономия Флинна,

- **Системы с одной инструкцией и едиными данными (SISD) –**

Вычислительная система SISD-это однопроцессорная машина, которая способна выполнять одну команду, работая с одним потоком данных. В SISD машинные инструкции обрабатываются последовательно, и компьютеры, использующие эту модель, обычно называются последовательными компьютерами. Большинство обычных компьютеров имеют архитектуру SISD. Все инструкции и данные, подлежащие обработке, должны храниться в основной памяти.

- **Системы с одной инструкцией и несколькими данными (SIMD) –**

Система SIMD-это многопроцессорная машина, способная выполнять одну и ту же инструкцию на всех процессорах, но работающая с разными потоками данных.

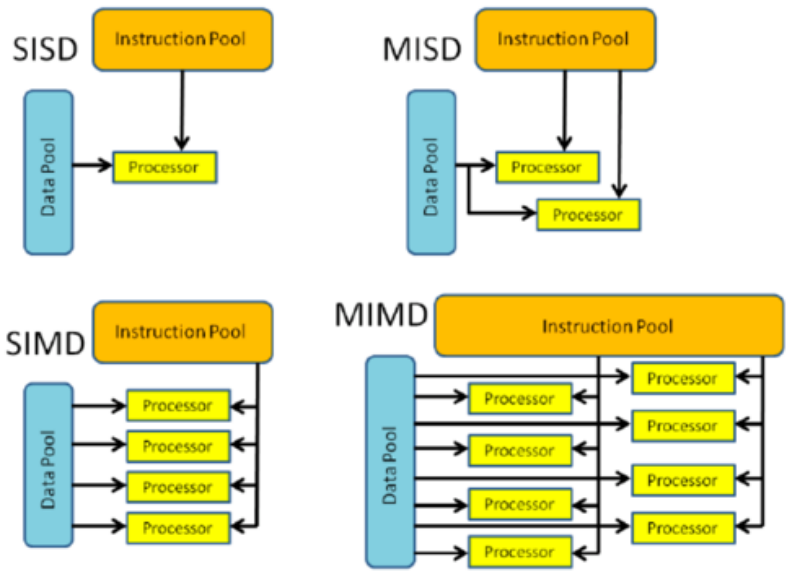
Машины, основанные на модели SIMD, хорошо подходят для научных вычислений, поскольку они включают в себя множество векторных и матричных операций. Чтобы информация могла быть передана всем элементам обработки (PEs), организованные элементы данных векторов могут быть разделены на несколько наборов(N-наборов для N систем PE), и каждый PE может обрабатывать один набор данных.

- **Системы с несколькими инструкциями и едиными данными (MISD) –**

Вычислительная система MISD-это многопроцессорная машина, способная выполнять различные инструкции на разных PE, но все они работают с одним и тем же набором данных .

- **Системы с несколькими инструкциями и несколькими данными (MIMD) –**

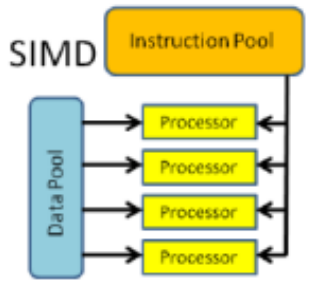
Система MIMD-это многопроцессорная машина, которая способна выполнять несколько инструкций для нескольких наборов данных. Каждый PE в модели MIMD имеет отдельные потоки команд и данных; поэтому машины, построенные с использованием этой модели, способны работать с любым видом приложений. В отличие от машин SIMD и MISD, PE в машинах MIMD работают асинхронно.



29. Классификация Флинна. SIMT архитектура. Области применения и особенности функционирования. Отличия от архитектуры фон Неймана.

Классификация Флинна - классификация вычислительных систем в зависимости от числа потоков команд и данных. Потоки бывают одиночные и множественные, таким образом согласно этой классификации существуют 4 типа.

	Поток команд	Поток данных
Одиночный	SISD	SIMD
Множественный	MISD	MIMD



Представителями SIMD архитектуры являются векторные процессоры. Они позволяют благодаря множественному потоку данных обрабатывать параллельно большие массивы данных, выполняя над ними однотипные операции(одиночный поток команд). Так работают, например, графические и физические акселераторы. Подобная архитектура также применяется в суперкомпьютерах, где требуется высокая степень параллелизации математических вычислений

Отличие от фон Неймана довольно очевидно - классическая архитектура фон Неймана по классификации Флинна - SISD. Соотв, отличие в количестве потоков данных.

30. Изоляция потоков команд. Банки памяти, сегментная память, виртуальная память. Роль в развитии компьютерных систем. Применение на практике. Достоинства и недостатки.

Изоляция потоков команд нужна, чтобы поток команд оперировал лишь своими данными и не мог прямо влиять на другие потоки команд/данных. Такая изоляция в реальных системах является многоуровневой и реализуется как средствами аппаратными(благодаря архитектуре процессора, которая учитывает права доступа), так и на уровне программного обеспечения(операционной системой)

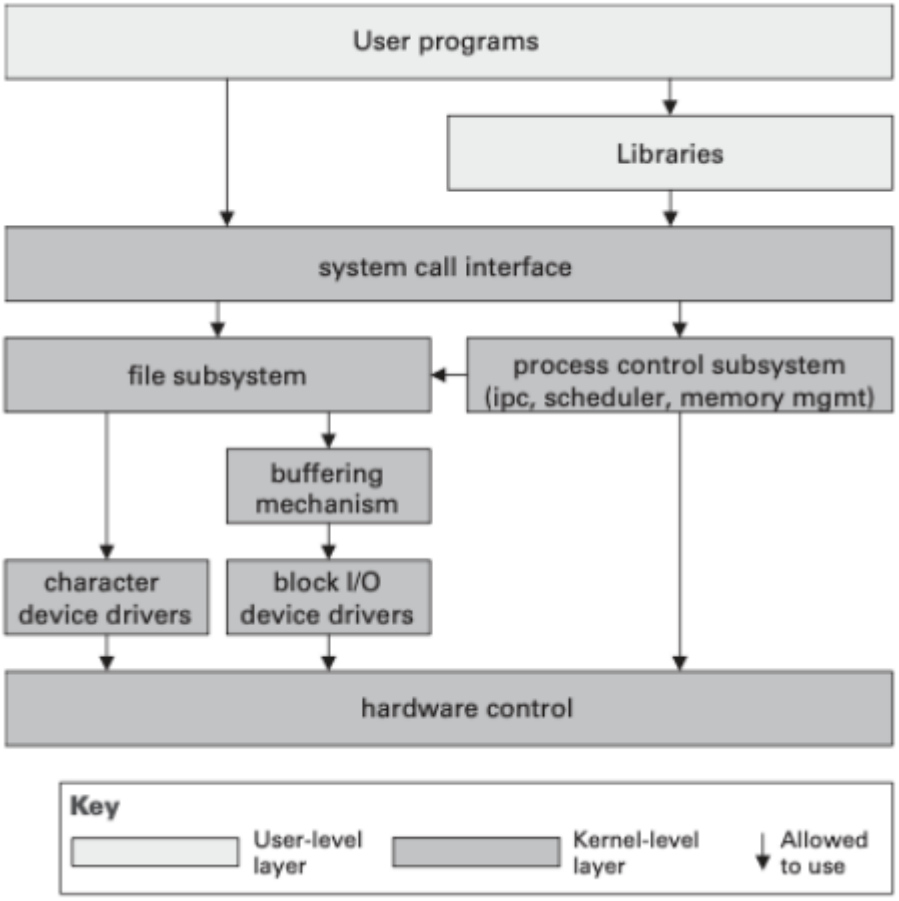
Memory Bank.
Хардварное решение для увеличения адресного пространства за размеры машинного слова. Кроме адреса, задаваемого процессором, при обращении к памяти используется также номер банки. Таким образом, размер адресного пространства можно увеличить на несколько двоичных порядков.

Сегментная память - способ организации памяти.
Память состоит из сегментов. Первая часть адреса ячейки - номер сегмента, или селектор. Вторая часть адреса - смещение относительно начала сегмента.
Сегментная память решает задачу обеспечения безопасности, так как реализованы таблицы соответствия задач процессора доступным ей селекторам памяти. Тогда сегмент памяти обладает дескриптором сегмента, который содержит связанный с ним набор полномочий (например, чтение, запись, выполнение). Процессу позволяют сделать ссылку в сегмент в том случае, если тип ссылки разрешен полномочиями, и если смещение в сегменте находится в диапазоне, определенном длиной сегмента. Иначе возникает ошибка сегментации. На аппаратном уровне реализуется в аппаратном блоке управления памятью (MMU).

Виртуальная память
Виртуальная память обеспечивает расширение адресного пространства посредством таблиц соответствия виртуального адреса реальному адресу в памяти.
Абстрагирует физическую память, дополнительно обеспечивая контроль доступа к ней(на уровне операционной системы), а также скрывая от пользователя фрагментацию данных. В современных вычислительных системах используется сегментно-страничная организация памяти. Сегменты виртуальной памяти разделяются на страницы. Такая организация позволяет разграничивать доступ процессов к сегментам памяти, а также реализовывать пейджинг - выгрузку страниц в долговременную память
Реализуется программно-аппаратно.

31. Уровневая организация компьютерной системы. Примеры. Что такое уровень компьютерной системы и что он определяет? Системные свойства компьютерных систем.

Вычислительные, программно-аппаратные системы имеют уровневую организацию. Каждый уровень выше - абстракция над предыдущим, в целях избавить пользователя реализации интерфейса более низкого уровня от необходимости разбираться в деталях этой реализации при решении прикладных задач.



Уровень компьютерной системы - степень абстракции, определяющаяся платформой, на которой производятся операции. Так, на уровне операционной системы такой платформой является процессор и прочее железо, на уровне процессора - полупроводники, транзисторы и двоичная логика, построенная на их основе. На уровне пользовательских программ платформа - фреймворк, библиотеки или интерфейсы операционной системы.
Простые компьютерные системы оперируют только на одном уровне. Сложные компьютерные системы требуют оперирования несколькими уровнями.

Свойства компьютерных систем

- Производительность
- Энергоэффективность

- Надежность
- Детерминизм и предсказуемость
- Требование реального времени
- Информационная безопасность

Достижение всех этих качеств требует принятия верных инженерных решений на всех уровнях программно-аппаратной системы.

32. Явление дезагрегации. Место явления в развитии компьютерных систем. Тенденции и перспективы. Принцип развития иерархических систем Седова.

Процесс дезагрегации в компьютерных системах – это процесс изменения рынка, в ходе которого рынок разделяется на экосистемы вокруг вычислительной платформы.

По сути: с развитием компьютерных технологий на рынок выходит все больше разработчиков, что в свою очередь усложняет процесс системной интеграции продуктов. Получается, что компании стараются отделиться от всей остальной массы, (как говорилось выше) создают свои экосистемы и концентрируют внимания на своих продуктах. Пропадает стандартизация процессов.

Принцип развития иерархических систем Седова: в сложной иерархически организованной системе рост разнообразия на верхнем уровне обеспечивается ограничением разнообразия на предыдущих уровнях, и наоборот, рост разнообразия на нижнем уровне разрушает верхний уровень организации (то есть, система как таковая гибнет).. Фактически, данный принцип описывает процесс дезагрегации.