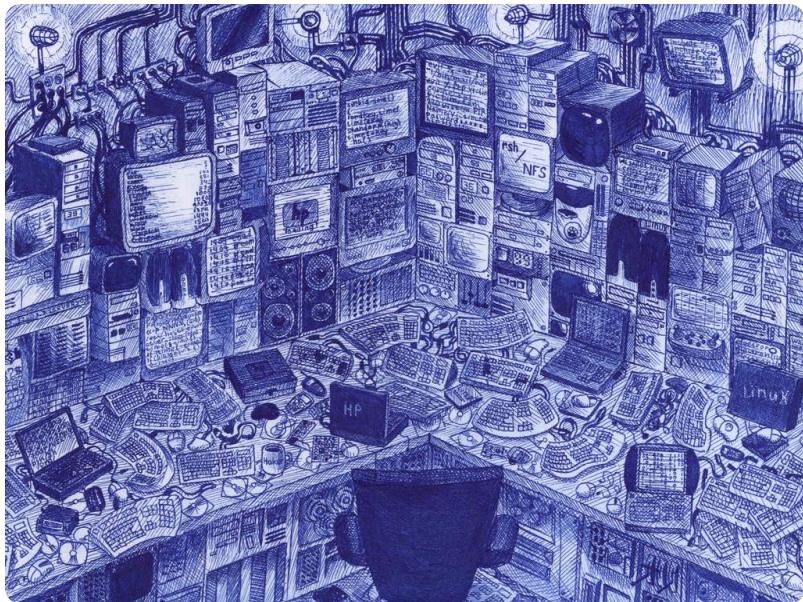
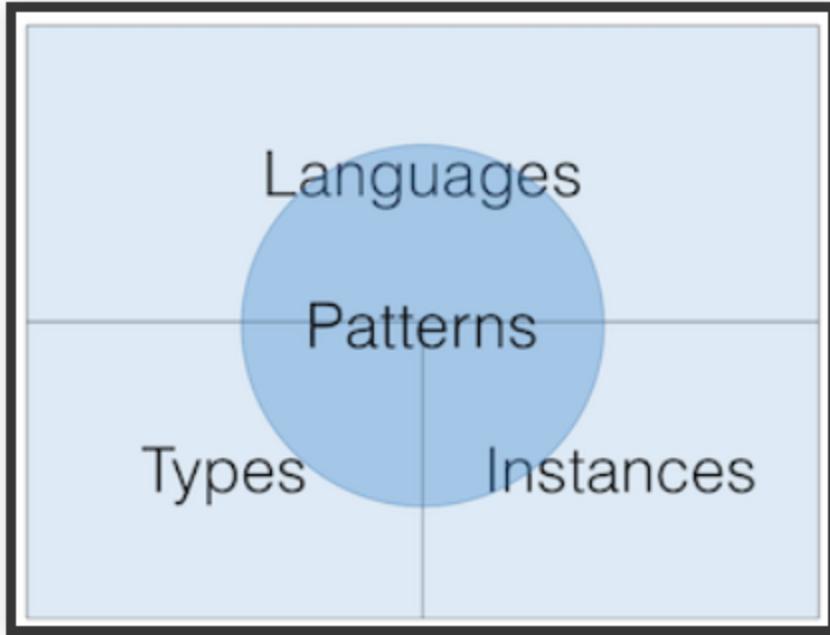


Экзамен(АК 2025)



1)Что такое "вычислительная платформа" с точки зрения пользователя? Каков её состав?

Вычислительная платформа - совокупность: языка(при помощи которого мы можем описать вычислительный процесс, его модель), набор типов и базовых операций, которые являются частью вычислительной платформы(неотъемлемая составляющая. Пример: процессор, есть машинный код, базовые типы данных(только машинное слово), и операции по типу сложения и вычитания, которые вбиты на его аппаратном уровне). Связующее звено - паттерны, те принципы и идеи как реализовывать или описывать те или иные алгоритмы



С точки зрения пользователя - среда, позволяющая выполнять вычисления и решать задачи.

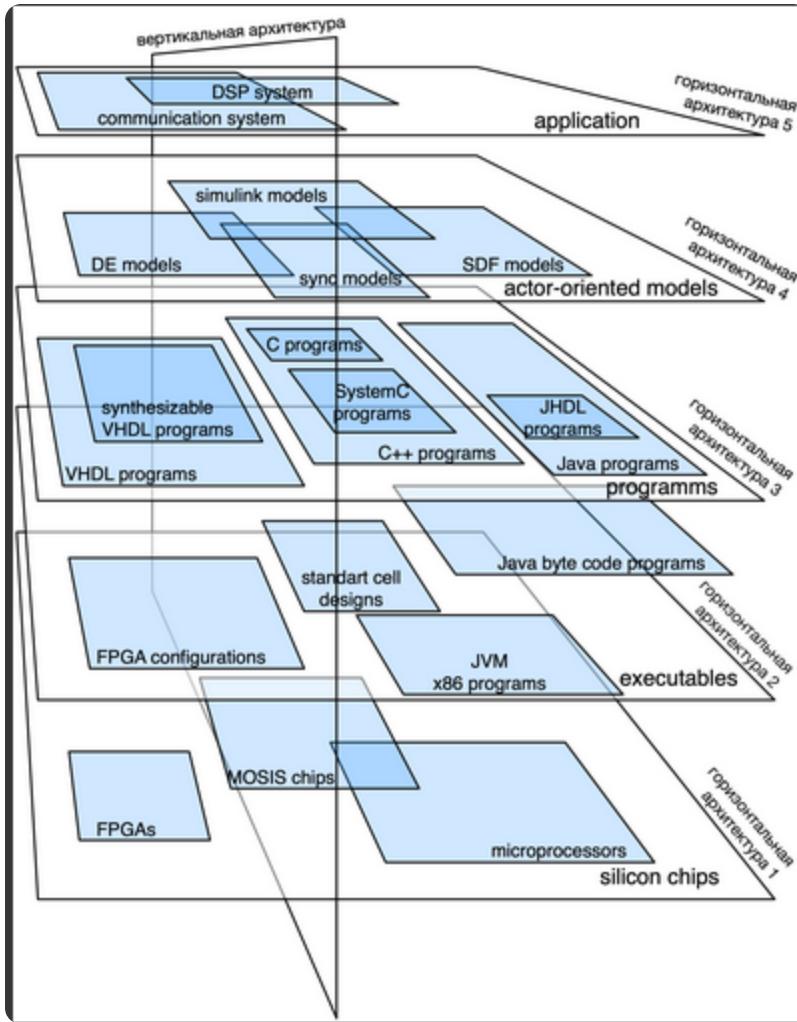
2)Какие уровни абстракций относятся к дисциплине "Архитектура компьютера"?

К дисциплине "Архитектура компьютера" относятся следующие уровни абстракции: ![



Figure 1.1 Уровни абстракции для электронно-вычислительной системы

или вот более удачный и подробный пример:



Здесь буквально:

1. кремневые чипы
2. уровень базового конфигурирования/программирования, где можно определить программы и/или конфигурацию устройства и определить ее взаимодействие
3. Спецификации что запускается
4. акторные модели(высокоуровневые вычислительные модели, которые никак не привязаны к реальным вычислениям)
5. уровень пользовательский

по сути тут от прикладного до физического уровня

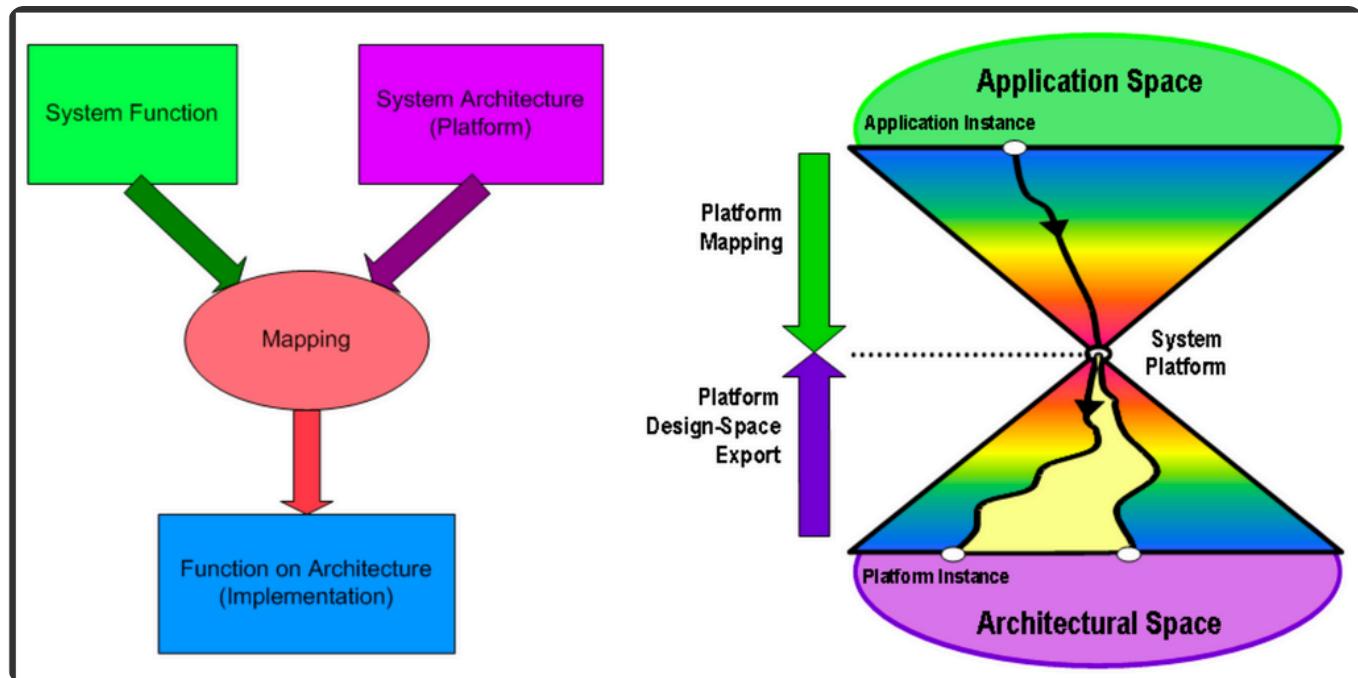
но взаимодействие уровней абсолютно разное, для примера веб приложение, где взаимодействие между уровнями абсолютно разное

3) В чём заключается отличие Platform-Based Design от проектирования для конкретной платформы?

Platform-based design - методология проектирования, основанная на использовании абстрактных платформ для разделения задач и оптимизации разработки.

Есть задачи, где надо выстроить определенные уровни под определенную задачу.

Подход заключается в следующем: с одной стороны вы имеете пространство проектных решений(нижняя часть картинки, вычислительные платформы, их элементы, паттерны и etc...), а с другой стороны пространство проектных решений с точки зрения решения задачи(способ решения задачи, необходимые процессы, бизнес решения), в итоге получаем что нижний и верхний уровень сужаются, чтобы предоставить оптимальную вычислительную платформу для разрабатываемой системы



как пример: язык python и поверх него django, fastapi, flask, которые упрощают создание веб серверов или сайты.

В итоге пространство проектных решений можно заузить на конкретную область и в итоге получить DSL(domain specific language) для создания системы

4) Почему большинство современных компьютерных систем считаются системами с преобладающей программной составляющей? Приведите примеры.

Системы с преобладающей программной составляющей:

Современный компьютер очень ну прям очень редко является исключительно железякой, почти всегда у нас есть огромное количество ПО которое и определяет функциональность и поведение системы

Сам софт позволяет создавать аппаратуру быстрее и дешевле.

Доминирование ПО обусловлено требованиями гибкости, масштабируемости и экономии,

тогда как аппаратура играет роль универсальной вычислительной основы.

Примеры:

Смартфоны(новые функции через обновления)

ПК(одно железо - разные оськи)

да и в целом что угодно: лампы, умные часы, автомобили и тд и тп...

5)Что такое информационная и управляемая система? Каковы их отличия (функции, ПО, аппаратура)?

Информационная система - главная задача: взять информацию и как то ее изменить, или сказав иначе - получить данные, преобразовать/накопить и выдать в измененном/обработанном виде.

Ее особенности:

- главный приоритет - производительность;
- спекулятивные вычисления
- параллелизм
- кластерные и облачные вычисления.

Управляющая система - взаимодействие с реальным физическим миром с целью контроля или управления посредством получения данных, преобразование/накопление и выдача в измененном/обработанном виде.

Ее особенности:

- Встроенное исполнение: интеграция в реальный мир, ограниченные ресурсы(энергия), специализация функций, специализация платформы, аппаратуры
- автономная эксплуатация
- ограниченные вычислительные ресурсы
- работа в режиме реального времени

Их отличия:

1. Основная функция

- **ИС** → сбор, хранение и обработка данных (например, базы данных, CRM).
- **УС** → управление процессами/устройствами в реальном времени (например, автопилот, промышленные контроллеры).

2. Критичные параметры

- **ИС** → важны объёмы данных и скорость доступа, но не жёсткие временные рамки.

- **УС** → обязательна **реакция в реальном времени** (миллисекунды), высокая надёжность.

3. Аппаратура

- **ИС** → серверы, облачные хранилища, ПК.
- **УС** → микроконтроллеры (STM32, Arduino), датчики, ПЛК.

4. ПО

- **ИС** → СУБД, веб-приложения, аналитика.
- **УС** → realtime-ОС (FreeRTOS, QNX), firmware, алгоритмы управления.

ИС — для данных, УС — для физического управления с жёсткими временными ограничениями.

6)Что такое "требование реального времени"? Примеры систем.

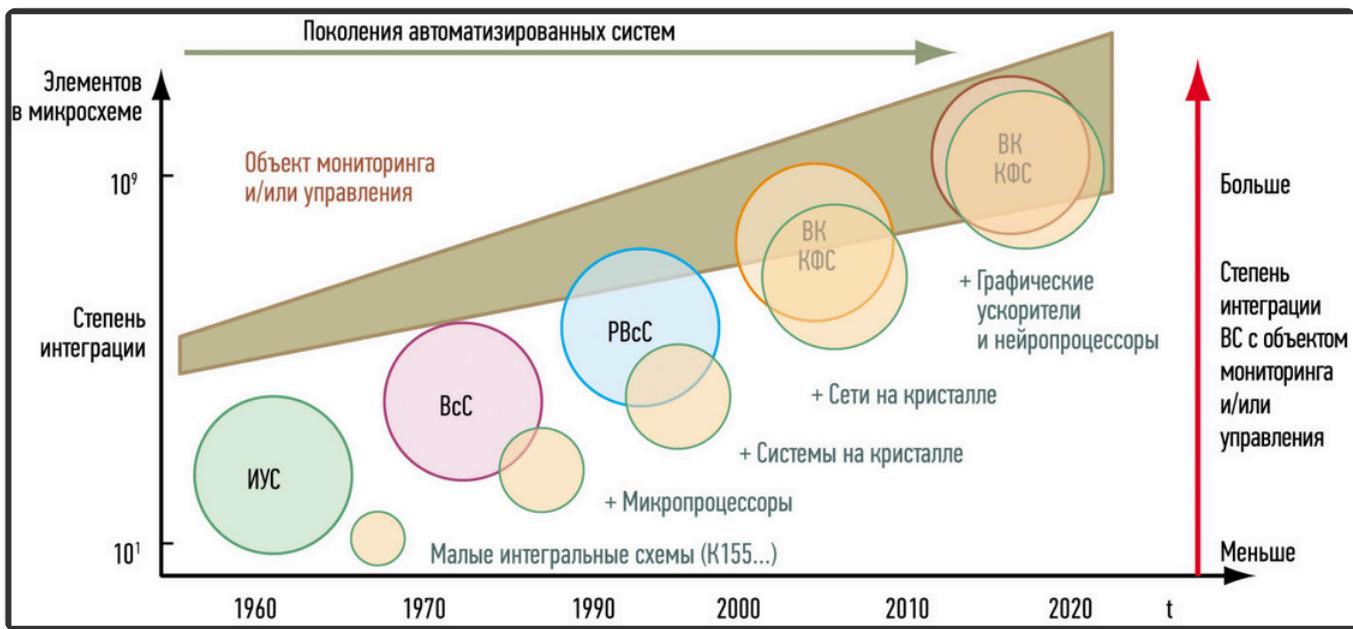
Требование реального времени - необходимость гарантированно выполнять вычисления в строго заданные временные рамки.

- != быстро
- != абсолютно точно
- != без отказов
- = предсказуемо и в срок(+-)

Примеры:

- ГЭС и водосброс: согласно методике, водосброс должен открываться примерно через 7 минут(для избежания помех(волна пришла)) и открываем медленно. Требования реального времени очевидно, но и огромных вычислительных мощностей нам тут тоже не надо

7)Каковы этапы эволюции управляющих систем: от ИУС до КФС?



ИУС - индивидуальные управляемые системы (малые интегральные схемы)

↓ от аналога к цифре

ВсС - Встроенные системы (+ микропроцессоры)

↓ сетевое взаимодействие

РВсС - Распределенные встроенные системы (+системы на кристалле)

↓ интеграция ии и облака

КФС - киберфизические системы (+сети на кристалле, + графические ускорители и нейропроцессы)

8) Каковы задачи и предмет дисциплины "Системная инженерия"? Какова роль системных инженеров?

Системная инженерия - это междисциплинарный подход и средство, позволяющее реализовать успешные системы. Он фокусируется на целостном и одновременном понимании потребностей заинтересованных сторон (стейкхолдеров); изучении возможностей; документировании требований; и синтезе, проверке, приемке и разработке решений при рассмотрении всей проблемы, от исследования концепции системы до вывода системы из эксплуатации.

Предмет дисциплины - сверхбольшие системы(например нефтедобывающая платформа), требующие множества дисциплин и участников.

Роль системного инженера:

- Координация и структура: команды разработки, процессов, передачи информации и etc..

- или иначе выступить посредником между всеми участниками проекта

**9)Что такое успешная система? Какие точки зрения необходимы для построения успешной системы?
TODO задать вопрос про то, что хочет услышать здесь Пенской, иначе я хуй его знает**

есть три точки зрения что такое система:

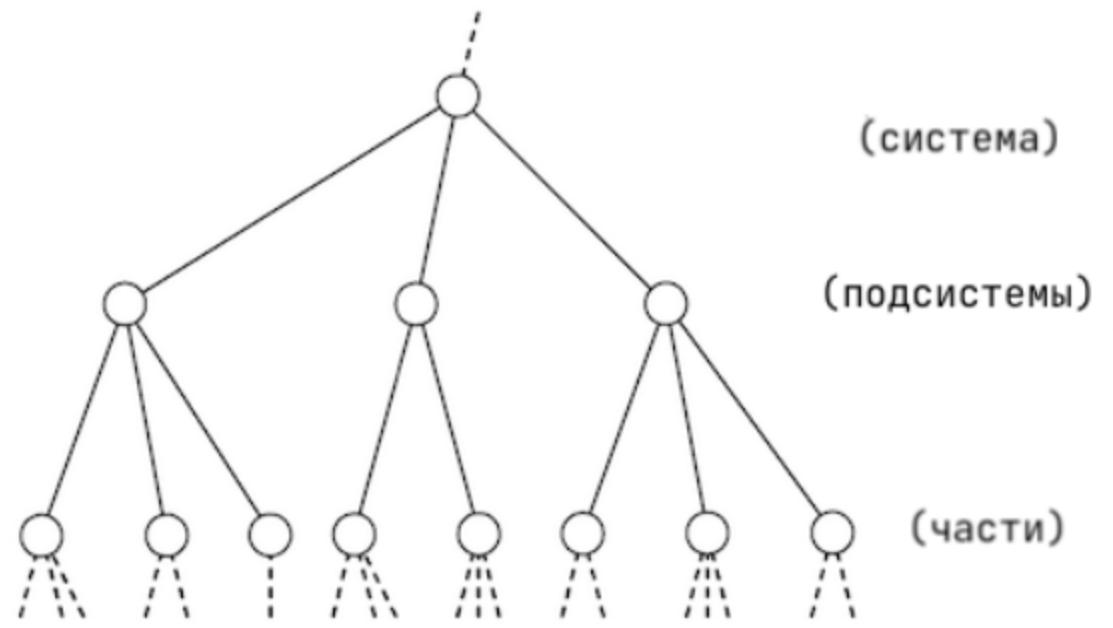
- система как совокупность частей
 - система как функциональное место
 - система как жизненный цикл
- Успешная система - система которой довольны все ее стейкхолдеры(заинтересованные стороны)

Разработка успешной системы требует:

- рассмотрения ее структуры
- рассмотрения ее функционального места
- рассмотрения ее операционного окружения
- рассмотрения ее жизненного цикла
- рассмотрение ее обеспечивающих систем

10)Как можно рассматривать систему с точки зрения структуры? Каковы причины множественности структуры?

Система как совокупность частей:



Система — это комбинация взаимодействующих элементов, организованных для достижения одной или нескольких заявленных целей.

Система может рассматриваться как продукт или как предоставляемые ею услуги.

Смотря на систему, мы должны сразу видеть основные ее части, как они взаимосвязаны между собой и взаимодействуют. Пример: разработка веб сервиса, для него необходимо знание питона, монго и джанго, чего будет достаточно, при этом нет необходимости понимания того, как работает процессор, полупроводники и etc.

Причины множественности структур

Одна и та же система может быть представлена разными структурными моделями в зависимости от:

- Уровня абстракций
- Цели анализа
- Масштабируемости и гибкости
- Технологического стека

Итого:

структурный подход позволяет:

- Декомпозировать сложные системы на управляемые части
- Оптимизировать взаимодействие компонентов
- Выбрать подходящие уровни детализации задач

11) Как можно рассматривать систему с точки зрения функционального места? Кто такие

заинтересованные стороны (Stakeholders)? Что такое операционное окружение?

Система как функциональное место:

Система должна выполнять функцию, которую требуют стейкхолдеры, которая приносит им пользу. Стейкхолдеры это те, кто может влиять на систему и имеют интерес в работе системы, как правило под этим подразумеваются инвесторы, но это может быть и не так в отдельных случаях. ОО - это окружение в котором развертывается система, в котором она функционирует

12)Как можно рассматривать систему с точки зрения жизненного цикла? Что такое обеспечивающая система?

Жизненный цикл системы - это эволюция системы во времени от концепции до выхода из эксплуатации. Типовые стадии системы:

1. Концептуальный этап(когда мы придумываем концепцию и понимаем, что "неплохо бы такое сделать")
2. Этап разработки(когда делаем документацию по которой мы можем её воспроизвести, создать) - для инженеров - чертежи, для прогеров - код.
3. Этап производства(для прогеров - компиляция, для инженеров - постройка)
4. Этап утилизации(используем)
5. Этап поддержки(система продолжает работать но дорабатывается)
6. Этап вывода из эксплуатации(остановка системы и использование остаточных ресурсов)

Обеспекивающая система - это система которая дополняет нашу систему на этапах её жизненного цикла, но не обязательно влияет на эксплуатацию. Например - вспомогательная производственная система, вспомогательная рекламная система. Они тоже имеют жизненные циклы.

13)Почему плохой менеджмент может увеличить бюджет проекта быстрее, чем другие факторы?

Ключевые проблемы разработки компьютерных систем:

- формулирование/генерация информации
 - передача/сохранение информации
 - использование информации
 - а также: неполнота, неоднозначность, нераспределенность, противоречивость, решение других проблем
- Этим всем и должны заниматься менеджеры, плохой менеджмент = полный набор ключевых проблем

14) Каковы цели архитектурного проектирования компьютерных систем?

TODO

Архитектурное проектирование направлено на создание эффективной, надежной и масштабируемой системы.

15) Что такое архитектура по Гради Бучу? Что представляют собой логическая и физическая структура?

Архитектура

логическая и физическая структура компонентов системы и их взаимосвязи, сформированные всеми стратегическими и тактическими проектными решениями, применяемыми во время разработки

Логический взгляд на систему учитывает концепции, созданные в концептуальной модели, и устанавливает существование и роль ключевых абстракций и механизмов, которые будут определять архитектуру и общий дизайн системы.

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

16) Что такое архитектура согласно ISO 42010? Что такое архитектурное описание?

Архитектура по ISO 42010:

фундаментальные концепции или свойства системы в ее среде, воплощенные в ее элементах, отношениях и в принципах ее конструкции развития

Архитектурное описание:

рабочий продукт, используемый для описания конкретной архитектуры(формализованное объяснение как устроена система)

архитектура системы есть всегда(что воплощено), архитектурное описание - точка зрения на систему

НО! огромное НО!

однозначного определения что такое архитектура нет, в самом документе ISO 42010 сказано следующее:

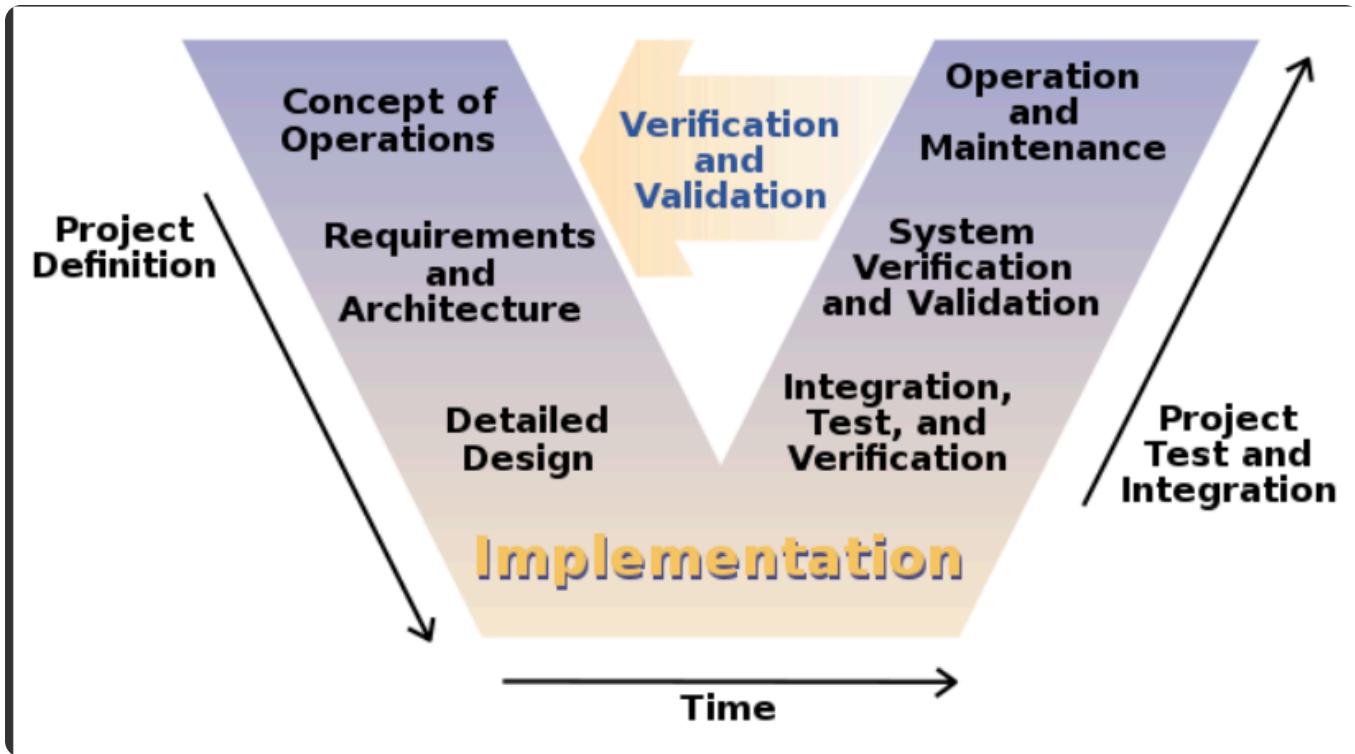
не существует единой характеристики, что является существенным или фундаментальным для системы, всё сильно зависит от проекта(буквально от компонентов или элементов системы, как системы организованы и взаимосвязаны)

17)Как архитектурные решения влияют на проектные метрики? Что такое V-диаграмма и какую особенность разработки она демонстрирует?

Архитектурные решения напрямую определяют ключевые характеристики системы, что отражается на проектных метриках:

- На производительности
- Масштабируемости
- Безопасности
- Сопровождаемости

V - Диаграмма это буквально модель разработки, которая связывает каждый этап проектирования с соответствующим этапом тестирования

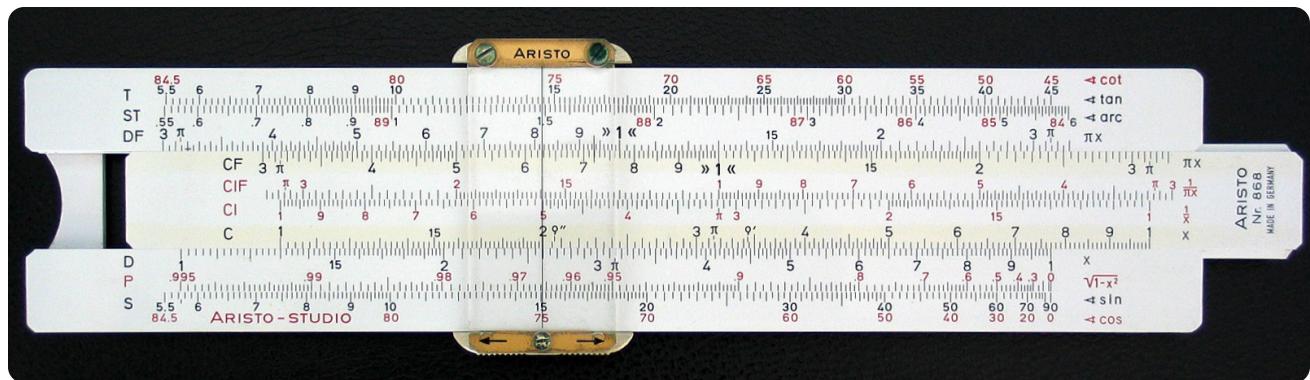


по левой ветви у нас этапы развития проекта, по правой - тесты и интеграция, в итоге получаем протестированную систему на каждом уровне.

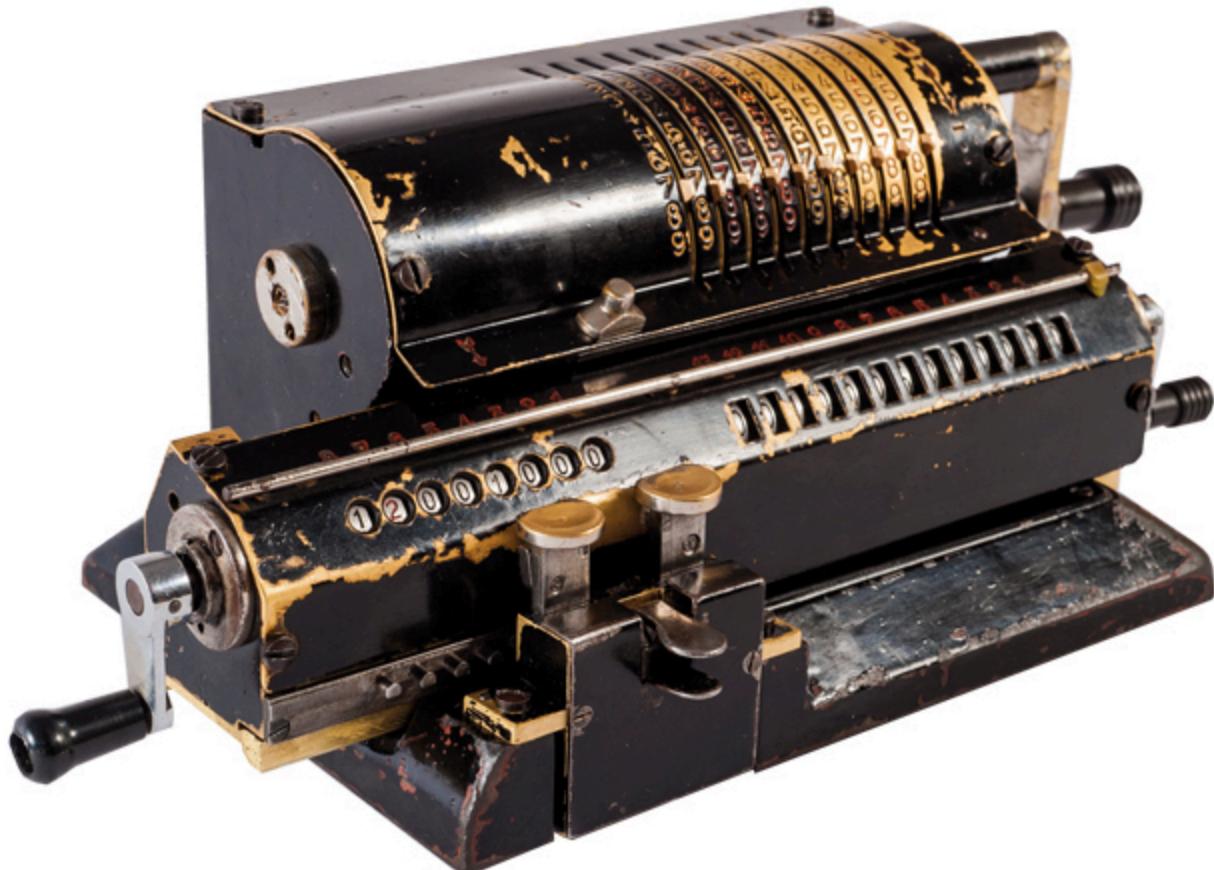
18) Сравните подходы к реализации вычислений в арифмометре и на логарифмической линейке. Какие аналогии можно провести с современными компьютерами?

Логарифмическая линейка:

- Устройство, позволяющее выполнять несколько математических операций: умножение, деление, возведение в степень(квадрат и куб), вычисление корней(квадрат и куб), логарифмов, вычислении тригонометрических и гиперболических функций и т.п.
- Принцип расчета - воплощение таблицы значений в виде, позволяющем быстро находить нужные зависимости.
- Расчет производится путем выставления линейки в требуемое положение(возможно в несколько приемов) и считывания результата



- Примеры использования(табличное исполнение): таблицы истинности(очень часто напрямую зашиваются в чипы), таблицы предрасчитанных значений: cache, memoization(Рекурсия и вычислительная сложность), SRT division(алгоритм деления бинарных чисел, какие то кусочки вычислений достает из таблицы). По сути напрямую логарифмическая линейка не используется(ну кто бы бл сомневался) НО! сама идея что мы можем взять вычисления и не проводить их, а просто сохранить результат в какой то форме активно используется в примерах выше
- Арифмометр:**



- Устройство, позволяющее складывать, вычитать, умножать и делить с фиксированным количеством разрядов.
- Принципы расчета воплощены в механизмах устройства

- Расчет производится по шагам(алгоритмически), с учетом переноса между разрядами.
Ключевое отличие арифмометра и логарифмической линейки
- Рост области определения функций:
 - для арифмометра: медленный рост сложности устройства, средний рост длительности расчёта (зависит от функции);
 - для линейки/таблицы: быстрый рост размера линейки, минимальный рост длительности расчёта.
- Рост области значений функции:
 - для арифмометра: медленный рост сложности устройства, средний рост длительности расчёта (зависит от функции);
 - для таблицы: средний рост размера таблицы, минимальный рост длительности расчёта.
 - для линейки: быстрый рост размера линейки или погрешности, минимальный рост длительности расчёта.
- Линейка - аналоговое устройство, Арифмометр - цифровое.

19)Каков был подход к расчёту артиллерийских таблиц группой людей? Какие аналогии можно провести с современными компьютерами?

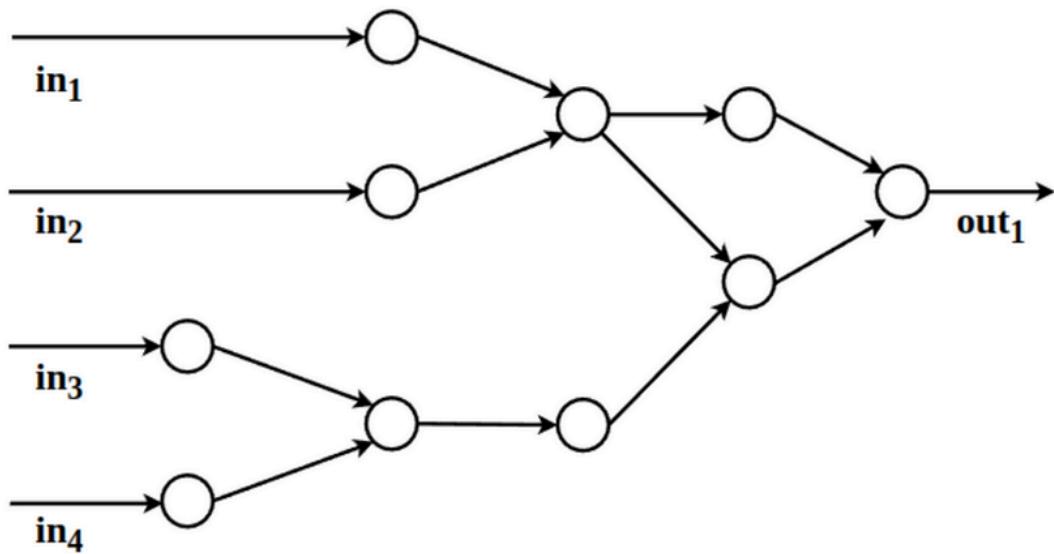
Карочи бля, я не знаю как относиться серьезно к таким вопросам, но ладно уж, распишу их:

Задача: расчёт артиллерийских таблиц для французской или российской армий.

Вычислительная задача: многократный расчёт сложной формулы с большим количеством операций для различных входных параметров.

из за ограничений(по типу простые средства расчета) и разных проблем(по типу арифметических ошибок)

Поэтому было предложено решение: **реконфигурируемый вычислитель с потоковой архитектурой**



а вот и сама [реализация](#)

1. расчёчная формула представляется в виде графа;
2. каждой вершине графа сопоставляется человек, способный выполнить соответствующую операцию (только одну);
3. люди размещаются в соответствии с графом, каждый знает, кто сообщит входные данные, кому сообщить результат;
4. на вход графа подаются параметры для расчётов;
5. на выходе собираются результаты;
6. для повышения надёжности расчётов используется двоирование (две роты, сравниваем результаты, несовпадение — повтор).

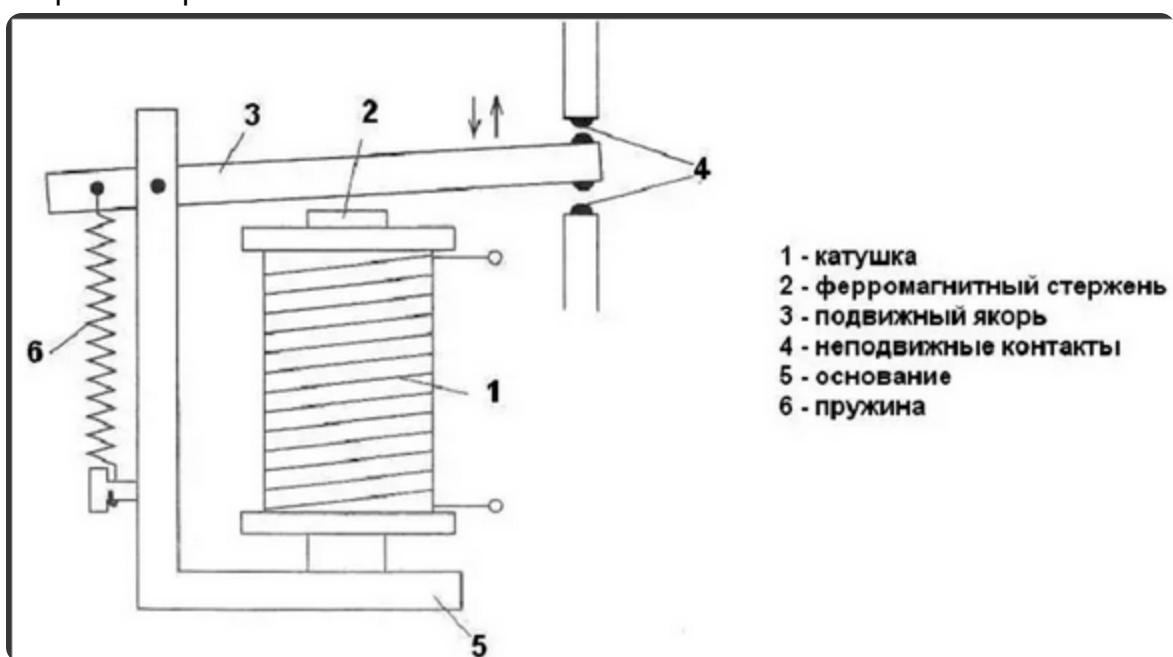
[Достоинства](#)

- Радикальный рост производительности
 - параллельное исполнение независимых операций
 - конвейеризация расчетов
- Радикальный рост надежности
- Возможность реконфигурации вычислителя под новую задачу.
И по сути, на практике все перечисленные механизмы встречаются в современных компьютерных системах(в процессорах общего назначения, в специализированных процессорах, в распределенных системах)

20)Каково устройство электрического реле? Какие виды реле существуют и каковы области их

применения?

Устройство реле:



- вход и выход, между ними ключ
- магнитная катушка; замыкает ключ
- без тока возвращается в нормальное состояние
- состояния: 0/1, есть питание/нет питания
- нет сигналов, есть уровни

Виды реле:

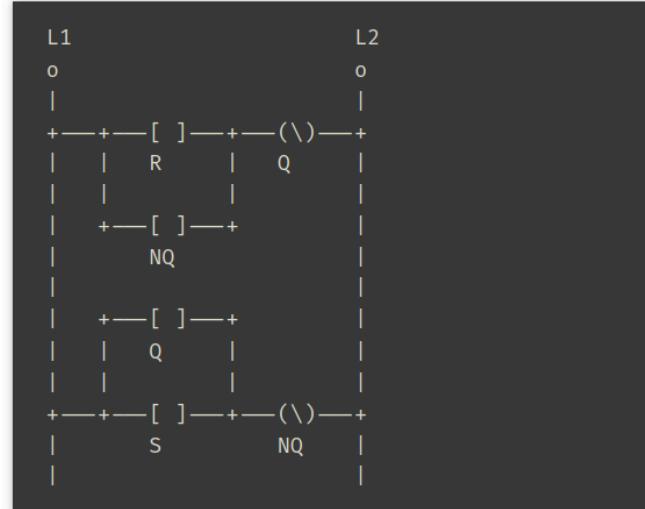
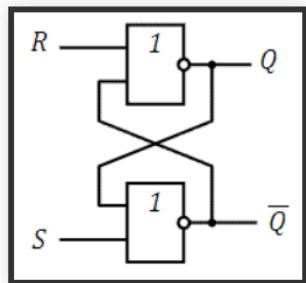
- механические
- пневматические(в военной сфере(помехи не повлияют))
- тепловые
- оптические
- акустические
- магнитные

Вообще реле используются в бытовой технике(для автоматического включения и выключения электродвигателей)

Когда то использовали чтобы делать компьютеры, непосредственно применяются в силовых шкафах управления, там где электроника не работает(неэлектрические реле) или Программируемые логические контроллеры (ПЛК)

еще при помощи реле можно реализовать RS-триггер(для хранения одного битика информации)

RS-триггер на реле



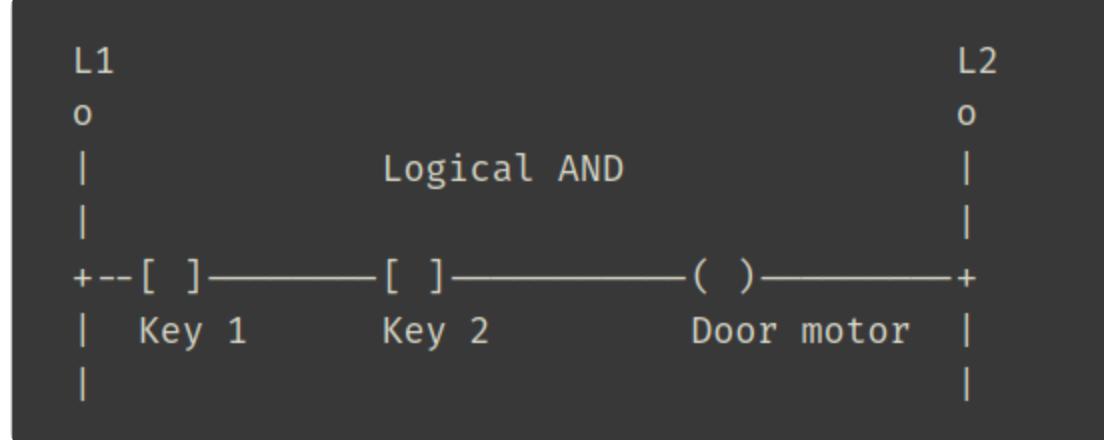
21) Как реализуется булев базис на электрических реле?

квадратные скобки - ключи, круглые - актуатор(устройство на которое подается питание, чтобы оно заработало)

- $- () -$ нормально неактивный актуатор
- $- (\backslash) -$ нормально активный актуатор

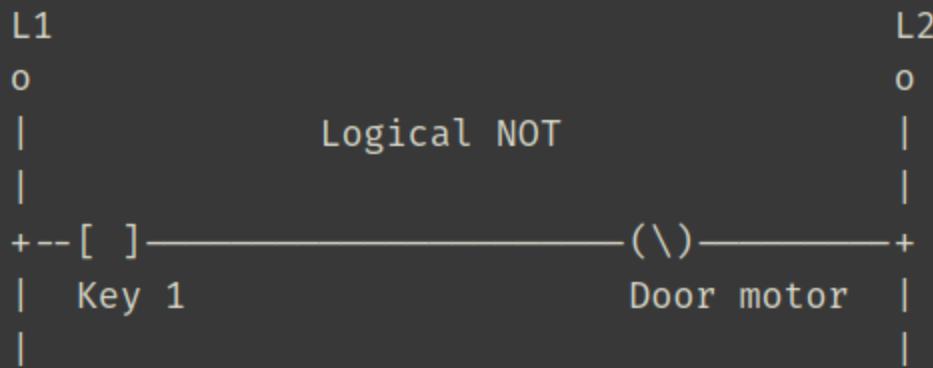
L1/L2 - шина с фазой и шина с нейтралью

Логическое и



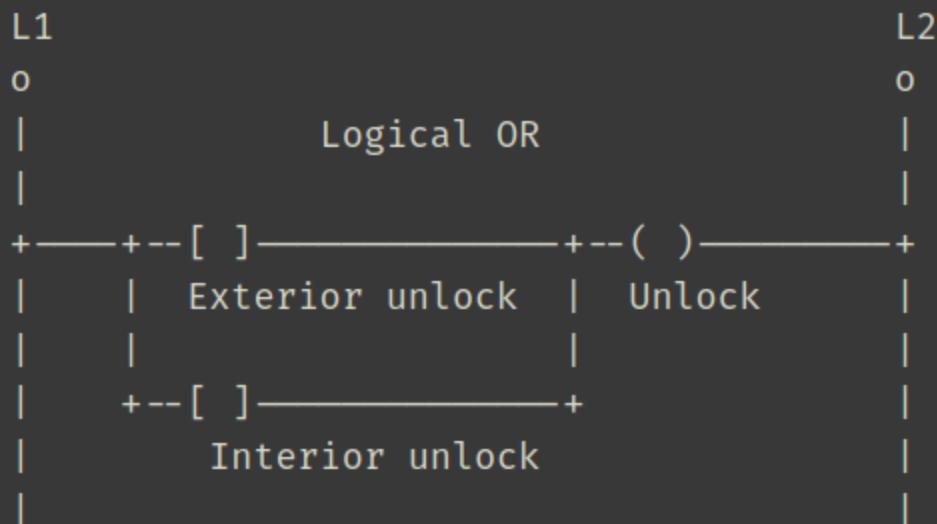
Две подряд установленные релешки, в итоге когда замыкаем оба, тогда и получаем сигнал

логическое не



тут банально, ключ разомкнут, актуатор - будет ток

логическое или



две параллельных релешки, одна из линий загорится и будет ток

22)Что такое программируемые логические контроллеры (ПЛК) и каковы области их применения?

ПЛК - специальная разновидность электронной вычислительной машины для оптимизации производств.



вот так оно примерно выглядит

плк по своей сути - это программируемое реле, плк позволяет написать программу, где такая же релейная схема, где проводочки только виртуальные и зашить необходимую логику.

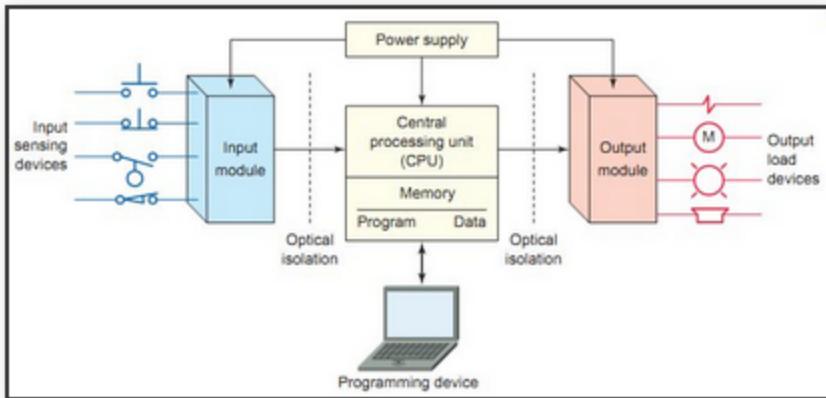
А зачем они появились:

были карочи проблемы автоматизации релейными схемами:

- плохо масштабируются
- сложная настройка и поддержка
- ненадежность сложных схем
- проблема ведения документации
к особенностям плк относится:
 - управляющие системы(Вообще реле удобны тем, что нет циклов, легко предсказать вычислительный процесс и легко рассчитать время реакции на разные события)
 - законченные устройства ввода-вывода для встроенного использования(конкретный девайс, конкретное по написанное для него, воткнули в шкаф)
 - специализация ввода-вывода, модульность
 - выделение инструментальной составляющей
 - эксплуатация в тяжелых условиях

23)Какова структура программируемых логических контроллеров (ПЛК)? Причины отделения инструментальной составляющей?

Типовое устройство ПЛК



Также существуют и полностью программные реализации.

карочи структура плк довольно типичная для управляющих систем, есть модули ввода/вывода и какой то CPU, который и преобразует информацию(в целом стандартно для управляющей системы).

ну еще и есть инструментальный компьютер, на котором и пишется вся необходимая логика

Инструментальный компьютер (ПК с ПО для разработки) физически и логически отделен от ПЛК по следующим причинам:

1. В целом, программирование плк не похоже на обычное программирование, оно всё создано для инженеров, а не программистов, отсюда и получаем такой набор примерно: во первых есть стандарт для программируемых контроллеров, во вторых оно всё визуализировано и сделано так, чтобы написанный софт работал предсказуемо: функциональные блоки(специальный язык, буквально можем соединить кнопку с двигателем), последовательные функциональные диаграммы(что-то типа конечных автоматов), ну и какой то ЯП нужен => получаем структурированный текст.
2. плюс сюда можно добавить сказ о безопасности и надежности, о разделение среды разработки и исполнения, гибкости разработки, да и просто в этом и есть вся разница между плк и реле, есть возможность написать софт на виртуальном железе и зашить эту логику в шкаф.

24)Каковы особенности аппаратного обеспечения программируемых логических контроллеров (ПЛК)?

Особенности:

- Релейно контактные схемы
- Функциональные блоковые диаграммы
- Последовательные функциональные диаграммы
- Структурированный текст
и всё это сделано для удовлетворения следующего:
- Использование не программистами
- визуальность
- формальные свойства, реальное время

25)Каковы особенности программного обеспечения программируемых логических контроллеров (ПЛК)?

к особенностям ПЛК относится:

- управляющие системы(Вообще реле удобны тем, что нет циклов, легко предсказать вычислительный процесс и легко рассчитать время реакции на разные события)
- законченные устройства ввода-вывода для встроенного использования(конкретный девайс, конкретное по написанное для него, воткнули в шкаф)
- специализация ввода-вывода, модульность
- выделение инструментальной составляющей
- эксплуатация в тяжелых условиях

26)В чём заключается принцип развития иерархических систем Седова? Иллюстрируйте применительно к вычислительным платформам.

Принцип развития иерархических систем Седова: - закономерности и механизмы, по которым развиваются сложные иерархические системы.

Ключевой принцип: ограничение разнообразия на нижележащем уровне для получения разнообразия на вышележащем уровне.

Иллюстрации к вычислительным платформам:

- если нам хочется получить большое количество разных мобильных приложений, то нам нужно ограничить количество операционок, которые будут стоять на этих

мобильниках

- если мы хотим получить эффективные компиляторы, то нам нужно унифицировать количество процессорных архитектур, которыми мы пользуемся.
да и можно сделать подобный вывод:
Повторное копирование того же самого, но на новой платформе(в новых системах)
это всегда очень дорого.

27)Что такое булев базис и какова его роль в вычислительной технике? Приведите примеры.

Вообще чтобы построить произвольную вычислительную систему нам надо:

1. инструментарий, который позволит работать с двоичной логикой
2. полный набор булевых функций
3. Комбинационные схемы
4. триггер - хранение состояния.

Булев базис - совокупность набора из функций **И, ИЛИ, НЕ**

есть еще штрих шеффера и стрелка пирса, через них можно тоже выразить любую булеву функцию.

или то же самое, но другими словами:

Булев базис - минимальный набор логических операций, через которые можно выразить любую булеву функцию.

Роль булева базиса в вычислительной технике:

- Основа цифровой логики(все процессоры и микросхемы работают на базе булевых операций)
- Минимизация аппаратной реализации(любую схему можно собрать на булевом базисе)
- Универсальность(с помощью булева базиса можно описывать арифметические операции, условные переходы, память)

Примеры:

- RS-триггер(собран на NOR и NAND)
- Во всяких сумматорах, вентильных схемах

28)Что такое двоичное кодирование сигналов? Каковы его достоинства и недостатки? Что такое

запретная зона?

Вообще, допустим в аналоговых системах есть большой набор значений, что создает большое количество проблем.

Двоичное кодирование - представление данных или сигналов в виде последовательности битов(0 и 1).

Пускай мы говорим, что у нас есть два значимых состояния: логический 0 или логическая 1. После мы фиксируем интервал значений для 1 и для 0, в которых ток может свободно прыгать и нам абсолютно пох на погрешности. В итоге это сильно упрощает нашу работу, так как множество переменных которые влияли как ток будет распространяться. В итоге наша задача не сделать как можно точнее, а получить попал ли сигнал в допустимое значение или нет.

Запретная зона - расстояние между логическим 0 и 1, диапазон значения, в котором система будет игнорировать изменения(он туда как бы попадать не должен, но он будет и мы будем его игнорировать)

Достоинства:

- надежно и помехоустойчиво
- простая арифметика
- диапазоны и точность наращиваются разрядностью
- погрешности by design, а не by implementation(by design - на этапе проектирования мы уже знаем наши разрядности(проектировщик сознательно делает расчеты в системе неточными), сколько до запятой, сколько после, by implementation - тут уже погрешности будут из за внешних факторов и всегда разные)

Недостатки:

- Нечитаемое представление
- Простые десятичные дроби записываются в виде бесконечных двоичных дробей
- Дискретное кодирование сигналов(точность).

29)Какова роль машинного слова в устройстве процессора? Что такое Big- и Little-endian?

Машинное слово - единица данных, естественная для обработки вычислителем.

Пример: сложение, пересылка и т.п.

Целые числа:

- Позиционные системы исчисления

- Доп код

Дроби:

- с фиксированной точкой(мантицы нет, в бинарном коде строго зафиксированное место точки)
- с плавающей точкой(база мантисса и знаковый бит)

Перечисления:

- Двоично-десятичное кодирование
- Символы

Big-endian - старший бит в машинном слове - старший(порядок от старшего байта к младшему), **Little-endian** - наоборот(порядок от младшего к старшему). А зачем вообще little-endian если он так нечитаемый? АТВЕТ: как пример можно по указателю вывести байты, просто итерируясь по ним, в big-endian придется делать хитрые пересчеты.

30)Какие существуют способы кодирования целочисленных данных? Что такое позиционное кодирование, код Грея, BCD?

Прямой, обратный и доп код, код грея, BCD(Binary-Coded Decimal)

Позиционное кодирование - метод для передачи информации о порядке элементов в последовательности.

Код Грея:

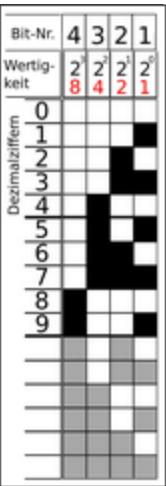
две "соседние" кодовые комбинации различаются только цифрой в одном двоичном разряде.

BCD:

форма записи рациональных чисел, когда каждый десятичный разряд числа

записывается в виде его четырёхбитного двоичного кода

Decimal	Gray Code	Binary	Bit-Nr.			
			4	3	2	1
Wertigkeit	2 ³	2 ²	2 ¹	2 ⁰		
0	0000	0000	0	0	0	0
1	0001	0001	1	0	0	0
2	0011	0010	0	1	0	0
3	0010	0011	0	0	1	0
4	0110	0100	1	0	1	0
5	0111	0101	1	1	0	0
6	0101	0110	0	1	1	0
7	0100	0111	0	0	1	1
8	1100	1000	1	1	0	0
9	1101	1001	1	1	0	1
10	1111	1010	1	1	1	0
11	1110	1011	1	1	1	1
12	1010	1100	0	0	1	0
13	1011	1101	0	0	1	1
14	1001	1110	0	1	0	0
15	1000	1111	0	1	0	1



The diagram shows a 4x8 grid where each row represents a decimal number from 0 to 15. The columns are labeled 4, 3, 2, 1 from left to right. The first three columns represent the Gray code, and the fourth column represents the binary equivalent. The binary values are: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000. The first three columns show the Gray code sequence: 000, 001, 011, 010, 110, 111, 101, 100, 110, 111, 101, 100, 010, 011, 001, 000.

31)Какие существуют способы кодирования бинарных данных? Что такое Base64, Base58?

Вообще бинарные данные требуется представить в текстовом виде для:

- передачи по текстовым протоколам(http, json)
- хранение в конфигурационных файлах
- удобного отображения

Base64 - стандарт кодирования двоичных данных при помощи только 64 символов ASCII. Алфавит кодирования содержит латинские символы A-Z, a-z, цифры 0-9(всего 62 знака) и 2 дополнительных символа(произвольные бинарные данные можно записывать в формате аски)

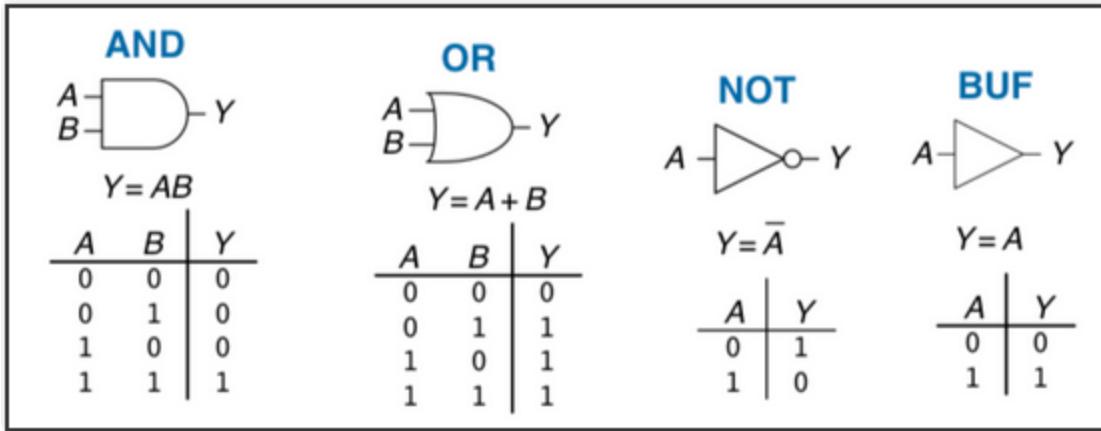
Base58 - аналогично base64, но изменено, чтобы избежать небуквенно-цифровых символов и букв, которые могут выглядеть неоднозначно при печати(I, O, 0, 1)

32)Что такое комбинационные схемы? Что такое переходный процесс? С чем связаны задержки и накопление ошибки? Как связаны комбинационные схемы с параллелизмом уровня бит?

Комбинационные схемы - схема, составленная из набора логических элементов, в совокупности реализующая заданную таблицу истинности(или другими словами - реализация функции, которая отображает множество А в множество В)

Требования: А и В - конечны

Вот простейшие элементы комбинационных схем:



Переходный процесс - кратковременный ложный сигнал на выходе комбинационной схемы, возникающий из за разной задержки распространения сигналов через логические элементы.

Задержки и накопление ошибки:

- Каждый логический элемент вносит задержку(в сложных схемах задержки суммируются)
- Накопление ошибки в физическом процессе, что может привести к ошибке на логическом уровне(ее в целом не будет если нет длинных проводников)
Комбинационные схемы - основа **битового параллелизма** в процессорах(узлы с параллельным включением работают параллельно, вид параллелизма основанный на ширине слова, несколько бит обрабатываем параллельно в рамках машинного слова
 \Rightarrow оно параллельно на уровне отдельных бит):
- Одновременная обработка бит(как пример сумматор обрабатывает все биты параллельно за 1 такт)
- SIMD-инструкции(одна команда применяется ко множеству бит одновременно, SIMD - Single Instruction Multiply Data)

33)Какие состояния существуют в комбинационных схемах (0, 1, x, z) и что они означают?

0 - логический ноль

- Низкий уровень напряжения
- Соответствует False

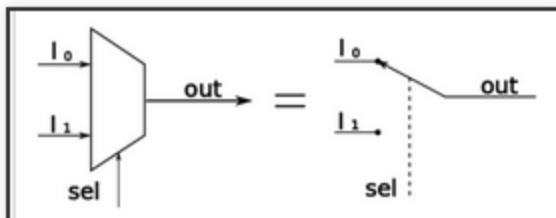
1 - логическая единица

- Высокий уровень напряжения

- Соответствует True
 X - неизвестное, значение может быть произвольным, так как таблица истинности может быть произвольным.
- Неизвестное значение
- возникает в случае: конфликтов входов(кз между 1 и 0), Неинициализированной памятью
- деление на ноль
- квадратный корень из отрицательного числа
 Z - отключено, когда источник данных висит в воздухе!
- Отключенный вход
- Очень высокое сопротивление(разомкнута цепь)

34)Каковы особенности реализации "условного оператора" в комбинационных схемах?

Ваше они реализованы через мультиплексор, вот пример реализации условного оператора:



Если говорить про то, как оно описывается то вот:

```
in0 = f0(...)  
in1 = f1(...)  
  
out = in0 if sel else in1
```

PYTHON

или на verilog

```
assign in0 = f0(...);  
assign in1 = f1(...);  
  
assign out = sel > in0 : in1
```

Тут из приколов можно отметить следующее:
вообще подобные условные операторы уже имеют на входе вычисленные значения,

поэтому независимо от условия, они оба будут просчитаны(но сами результаты не обязательно должны иметь предсказуемые значения)

35)Что такое триггеры в цифровых схемах? Каковы варианты их использования?

Триггер - класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием сигналов

триггеры позволяют

- Зафиксировать состояние линии на длительное время
- Ограничить распространение сигнала по схеме(те 1) разорвать цикл, 2)Сократить задержку в комбинационной схеме)

36)Что такое D-триггер и RS-триггер? Какие существуют варианты условия изменения состояния?

D-триггер(триггер задержки) - запоминает состояние входа и выдает его на выход.

RS-триггер(reset/set) - асинхронный триггер, который сохраняет свое предыдущее состояние при неактивном состоянии обоих входов и изменяет свое состояние при подаче на один из его входов активного уровня.

Условия изменения состояния:

- по фронту
- по уровню

37)Что такое пространственные и временные вычисления? Как они могут быть использованы для оптимизации процессоров?

Пространственные вычисления - параллельная обработка данных, когда несколько операций выполняются одновременно на разных блоках процессора

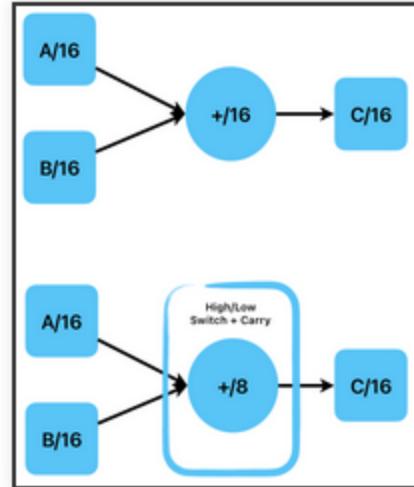
Временные вычисления - выполнение операций последовательно, но с перекрытием

стадий конвейера или предвыборкой данных.

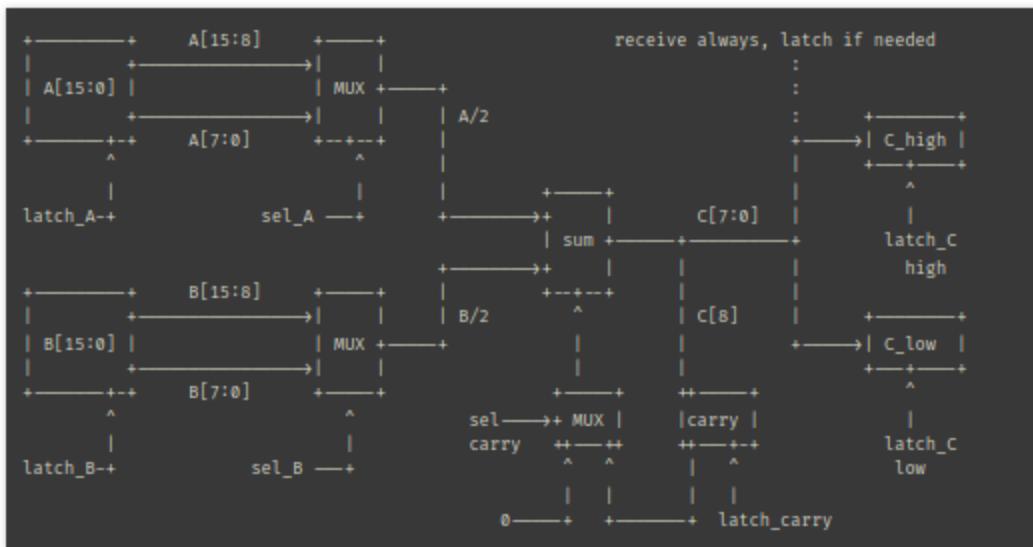
- Сложение 16 битных

чисел:

- сумматор на 16 бит и 1 такт.
- сумматор на 8 бит и 2 такта + система управления.



Сумматор на 8 бит и 2 такта + система управления



тут как пример можно рассказать про сумматор на 16 бит и на 8, который сработает в 2 такта(+ можно сказать про вычисление биг инт чисел, логику которых можно реализовать через систему управления, а не аппаратно изменять все кампуктеры на 128 бит вместо 64, ведь спектр задач небольшой)

38)Что называют синхронной схемотехникой? Каковы её достоинства и недостатки?

Синхронная схемотехника - мы запускаем наши комбинационные схемы не от входа к выходу максимально быстро, а у нас есть четкие такты сигналов, которые происходят с определенной регулярностью и между этими регулярными событиями у нас должны

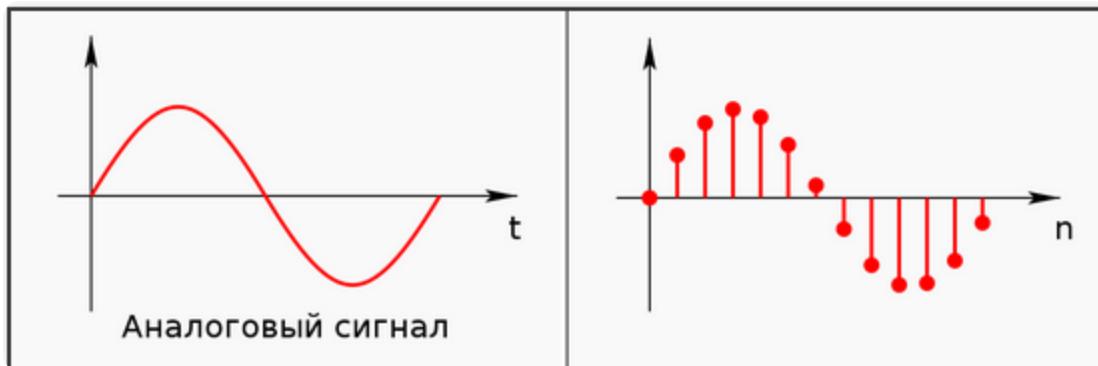
закончиться все переходные процессы внутри нашей схемотехнике.

Достоинства:

- меньше гонок, проще синхронизация
- частота по самой медленной комбинационной схеме
- синхронность в заданном диапазоне

Недостатки:

- ненужные задержки(теоретически)
- дискретизация входных сигналов по времени(на картинке, из аналогового сигнала получаем его подобие в отдельных точках времени)



- энергопотребление(синхросигнал постоянно отщелкивает и работает постоянно => энергопотребление)

39) Почему цифровая схемотехника оперирует уровнями, а не сигналами? Какие возможности это открывает и какие проблемы создаёт?

Сигнал это что то мгновенное во времени(буквально аналогия отправить запрос на сервер, он мгновенно ушел и мы когда то получим ответ). Уровень же выставляется, и пока его никто не трогает он будет стоять.

Есть регистр и линия входа, выставили уровень и он упрется в этот регистр пока мы не его не прочитаем.

Цифровая схемотехника использует дискретные уровни напряжения(0/1) вместо аналоговых сигналов.

- Устойчивость к помехам(Аналоговые сигналы чувствительны к шумам, искажениям и падениям напряжения.)
- Простота обработки и хранения (- Логические элементы (И, ИЛИ, НЕ) работают только с двумя состояниями, что упрощает проектирование; - Память (триггеры, регистры) хранит чёткие значения, а не плавно меняющиеся сигналы.)

- Масштабируемость и предсказуемость (Цифровые схемы легко комбинируются в сложные системы + поведение системы детерминировано)

Какие возможности открывает:

- Высокая надежность(возможна работа с помехами)
- Автоматизация проектирования(логику можно описывать языками по типу Verilog и потом синтезировать в схему)
- Можно строить кеши, конвейеры

А вот и проблемы:

- Точный контроль уровней(если напряжение проваливается, то будут ошибки(псы): можно пофиксить буфферами, повторителями и не делать длинных проводников))
- Задержки распространение сигналов
- Потребление энергии при переключении синхросигнала(выше описывал)

Вот еще хороший слайд с выводами по схемотехнике:

Ключевые отличия схемотехники от программирования

1. Все процессы между регистрами всегда происходят параллельно. Не читайте код как алгоритм, рисуйте схему.
2. Передача сигнала – физический аналоговый процесс. Есть питание и контакт – есть передача. Сигнал не может не идти.
3. Нет понятия "система остановилась". Она всегда работает, если есть питание.
4. Таблица истинности неполна – результат будет случайным, но будет (и возможно воспроизводимым).

40)Каковы ключевые тенденции в производстве радиоэлектронной аппаратуры и связанные с этим проблемы?

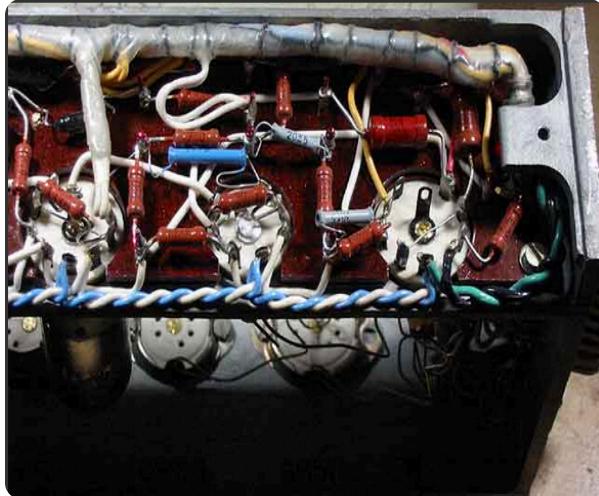
Ключевые тенденции:

- Рост уровня интеграции

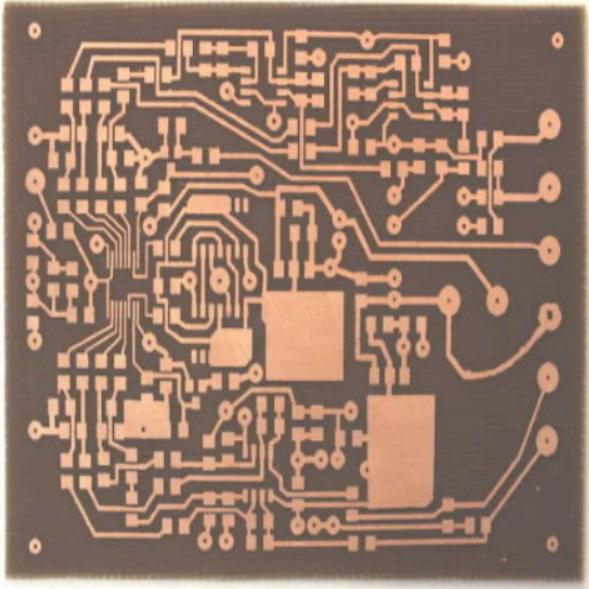
- снижение затрат на производство изделия
- усложнение производственной цепочки

41)Что такое навесной монтаж, монтаж на печатную плату, штыревой и поверхностный монтаж? Как они обеспечивают "гибкость"?

Навесной монтаж - способ монтажа электронных схем, при котором расположенные на изолирующем шасси радиоэлементы соединяются друг с другом проводами или непосредственно выводами.(буквально детали прикручены к шасси и соединены проводами)



Монтаж на печатную плату - есть карочи печатная плата - это пластина из диэлектрика, на поверхности и/или в объеме которой сформированы электропроводящие цепи электронной схемы. Электрическое и механическое соединение компонентов(буквально детали вставляются в отверстия и запаиваются, механическое соединение деталей и электронных компонентов в последовательности, обеспечивающий их требуемое расположение и взаимодействие для тех. требований)



Поверхностный монтаж(SMT/SMD - Surface Mount Technology/Device, разновидность монтажа на печатную плату) - маленькие детали припаяны к поверхности платы без отверстий. (буквально фиксируем электронные компоненты непосредственно на внешние стороны платы, выводы компонентов монтируются на металлизированные площадки на печатной плате, тем самым получаем электрические цепи).

Штыревой монтаж - детали с ножками вставляются в отверстия платы и запаиваются с обратной стороны.

Как это обеспечивает гибкость:

- Например навесной монтаж - прост в производстве, подготовки производства и гибок(но он никак не поддается автоматизации, так как вручную собирается монтажниками2)
- Монтаж на печатную плату - гибкость в ручной сборке и ремонте
- Поверхностный монтаж - гибкость в автоматизации
Каждый способ имеет свои плюсы и минусы => получаем гибкость и разнообразия выбора под разные цели и задачи

Поверхностный монтаж: Достоинства

- автоматизация
- плотность размещения компонент
- размеры
- стоимость в серии

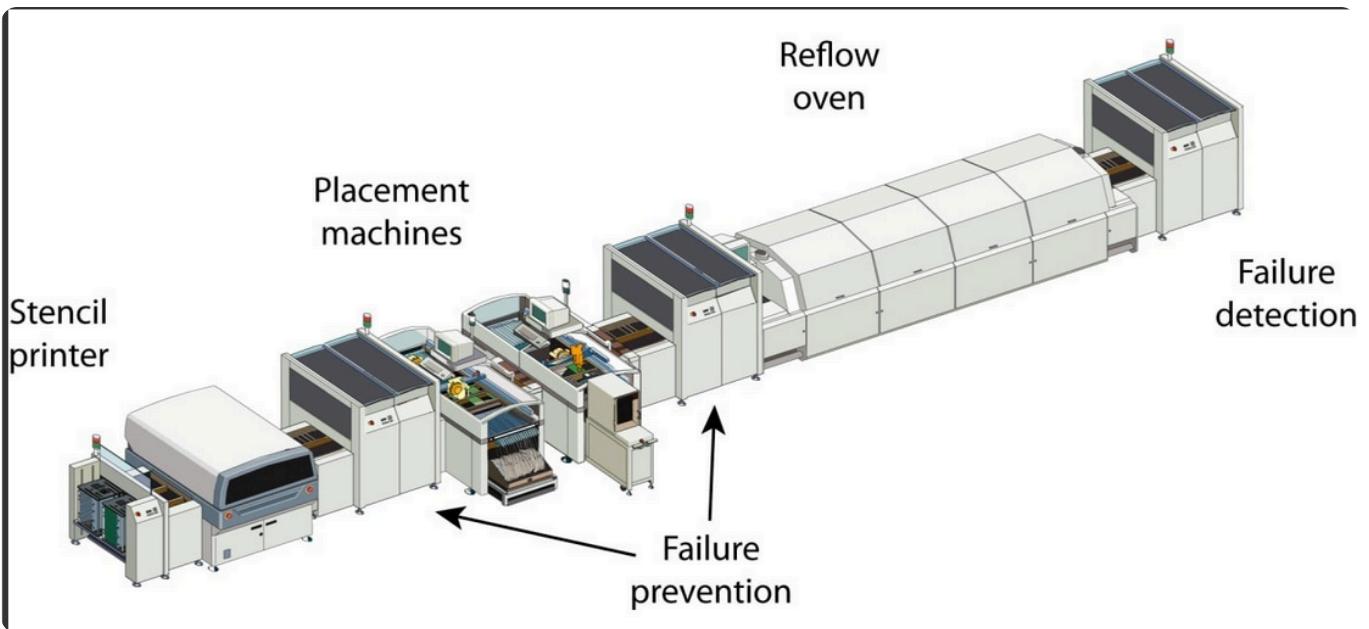
Поверхностный монтаж: Недостатки

- необходимость подготовки производства
- удлинение производственной цепочки
- сложность внесения исправлений

Поверхностный монтаж: "Гибкость"

- джампер (by design)
- "перерезать дорожку"
- "навесная дорожка"

42) Какова производственная цепочка поверхностного монтажа? Каковы её этапы?



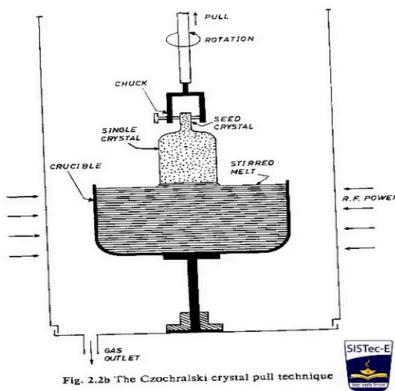
1. Нанесение паяльной пасты(через металлический трафарет на плату наносится паяльная паста)
2. Установка компонентов(размещаются SMD компоненты на паяльную пасту)
3. Пайка(плата проходит через печь, где паста плавится и формируются соединения)
4. Обнаружение косяков(проверяются пропущенные/перекошенные компоненты и качество пайки)

43) Какова производственная цепочка кремниевого производства? Каковы особенности формирования цены изделия?

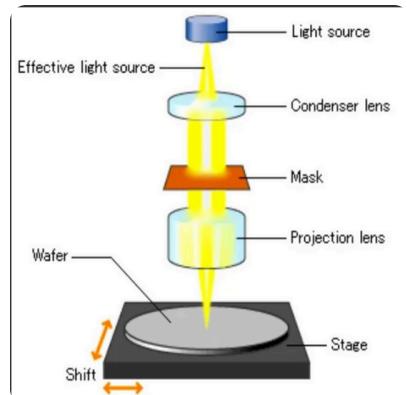
Производственная цепочка:

1. добываем и чистим кремниевый песок
2. выращиваем монокристалл кремния(монокристалл - кристаллическая цепочка ровная, в природе таких не бывает, такая точность необходима, так как размеры транзисторов

очень маленькие)



3. Производим подложку будущих чипов(пилим монокристалл на тонкие блинчики и после на каждом блинчике будем выращивать чипы)
4. после на эту кремниевую подложку наносят материал для рисунка, наносится фоторезист, экспонируется через фотошаблон и после удаляют отработанный фоторезист.



Особенности формирования цепи:

Вообще сделать маску для 4 этапа стоит большое количество деняк, в итоге для производства 1 чипа = производство 10млн чипов с такой же маской, отсюда и появляется эффект масштаба.

44)Какие трудности связаны с аппаратным обеспечением на этапах производства и эксплуатации?

Карочи, тут куча всяких проблем, начиная с размерности производств, колосальная серийность, безумно сложные этапы производства.

можно в целом упомянуть про этапы производства, и немного рассказать про сами этапы или можно отойти от кремниевых чипов и рассказать в общих чертах:

На производство влияют многие факторы:

- Логистика
- склады
- специалисты
- производственная цепочка
- тестирование
- упаковка
- дистрибуция
- гарантийный ремонт

Во время эксплуатации:

- Выход оборудования из строя
- Необходимость физического доступа
- Особенности среды эксплуатации
- Долгосрочные эффекты(по типу оловянные нитевидные кристаллы в электронной технике, ссадится аккум, иссыхаются элементы)
- Ну и обновления систем, чем совершеннее технология, тем сложнее внести изменения

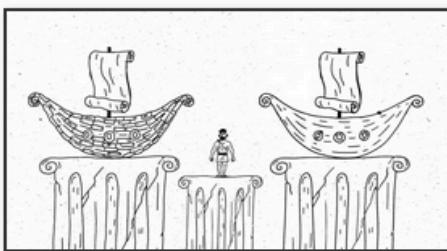
45)Какие подходы к решению проблемы "устаревающей аппаратуры" существуют с точки зрения аппаратуры и ПО?

Обслуживание стареющей аппаратуры

1. Срок службы. Деградация надёжности. Стоимость обслуживания.
2. Ремонт требует запасных компонентов.
 - Дорогое хранение.
 - Конечный запас.
3. Воспроизведение:
 - Вытеснение устаревшей элементной базы.
 - Устаревшую элементную базу дорого производить.
 - Низкий спрос. Штучное производство.
 - Барахолки.
4. Вывод из эксплуатации сопутствующего оборудования:
 - CD-ROM в компьютерах не устанавливается.
 - 2.5 дискеты больше не производят.

Варианты замены устаревшей аппаратуры

1. Перепроектирование на новой элементной базе.
2. Модульная организация.
Стандартные интерфейсы.
Имитация старых систем.
3. Виртуализация.
4. Проблема пользователяского опыта: компьютер 80-ых позволяет внести запись в БД, пока современный только загружается.



Парадокс Тесея.

**46)Что такое концепция 2-этапного производства?
Какие существуют варианты этапа
"конфигурирования" для разных уровней
организации вычислений?**

грубо говоря это подход при котором аппаратное обеспечение изготавливается в универсальном виде, а затем настраивается под конкретные задачи на этапе конфигурирования

2ух этапное производство состоит из:

1. Производства "универсальной компьютерной системы(Hardware)"
2. Настройки прикладного поведения(Software)

Возможности конфигурирования определены заранее.

варианты конфигурирования:

- заложенные при проектировании
- незадокументированные(нечеловое использования, дыра в безопасности)
Метод конфигурирования зависит от стадии жизненного цикла

47)Каково определение программной системы согласно OMG Essence? Каковы её части?

Определение программной системы согласно стандарту OMG Essence - система, состоящая из программного обеспечения, оборудования и данных, которая обеспечивает свою основную ценность посредством выполнения программ.(ну или просто в двух словах совокупность software, hardware and data)

В итоге в любой программной системе есть:

- software - ПО
- hardware - аппаратное обеспечение
- data - какие то необходимые для работы данные, являющиеся частью системы.

48)Что такое программное и аппаратное обеспечение? Что означают понятия Hardware и Software? Сопоставьте их.



- **Аппаратное обеспечение** - электронные и механические части вычислительного устройства, входящие в состав системы или сети, исключая программное обеспечение и данные(информацию, которую хранит вычислительная система и обрабатывает). Аппаратное обеспечение включает: кампьютеры и логические устройства, внешние устройства и диагностическую аппаратуру, энергетическое оборудование, батареи и аккумуляторы.
 - **Программное обеспечение** - совокупность программ, систем обработки информации и программных документов, необходимых для эксплуатации. Позволяет аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.
- блять вот тут пенской наговорил какой то хуйни, которую я не выкупил ваще, какая то абстракция на абстракции, доеб до слов и перевода с английского на русский, в этом моменте складывается ощущение что итмо и курс ак это шиза.
поэтому вот два слайда с лекции

Software -- это та часть компьютерной системы, которая приспосабливает технику к различным видам использования. Например, на одном и том же компьютере, но с разным программным обеспечением, вы можете играть в игру, рассчитывать налоги, писать письмо или книгу, или получать ответы на вопросы о свиданиях.

Затем я объяснил ей, что, к сожалению, в начале истории развития компьютеров этой функции дали название "программное обеспечение (software)", в отличие от "аппаратного обеспечения (hardware)". Ее следовало бы назвать "гибкое программное обеспечение (flexibleware)".

К сожалению, термин "soft" многие интерпретировали как "легкий (easy)", что совершенно неверно. Не заблуждайтесь.

То, что мы называем "hardware", должно было называться "easyware", а то, что мы называем "software", можно было бы назвать "difficultware".

Аппаратура и ПО \neq Hardware и Software

1. Программная/аппаратная составляющая слабо связаны с SW/HW
 - Minix – одна из самых популярных ОС.
2. Часто наименование программной или аппаратной составляющей зависит от языка:
 - Virtual-Computer, -Network, -Volume, etc.
 - Почему ПЛИС – это аппаратура, так как работает как схема.
3. Разделение на SW/HW зависит в большей степени от способа использования элементной базы.
 - Hardware – то, что тяжело/долго/дорого поменять;
 - Software – то, что легко/быстро/дешево поменять.
4. HW совпадает с аппаратной составляющей, если нет альтернатив: питание, антенны, аналоговые сигналы и т.п.

49)Какие возможности открывает программное обеспечение (цикл разработки, гибкость, контроль, и

т.д.)?

1. Быстрый цикл разработки
2. Легко заменяется прямо у пользователя
3. Пользователь как Beta-тестер
4. Возможно удаленное обновление, в том числе без информирования согласия
5. Сервис - вершина владения компьютерной системы
6. Процесс создания и внедрения ПО автоматизируется(CI/CD)
7. Высокая сложность программ

50) Чем отличается типовое проектирование Hardware/Software от совместного (CoDesign) проектирования Hardware/Software? Каковы их достоинства и недостатки?

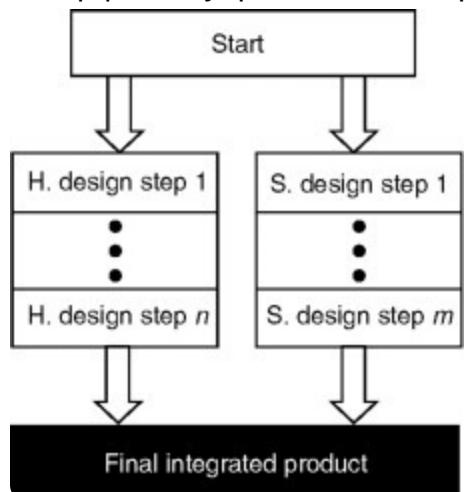
Типовое проектирование:

Проектируем аппаратуру:

- Система команд и вычислительные механизмы
- интерфейсы управления аппаратурой
- цифровая схемотехника, система команд

Пишем программу:

- прикладная задача
- языки и либы
- система команд и вычислительные(вирт. память, прерывания)
- интерфейсы управления аппаратурой



в итоге получаем, что сначала разрабатываем hw, а потом уже sw(хотя на картинке нихуя не так *но я нихуя не понял что пенской пытался этим сказать(потому что в конце он добавил в БОЛЬШИНСТВЕ случаем вы сначала делаете hw, а потом sw)*)

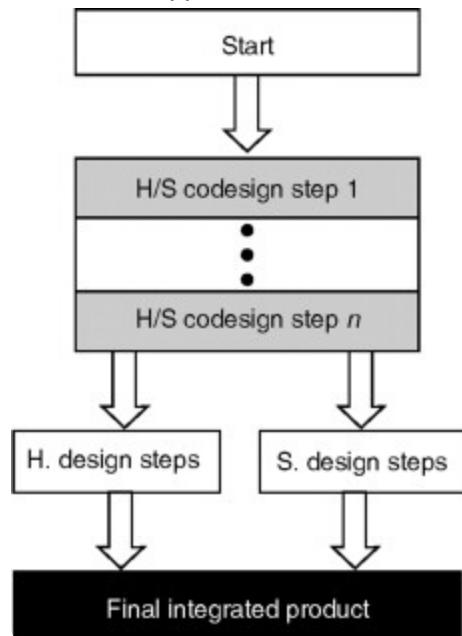
Проблемы подобного:

- излишне шаблонное проектирование
- высокие интеграционные риски
- последовательная разработка(сроки)

CoDesign проектирование:

Проектируем систему:

- Разработка интерфейса HW/SW(системы команд)
- Разработка моделей HW/SW для получения обратной связи
- Разработка HW/SW, верификация моделей
- Замена моделей на HW/SW



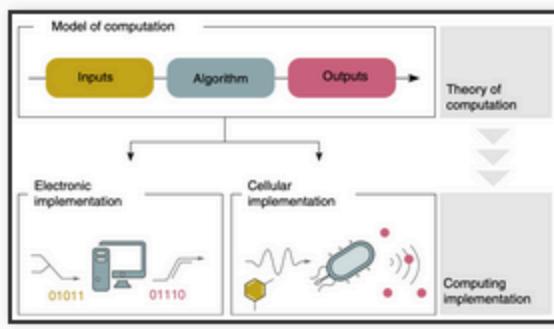
В итоге основная разница, в том, что мы пытаемся избежать этой поэтапности в типовом проектировании, в итоге находим общие этапы, где можно параллельно разрабатывать HW/SW и в полной интеграции. Ну и конечно же, codesign оказывается быстрее в скорости разработки

51)Что такое "Модель вычислений"? В чём назначение моделей вычислений? Приведите примеры.

Модель вычислений

1. MoC предоставляет язык для описания моделей процессов/программ/структур.
2. MoC определяет возможности вычислительной машины.
3. MoC характеризует, как исполняется модель (программа): возможные состояния вычислителя, их последовательность, правила переходов (возможно детерминированное).
4. MoC, обычно, минималистичны относительно реальных вычислителей.

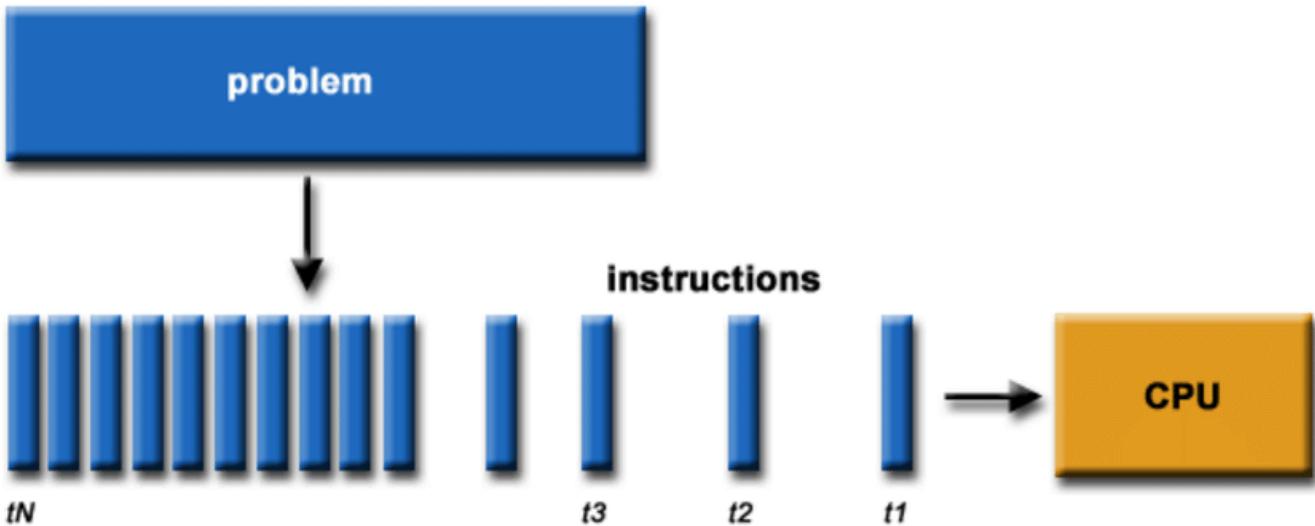
Model of Computation –
MoC



Источник

в целом вся инфа на 1 слайде.

52)Что такое последовательные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс?



Последовательная модель вычислений позволяет описать последовательный процесс, который может быть представлен как последовательность переходов состояний:

- Конечные автоматы(переключение между состояниями)
- Автомат с магазинной памятью(Pushdown automata)(запоминать данные в стеке, но обрабатывать только верхние)
- Машины Тьюринга(есть лента с ячейками, читает символ, меняет его по правилу, переходит в новое состояние)
- Машины с произвольным доступом(как пример переход к разным ячейкам памяти)
- Машина Фон Неймана(инструкции выполняются по очереди, взять -> декодировать -> выполнить)

53)Что такое машина Тьюринга и почему она важна для теории вычислений?

Машина Тьюринга - это абстрактная вычислительная модель, состоящая из:

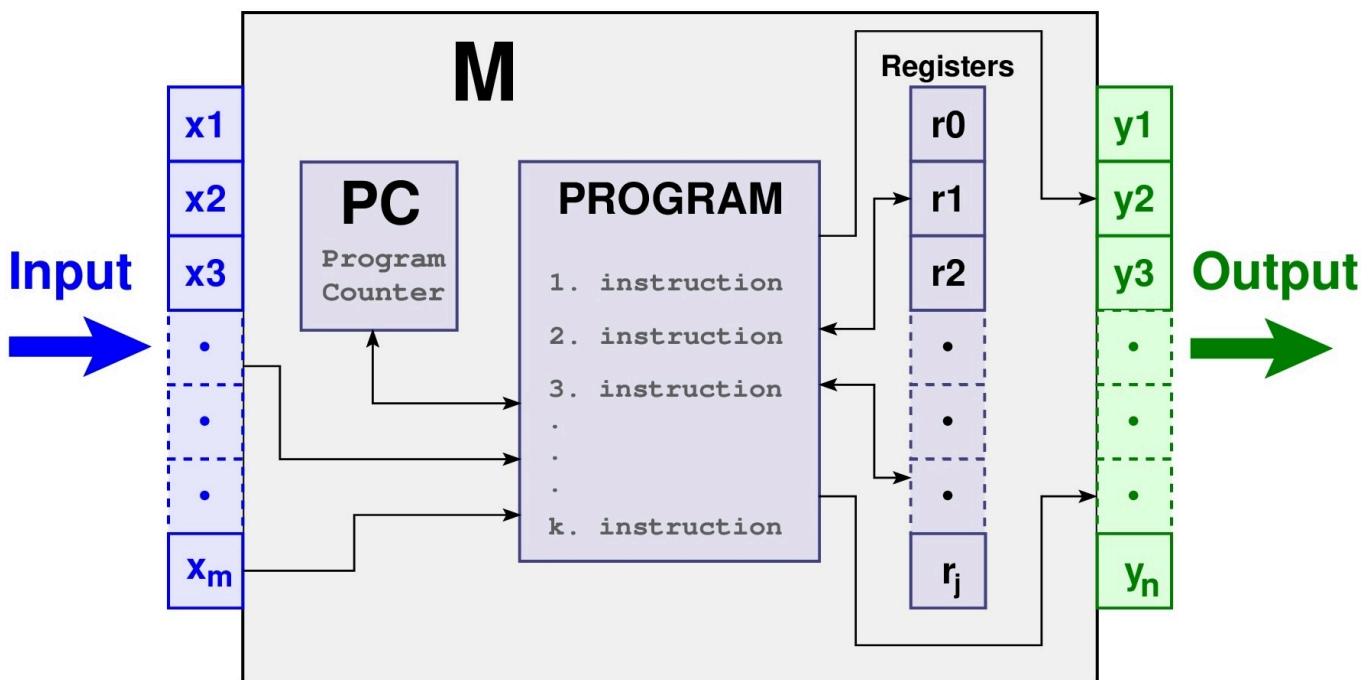
1. **Бесконечной ленты**, разделённой на ячейки (с символами).
2. **Головки**, которая может читать/записывать символы и двигаться влево/вправо.
3. **Набора правил** (программы), определяющих действия на основе текущего состояния и прочитанного символа.

Почему важна:

- Любой алгоритм можно реализовать на машине Тьюринга
 - Понятия "вычислимости" и "сложности" опираются на эту модель
- и еще немногой важной информации:

- Не может быть реализована на практике
- обладает полнотой по Тьюрингу(позволяет реализовать любой известный алгоритм)
- проблема остановки(вроде звучит примерно так: невозможно создать программу, которая всегда точно предскажет, завершится ли другая программа или будет работать бесконечно)
- данные и управление отделены
- ориентирована на реализацию

54)Что такое Random Access Machine? Каково её устройство и место сегодня? Какова её связь с машиной Тьюринга?



Random Access Machine - берем ленту из машины Тьюринга и заменяем ее адресуемой памятью с конечным или бесконечным количеством регистров. В итоге получаем то же самое(только можно читать и писать куда нам надо в память, а не последовательно как на ленте). Есть инпут, есть РС с помощью которого берем из памяти что нам надо, выполняем это действие, записываем в регистры и что то отдаём на аутпут. В целом все современные процессоры описываются этой random access machine(с небольшими оговорками)

а связана она с машиной Тьюринга тем, что обе модели формально эквивалентны и могут решать одни и те же задачи. Ну и Random Access Machine это своего рода математическая абстракция, но просто очень и очень приближенная к реальным процессорам.

55)Что такое функциональные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс?

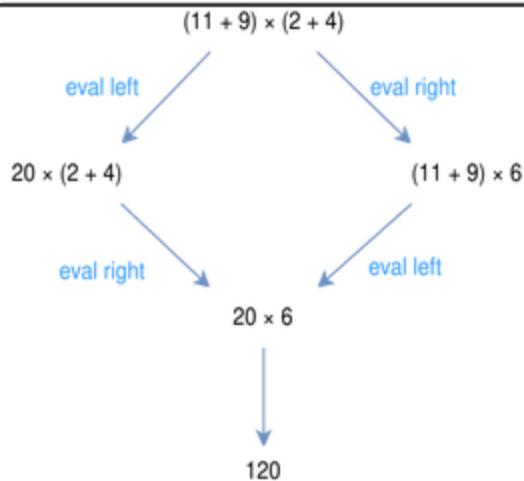
Функциональные модели вычислений - представление вычислительного процесса как совокупности символов и правил преобразований:

1. Арифметика
2. Лямба исчисления(полнота по Тьюрингу)
3. Комбинаторная логика
4. Общие рекурсивные функции
5. Абстрактные системы перезаписи?(Abstract rewriting systems)

Но есть такие еще приколы:

- Программа/данные - математический объект
- Возможна Тьюринг неполнота

Если в последовательных моделях вычислений, у нас есть какая то машина, которая шаг за шагом выполняет какие то действия, то в функциональных моделях мы больше работаем по математическим законам и правилам(на картинке).



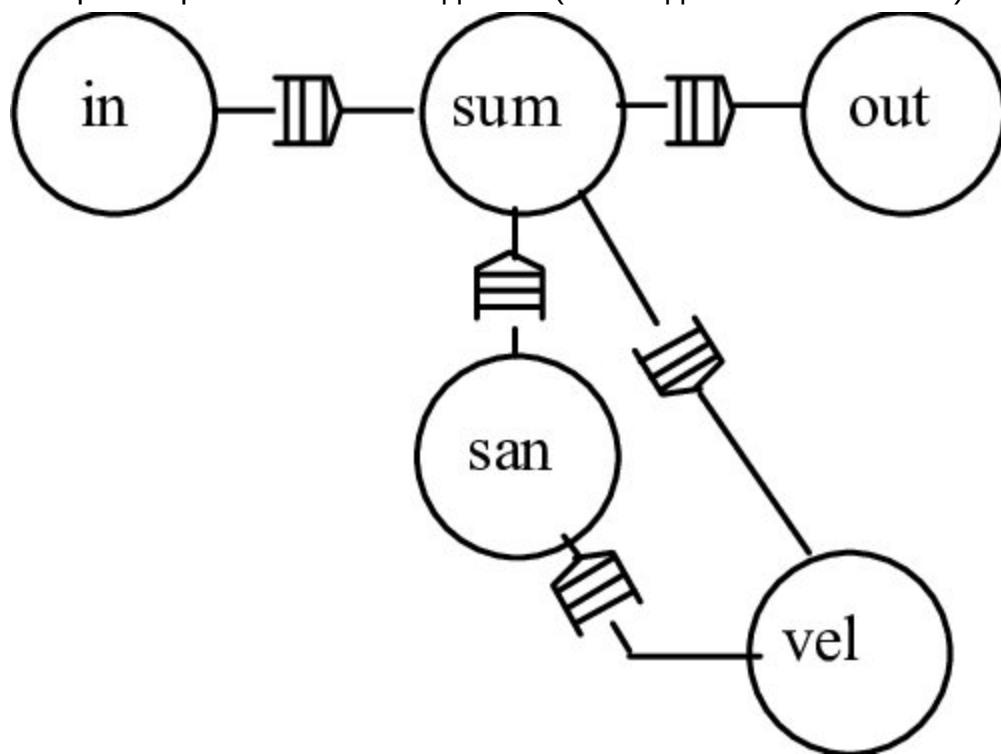
в RAM есть фундаментальное разделение между программной и данными, то в случае функциональных моделей данные и программы неразличимы, и мы получаем что то типа гомоморфность, что позволяет оперировать данными как программами и наоборот.

какая то душегубка, почему половина билетов это просто абстракции одного над другим, которое еще такое душное...., лан лан еще немного и будут нормальные вопросы

56)Что такое параллельные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс?

Параллельные модели вычислений - системы включающие несколько взаимодействующих процессов:

- Kahn process networks - односторонние каналы с неограниченными буферами(пример яп Gol)
- Акторные модели(отправка асинхронных сообщений в почтовые ящики, пример банковские транзакции да и в целом в распределенных системах)
- События в заданные моменты времени(процессы выполняются в заданные моменты времени реагируя на события)
- общая память(потоки/процессы шарят общие переменные)
- синхронизированные потоки данных(обмен данными по тактам)



такое написал мне дипсик

Как представляется вычислительный процесс?

- **Взаимодействие:** Через **сообщения**, **общую память** или **события**.
- **Параллелизм:**
 - **Физический:** Несколько ядер CPU/GPU.
 - **Логический:** Чередование потоков на одном ядре (coroutines, green threads).
- **Синхронизация:**
 - Блокировки (locks), барьеры, транзакционная память.

блаблаблаблаблаблабла

57)Что такое Model-Driven Engineering (MDE)? Чем цепочка трансформации в MDE отличается от языков высокого уровня? Приведите примеры.

MDE - подход к разработке программных компьютерных систем, который заключается в том, что не надо давать программисту использовать языки, а надо дать использовать набор моделей, в рамках которых он сможет описать ту прикладную задачу, те процессы с которыми он непосредственно работает

или другими словами:

1. разработка через моделирование предметной области
2. автоматическая генерация кода из моделей
3. повышение уровня абстракции при разработке
4. Сокращение разрыва между требованиями и реализацией
5. улучшение коммуникации между заинтересованными сторонами
6. снижение сложности разработки сложных систем

что говорить за цепочку трансформации я хз, пенской про это ничо не сказал, дипсик выдает буллшифт какой то, скип скип скип
примеры:

- что то типа no-code solution для android/ios
- case-технологии(набор инструментов и методов программной инженерии для проектирования ПО, обеспечивающих качество, надёжность и поддерживаемость продуктов)
- LabVIEW (графическая среда разработки, ускоряющая создание тестовых систем через интуитивное программирование и поддержку любого измерительного оборудования)
- и тд и тп подобные странные технологии

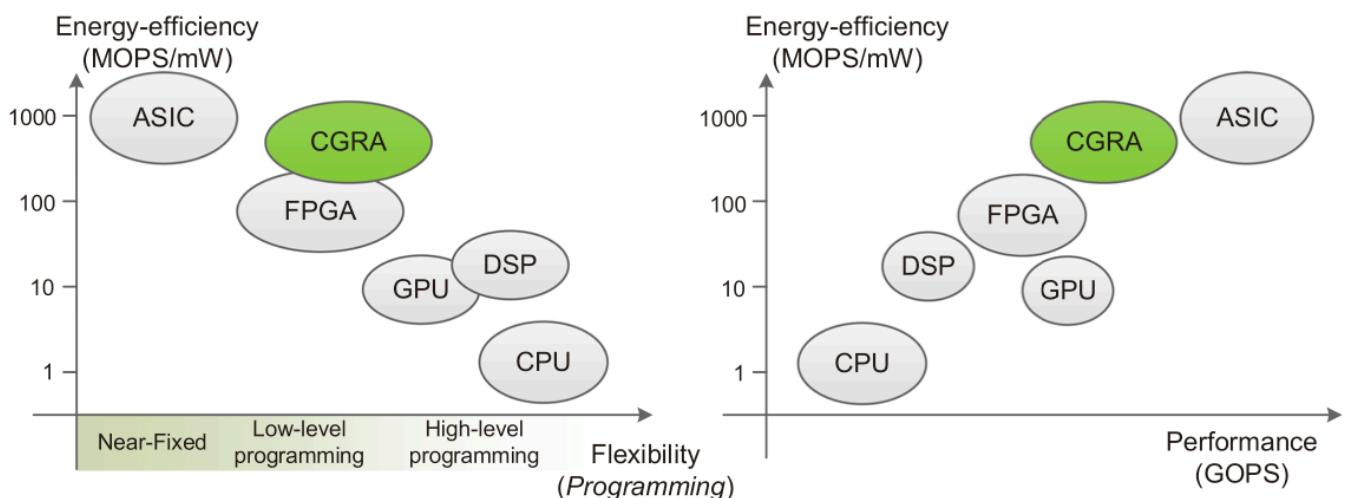
58)Что такое универсальный процессор? Каковы его особенности и свойства?

Универсальный процессор позволяет решать широкий круг задач, настройка которых производится после производства "по месту" или в run-time

НО пара примечаний важных:

- Универсальность != полнота по Тьюрингу
 - Теоретическая универсальность != практическая универсальность
 - Универсальность противоречит эффективности
 - Современный процессор - совокупность программного и аппаратного обеспечения
- Свойства:
- 2-этапное производство(Hardware и Software)
 - Полнота по Тьюрингу(необязательно но в общем случае должен)
 - Отсутствие "серьезных" ограничений "на объем" программы
 - Изменяемость ПО

59) В чём заключается противоречие между универсальностью и эффективностью в разных видах процессоров (СБИС, FPGA, CGRA, GPU, DSP, CPU)?



очень удачная пикча

слева(по горизонтали) - гибкость, программируемость?(буквальная цитата Пенского), по вертикали энергетическую эффективность.

ASIC - кремниевое производство(большие интегральные схемы)

CGRA, FPGA - способ программной реализации схем

GPU - графические процессора

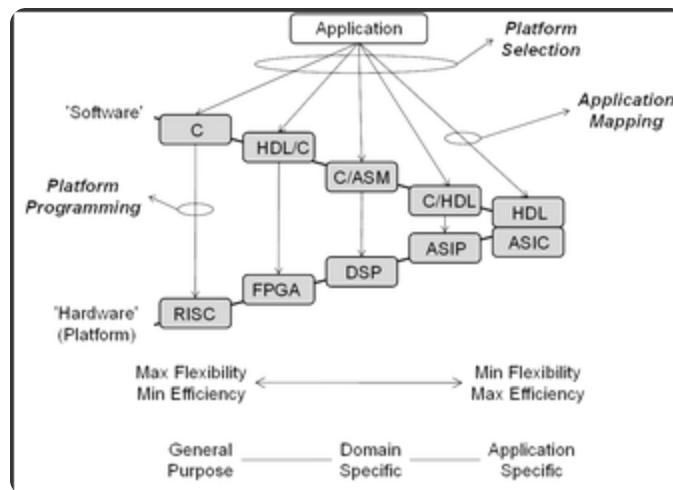
DSP - сигнальные процессора

ну и CPU.

В итоге с графика, уходя в универсальность - получаем высокое энергопотребление,

уходя в специализированное железо - получаем низкое потребление.
на правом графике показана производительность на единицу операции?

60)Как соотносятся платформы и языки описания вычислительного процесса? Как это связано с эффективностью и областью применения?



да тут примерно та же история, что и в прошлом вопросе, поэтому можно в целом ограничиться анализом графика

61)Что такое архитектура и микроархитектура процессора? В чём различие между ними?

Архитектура процессора - то, как видит компьютер программист. Определена набором команд(язык), местом нахождения operandов(регистры и память) и вычислительными механизмами(кеш, прерывания)

Микроархитектура процессора - соединение простейших цифровых элементов в логические блоки, предназначенные для выполнения команд определенной архитектуры.

Важные примечания:

1. Описывает, как в процессоре расположены и соединены друг с другом регистры, АЛУ, конечные автоматы, блоки памяти, интерфейсы ввода-вывода и т.п.
2. У каждой архитектуры может быть много микроархитектур, обеспечивающих разное соотношение производительности, цены, сложности, технической эстетики. Они смогут выполнять одни и те же программы.

3. Если ISA и микроархитектура отличаются на уровне MoC — обычно требуется уровень виртуализации (ПО, аппаратура, транслятор...).

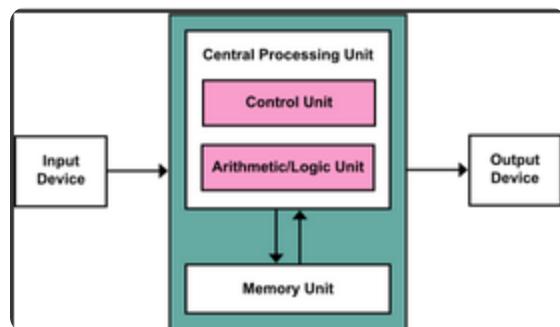
62)Что такое система команд и какова её роль в архитектуре процессоров? Что определяет система команд?

Система команд процессора(ISA) - абстрактная модель процессора, формирующая интерфейс взаимодействия между программным обеспечением и процессором.(каючи по простому набор инструкций процессора)

ISA определяет:

1. типа данных
2. модель памяти, система и метода адресации
3. набор инструкций
4. механизмы обработки прерываний и исключений
5. методы ввода и вывода

63)Что такое машина фон Неймана? Какова её связь с машиной Тьюринга? Каковы её ключевые принципы?



Машина Фон Неймана - базовая архитектура компьютера (принцип совместного хранения команд и данных в памяти компьютера)

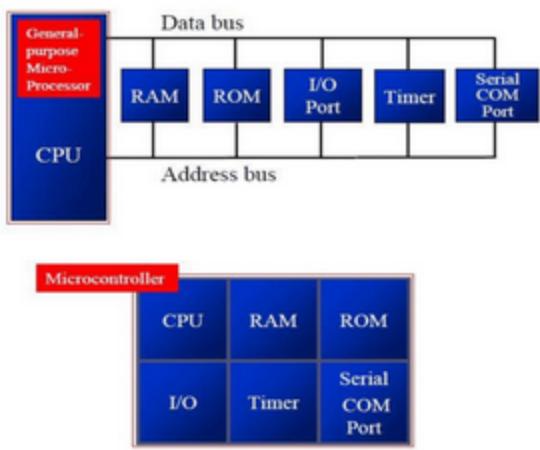
По сути своей, машина Фон Неймана = развитие машины Тьюринга, только с тем отличием, что лента заменена на RAM(Random Access Memory)

Основные принципы:

- Инструкции и данные объединены
- призвана быть максимально простой в реализации и производстве

- последовательное выполнение
- Использование двоичного кодирования
- Ячейки памяти имеют адреса(что внатуре)
- Возможность условного перехода

64)Что такое микропроцессор и микроконтроллер? В чём их отличия?



Микропроцессор - цифровая схема, которая выполняет операции с внешним источником данных (обычно памятью или потоком данных).

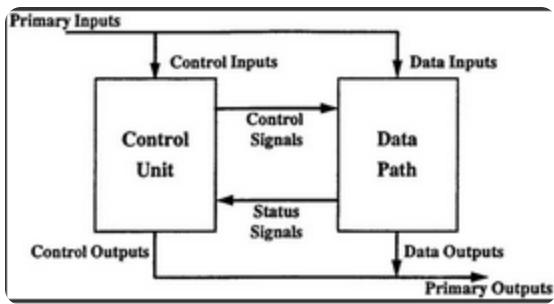
Микроконтроллер - микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает функции процессора и периферийных устройств, содержит ОЗУ и (или) ПЗУ.

В целом по картинке становятся тривиальными их отличия.

65)Что такое Control Unit и Data Path? Каковы их назначение и принципы взаимодействия?

Control Unit - компонент CPU, который управляет его работой. СУ обычно состоит из бинарного декодера для преобразования инструкций в управляющие и операционные сигналы, которые управляют работой других блоков(+ дословный перевод текста со слайда)

Datapath - АЛУ, набор регистров и внутреннюю шину(ы) СУ, которые позволяют данным передаваться между ними(datapath как раз таки и реализует всю эту машинерию по прикладному изменению данных)



по принципу взаимодействия тоже очень тривиально(особенно если делаешь 4 лабу)

66)Какие виды инструкций существуют в машине фон Неймана? Приведите примеры. Каковы принципы кодирования?

1. Работа с памятью: запись констант, копирование данных(память, порты I/O, регистры)
2. Арифметические, логические и битовые операции
3. Управляющие операции: безусловный, условный и косвенные переходы, вызов и возврат из подпрограмм
4. Инструкции сопроцессоров: загрузка и выгрузка данных, управление сопроцессором
Принцип кодирования тривиальный, можно пример risc-v привести, даже бэвм мне кажется сойдет за пример. карочи тут давайте сами от себя рассказывайте

67)Каковы особенности принстонской архитектуры? Каковы её достоинства и недостатки? Какова область применения?

Пристонская = Фон Неймана

Особенности я описывал уже выше

Достоинства:

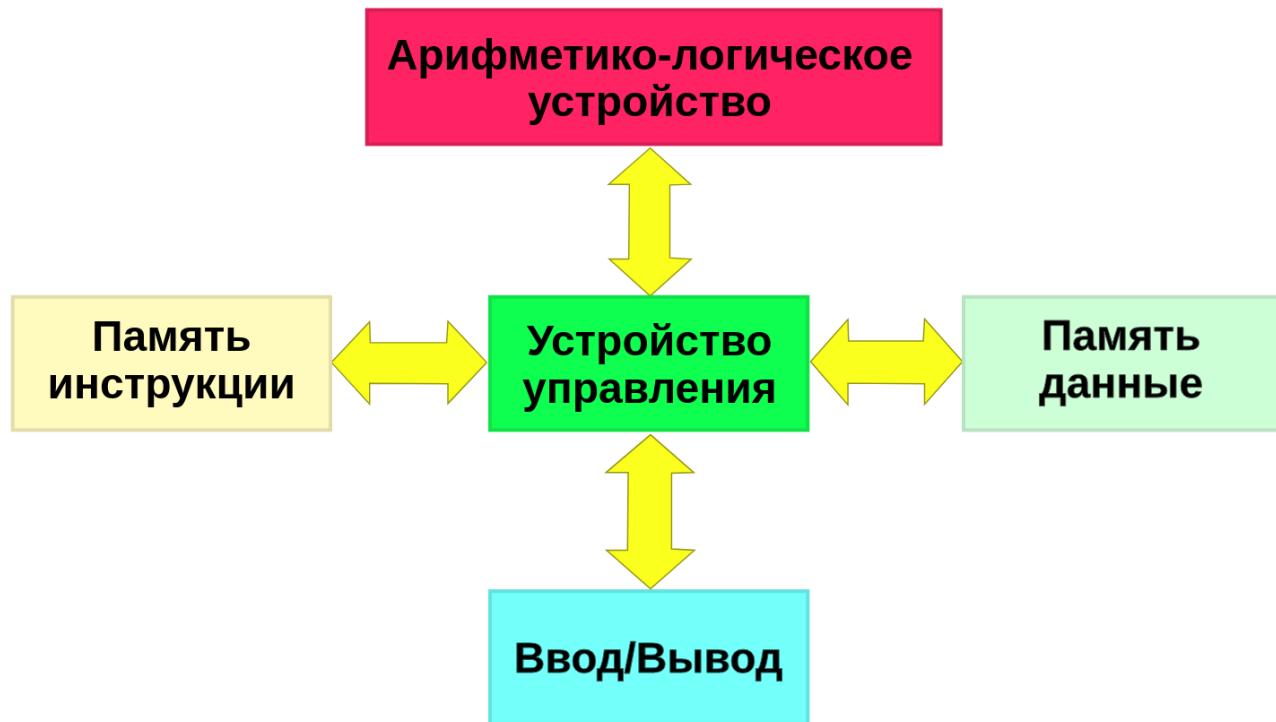
- Простота проектирования
- Универсальность
- Эффективное использование памяти
- Легкость программирования

Недостатки:

- Одна шина памяти - узкое горло
- Невозможно одновременно читать инструкцию и данные

- Проблема с безопасностью(никак не понять, что инструкция, а что данные)

68)Каковы особенности гарвардской архитектуры? Каковы её достоинства и недостатки? Какова область применения?



Особенности:

- Раздельная память команд и данных
- Параллельный доступ

Достоинства:

- Два физических канала между процессором и памятью
- Одновременный доступ к памяти
- Разная ширина машинного слова и адреса для данных и программ(оптимизация под решаемую задачу)
- Изоляция памяти инструкций

Недостатки:

- Сложность и стоимость реализации
- Изоляция инструкций и данных(запуск результата компиляции, указатели на функции)

69)Как организовано адресное пространство в гарвардской и принстонской архитектурах?

Пристонская архитектура - единное адресное пространство, код и данные в одной памяти и используют общую шину.

Гарвардская - раздельные адресные пространства, память команд и память данных, для каждой отдельные шины.

Принстонская - универсальность, Гарвардская - быстродействие и надежность

Вот еще вопрос интересный с лекции:

от чего зависит размерность машинных слов и адресных пространств? Что будет расти быстрее, адресное пространство инструкций или адресное пространство данных при решении задач?

Ответ: всё сильно зависит от задачи, которую мы решаем, в итоге мы сваливаемся к вопросу об информационной или управляющей.

Делаем архиватор - нужна память данных, есть фиксированный алгоритм, которые обрабатывает данные. В итоге объем данных будет расти быстрее.

В информационных системах - быстрее будет расти память данных, в управляющих - память команд.

70)Какие существуют варианты обхода ограничений гарвардской архитектуры, включая "Модифицированную гарвардскую архитектуру"?

Карочи есть несколько неприятных ограничений:

- фон неймановский процессор самодостаточен, один и тот же процессор и инструментальным и целевым. Мы можем сами генерировать программу и выполнить ее. Нет проблемы с расположением данных(из одного типа памяти в другую).

В итоге есть такие вариации гарвардской архитектуры:

1. Архитектура "Память инструкций как данные"
 - Реализуется возможность читать и писать данные в память инструкций. Позволяет генерировать и запускать машинный код
2. Архитектура "Память данных как инструкции"

- Реализует возможность запускать инструкции из памяти данных. Позволяет генерировать и запускать машинный код, но параллельный доступ ограничен

3. Модифицированная Гарвардская архитектура

- Доступ к памяти реализуется через независимые кеши(L1) для данных и программ, за счет чего, с точки зрения внутренней организации процессора, доступ реализован независимо, при этом канал между процессором и памятью один.

Вообще не особо выкупаю прикола первых двух вариаций, зачем они, если убивают все преимущества гарвардской архитектуры.

71)Что такое "аккумуляторная система команд"? Каковы основные элементы? Приведите примеры. Каковы достоинства и недостатки?

Аккумуляторная система команд - основные действия с альве используют специальный регистр для взятия данных от туда и для сохранения - аккумулятор

Основные элементы: аккумулятор, альве, память

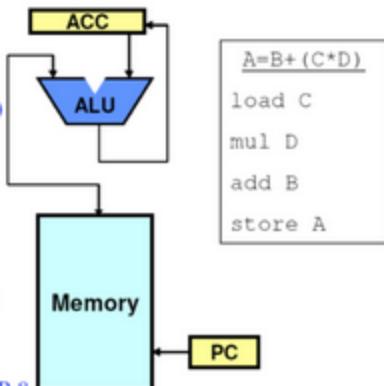
Достоинства:

- мало опкодов, мало занимает сама инструкция и фикс размер
- аппаратурная реализация тривиальная
- легкость программирования(тк как и правда тривиально)

Недостатки:

- низкая производительность
- сложно распараллелить такое(зависимость от одного регистра)
Как примеры:
 - Какие нибудь старые советские ЭВМ(хз я не знаю их названий)
 - БЭВМ!!!!!!!
 - Ну и на самом слайде: IBM 7090, DEC PDP-8(как примеры старых эвм), а сегодня DSP(digital signal processor) архитектура

- Single register (accumulator)
- Instructions
 - ALU ($\text{Acc} \leftarrow \text{Acc} + *M$)
 - Load to accumulator ($\text{Acc} \leftarrow *M$)
 - Store from accumulator ($*M \leftarrow \text{Acc}$)
- Instruction operands
 - One explicit (memory address)
 - One implicit (accumulator)
- Attributes:
 - Short instructions
 - Minimal internal state; simple design
 - Many loads and stores
- Examples:
 - Early machines: IBM 7090, DEC PDP-8
 - Today: DSP architectures



A=B+ (C*D)
load C
mul D
add B
store A

72)Что такое Register-to-Memory системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки?

R/M система команд - команды напрямую могут работать с данными в памяти, используя регистры только для промежуточных операций. (логическое развитие аккумуляторной архитектуры, просто большее количество регистров, что аппаратно усложняет, но устраняет проблему с чтением и записью в один и тот же регистр)
почти все современные CISC процессора построены на этом принципе
Основные элементы: регистры общего назначения, алу, память

Достоинства:

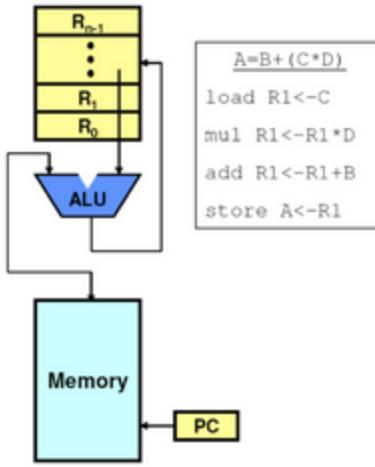
- Плотное кодирование
- небольшое количество инструкций

Недостатки:

- Результат уничтожает operand
- Переменная длина инструкций
- Количество тактов на инструкцию отличается
- Ну и тоже проблема с конвейеризацией

Примеры: IBM 360/370

- One memory address in ALU ops
- Typically 2-operand ALU ops
- Advantages
 - Small instruction count
 - Dense encoding
- Disadvantages
 - Result destroys an operand
 - Instruction length varies
 - Clocks per instruction varies
 - Harder to pipeline
- Examples
 - IBM 360/370, VAX



73)Что такое Register-to-Register системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки?

R/R системы - арифметико логические операции выполняются только между регистрами, а доступ к памяти только через специальные инструкции по типу load/store

Достоинства:

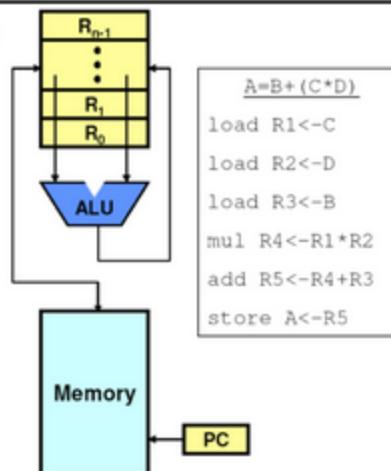
- простые инструкции фиксированной длины
- легко конвертируются

Недостатки:

- большее количество инструкций

Примеры: CDC6600, большинство risc процессоров

- No memory addresses in ALU ops
- Typically 3-operand ALU ops
 - Bigger encoding, but simplifies register allocation
- Advantages
 - Simple fixed-length instructions
 - Easily pipelined
- Disadvantages
 - Higher instruction count
- Examples
 - CDC6600, CRAY-1, most RISCs



74)Что такое Memory-to-Memory системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки?

M/M системы - (скорее гипотетические) - арифметико логические операции выполняются с операндами из памяти

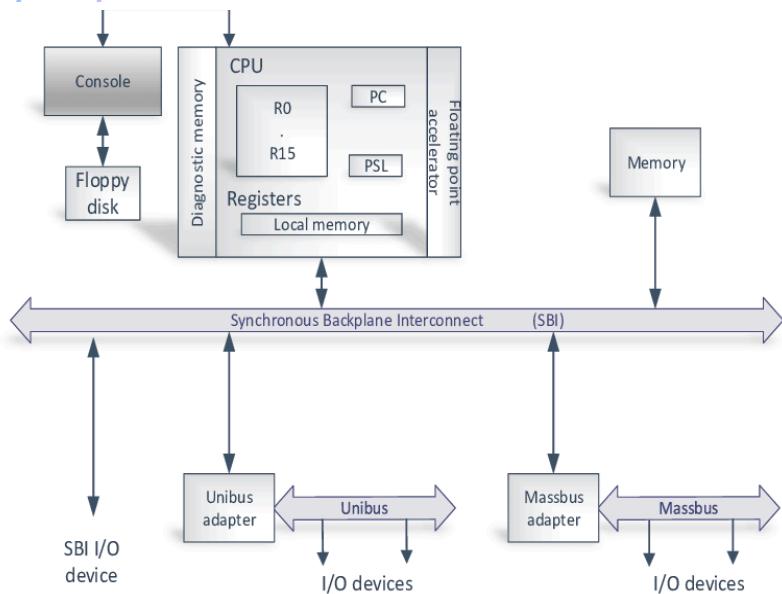
Достоинства:

- нет потерь регистра?(no register wastage idk)
- наименьшее количество инструкций

Недостатки:

- Большое количество вариаций в длине инструкции
- Большое количество вариаций тактов на инструкцию
- Огромный трафик памяти

Пример: VAX



75)Что такое стековые системы команд? Каковы основные элементы? Что такое неявная адресация аргументов? Каковы их достоинства и недостатки?

Стековые системы - вместо набора регистров - стековая память, аргументы не упоминаем в инструкциях, сами команды оперируют вершиной стека. В итоге(из за подобного подхода) мы естественным образом получаем процедуры. В итоге операции

для работы с алу не используют операнды.

Достоинства:

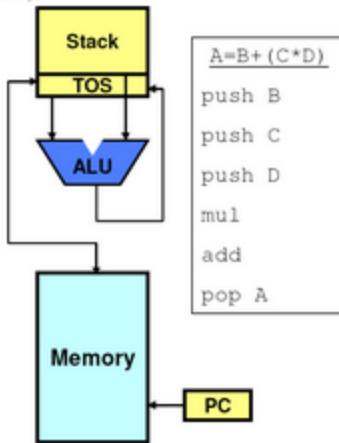
- Короткие инструкции
- Аппаратурная реализация простая
- Легко написать компилятор под такую систему

Недостатки:

- Очень быстро разваливается, когда надо ходить по стеку хитрым образом, те надо постоянно дублировать, тасовать данные на стеке, на что уходит лишнее время
- Такие инструкции сложно оптимизировать
- Ну и есть ограничения в виде размера стека

Примеры: раньше - Burroughs B5500/6500, а сейчас есть java вм(фу жава)

- Stack: First-In Last-Out data structure (FILO)
- Instruction operands
 - None for ALU operations
 - One for push/pop
- Advantages:
 - Short instructions
 - Compiler is easy to write
- Disadvantages:
 - Code is inefficient
 - Fix: random access to stacked values
 - Stack size & access latency
 - Fix : register file or cache for top entries
- Examples
 - 60s: Burroughs B5500/6500, HP 3000/70
 - Today: Java VM



вот тут в целом удобно различать все эти системы команд, их тут удобно сравнить

Система команд и линейная функции

```
load ACC ← A      ; acc
mul ACC ← * B    ; 1 operands
add ACC ← + C
store Y ← ACC     ;           (1)
```

```
load R1 ← A      ; reg-to-mem
mul R1 ← R1 * B  ; 1 operands
add R1 ← R1 + C
store Y ← R1      ;           (2)
```

```
load R1 ← A      ; reg-to-reg
load R2 ← X      ; 2 operands
load R3 ← B
mul R4 ← R1 * R2
add R5 ← R4 + R3
store Y ← R5      ;           (3)
```

```
load R1 ← A      ; reg-to-reg
load R2 ← X      ; 3 operands
load R3 ← B
lnf R4 ← R1 * R2 + R3
store Y ← R4      ;           (4)
```

$$Y = A * X + B$$

```
mul R1 ← A * B    ; reg-to-mem
add Y ← R1 + C    ; 2 operands (5)
```

```
lfn Y ← A * B + C ; mem-to-mem
; 3 operands (6)
```

```
A @ B @ *      \ stack, 0 operands (7)
C @ + Y !      \ @ - read, ! - write
```

Что лучше?

- (1) – начало
- CISC (2, 4, 5, 6)
- RISC (3)
- Stack (7)

§

76) Какие виды стеков есть в стековых процессорах? Их количества?

Стек данных(1 - 2), стек возврата(1, вызова)

77) Каковы основные принципы описания алгоритмов для стековых процессоров? Что такое Forth и High- level language computer?

Forth - императивный язык программирования на основе стека.

Особенности: структурное программирование, отражение (возможность исследовать и изменять структуру программы во время выполнения), последовательное программирование и расширяемость(новые команды)

Синтаксис - обратная польская нотация(почитайте сами если не знаете).

High-level language computer: - яп дает функции высокоуровневых япов

- процедуры
 - автоматическая память
 - рекурсия
 - простой компилятор
 - выражения по типу: $X=(A+B)(C+D)$
- вот достоинства стековых процессоров:
- High-level language computing
 - Простая система команд, высокая производительность
 - Отлично работает с кешами и потоками
- а вот недостатки:
- Эффективность страдает при большом количестве данных на стеке
 - сложно положить динамические структуры данных на стек
 - Параллелизм уровня инструкций
 - Сильно отличается

78)Что такое CISC процессоры? Каковы их ключевые особенности, достоинства и недостатки?

CISC(Complex Instruction Set Computer) - архитектура компьютера, в которой отдельные инструкции могут выполнять несколько низкоуровневых операций(загрузку из памяти, арифметическую операцию и сохранение в памяти) или способны выполнять многошаговые операции или режимы адресации в рамках отдельных инструкций.

x86 - cisc процессор

это было строгое определение, если говорить проще, то:

идея вот в чем: если у нас есть процессор, есть его устройства то давайте будем управлять ими наиболее эффективным способом, для этого будем усложнять систему команд таким образом, чтобы наш процессор делал как можно меньше ненужных телодвижений.

В итоге получаем следующие **причины появления:**

- были только низкоуровневые япы
- Разнообразие архитектур
- Неразвитость компиляторов

- Удобство программирования
- Высокая производительность
- Минимизация объема программы
- Минимизация накладных расходов

Особенности:

- Сложные инструкции(одна инструкция может выполнять несколько операций)
- Разнообразие форматов инструкций
- Микрокод
- Мало регистров общего назначения
- Прямая работа с памятью

буквально можно прочитать выше билет про Register-to-Memory системы команд :3

Достоинства:

- Эффективность для специфичных задач
- Компактность набора инструкций уменьшает размер программы и уменьшает количество обращений к памяти.
- Набор инструкций включает поддержку программного обеспечения высокого уровня.

Недостатки:

- Сложная система команд(использование, анализ)
- Сложное устройство процессора и СУ
- Сложно генерировать эффективный машинный код

79)Какие виды адресации аргументов инструкции в CISC процессорах?

ващие любые

1. регистровые(когда operand находится в регистре)

```
mov eax, ebx
```

2. Непосредственные

```
add eax, 42
```

3. Прямая адресация памяти

```
mov eax, [0x1234]
```

4. косвенная

```
mov eax, [ebx]
```

5. через смещение

```
mov eax, [ebx + 10]
```

80) В каких случаях CISC инструкции эффективны? Приведите примеры.

cics процессора выигрывают там, где одна сложная команда заменяет несколько простых будь то операции со строками, математические вычисления, системные вызовы а вот примеры:

```
rep movsb ; копирование блока памяти
```

```
aam ; (ASCII Adjust after Multiplication) иначе коррекция BCD-чисел
```

81) В чём выражается комплексность систем команд CISC?

Комплексность в CISC процессорах - принцип построения системы команд, при котором:

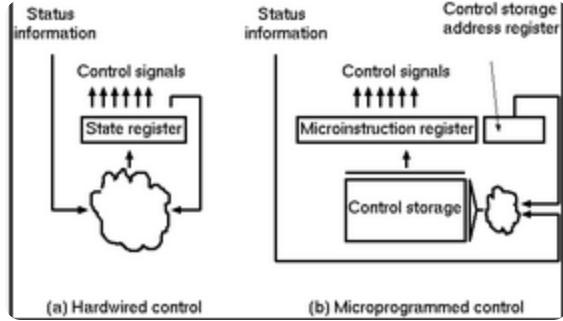
1. одна инструкция выполняет сложную операцию, заменяя несколько других
 2. Поддерживаются разные режимы адресации
 3. Инструкции имеют переменную длину и формат
- ну и после понятия комплексности, надо просто собрать базовую инфу из билетов выше по CISC процессорам.

82) Какие существуют подходы к реализации Control Unit? Достоинства и недостатки Hardwired и микропрограммного управления?

Существуют следующие подходы к реализации CU:

1. Hardwired: при помощи аппаратных комбинационных схем, декодирующих инструкции в последовательности сигналов
2. Microcoded: при помощи исполнения микропрограммы, реализующей необходимые функции

Микропрограмма - программа, реализующая набор инструкций процессора.



Достоинства hardwired:

- Высокая скорость работы
- Предсказуемость

Недостатки hardwired:

- сложность модификации
- Ограниченнная сложность ISA
- Высокая стоимость разработки

Достоинства microcoded:

- Простота реализации сложных ISA(cisc)
- "Программирование" системы команд
- Доступ к микрокоду для программиста(очень спорный пункт, кому нужно смотреть в микрокод, надо тогда воротить всю внутренку процессора)
- Генерация ISA под задачу(сократить объем, повысить эффективность)

Недостатки microcoded:

- Хранение микрокода в процессоре
- CISC долго учить(буквально инфа со слайда, но тоже спорный очень момент)
- Разнообразие архитектур -> проблемы инструментария
- Разнообразные команды(форматы, размеры, длительность, доступ), что усложняет оптимизацию процессора и инструментарий

- Микрокод привносит все проблемы программирования: сложность, отладка, методы....

83)Что такое микропрограммное управление? Что такое микроинструкция? В чём отличия инструкций и микроинструкций?

Микропрограммное управление - способ реализации управления процессором, при котором каждая инструкция выполняется как последовательность микроинструкций.
Микроинструкция - элементарная команда, управляющая работой компонентов процессора(открытие/закрытие защелок, чтение/запись и тд)

важное примечание, пенской сказал, что любит спрашивать вопрос про разницу инструкции и микроинструкции, вот его ответ:
инструкция - это то, с чем работает программист, микроинструкция или список микроинструкций - это то, во что должна быть странслирована инструкция в момент исполнения. Микроинструкции выполняются за такт, микроинструкции не выполняются параллельно. **ЭТО БУКВАЛЬНО ПРЯМАЯ ЦИТАТА, ЗНАЧИТ ОТВЕТ 100% правильный.**

84)Как можно оптимизировать инструкции при помощи микропрограммирования? Приведите пример.

todo

idk

85)Какие существуют подходы к поиску первой микроинструкции для заданной инструкции?

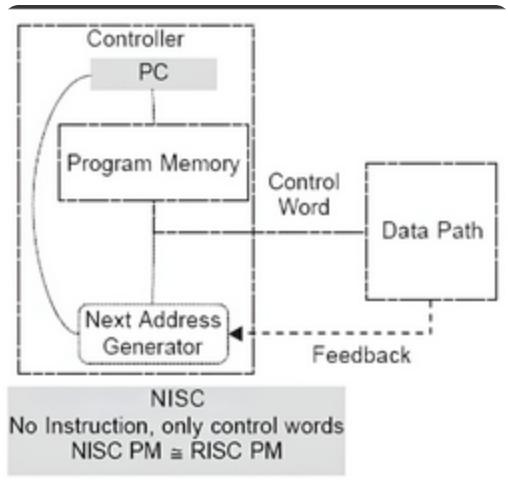
1. Алгоритмически: сопоставление инструкции с условием + условный переход в рамках микропрограммы
2. Отображение кода операции на адрес микроинструкции:
 - Прямое отображение - опкод выступает в роли адреса. К примеру опкод 0101 - 0x5 адрес микроинструкции
 - Непрямое отображение - требует использования look-up-table для поиска нужного адреса(LUT - таблица, где каждому опкоду сопоставлен нужный адрес)

86)Что такое архитектура NISC? Каковы область её применения, достоинства и недостатки?

NISC(No Instruction Set Computing) - архитектура процессора без фиксированного набора инструкций, где управление выполняется напрямую микрокодом или конфигурационными битами, а не машинными командами

или если говорить проще

NISC это как взять принцип микропрограммного управления и сделать его главным. Зачем нам инструкции, если процессор управляется микроинструкциями, а не инструкциями. Пускай компилятор будет выдавать на выход трассы из микроинструкций, которые мы будем напрямую гнать в процессор.



Достоинства:

- Упрощение аппаратуры
- Максимальная эффективность программного управления
- Нет ISA - нет проблем ее проектирования

Недостатки:

- Невозможность бинарной совместимости
- Низкая плотность машинного кода

Использование:

- Применяется в ускорителях, в высокоуровневом синтезе (HLS), спец вычислителях
- Проект NITTA - CGRA процессор, где вычислительные блоки управляются в стиле NISC.

87)Каковы источники роста производительности процессоров?

Источники роста производительности:

- Частота
- Специализация системы команд, аппаратуры
- Параллелизм уровня бит, инструкций, задач
- Адаптация структуры вычислителя под задачу и параллелизм
- Динамическая адаптация
Препятствия на его пути - законы из билета ниже

88)Какие законы и ограничения влияют на производительность современных процессоров?

1. Закон Мура
2. Закон Амдала
3. Закон Деннарда
4. Power Wall
5. Memory Wall

89)Что представляют собой закон Мура и закон Деннарда?

Закон Мура звучит примерно так, что каждые ~2 года количество транзисторов на кристалле увеличивается вдвое, а производительность растет в 1.5 раз. Сегодня он уже не работает, так как простое наращивание транзисторов на чипе уже практически ничего не дает. Можно конечно сделать такие чипы, но проблема в том, что мы не сможем их загрузить. Самый простой способ загрузить большое количество ядер - выполнять задачи в параллель, и в итоге мы наталкиваемся на закон Амдала.

Закон Деннарда - уменьшая размеры транзистора и повышая тактовую частоту процессора, возможно пропорционально повышать производительность. Можно уменьшать размер чипа и засчет этого получать большую производительность(во первых электрическим сигналам меньше пространства засчет этого можно поднимать частоту, во вторых уменьшение чипов позволяет уменьшить энергопотребление)
но есть но!

- Дорого + физически невозможно
- чем меньше транзистор -> больше утечки тока

- Powerwall и Dark Silicon.

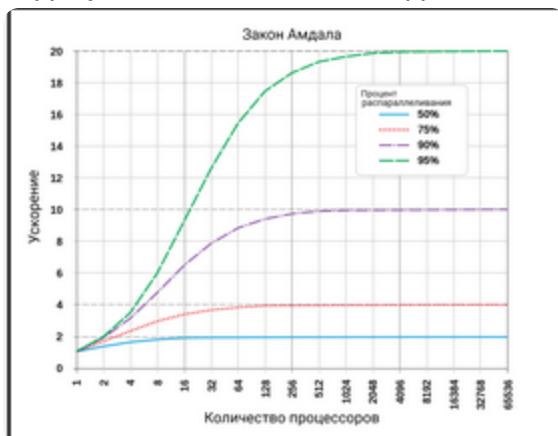
90)Что представляет собой закон Амдала? На каком уровне параллелизма он работает?

Закон Амдала - по сути это ограничение ускорения программы при параллельном выполнении из за последовательной части задачи.

вот на картинке супер показано

В любой задаче есть часть работы, которая так или иначе будет сделана последовательно, и получается, если брать тот процент задачи, который можно распараллелить между разными исполнителями. Если допустим взять 50% распараллеливания задачи, то после 8 ядер уже это не имеет особого смысла, там ускорение прям минимальное.

Да и в целом увеличение ядер вдвое это дорого, так как нужны обвязки управляющие, системы кешей. Да и в целом переключение задач тоже тратит ресурсы, поэтому бить задачу на меньшие не всегда есть смысл.



91)Что такое Power Wall и тёмный кремний?

Power wall в чем проблема, если есть 100 ватт энергии, которые тратятся на работу чипа, уменьшаем чип -> те же 100 ватт должны рассеяться на меньшей площади, из за этого локальное излучение тепла улетает в небо(карочи большим становится). В итоге мы буквально не можем развеять такое тепло

Dark Silicone - это такая часть чипа, которая не используется из за ограничений энергопотребления и нагрева. Специально на чипе размечается такая площадь, на которую не отводится никакая задача, чтобы тепло просто эффективнее рассеивалось. (есть еще паттерны, с которыми будет эффективнее рассеиваться тепло)

92)Какие подходы существуют к решению проблемы Memory Wall в рамках процессоров семейства фон Неймана?

Memory wall проблема заключается в том, что есть разрыв между скоростью выполнения инструкций процессором и скоростью доступа к памяти. Раньше этого разрыва не было(в 80-х, условная эпоха CISC процессоров и memory-to-memory), но со временем процессора стали ускоряться сильно быстрее чем память, в итоге получаем мы можем создать довольно быстрый вычислитель, но не сможем загрузить достаточное количество данных. В итоге получаем, что мы как бы можем сделать больше ядер, процессоры сделать быстрее, но есть ли в этом смысл когда мы не можем загрузить необходимые данные

93)Какие подходы существуют к решению проблемы Memory Wall в рамках ASIC и CGRA?

TODO

94)Что такое RISC? Каковы особенности и предпосылки появления (ПО и аппаратура)?

RISC (Reduced Instruction Set Computer) - подход к проектированию процессоров, где быстродействие увеличивается за счет простого кодирования упрощенного набора инструкций(Ускорение выполнения ограниченного количества инструкций).

Предпосылки:

- Многие из функциональных особенностей традиционных ЦПУ просто игнорировались(условно говоря ускорение сложных операций, которые редко используются не дает никакого буста для общей производительности процессора)
- Появились языки высокого уровня + нормальные компиляторы
- Единый формат инструкций(Проблема CISC - мпзу занимает кучу памяти, в котором кодируется как надо вычислять инструкции, что занимает площадь, потребляет энергию. Так же сделаем проще, в машинное слово затолкаем все возможные инструкции)
- Место памяти микрокоманд и декодера можно использовать для регистров и кеша.
- Оптимизация малого количества однообразных команд

- Параллелизм уровня инструкций, pipeline
даже раньше для cisc процессора требовались целые компании и куча времени, а сейчас risc процессор могут написать два аспиранта за неделю. И такой процессор будет производительней за счет ускорения частых инструкций.

95)Как сравниваются CISC и RISC с точки зрения архитектуры?

CISC - register to memory

RISC - register to register

в cisc большие сложные инструкции, с переменной длиной

в risc упрощенный набор инструкций, с фиксированной длиной + ограничены возможности команд для работы с памятью(нет операций вида read-modify-write)

да и в целом вся база про register-to-memory и register-to-register можно навалить ну и про параллелизм, что в cisc с ним проблемы, в risc же иначе(далее будет про конвейризацию)

96)Как сравниваются CISC и RISC с точки зрения микро-архитектуры?

В cisc сложные инструкции декодируются в микрооперации + в cisc есть аппаратно заложенная логика для сложных инструкций

в risc же нет микроархитектуры, так как команда, поступающая в CPU уже разделена по полям и не требует дополнительной дешифрации и каждое действие выполняется в 1 такт(однотактное выполнение большинства инструкций)

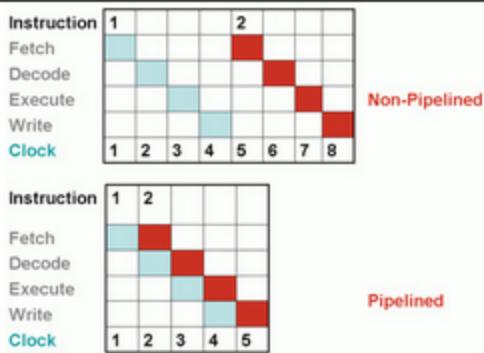
97)Каковы особенности кодирования инструкций при сравнении CISC и RISC?

в cisc переменная длина и плотный код, 1 инструкция = несколько операций высокого уровня.

в risc фиксированная длина, машинное слово четко разделено на поля.

98) Каковы принципы конвейерного исполнения инструкций? Как это влияет на загрузку и производительность?

как только мы получили все инструкции одинакового размера и возможность загружать 1 инструкцию каждый такт => можешь исполнять инструкции не последовательно, а можем шаги выполнения инструкции(fetch, decode, execute, write) вытянуть в конвейер



Конвейеризация (pipelining) — это метод выполнения инструкций, при котором различные этапы обработки команды выполняются **параллельно** для разных инструкций в последовательности.

Вообще, на картинке очень понятно должно стать, как это примерно работает

Вот как построить конвейер:

1. Выделить стадии выполнения команд;
 2. организовать внутренние структуры процессора так, чтобы:
 - у процессора был входной (поступают команды) и выходной конец (команды "покидают" процессор);
 - структура процессора должна соответствовать стадиям выполнения команд;
 - сегменты связаны регистрами, комбинационные схемы сбалансированы;
 - все части процессора управляются одним тактовым сигналом;
 3. загружать в процессор команды каждый такт;
 4. получать результаты выполнения команд каждый такт;
 5. разрешать конфликты параллельно выполняемых команд.
- в итоге мы получаем такое, что мы можем получать каждый такт несколько выполняемых инструкций(их количество равно количеству стадий процессора, в risc их 5, см. ниже)

99) Каковы типовые стадии RISC процессора?

1. **Instruction Fetch**. Чтение инструкции по счётчику команд.
2. **Instruction Decode**. Декодировать инструкцию и считать регистры.
3. **Instruction Execute**. Операций изменения данных.
4. **Memory Access**. Чтение/запись операндов из памяти/в память.
5. **Write Back**. Запись результата в регистры

100)Какие проблемы и архитектурные ограничения связаны с конвейеризацией?

1. Структурные конфликты:
 - Конфликт из за ресурсов. Аппаратура не позволяет выполнить все возможные комбинации инструкций
 - пример: одновременный доступ к единной памяти
2. Конфликты по данным(все они связаны с тем, что если мы выполняем несколько инструкций на разных стадиях параллельно, то может возникнуть ситуация, когда одна инструкция будет требовать данные от другой):
 - read after write
 - write after read(проблема перестановок из за оптимизации компиляторов)
 - write after write(проблема с кешами)
 - read after read(не проблема выше)
3. Конфликты по управлению(конфликт из за операций условного и/или безусловного перехода. Проблема в том, что в конвейер загружены команды, которые не должны быть исполнены):

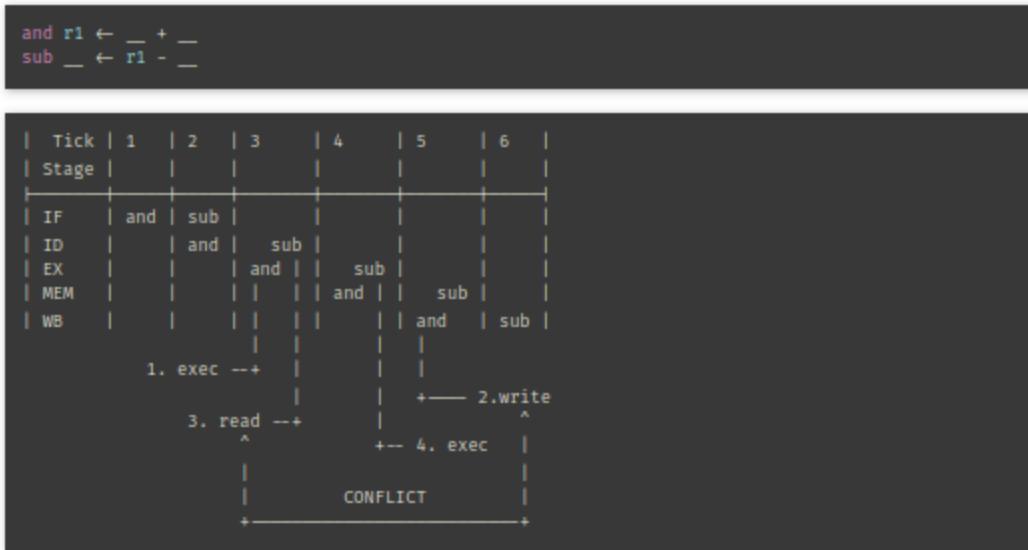
101)Как разрешаются структурные конфликты в конвейере?

Варианты полного решения проблемы:

- Гарвардская архитектура
- Двухпортовая память
- пузырек в контейнере(описание ниже)
итого надо просто обеспечить два канала для данных и для памяти.

102)Как разрешаются конфликты по данным в конвейере?

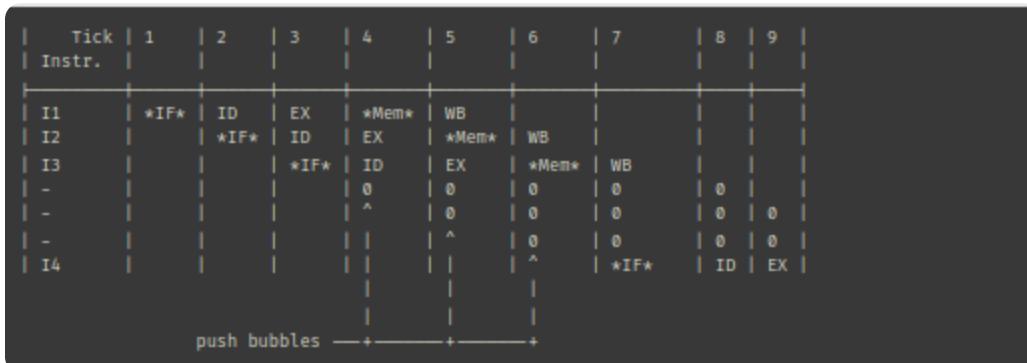
RAW – Read after Write (Data-dependency)



вот пример конфликта.

1. Исполнение не по порядку(компилятор/процессор)
2. Переименования регистров. Если зависимость ложная то запись может быть переназначена на другой регистр(пример WAW)
3. Пузырек
4. Проброс операндов между стадиями процессора, минуя регистровый файл

103)Как происходит разрешение конфликтов при помощи пузырька в конвейере?



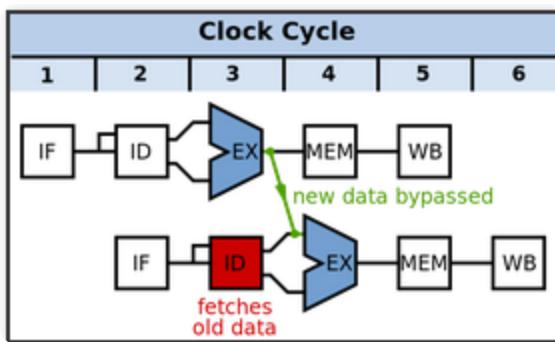
в рамках процессора надо сделать так, чтобы вместо выполнения он встал на условный холд, пока не выполнит текущие

буквально отказываем процессору в том, чтобы он получил новую инструкцию в результате, пока не закончатся все доступы к памяти(реализована схемотехникой) и каждый запущенный пузырек должен пройти все стадии конвейера

0 - пустая операция:

- занимает конвейер
 - не выполняет никаких действий
- это просто в реализации, но снижает эффективность

104)Как происходит разрешение конфликтов при помощи проброса операндов в конвейере?



на картинке показано как это выглядит примерно:

вместо записи в память, мы делаем петлю обратной связи, которая заставит процессор вместо прокидывания данных дальше, прокинуть их назад в выход аргумента предыдущей стадии.

Есть еще вариант, когда чтение из памяти, тогда можно сделать гибридный вариант: добавить сюда еще пузырек.

105)Какие существуют способы разрешения конфликтов по управлению во время компиляции?

предсказатели переходов(branch prediction)

106)Какие существуют способы разрешения конфликтов по управлению во время исполнения?

пузырек, сброс конвейера, минимизация количества условных переходов

107)Какие существуют способы предотвращения конфликтов по управлению через branch prediction? Что такое статические предсказатели переходов?

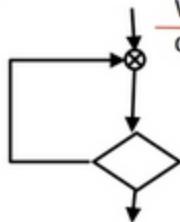
Branch Predictor работает за счет:

1)Эвристика, те статически, без запуска программы просто гляда на машинный код угадать что будет происходить. Есть два варианта джампов вперед и назад. Обычно джамп назад - почти всегда цикл, джамп вперед - обычный if statement. Отсюда можно сделать вывод, что для любого бренча, в случае прыжка назад, всегда считать, что мы его совершили и тогда наш конвейер будет работать эффективнее. Для джампов вперед вообще пох, но почему я не понял

Overall probability a branch is taken is ~60-70% but:

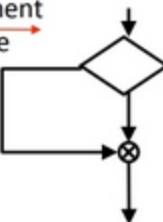
What C++ statement
does this look like

backward
90%



What C++ statement
does this look like

forward
50%



108)Что представляют собой динамические предсказатели переходов в конвейере?

109)Как влияют различные виды конфликтов на производительность конвейера?

Конфликты (hazards) в конвейерном процессоре нарушают его плавную работу, заставляя конвейер простаивать или выполнять лишние операции. Это снижает эффективность исполнения инструкций и общую производительность.

конвейер может останавливаться, простаивать в ожидании данных
да и в целом конфликты снижают реальную производительность.

110)Как связаны конвейерное исполнение и спекулятивное исполнение инструкций?

1. Конвейерное исполнение (Pipeline Execution)

Конвейеризация разделяет выполнение инструкции на несколько этапов (например, выборка, декодирование, выполнение, запись результата). Пока одна инструкция находится на этапе выполнения, следующая уже может декодироваться, а третья — выбираться из памяти. Это позволяет обрабатывать несколько инструкций одновременно, увеличивая пропускную способность.

Проблема: Если встречается условный переход (branch), процессор не знает, какая инструкция будет следующей, пока условие не вычислится. Это приводит к **простоем конвейера** (pipeline stalls).

2. Спекулятивное исполнение (Speculative Execution)

Чтобы избежать простоев, процессор **предсказывает** результат условного перехода (branch prediction) и начинает выполнять инструкции **спекулятивно** (т.е. в предположении, что предсказание верно).

- Если предсказание верное, спекулятивно выполненные инструкции сразу используются, и конвейер работает без задержек.
- Если предсказание неверное, результаты отбрасываются (брос конвейера — pipeline flush), и выполнение продолжается с правильной ветки.

Связь между конвейером и спекулятивным исполнением

- **Конвейер создаёт проблему** (зависимость от условных переходов → простои).
- **Спекулятивное исполнение решает её** (предсказание ветвлений + выполнение "вперёд" → минимизация простоев).

Таким образом, спекулятивное выполнение **улучшает эффективность конвейера**, позволяя ему работать с высокой загрузкой даже при наличии условных переходов.

overall по конвейерам

Преимущества:

- повышение производительности и уровня утилизации ресурсов.

Недостатки/проблемы:

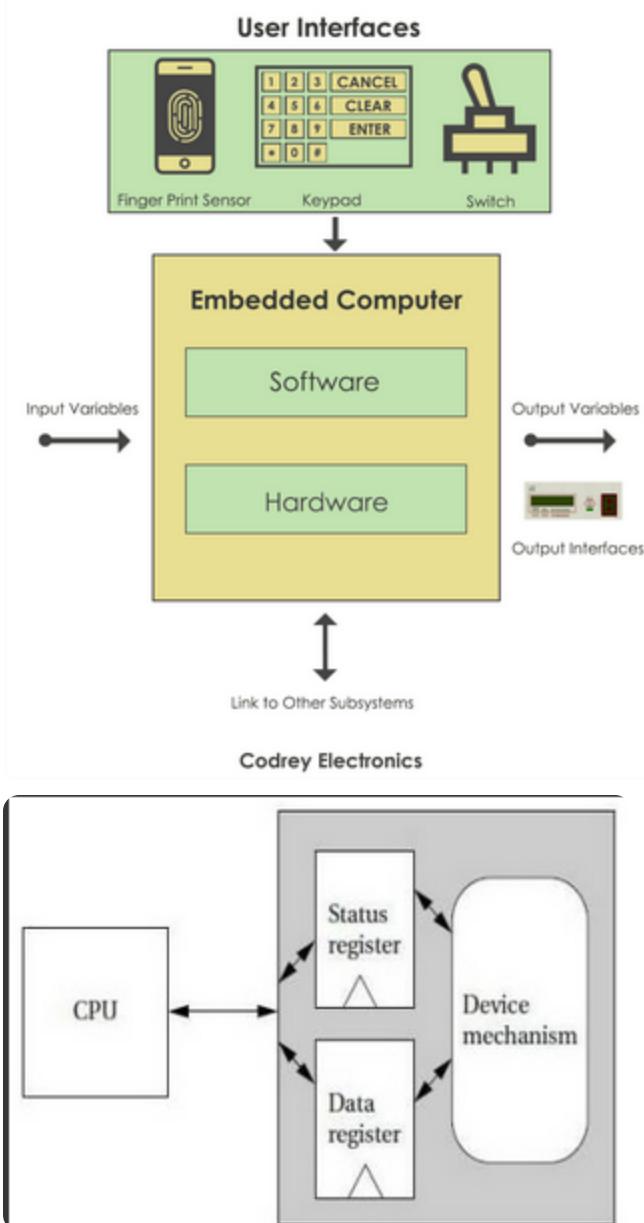
- снижение скорости исполнения отдельной команды;
- не все операции за один машинный цикл;
- необходимость разрешения конфликтов;
- непредсказуемое время исполнения;
- уязвимости "косвенных каналов" (*Meltdown*, *Spectre*);
- как быть с виртуальными методами?

111)Каковы шаги типового взаимодействия с устройством ввода-вывода? Каков интерфейс устройства для процессора?

Шаги типового взаимодействия с устройством ввода-вывода:

1. Конфигурирование устройства доступа(объясняем внешнему устройству что мы будем с ним делать)
2. проверка состояния
3. отправка данных
4. проверка статуса ответа

5. чтение ответа



Устройство ввода-вывода выглядит примерно как и изображено на картинке:

- регистры данных
- регистры статуса
- протокол взаимодействия

практически любое взаимодействие с устройством на низком уровне так или иначе сводится к тому, что у нас есть набор регистров, которые мы читаем или в которые мы пишем. При этом регистр внешнего периферийного устройства.

112)Как реализуется ввод-вывод с точки зрения системы команд в Port Mapped IO?

В системе команд процессора есть специальные инструкции по типу input/output и где мы еще указываем порт ввода/вывода, и данные будут туда отправлены или прочитаны. Адресация при этом этих устройств ввода-вывода никак не зависит относительно работы с памятью, с регистрами. Это отдельно выделенный набор системы команд специально для операций ввода-вывода.

Достоинства:

- Минимизация логики управления(малое адресное пространство), оптимизация IO
- Ввод-вывод и доступ к памяти разделены
- Адресное пространство памяти однородно
- Простота системы в целом

Недостатки:

- Усложнение ISA и процессора
- Данные ввода-вывода - данные второго сорта(особенно для cisc процессоров)
- "Лишние" копирования данных
- Очередное адресное пространство(чем их больше тем больше проблем с фрагментацией)

113)Как реализуется ввод-вывод с точки зрения системы команд в Memory Mapped IO?

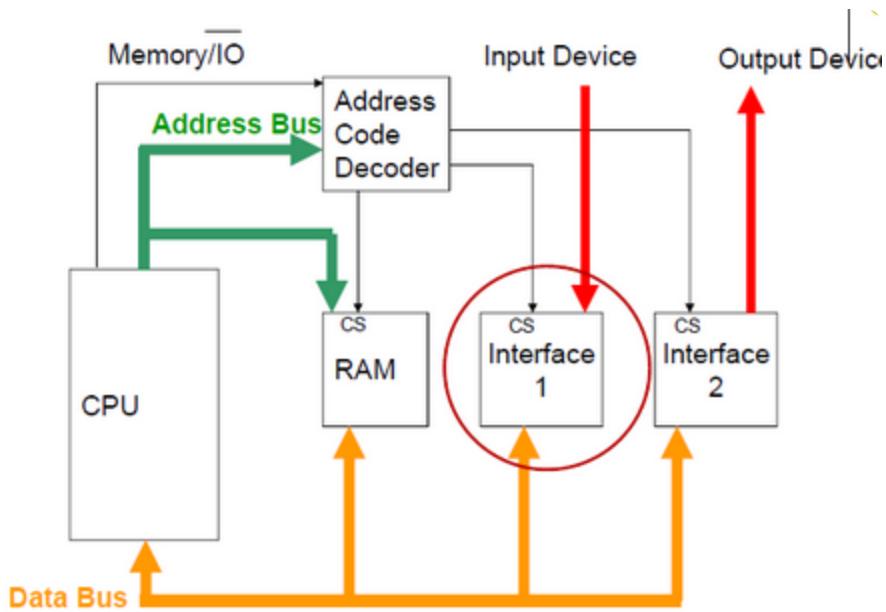
Тут же нет специальных инструкций по типу input/output, так как устройства ввода-вывода отображаются в адресное пространство памяти.

В итоге получаем, что часть адресного пространства резервируется под внешние устройства и обращение к этому устройству выглядит как обычное обращение к памяти.

114)Как устроен ввод-вывод в Memory Mapped IO с точки зрения устройства процессора?

В **Memory-Mapped I/O (MMIO)** устройства ввода-вывода отображаются в адресное пространство оперативной памяти, и взаимодействие с ними происходит через обычные команды работы с памятью

Устройствам ввода-вывода выделяются фиксированные адреса в общем адресном пространстве и контроллер памяти определяет куда направлять запрос.



1. Регистры внешних устройств отображаются в адресное пространство памяти.
2. Ввод-вывод реализуется через инструкции доступа к памяти.

Достоинства ММIO

1. Простота процессора. Без изменения микроархитектуры.
2. Единый набор механизмов доступа: автоинкремент, векторные операции, работа с барьерами.
3. Обработка данных без переноса в память.
4. Адресное пространство памяти.

Недостатки ММIO

1. Одна шина для ввода-вывода и памяти. Разница скорости шин.
2. Неоднородность памяти, сложная конфигурация системы.
3. Избыточный адрес для устройств ввода-вывода.
4. Проблемы с параллелизмом уровня инструкций и кешами:
 - flush при вводе-выводе;
 - порядок записи регистров.

115)Как работает программно-управляемый ввод-вывод без прерываний? Каковы его ограничения и

типовoy сценарий? Как связан с теоремой Котельникова?

Программно-управляемый ввод-вывод - операции реализуются процессором. Все действия реализуются через инструкции процессора.

Любую операцию ввода-вывода можно сделать примерно так:

Есть внешний контакт(грубо говоря нога процессора) и этот контакт можно либо завести в порт ввода-вывода процессора либо в одну из ячеек памяти и дальше просто читать ее. В итоге надо просто проверять эту шину, периодически проверять есть ли там данные, карочи буллшифт запара для процессора

или есть перестать слушать поток сознания пэнского и записать не его прямую цитату а по нормальному:

это метод, при котором процессор активно опрашивает состояние устройства, проверяя его готовность к обмену данными, вместо использования прерываний. Типичный подход к программированию в таком случае - polling(наблюдаем за состоянием устройства и реагируем)

карочи тут дальше сами вспоминайте бэвм нам там уже рассказывали это(будь то про бесмысленную работу процессора, про деръмовую масштабируемость и тд и тп)

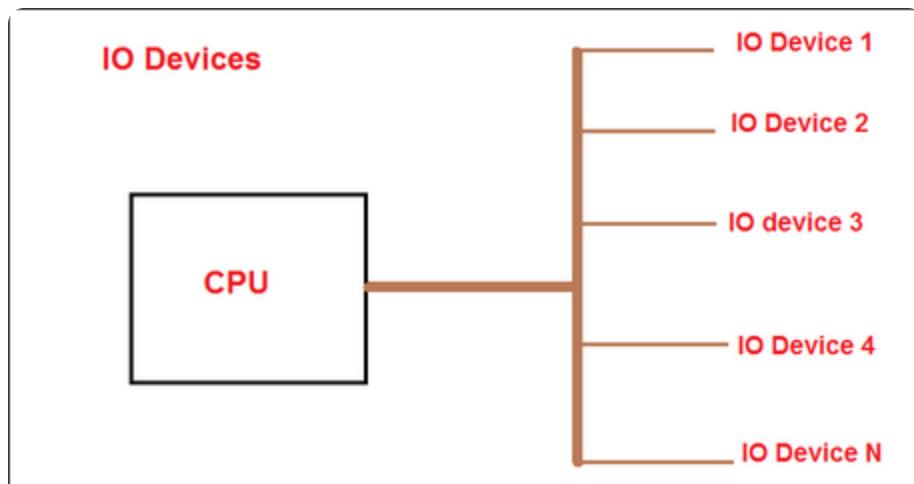
Про теорему Котельникова:

вообще теорема эта звучит так примерно:

чтобы точно восстановить аналоговый сигнал, частота дискретизации должна быть как минимум вдвое выше максимальной частоты сигнала. В итоге опрос устройства о его готовности - это условно дискретизация его состояния. И чтобы не пропустить событие частота опроса должна быть выше максимальной частоты событий(иначе данные будут потеряны), но высокая частота приводит к перегрузке ЦПУ.

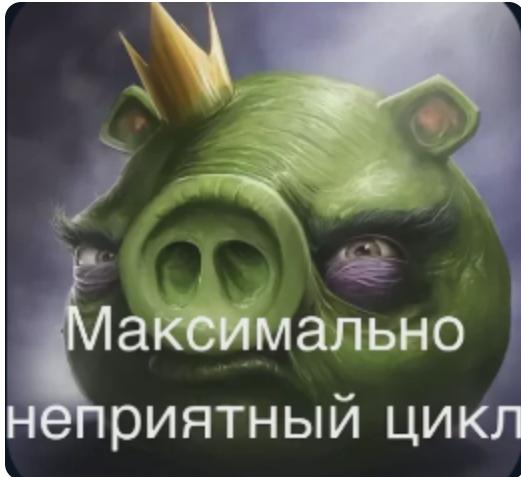
Программно-управляемый ввод-вывод. Проблемы

1. Занимает процессор, включая имитацию таймера.
2. Процессор (алгоритм) должен регистрировать сигнал на частоте в два раза выше частоты сигнала (теорема Котельникова).
3. Высокое энергопотребление.
4. Как работать с клавиатурой?
5. Как совмещать с другими задачами?
6. Как быть со сложным протоколом ввода-вывода (пример на следующем слайде)?



116)Как можно имитировать параллелизм уровня задач для параллельного программно-управляемого ввода-вывода?

TODO



душный билет лень писать чото сори

117)Какие существуют уровни параллелизма: бит, инструкций и задач? Приведите примеры.

Параллелизм:

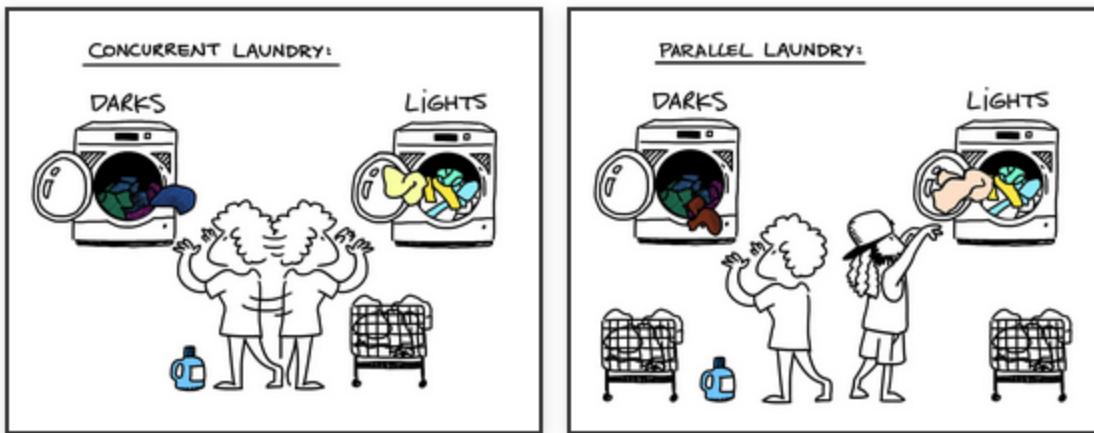
- Когда нужно работать сразу над несколькими задачами(ввод-вывод, системы управления)
- когда нужно повысить уровень утилизации (не пропускаем, а занимаемся чем то полезным)
- когда нужно повысить производительность компьютера(делаем больше дел за единицу времени)

Виды параллелизма:

1. Уровень битов: "Ширина" комбинационных схем, шин данных и машинного слова(Напоминаю условно есть машинное слово и отдельные битики этого слова пытаются считать параллельно за счет комбинационных схем, которые у нас есть или шин).
2. Уровень команд: Параллельное выполнение нескольких инструкций(Risc процессора)
3. Уровень задач: Параллельное выполнение нескольких программ

118)В чём разница между параллелизмом и конкурентностью (concurrency)?

конкуренци - 1 процессор, который переключается между задачами
параллельность - несколько процессоров, которые выполняют разные задачи



119) В чём заключается проблема реализации параллелизма уровня задач в архитектуре фон Неймана?

Архитектура фон Неймана не рассчитана на параллелизм(рассчитана на последовательное выполнение инструкций и в едином потоке исполнения). Поток инструкций - один и процессор должен "молотить" инструкции до Halt(условно начиная выполнение инструкции он не остановится пока не выполнит карочи тупая машина).

120) Каковы достоинства и недостатки кооперативной многозадачности?

Кооперативная многозадачность - поток инструкций не просто считается, а учитывает тот факт, что он не единственный процесс и он отдает поток управления кому то другому. Многозадачность, при которой следующая задача выполняется, когда текущая задача явно объявит о готовности отдать процессорное время(грубо говоря есть Pause).

1. Активная программа получает всё процессорное время.
2. Фоновые — заморожены.
3. Приложение захватывает столько ресурсов, сколько хочет.
4. Приложения делят процессор, передавая управление следующему.

Достоинства

- Контроль за ресурсами со стороны задачи.
- Известные точки остановки. Отсутствие гонок.
- Легкость и эффективность (при программной реализации).

Недостатки

- Контроль за ресурсами осуществляется задачей.
- Сбой задачи может быть глобальным.
- Низкая эффективность и трудоёмкость ввода-вывода.
- Сложность разработки интерактивных приложений.

121)Какие механизмы необходимы для реализации кооперативной многозадачности?

1. Механизм **остановки** выполнения задачи: добровольная передача управления "диспетчеру".
2. Механизм **сохранения** состояния задачи: регистры, стек, состояния сопроцессоров, память, ввод-вывод, кеш, состояние предсказателя переходов, и т.п.
3. Механизм **планирования** — какой задаче отдать процессорное время следующей.
4. Механизм **возобновления** остановленного процесса: восстановление состояния и передача управления.

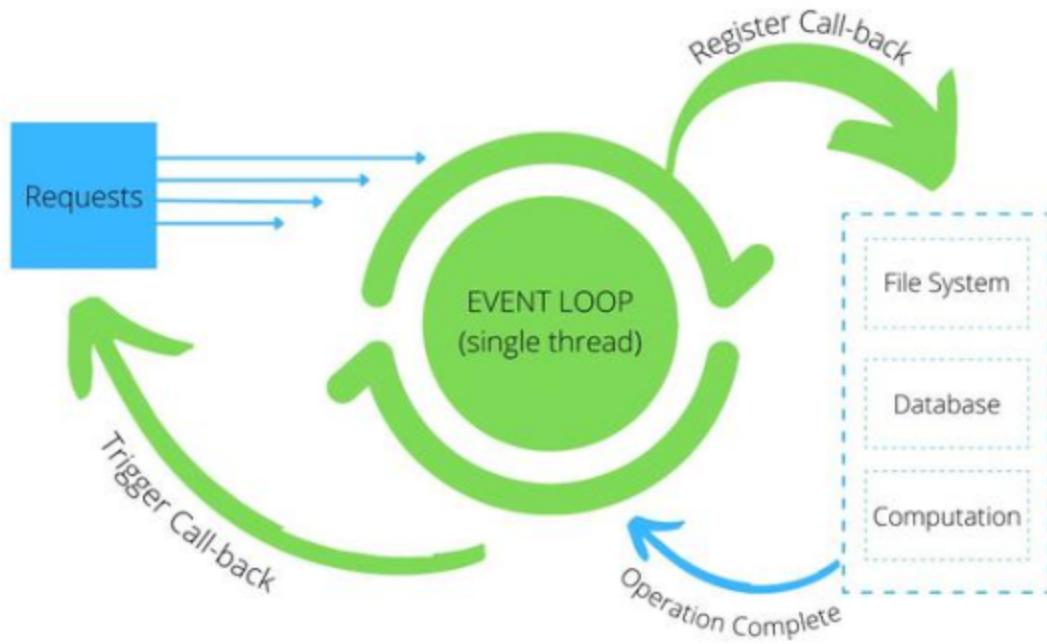
следующие механизмы опциональны.

1. Механизмы **изоляции** задач: независимое выполнение, безопасность.(с точки зрения выделяемой памяти - каждый процесс может захватить столько ресурсов сколько ему надо)
2. Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.(целостность данных)

Есть еще разные подходы к реализации кооперативной многозадачности:

- Имитация через конечные автоматы
- С диспетчером задач на уровне:
 1. ОС(системные вызовы/передачи управления)
 2. ВМ или умный рантайм
 3. либо в программном коде(условные event-loop + callbacks) или async/await
(запускаем задачу, помечаем ее как асинхронную и где нам нужен результат помечаем await. Это значит, буквально есть единственный цикл event loop, после регистрируем операцию в пуле, в котором появится отметка что задача выполнена, мы обратно попадает в event-loop и обнаружить что задача выполнена и вызвать

необходимого обработчика. Карочи на картинке показано, как выглядит подобное управление через общий поток управления)



122)Как взаимодействуют кооперативная многозадачность и система прерываний?

во первых прерывания это дорого, надо переключать контекст, менять страницы памяти и сохранять где то регистры, а потом еще восстанавливать их.

И взаимодействие строится примерно так:

прерывание устанавливает флаги или добавляют события в очередь, потом основная обработка происходит когда задача добровольно получает управление. Сами обработчики прерываний сохраняют данные, устанавливают флаги событий и пробуждают ожидающие задачи через механизм событий.

карочи тут Пенской почти ничего не сказал, эту информацию я взял из интернета.

но в самой презе и в лекции про прерывания и кооперативную многозадачность не было сказано, а было про прерывания и вытесняющую многозадачность
дать определение кооперативной многозадачности + прерываний

123)Как реализуется кооперативная многозадачность на уровне приложения через Event-loop?

описано выше (см. билет 121)

124)Каковы достоинства и недостатки вытесняющей многозадачности?

Вытесняющая многозадачность(истинная многозадачность) - ос передает управление между программами в случае завершения операций ввода-вывода, событий в аппаратуре компьютера, истечения таймеров и квантов времени, поступления сигналов.

по большому счету, это та же кооперативная многозадачность, но с тем отличием, что передача управления происходит не в фиксированный момент времени, когда об этом сказал сам процесс, а в случайный момент времени, чаще реализуется через механизм прерываний.

1. Переключение процессов происходит буквально между любыми двумя инструкциями.
2. Распределение процессорного времени осуществляется планировщиком
3. Возможна "мгновенная" реакция на действия пользователя.

Преимущества:

- Простота разработки ПО с 1 процессором
- Контроль за ресурсами со стороны ОС. Изоляция
- Интерактивность системы(почти мгновенная реакция)

Недостатки:

- Лишние переключения. Тяжесть процессов(прерывания, состояния и тп)
- Разделяемые состояния и непредсказуемые переключения, синхронизация, гонки
- непредсказуемая длительность работы

125)Какие механизмы необходимы для реализации вытесняющей многозадачности?

Механизмы практические такие же, как и в кооперативной многозадачности:

1. Механизм прерывания процесса - забрать процессор у задачи независимо от ее желания.
2. Механизм сохранения состояния задачи: регистры, стек, состояния сопроцессоров, память, ввод-вывод, кеш, состояние предсказателя переходов и тп.
3. Механизм планирования - какой задаче отдать процессорное время следующей
4. Механизм возобновления остановленного процесса: восстановление состояния и передача управления

следующие механизмы опциональны.

1. Механизмы **изоляции** задач: независимое выполнение, безопасность.(с точки зрения выделяемой памяти - каждый процесс может захватить столько ресурсов сколько ему надо)
2. Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.(целостность данных)

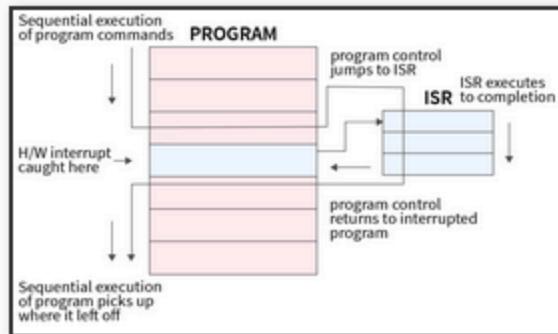
просто важная инфа про систему прерываний

Система прерываний

Архитектура фон Неймана характеризуется:

1. последовательным исполнением команд и [без]условным переходом;
2. процессор будет "молотить" инструкции столько, сколько сможет;
3. оптимизация "числодробилки" для одной задачи.

Система прерываний позволяет **переключаться** (совершить переход к заданной инструкции) по внешнему событию и вернуться назад после обработки.



126) В чём преимущество ввода-вывода с использованием системы прерываний?

Экономия процессорного времени(нет поллинга, когда процессор постоянно опрашивает внешнее устройство)

ЦПУ занято полезной работой, внешнее устройство само уведомляет процессор через прерывание о готовности обмена

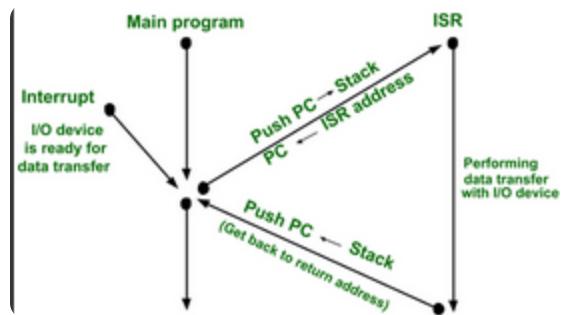
Прерывания мгновенно реагируют на событие ввода-вывода

нет нужды в ручных проверках готовности, так как логика обработки данных может быть инкапсулирована в обработчик прерываний.

127)Как по шагам происходит обработка прерывания?

1. Выполнение "основного" потока
2. Запрос прерывания
3. Сохранение адреса возврата
4. Вызов обработчика прерывания(ISR) - обычного кода в специальном месте(обычно имеют зафиксированные адреса в памяти, так как вызов обработчика прерывания должен проходить на аппаратном уровне)
5. Завершение ISR
6. Восстановление адреса возврата
7. Продолжение основного потока

Важная сводочка по картинке, вообще из прерывания мы не обязаны возвращаться в тот же поток управления, мы можем вернуться в целом куда нам захочется.



128)Почему появление системы прерываний приводит к автоматическому появлению параллелизма уровня задач? Охарактеризуйте его.

Система прерываний де facto вводит параллелизм уровня задач даже в однопроцессорных системах.

Система прерываний обеспечивает минимум два уровня задач(или два независимых контекста выполнения):

- Основной поток исполнения
- Поток(потоки) обработчиков прерываний
 - А это уже параллелизм, так как ISR существует независимо от основного кода
 - В итоге такой параллелизм характеризуется:
- это аппаратно программный уровень(хотя мне кажется что чисто аппаратный, так как вызов обработчика прерывания происходит на аппаратном уровне)
- Что касаемо детерминизма, то порядок прерываний зависит от внешних событий

- ресурсы общие: регистры, память.
 - есть всякие проблемы по типу дедлоков(когда isr заблокирует ресурс, который ожидается в основном потоке), ну гонки еще всякие(пример додумайте сами)
- И еще это автоматический параллелизм уровня задач, так как не требует явной программной реализации.

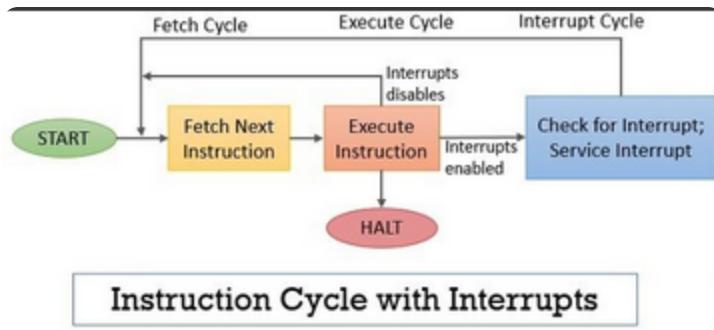
129)Как классифицируются прерывания по источнику? Их отличительные особенности.

Вот их классификация:

1. Аппаратные:

- Внешние(асинхронные для внутренних циклов процессора): переполнение таймера, нажатие клавиши, сетевой пакет, карочи вызывают устройства вне цпу
- Внутренние(синхронные): деление на 0, ошибка доступа к памяти и тп, карочи вызывает процессор, при ошибках или особых условиях

2. Программные(вызывается инструкцией): взаимодействие программы и ос()



130)Каковы задачи контроллера прерывания и какие виды прерываний бывают?

сама по себе система прерываний почти всегда работает аппаратно, обычно это сбоку стоящая аппаратура, в ней может быть программное обеспечение. Она делается аппаратно, чтобы работала быстро и не загружала процессор, иначе идея теряется.

Задачи контроллера прерываний:

1. Приоритизация прерываний(какое из них обработать)
2. Маскирования(блокировка) прерываний(отключить определенные прерывания, чтобы не мешали критическому коду)

3. перенаправления прерываний к цпу
4. обработка вложенных прерываний



виды прерываний:

- максириуемые/немаксириуемые(те прерывания, на которые мы можем повесить маску. Буквально мы обратываем нажатие кнопки, значит игнорим все другие нажатия. Или как вариант с критичным куском кода и маскируем все прерывания)
- относительные/абсолютные(прерывает прерывание. Относительные те прерывания, если мы уже в прерывании, то другое нас прервать не может. В относительном всё иначе)

131)Какие виды событий существуют в системе прерываний и как они обрабатываются?

Виды событий/прерываний:

- По фронту(Есть линия, на нее может прийти либо положительный фронт сигнала либо негативный. По такому фронту срабатывает система прерываний и вызывается тот самый переход по обработчику прерывания)- По уровню("требует сброса"). По уровню работает так: устройство выставило высокий уровень сигнала, региструем это как прерывание и потом его обрабатываем. Преимущество с предыдущим такое: если приходит несколько прерываний, то по фронту, надо будет сохранять этот сигнал в какой нибудь буффер, иначе будет утерян. По уровню надо будет просто притушить сигнал, чтобы показать, что прерывание обработано)
- По сообщению(Есть очередь сообщений, в которой мы сохраняем информацию о том, какие прерывания произошли(опять же реализована аппаратурно))
- По дверному звонку. Сигнал и информация разделены(Призван бороться с ситуацией, когда будет много прерываний и векторов прерываний будет недостаточно. А дальше я не понял, что там с регистром, в который будем сохранять номер прерывания и чототамеще)

132)Что такое сторожевой таймер и каков принцип его работы?

Сторожевой таймер - аппаратно реализованная схема контроля от зависания системы
принцип работы:

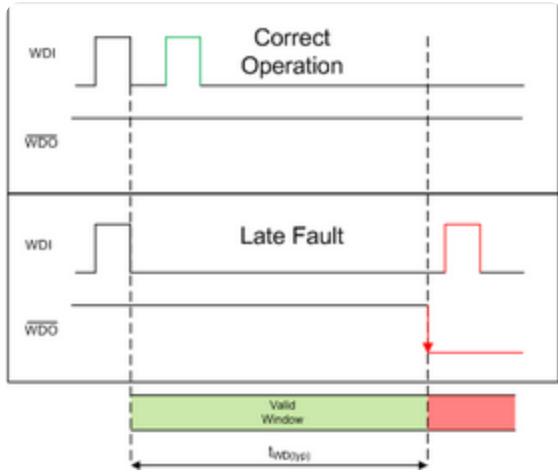
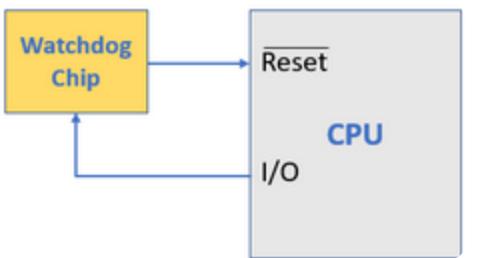
Так как в ирл в системах управления никто не может гарантировать, что процессор и его аппаратура будет работать корректно(физически невозможно). Тогда будет возможна ситуация, когда из за случайной инверсии отдельных битов или ошибок программирования процессор начнет зависать.

Есть сам процессор и внешнее устройство(этот самый сторожевой таймер), у которого есть два сигнала к процессору: первый от процессора к этому устройству, по которому идет сигнал, который сообщает что процессору по кайфу и он работает в штатном режиме. Второй сигнал уже от watchdog timer'a, с сигналом перезагрузки процессора. Часто бывает что перезагрузка аппаратная, которая ультимативно перезагрузит процессор.

Работает это как показано на 2 картинке. У нас есть какой то определенный таймаут, после которого процессор должен быть перезагружен, так как скорее всего это ошибка. Задача процессора в рамках этого цикла, сбрасывать watchdog timer до наступления критического момента. Если система проваливается в бесконечный цикл, то мы автоматически вываливаемся за этот цикл, и так автоматически будет перезагружен процессор.

НО!

это не является системой прерываний в классическом понимании, так как мы не передаем управление другой задаче, но это является системой прерываний, так как поток выполнения нарушается, если что то пошло не так.



133) Какие подходы существуют к решению проблемы изоляции регистров и инструкций в памяти при многозадачности?

Параллелизм требует:

- обеспечить совмещение и изоляцию между задачами
- обеспечить взаимодействие между задачами

у нас надо совместить/изолировать:

- Регистры
 - Адреса инструкций(переходы)
 - Адреса данных(переменных)
 - Динамическую память(куча)
 - Автоматическую память(стэк)
- когда надо совмещать/изолировать:
- Compile-time(состав задач фиксирован)
 - Run-time(состав задач динамический)
- а вот и сами подходы:

1. Инструкции

- Размещаем инструкции в разных областях памяти

- Корректируем все абсолютные адреса переходов

2. Данные:

- Размещаем данные в разных областях памяти
- Корректируем все абсолютные адреса

3. Регистры:

- Формируем соглашение о вызове процедур внутри задач
- Сохраняем и загружаем регистры между задачами

4. Автоматическая память:

- Оцениваем потребности в памяти
- Выделяем подходящие диапазоны(фрагментация)

5. Динамическая память: выделяем диапазон

134)Какие подходы существуют к решению проблемы изоляции данных в памяти (статика, куча, стек) при многозадачности?

из билета выше 3-5 пункт

135)Как банки памяти используются для расширения адресного пространства?

Банки памяти - это буквально блок ячеек памяти, к которому процессор обращается как к единому целому. Это нужно для увеличения объема доступной памяти без усложнения адресации. Или другими словами: есть несколько чипов памяти к которым мы можемходить связано или независимо друг от друга.

Используются когда:

- Память имеет большее адресное пространство, чем процессор
- Расширение машинного слова
- Переключение режима работы(аппаратная изоляция)
- Изоляция задач - редко

Работает он так(на примере процессора, которому надо больше памяти, чем он может адресовать):

Память разбивается на банки, в каждый момент времени активен только 1 банк, но его

можно переключить. Процесс обращается к одному и тому же адресу, но в зависимости от выбранного банка адрес указывает на разные физические ячейки.

136)Как банки памяти используются для расширения машинного слова?

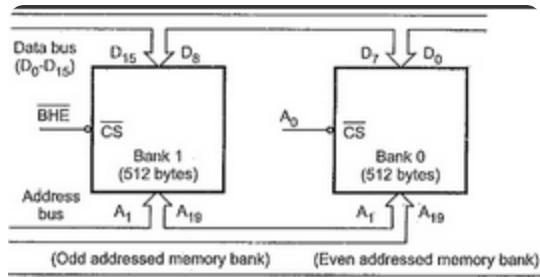


Fig. 10.11 Memory Interfacing

Представим, что у нас 16-ти битный процессор, ячейки памяти - 8 бит.

Как за один такт сделать так, что могли читать и записывать машинное слово.

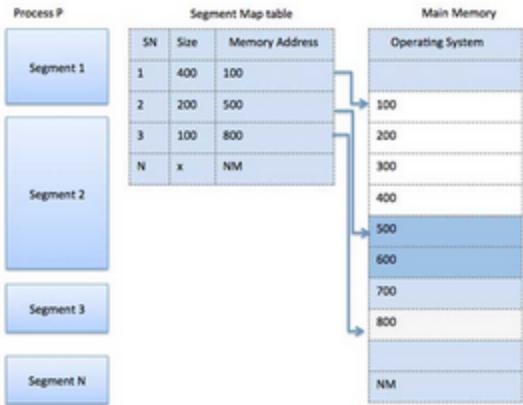
Почти все адреса приходят на каждую из ячеек памяти и младший бит мы отправляем либо в одну ячейку памяти, либо в инверсном в другую.

Это значит, что при записи в память, машинное слово бьется на 2 половинки, в 1 чип памяти попадут старшие биты, во второй чип запишется младшие биты.

Карочи ниче не понятно, но это прямая цитата с лекции.

Можно в целом взять пример из билета выше, потому что вся разница лишь в том, что в расширении памяти используется еще 1 линия, по которой выставляется нужный банк памяти.

137)Что такое сегментная память? Как происходит трансляция адресов? Каковы достоинства и недостатки?



Сегментная память - разметим память процессора по назначению, как внутри задач, так и между ними.

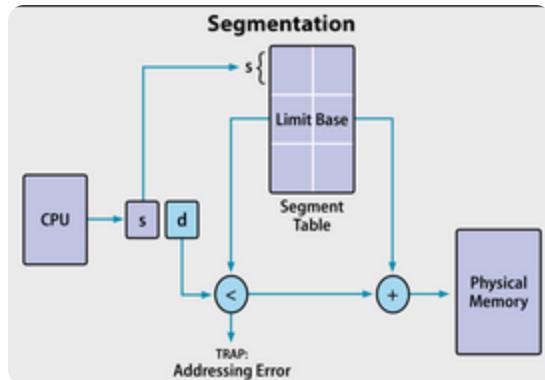
Сегментная адресация памяти - способ логической адресации памяти, где адрес = сегмент + смещение

Сегмент - выделенная область адресного пространства определенного размера

Смещение - адрес ячейки памяти относительно начала сегмента

таблицы сегментов состоят из номера, размера, адреса откуда начинается

Трансляция:



есть процессор, каждый адрес задачи имеет сегмент и сдвиг внутри сегмента, дальше он по таблице сегментов понимает в каком физическом адресе он начинается, проверяет размер сдвига для текущего сегмента(проверка не вышли за пределы при доступе), складывает сдвиг и первый адрес сегмента и получаем адрес физической памяти.

Достоинства:

- Таблицы сегментов относительно малы.
- Таблицы сегментов просты в обработке и перемещении.
- Средние размеры сегментов больше, чем размеры большинства страниц, что позволяет хранить в сегментах больше данных процесса.
- Отсутствует внутренняя фрагментация.

Недостатки:

- Non-von Neumann way
 - Поддержка со стороны компилятора, ПО.
 - Участие программистов (количество сегментов, размер сегментов).

- Считается устаревшей, имеет ограниченную поддержку ОС, так как в таком случае на момент запуска любой тулзы надо четко знать сколько памяти ей надо выделить, получаем такое, что если указать сегмент меньшего размера, то упадем сегфолтом.
- Внешняя фрагментация.

138)Что такое виртуальная память? Как происходит трансляция адресов? Каковы достоинства и недостатки?

https://www.youtube.com/watch?v=t3YkDqw_qDQ <- посмотрите этого дядьку, я в целом обожаю у него этот ролик(хотя я не понимаю зачем меня уже второй год просят рассказывать одно и то же, кстати на ОСах будет то же самое)



но если говорить про материал лекции, то вот:

Предоставим каждой задаче своё виртуальное адресное пространство. Пусть каждый процесс думает, что всё адресное пространство его.

В итоге получаем, что каждый процесс работает в своем изолированном адресном пространстве. Физическая память динамически расширяется между процессами. + можно вспомнить про файл подкачки aka своп/swap файл

Сам процесс преобразования виртуального адреса в физический называется трансляцией адресов.

И выглядит он примерно так:

Виртуальные адреса режутся на страницы, сам виртуальный адрес состоит из номера страницы и смещения. Есть еще таблица страниц, которая хранит соответствие виртуальных страниц физическим фреймам и у каждого процесса своя таблица. карочи посмотрите ролик и будем вам счастье, можете отдельно почитать про tlb, mmu, swap.

вот еще материал из лекции:

виртуальная память позволяет:

1. Прозрачно для программиста изолировать задачи.
2. Нелинейное физическое размещение данных.
3. Не фиксировать объём памяти, используемый задачей.
4. Использовать больше памяти, чем есть физически. Выгрузка на диск части задачи (задач).
5. Права доступа. Отображение страниц на разные адресные пространства.

Достоинства:

- Прозрачна для программистов.
- Работа с "бесконечной" памятью, динамическое распределение памяти.
- Повышает общую стабильность системы (аналогично Preemptive Multitasking).
- Отсутствует внешняя сегментация.

Недостатки:

- Большой объём таблиц страниц, длительный поиск (кеш).
- Высокие накладные расходы (ввод-вывод, перенос страниц...)
- Нет изоляции внутри адресного пространства.
- Непредсказуемая длительность доступа к памяти.
- Высокая сложность реализации.

139)Что такое внутренняя и внешняя фрагментация в контексте сегментной и виртуальной памяти?

Внутрення фрагментация возникает когда выделенный блок памяти больше, чем требуется процессу

Внешняя фрагментация - свободная память разбита на множество фрагментов, которые не могут быть эффективно использованы.

140)Как появление сегментной памяти улучшило пользовательский опыт?

изоляция процессов, логическая организация памяти, защита, гибкость и стабильность

141)Как появление виртуальной памяти улучшило пользовательский опыт?

"увеличение" памяти, изоляция процессов, защита и безопасность, упрощение разработки

142)Какие уровни (виды) задач существуют: основной поток, прерывание, процессы, потоки, зелёные потоки?

сам список очень условный

Типовые виды задач:

1. Main/Kernel/основной поток - исходный поток инструкций
2. Прерывания. Особый исходный код обработчиков прерываний
3. Процессы. Изолированные адресные пространства. Нет прямого доступа(задачи, которые запускаются в рамках user-space, их задача - решать прикладные задачи, с ограниченным доступом)
4. Потоки. Работают в адресном пространстве процесса. Прямой доступ ко всем его данным.(поток исполнения, т.е. последовательности инструкций которые будут выполняться процессором, которые будут выполняться в одном адресном пространстве)
5. Зеленые потоки. В рамках Run-time или виртуальной машины.

143)Как могут взаимодействовать процессы через основной поток? Какие бывают разделяемые ресурсы?

Само взаимодействие процессов, работающих в разных адресных пространствах строится только через взаимодействие с ОС.

и это либо:

1. Shared memory: берем отдельные страницы виртуальной памяти и даем к ним доступ в разных процессах
2. Signals: какое то подобие системы прерываний, но для конкретной задачи. Есть процесс, он выполняет свои инструкции. Идея сигнала заключается в том, что ядро ос

возвращает управление не в задачу куда должно, а в обработчик сигнала. КАРОЧИ АНАЛОГ ПРЕРЫВАНИЯ В РАМКАХ ПРОЦЕССА И ХУЙ С НИМ, Я НЕ ЗНАЮ КАК ЭТО ОБЪЯСНИТЬ

3. IO: network, files, pipes: всё то же самое, только через операции ввода-вывода.

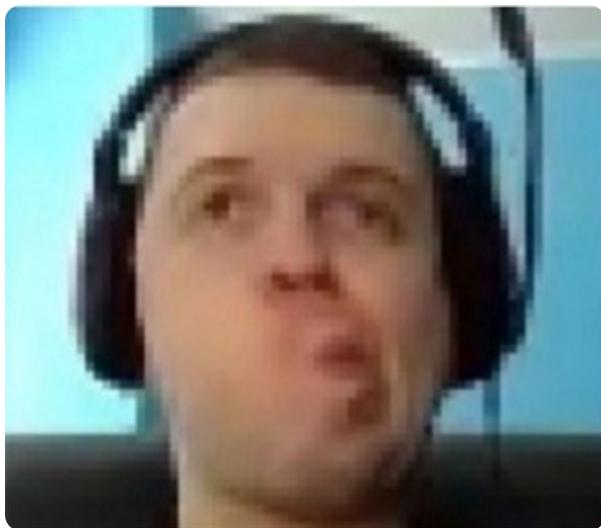
Разделяемые ресурсы:

- память
- файлы
- устройства ввода-вывода
- системные объекты
- глобальные переменные

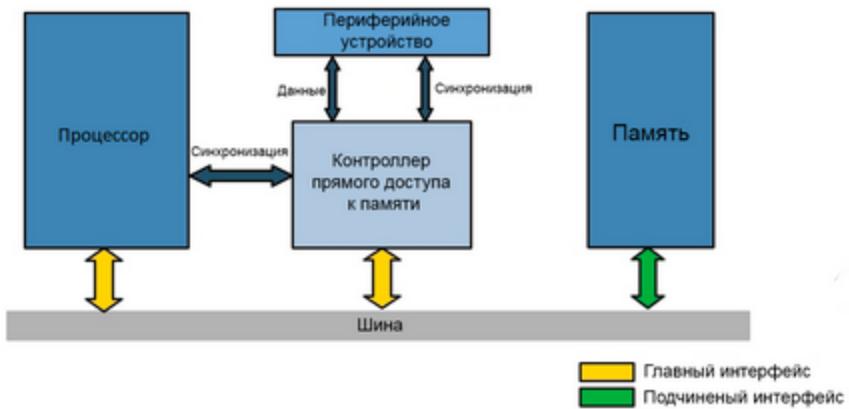
144)Что такое процессоры ввода-вывода? Каковы их назначение, интерфейс, достоинства и недостатки?

skip boring shit

145)Что такое контроллер прямого доступа к памяти (DMA)? Каковы его назначение, интерфейс, достоинства и недостатки?

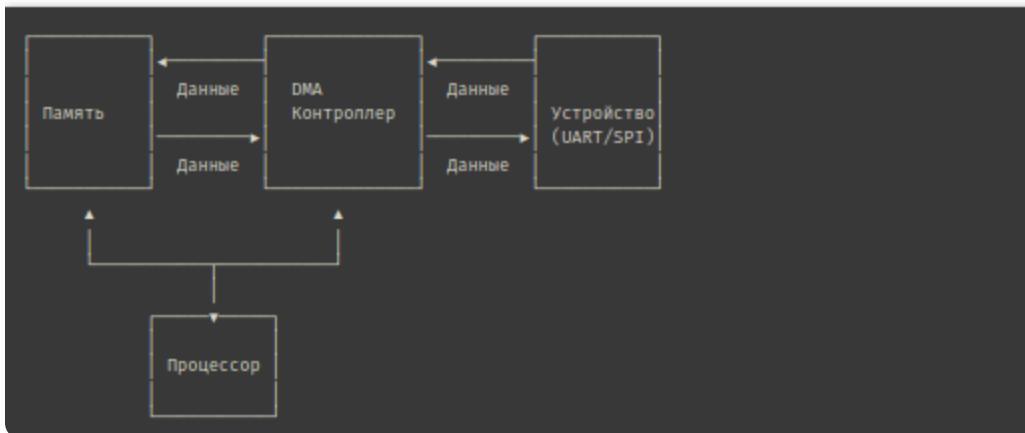


DMA (Direct Memory Access) - аппаратный модуль, позволяющий устройствам ввода-вывода обмениваться данными с оперативной памятью без участия ЦПУ.



```

DMA_SRC_ADDR // Исходный адрес (откуда читать)
DMA_DST_ADDR // Целевой адрес (куда писать)
DMA_COUNT // Количество байт/слов для передачи
DMA_CONTROL // Управление (направление, режим работы, запуск/остановка)
DMA_STATUS // Статус (занят/готов, ошибки)
    
```



вот так выглядит примерно интерфейс DMA:

- указываем адрес откуда читаем
- адрес куда пишем
- количество байт для передачи
- управление(режим работы, направление)
- статус (занят/готов, ошибки)

DMA просто копирует данные из одной области в другую

Достоинства:

- Скорость и эффективность контроллера
- Интерактивность, так как процессор разгружен от "рутиной"
- Параллелизм: DMA может иметь несколько каналов для параллельной работы.

Недостатки:

- Проблемы совместимости
- Сложность при непоследовательном доступе к памяти
- Ограниченный контроль за системной шиной, синхронизация работы процессора и DMA

- Конфликт использования DMA разными устройствами ввода-вывода

146)Какие существуют способы интеграции и взаимодействия контроллера прямого доступа к памяти (DMA) с процессором?

способы интеграции:

1. Third-party: Управление DMA осуществляется процессором(есть специальное отдельное устройство, которое висит на шине рядом с памятью и мы с ним можем взаимодействовать. Но в таком случае контроллер dma тупеньким становится, он не может обрабатывать прерывания, может только переносить данные)
2. Bus mastering: Управление DMA может осуществляться и устройствами ввода-вывода(на шину ставим самостоятельное устройство, которое имеет свою систему прерываний, информировать процессор о готовности данных к чтению только после прочтения всех данных. Карочи весь ввод-вывод делегировать на DMA устройство)

Способы взаимодействия:

1. Пакетный режим(Burst mode). Приоритет DMA. Передача данных осуществляется единой операцией, которая не может быть прервана процессором.
2. Циклический режим(Cycle stealing mode): Приоритет конфигурируется. Для процессора и DMA выделяется фиксированный слот времени в рамках цикла
3. Прозрачный режим(Transparent Mode): Приоритет процессора. Передача данных, когда процессор не взаимодействует с памятью.

147)Почему необходима иерархия памяти в современных компьютерных системах?

THE MEMORY HIERARCHY



на схеме:

- триггеры, работают на той же частоте, что и процессор
 - кэши трех уровней
 - физическая память
 - SSD
 - виртуальная память, файловые системы, жесткие диски и всякая такая история
- Основная проблема: с ростом скорости памяти растет ее стоимость
Иерархия памяти — это многоуровневая структура запоминающих устройств, организованная так, чтобы максимизировать быстродействие и минимизировать стоимость хранения данных.

вот почему необходима иерархия памяти:

2. Компромисс между скоростью, стоимостью и объёмом
3. Устранение "узкого горла" процессора (Memory Wall)
4. Локализация обращений
5. Энергоэффективность

148)Какие основные виды памяти входят в иерархию памяти и каковы их характеристики?

нужны ли тут точные характеристики я хз, но ладно

1. Регистры цпу: их мало, они быстрые и управляются напрямую процессором и работают на той же частоте, что и процессор
2. кэши L1-L3:
 - L1: вообще разделен на L1i(instruction) и L1d(data). Тоже маленькие и быстрые. Работает на частоте ЦПУ
 - L2: как L1(только он унифицирован), только чуть больше и медленнее
 - L3: общий для всех ядер, используется для синхронизации между ядрами
3. Оператива(DRAM): Динамическая память. Основная рабочая память системы
4. ssd: быстрая энергозависимая память, живут еще долго, так как в них нет движущей части
5. hdd: медленный, но дешевый и емкий, используется для долговременного хранения

149) В чём различие между явной иерархией памяти и скрытой? Приведите примеры.

есть две иерархии памяти: явная(управляется программистом) и скрытая(управляется аппаратно/автоматически)

Явная иерархия: - программист точно знает в какую память заступается, знает ее задержки и как с ней работать

- Регистры, SRAM, DRAM, диски доступны как альтернативные хранилища
- Посыл простой: ИСПОЛЬЗУЙТЕ С УМОМ

Скрытая иерархия: - идея в том, типа го сделаем такую абстракцию, что программист будет работать как с одной памятью, не различая ее, он будет просто обращаться к данным и получать ее. Главное только реализовать этот слой абстракции, реализовать всю эту магию, что программист чаще будет обращаться к самой быстрой памяти и дальше по мере частоты все медленнее и медленнее.

- Модель: одна память, одно адресное пространство
- Зависит от использования. Определяется прозрачно
- Посыл: мы всё сделаем за тебя
и выше скрытая иерархия какой то черный ящик, так как поведение нестабильное нифига, но используются обе.

150) Каково устройство памяти с произвольным

доступом? Как устроена ROM ячейка?

RAM (Random Memory Access) - это энергозависимая память, позволяющая читать и записывать данные в произвольном порядке(+ Задержка доступа не зависит от истории запросов). Пример с книгами: в полке с книгами может достать любую за фиксированное и одинаковое время.

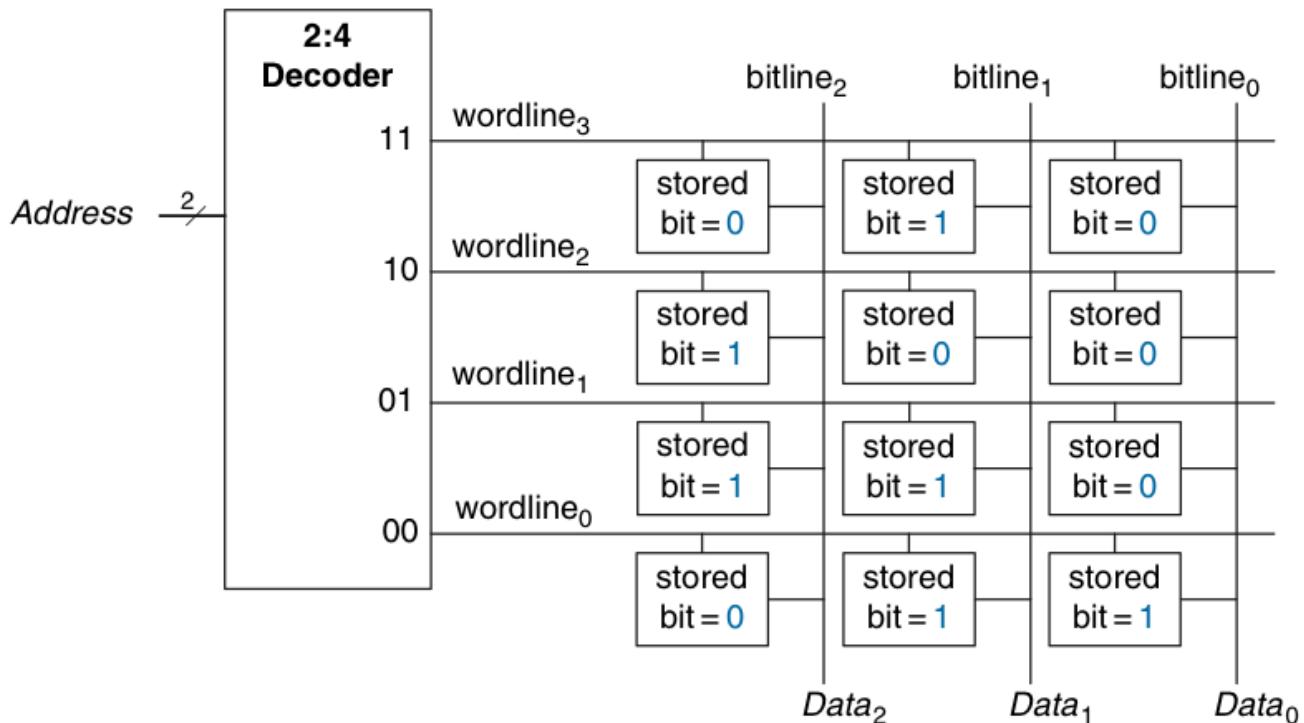


Figure 5.42 4 × 3 memory array

- Address - адрес ячейки памяти шириной два бита
 - Decoder - дешифратор, активирует нужную линию
 - wordline_i - линия, активирующая ячейки требуемого машинного слова.
 - bitline_i - линия, на которую выставляется/читается значение бита определенной позиции
 - stored_bit - ячейка памяти
- RAM организована в виде двумерного массива ячеек.
да и на рисунке все показано

Теперь поговорим про **ROM(Read Only Memory)** - память только для чтения
способы ее реализации:

- Физическое размещение транзисторов или наоборот не ставим его

- Пережигание перемычек при однократном программировании

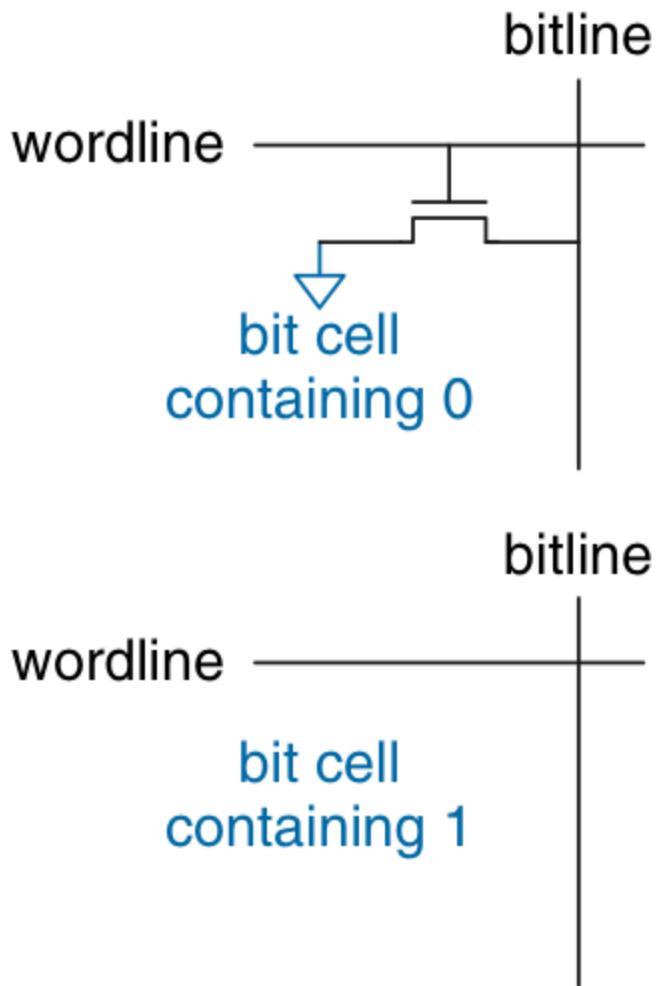


Figure 5.48 ROM bit cells containing 0 and 1

на картинке ее устройство:

есть wordline(выбирает строку в памяти), bitline(читывает бит), транзистор. Если мы на wl подаем сигнал, то транзистор открывается и соединяет bl с землей(то есть выставляет 0) или с питанием(выставляется 1). В итоге память представляет из себя наличие или отсутствие транзистора.

151)Каковы технологии реализации SRAM ячеек? Как происходит чтение и запись?

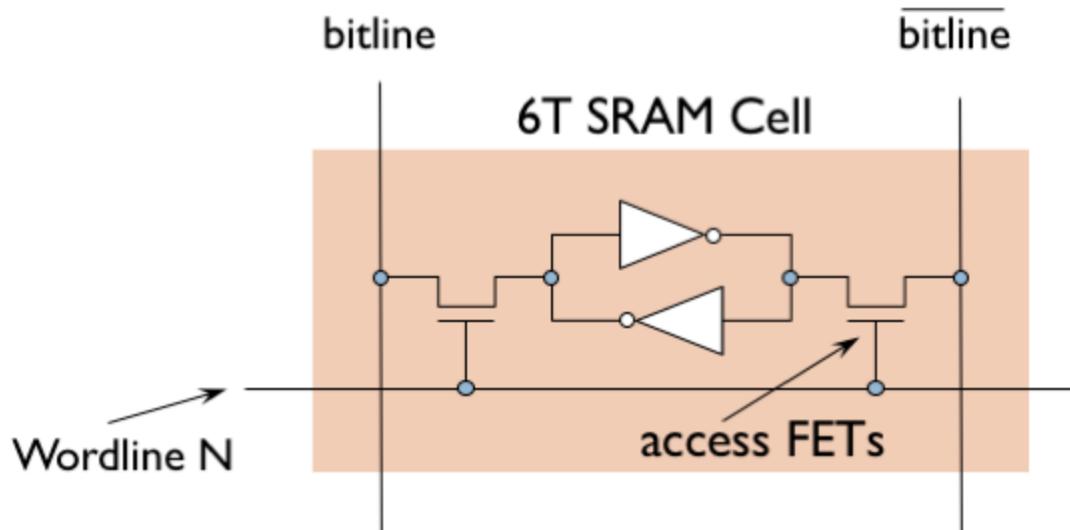
SRAM(Static Random Access Memory) - это статическая память с произвольным доступом, которая хранит данные в виде состояний триггера(бистабильного элемента). В отличие от DRAM она не требует периодического обновления и работает быстрее, но занимает больше места на кристалле.

используются в кешах, раньше даже использовались во внешней памяти, хотя сегодня уже нет.

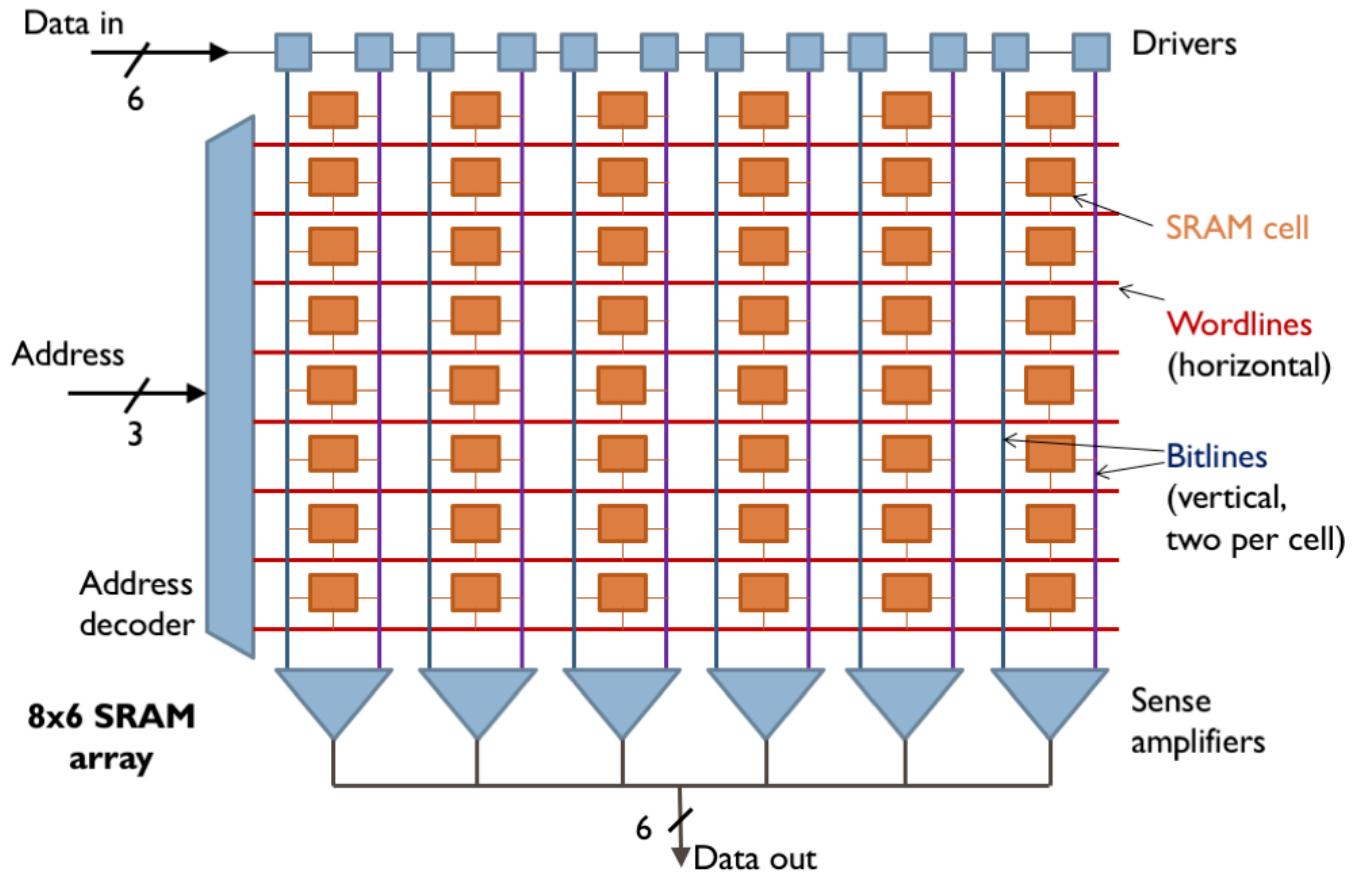
Static значит, что данные записанные в такую ячейку памяти будут там храниться произвольно долго, ровно до того момента, пока их не изменят.

Характеристики:

- Быстрый доступ на чтение и запись
- значение хранится до отключения питания
- требует довольно большое количество транзисторов



чем то похож на рс-триггер. Состоит группы транзисторов(4 - инверторы, 2 - доступ). Есть инвертное и прямое значение в памяти.



а вот как оно примерно всё работает и выглядит:

вот на картинке матрица из ячеек.

есть декодер, на который адрес подается и надо выставить адрес на него, который выставляя линию, мы коммутируем состояние замкнутое в ячейки на этой линии. На выходе стоит усилитель сигнала, задача которого понять, с какой стороны единичка, а с какой нолик.

Ща разберем чтение:

1. Data in выставляет битовые значения в единичку.
2. декодер выставляет линию и конкретное машинное слово
3. в самой ячейке(вот в этом буквально прямоугольнике) ток может уйти в нолик и упасть в битлайн, а может остаться неизменным, а дальше уже усилитель расшарит эту разницу и интерпретирует сохраненный сигнал в 0 или 1.

Вот запись:

- Выставляем в data in нужные сигналы, а не все единички
- на битлайне выставляем единичный сигнал
- и ждем пока внутренние инверторы не будут притянуты в нужное состояние и не сохранят его
- декодер активирует нужную линию

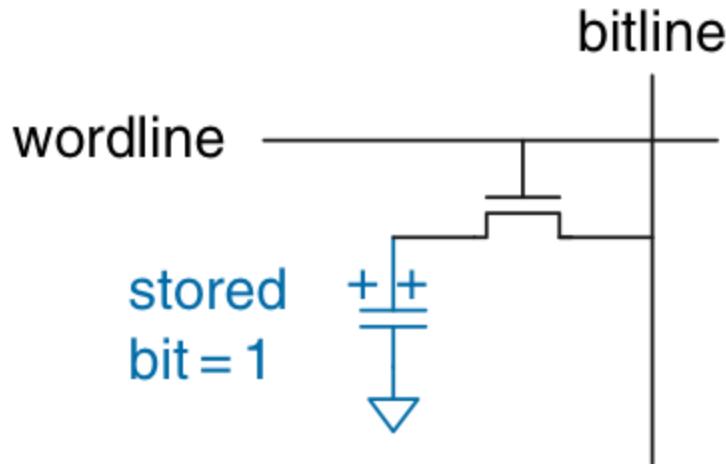
152) Каковы технологии реализации DRAM ячеек? Как происходит чтение и запись?

DRAM(Dynamic Random Access Memory) - динамическая память, которая хранит данные в виде заряда на конденсаторе. В отличие от sram она требует периодического обновления, но зато дешевле и компактнее.

Dynamic - сохраняем данные не в стабильном состоянии, а в конденсаторе, те какое то время там пролежит.

Характеристика:

- Состояние конденсатора можно считать лишь раз
- состояние конденсатор утекает
- требуется контроллер памяти для регенерации(что увеличивает длительность доступа, но блокирует доступ к памяти во время регенерации)
- один транзистор и конденсатор на ячейку памяти.
- один транзистор и конденсатор на ячейку памяти.



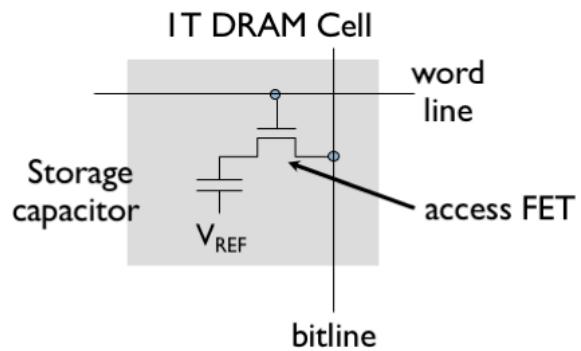
(a)

когда мы хотим записать в него нужное значение, то на битлайне мы выставляем нужное значение, заряд натекает в наш конденсатор(если 1), иначе ничо не выставляется, остается нолик()

- Writes: Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- Reads:

1. Precharge bitline to $V_{DD}/2$
2. Activate wordline
3. Capacitor and bitline share charge
 - If capacitor was discharged, bitline voltage decreases slightly
 - If capacitor was charged, bitline voltage increases slightly
4. Sense bitline to determine if 0 or 1
 - Issue: **Reads are destructive!** (charge is gone!)
 - So, data must be rewritten to cell at end of read



что касаемо записи:

- битлайн переводится в нужное состояние,
- активируется строка, транзистор открывается
- конденсатор заряжается(1) или разряжается(0)

чтение:

- битлайн предзаряжается(что то типа половинного напряжения)
- активируем вордлайн
- происходит перераспределение заряда: в конденсаторе была 1 - заряд перетекает на битлайн и его напряжение немного повышается, с 0 - наоборот
- усилитель фиксирует изменение
- данные перезаписываются обратно в ячейку(иначе всё сломается)
в современных кампьютерах стоит внешняя dram память, из за более высокой плотности + процессор с ней давно напрямую не взаимодействует(есть dma, кеши).

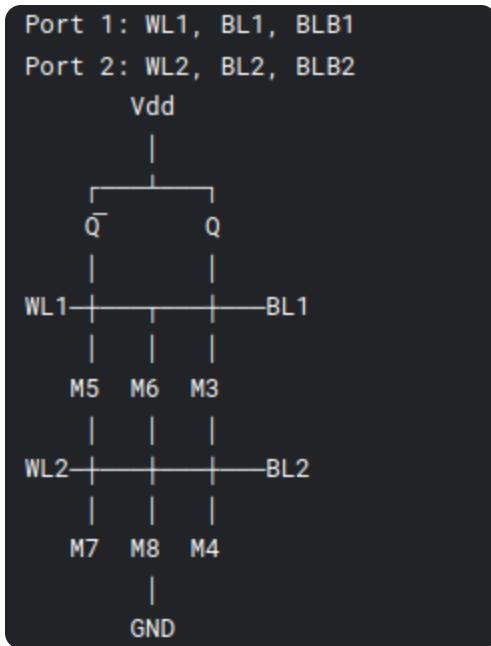
153)Каково устройство многопортовой памяти? Как связана площадь с количеством портов?

Многопортовая память - это память, которая позволяет нескольким устройствам(процессорам, блокам в SoC) одновременно читать и/или писать данные в одну и ту же ячейку.

Используется это в кеш-памяти CPU/GPU, сетевых процессорах

Классическая **8T-SRAM** ячейка для двух портов:

- **6 транзисторов** (как в 6T-SRAM) — триггер для хранения данных.
- **+2 транзистора доступа** (по 1 на каждый порт).



- **M1-M4** — ядро триггера (2 инвертора).
- **M5-M6** — транзисторы доступа для Порта 1.
- **M7-M8** — транзисторы доступа для Порта 2.

Каждый порт имеет свои **Word Line (WL)** и **Bit Lines (BL/BLB)**.

Порт 1 и Порт 2 могут независимо читать/писать (если не конфликтуют по адресу)

Площадь ячейки растет **почти линейно** с числом портов:

- **1 порт (6T-SRAM)**: 6 транзисторов.
- **2 порта (8T-SRAM)**: $6 + 2 = 8$ транзисторов.
- **N портов**: $6 + 2N$ транзисторов (для True Multi-Port).

Если два порта пытаются **писать в одну ячейку** — нужна арбитрация (например, приоритет одного порта).

Решение: **Pseudo Multi-Port** (мультиплексирование во времени).

Увеличение энергопотребления

- Многопортовая память **усложняется с каждым новым портом** (больше транзисторов, проводов).
- Площадь растет **линейно**, но технологии вроде 3D и мультиплексирования помогают сократить затраты.
- Используется там, где критична **параллельная обработка данных** (CPU, сетевые чипы).

в целом ответ гpt, так как в лекциях ни слова не было сказано

154)Что такое кеш? Каково его назначение, место в иерархии памяти и основные метрики?

Кеш - промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью.

цель кеша - уменьшить задержки доступа к информации и ускорить работу системы(если по пунктам, то сократить время доступа к памяти, снизить нагрузку на RAM, оптимизировать пропускную способность).

ну про иерархию памяти:

кеш находится между процессором и RAM.

Три уровня кеша:

- L1: вообще разделен на L1i(instruction) и L1d(data). Тоже маленькие и быстрые.
Работает на частоте ЦПУ
- L2: как L1(только он унифицирован), только чуть больше и медленнее(тут доступ 10-20 тактов)
- L3: общий для всех ядер, используется для синхронизации между ядрами(тут 20-50 тактов)

еще инфы про кеш:

- Кеш состоит из набора кеш-линий (блоков кеша, записей).
- Кеш-линия ассоциирована с элементом в медленной памяти.
- Кеш-линия имеет идентификатор (тег), определяющий соответствие.
- Доступ к памяти реализуется прозрачно для программиста.
- Память может быть изменена вне зависимости от кеша: DMA, другое ядро.

Вот что касаемо основных метрик:

2. Hit Rate(HR) - доля запросов, которые кеш смог обработать без обращения к RAM

Основная характеристика эффективности кеша

3. Miss Rate(MR) - уже наоборот

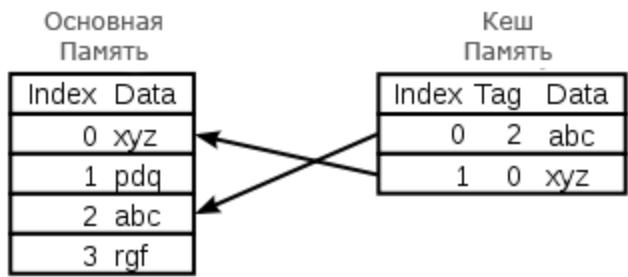
4. Latency(Задержка доступа): время на чтение и запись данных

5. Ассоциативность: Определяет, сколько мест в кеше может хранить данные по одному адресу(либо 1 строка на адрес, либо N вариантов размещения, либо любая строка кеша)

6. Размер строки кеша: объем данных, загружаемых за 1 раз

7. write policy(политика записи?): write-through - данные пишутся одновременно в кеш и RAM; write-back - сначала только в кеш, в Ram - при вытеснении

А вот функционирование кеша:



- Кеш таблица из кеш линий
- Кеш состоит из набора кеш-линий(или блоков кеша, записей - это всё синонимы)
- Кеш линия ассоциирована с элементом в медленной памяти
- Кеш линия имеет индентификатор(тег), определяющий соответствие
- Память может быть изменена вне зависимости от кеша: DMA, другое ядро

155)Что такое принцип локальности и каковы его виды (временная, пространственная) применительно к механизму кеширования?

Принцип локальности — это свойство компьютерных программ обращаться к данным и командам не случайным образом, а сосредотачиваться на определенных областях памяти в течение короткого времени. Этот принцип лежит в основе эффективной работы кеш-памяти, позволяя хранить часто используемые данные ближе к процессору для ускорения доступа.

- **Временная локальность (Temporal Locality)**
 - Если программа обращается к данным или инструкциям в определенный момент времени, высока вероятность, что они понадобятся снова в ближайшем будущем.
 - **Пример:** Циклы, повторное использование переменных.
 - **Применение в кешировании:** Кеш сохраняет недавно использованные данные, чтобы сократить время доступа при повторном обращении.
- **Пространственная локальность (Spatial Locality)**
 - Если программа обращается к данным по определенному адресу, высока вероятность, что вскоре потребуются соседние данные.
 - **Пример:** Последовательный перебор элементов массива, выполнение инструкций программы.
 - **Применение в кешировании:** Кеш загружает не только запрошенные данные, но и соседние блоки (кеш-линии), чтобы уменьшить количество промахов.

156)Как пошагово происходит процесс чтения данных через кеш-память процессора?

есть два варианта: обращаемся и находим тег, иначе ненаход

1. Тег найден -> кеш попадание(cache hit). Данные читаются в процессор из кеш линий
 2. Тег не найден -> кеш промах(cache miss). Запрашиваем данные из памяти или следующего кеша. Выбираем линию для замещения:
 - Есть пустая - подходит
 - Все заняты - принимаем решение о вытеснении линии
 - Длительность полученных данных - произвольна, тк:
 - Возможен многоуровневый кеш
 - возможна блокировка памяти(восстановление памяти, DMA)
- выше как это описано в лекции, вот вариант на словах:

157)Как пошагово происходит процесс записи данных через кеш-память процессора и какие существуют политики записи?

В случае **кеш попадания**, есть несколько вариантов развития. При записи данных в кеш есть:

1. Немедленная запись(write-throgh). Изменение вызывает синхронное обновление памяти. Иногда медленнее чем без кеша(Не самый лучший вариант, так как записали в кеш и сразу же продолжили писать в память, те не самая лучшая опция с точки зрения производительности. Она завязывает циклы процессора и циклы памяти друг на друга.)
2. Отложенная запись(write-back): Обновление памяти при вытеснении кеш-линии, периодически или по запросу.
 - Требует хранения признака модификации(те данные которые должны будут сохранены в память)
 - Группировка изменений, скрытие промежуточных состояний
 - Возможно неконсистентное состояние кеша и памяти. Для процессора - невидимое, для других устройств(DMA) - требуется принудительная запись.(в разных процессорных ядрах, в разных устройствах будут разные представления о данных)

3. Есть еще всякие гибридные варианты, по типу немедленная запись с буферизацией: почти как write-back, вместо вытеснения и только записи после вытеснения, мы будем записывать через n количество тактов. Мы в итоге невелируем проблемы немедленной записи, но и устранием некоторые проблемы отложенной записи.

Теперь в случае **кеш промаха**

1. Кеш с размещением(Write allocate, fetch on write)
 - Данные загружаются в кеш, после в них вносится исправление
 - Два обращения: запись вытесняемого и чтения необходимого
 - Поведение аналогично промаху по чтению
2. Запись без размещения(No-write Allocate, Write around)
 - Запись производится напрямую в память или через буффер (группировка изменений)

158)Что такое кеш-промах и какие существуют типы кеш-промахов?

Кеш промах - результат обрабатываемого запроса отсутствует в кеше и чтобы его получить необходимо обращаться к внешней памяти.
бывают разные типы:

1. Чтение данных: средняя задержка, может компенсироваться параллелизмом уровня инструкций
2. Запись данных: минимальная задержка, запись может быть отложена
3. Чтение инструкций: большая задержка, процессор простояивает в ожидании инструкции.

Кеш. Внутреннее устройство

Кеш – это память и логика. Логика реализует:

1. Поиск кеш-линии по тегу (компараторы).
2. Вытеснение, replacement (определение ненужной кеш-линии).
3. Предзагрузка (prefetch) данных и инструкций.
4. Взаимодействие с памятью (группировка операций и т.п.).
5. Синхронизации кешей разных уровней.

Больше данных → больше логики.

Масштабирование по времени или по площади:

- 2 линии, 2 компаратора → 1 такт,
- 2 линии, 1 компаратор → 2 такта.

159) Что такое ассоциативность кеша? Какова "структуря" адреса с точки зрения кеша?

Главная характеристика кеша - его ассоциативность.

Ассоциативность кеша определяет как данные из оперативной памяти распределяются и хранятся в кеш памяти. Это ключевой параметр, которые влияющий на скорость доступа к данным и количеству промахов кеша.(Или если говорить грубо, то ассоциативность кеша - то, как кеш линии связаны с памятью и каким образом построен механизм поиска нужного адреса в памяти)

тег - идентификатор области памяти, сохраненной в кеш линии.

когда процессор обращается к памяти, то адрес разбивается на несколько частей, которые определяют где и как данные будут храниться в кеше. Структура зависит от организации кеша и его параметров(будь то размер строки, количество наборов и всего подобного)

Адрес памяти обычно делится на:

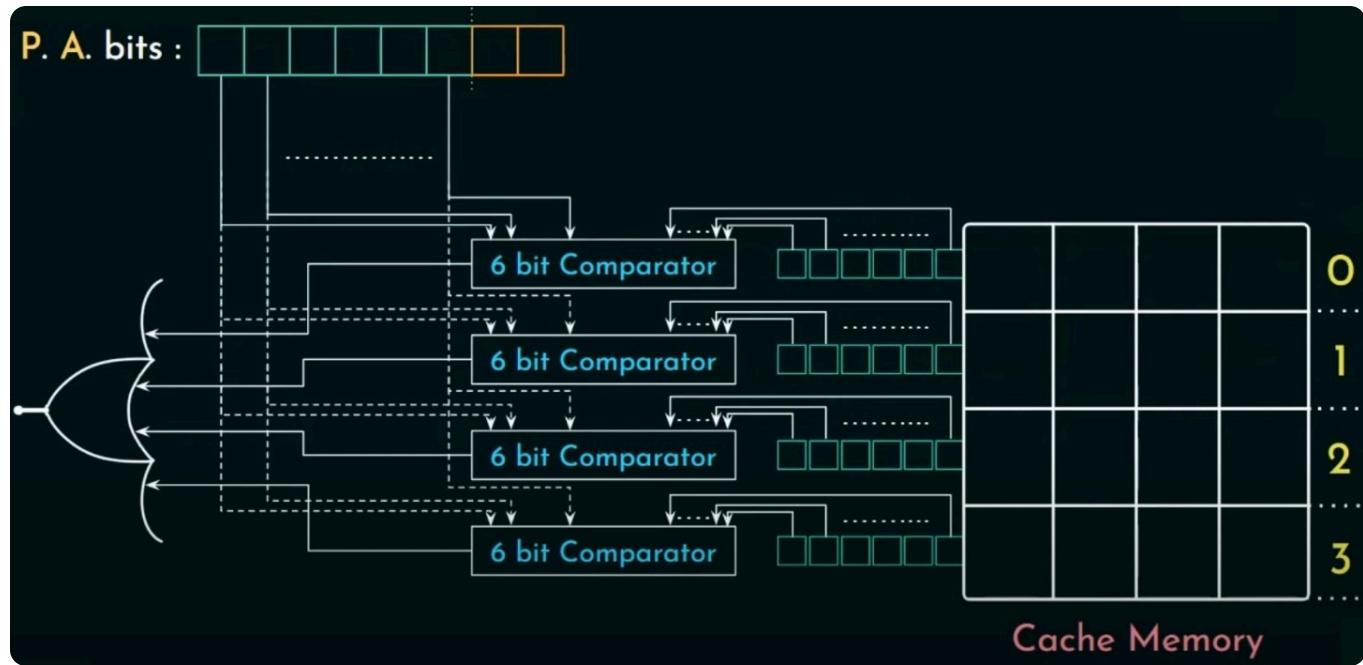
- тег
- индекс, определяет набор, в котором может находиться каждая строка

- смещение указывает на конкретный байт внутри кеш строки
Еще есть следующая инфа про адрес:
- Полный адрес - много памяти, много больших компараторов
- Младшие биты: не нужны если работать блоками, а не байтами
- Средние биты: циклически повторяются в адресах памяти(Принцип временной и пространственной локальности)
- Старшие биты: относительно уникальны(если используются)

160)Каковы особенности реализации полностью ассоциативного кеша?

Полностью ассоциативный кеш, тот кеш который работает наиболее быстро и наиболее эффективно среди остальных, потому что в каждый такт может убедиться и проверить есть ли нужный адрес в памяти.

в итоге получаем, что любая строка памяти может быть отображена в любую кеш линию. Отсюда получаем лучшую эффективность, но при этом много логики: компараторы и вытеснения.



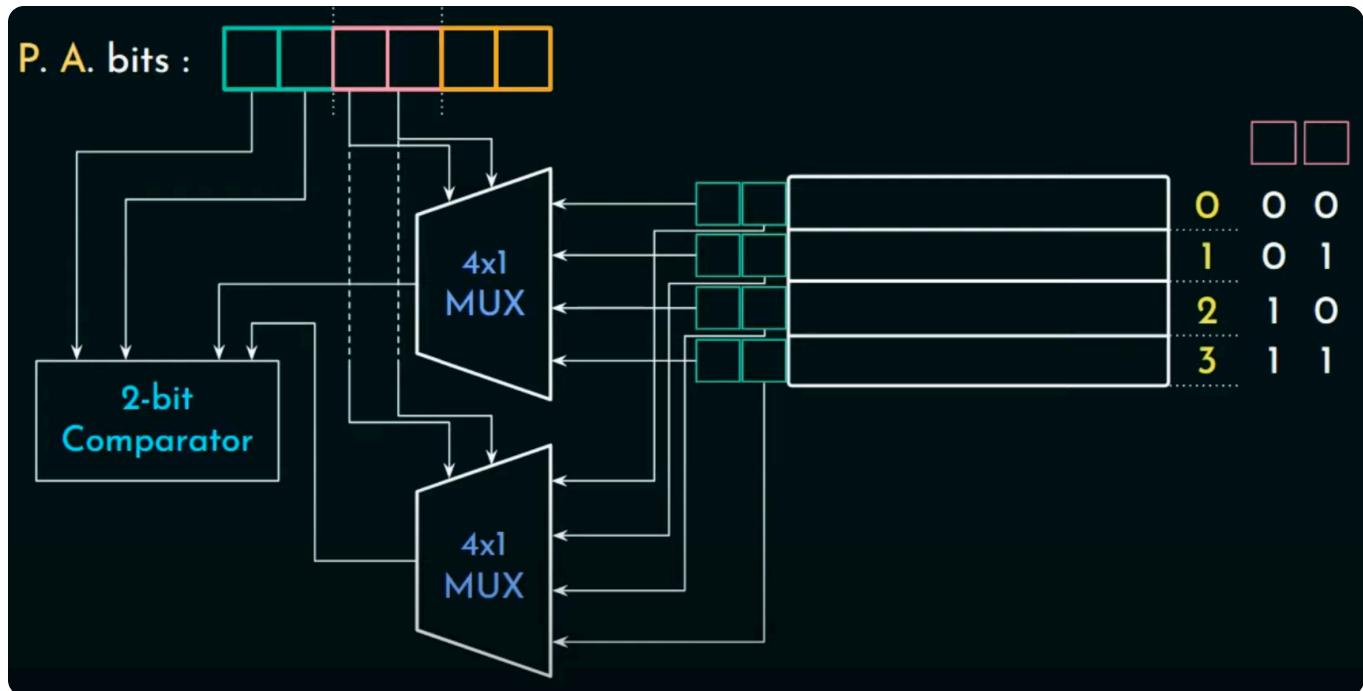
Младшие биты мы откидываем, в итоге по старшим битам ищем физические вхождения.
С точки зрения схемотехники работает оно так:

Вот кеш линия, рядом лежит тег. В итоге старшие разряды мы записываем в компаратор параллельно с тегом. Если кеш хит происходит, то через логическое или вылетит 1. Это всё круто тем, что эффективно. Нет искусственных ограничений, наложенных со стороны архитектуры. Неприятность только в том, что надо компараторов будет много. Условно их станет больше чем памяти для кеш линий, а это уже очень печально

161) Каковы особенности реализации кеша с прямым отображением?

Кеш с прямым отображением = Каждая ячейка может быть отображена только в 1 кеш-линию(одна кеш линия - много ячеек) или иначе каждый блок может быть размещен только в одной строго определенной строке кеша.

он базируется на том, что средняя часть адресации будет повторяться циклически + если есть принцип временной и пространственной локальности, то мы можем сказать о том, что мы либо попадем в начало такого сектора либо в его конец. Если мы линейно идем по памяти, то этот цикл и будет проворачиваться через наш кеш. Вероятность того, что внутри такого цикла нам нужно будет складывать данные из старших бит очень мала. В итоге получаем следующую логику:



Мы берем наш физический адрес, выкидываем младшие битики(они компенсируются выравниванием), средние биты заводим в мультиплексор и говорим, что в эту кеш линию могут быть записаны только адреса, в которых в адресах захардкожены только эти значения из центра и никакое другое.

Это позволяет упростить схемотехническую составляющую, а не как это было в полностью ассоциативном кеше. В итоге в кеш линии выводятся все линии в мультиплексор. Тег передается в компаратор вместе со старшими битами физического адреса.

В итоге сокращается количество используемой памяти и упрощается аппаратура. Недостаток такой, что мы не можем в кеше с прямым отображением принимать решения о том, какие данные надо вытеснить. Вытесняются верхние элементы из кеша(ну как это

показано на картинке).

Вывод таков:

- Меньше памяти для тэгов
- Один меньший компаратор
- Нет вытеснения(однозначно)
- Коллизия и техническая выгрузка

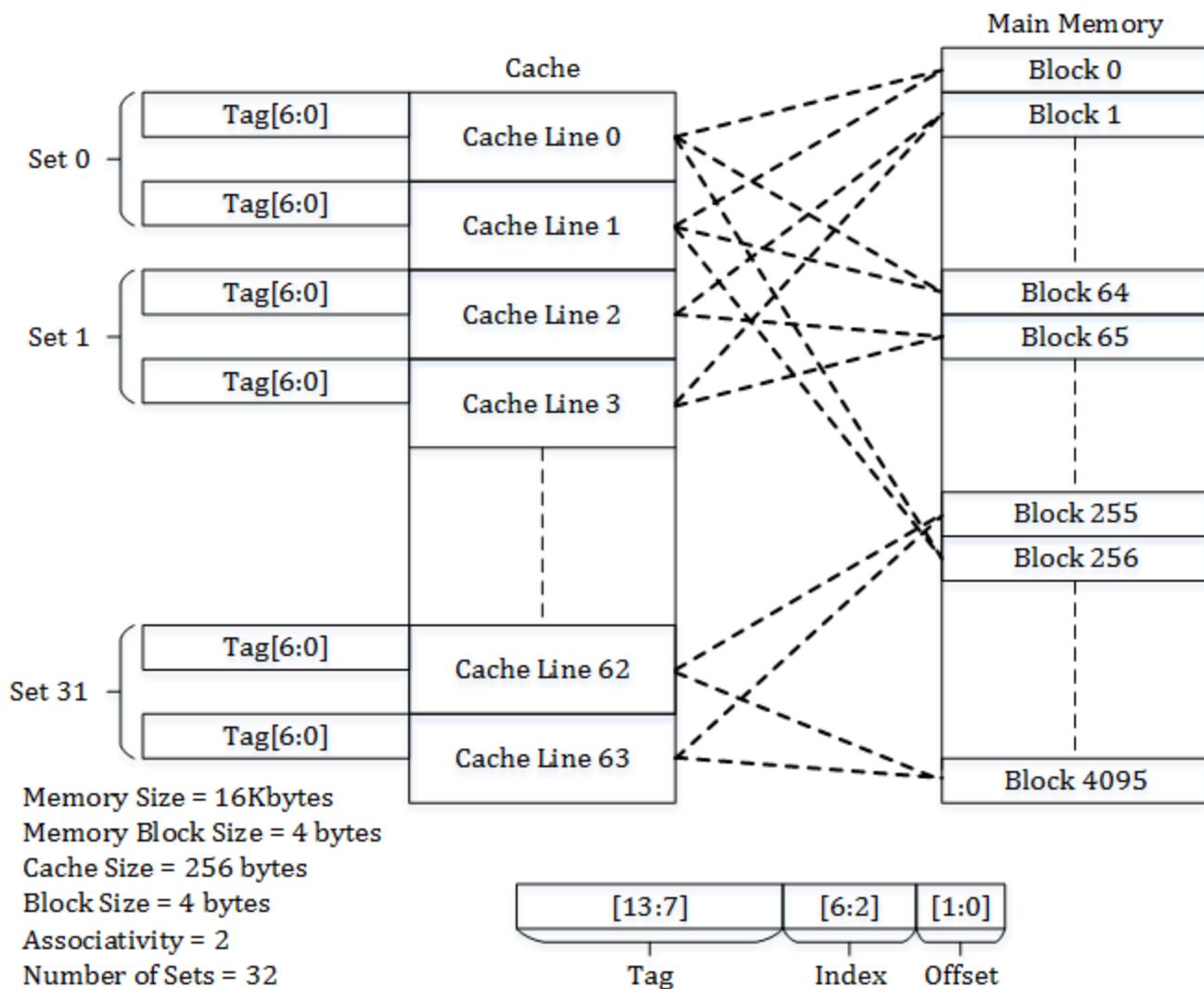
162)Что такое множественно-ассоциативный кеш и что определяет уровень множественности?

Множественно-ассоциативный кеш - по большому счету, мы ставим в параллель работать два/четыре/восемь кешей с прямым отображением.

Кеш с прямым отображением (1 банк) воспроизведенный N раз.

У него больше ассоциативность, ниже скорость, выше эффективность

Выбор банка осуществляется на основе алгоритма вытеснения



Уровень ассоциативности определяет количество строк в одном наборе.

Грубо говоря, каждый блок памяти из прямого доступа может попасть в две кеш линии. В итоге как раз и появляется эта возможность управлять вытеснением данных.

в итоге получаем, что у нас стоит кеш с прямым отображением(просто потому что его легко реализовать),

163)Как связана ассоциативность кеша с механизмом вытеснения и замещения?

Ключевой механизм: эвристика выбора наиболее не требующейся кеш линии среди доступных

Ассоциативность кеша напрямую определяет как часто возникают конфликты и какие алгоритмы требуются для эффективной работы. Чем выше ассоциативность, тем:

- Меньше принудительных вытеснений, так как у блока памяти больше возможных мест в кеше.
- Но сложнее механизм замещения, потому что надо выбирать одну строку из N кандидатов в наборе
В процессорах применяются алгоритмы:
 - LRU(least recently used)
 - Pseudo LRU.

164)Как работает механизм вытеснения и замещения LRU в кеше?

LRU (Least Recently Used) - стратегия замещения данных в кеше, при которой вытесняется самый давно использованный элемент.

Если грубо говоря, то маркируем записей временем последнего доступа, на картинке очень понятно показано в целом

Пример: A B C _D_ E _D_ F

A(0)
A(0) B(1)
A(0) B(1) C(2)
A(0) B(1) C(2) D(3)
E(4) B(1) C(2) D(3)
E(4) B(1) C(2) D(5)
E(4) F(6) C(2) D(5)

То мы буквально простым поиском определяем где наименьший индекс и перезаписываем то значение.

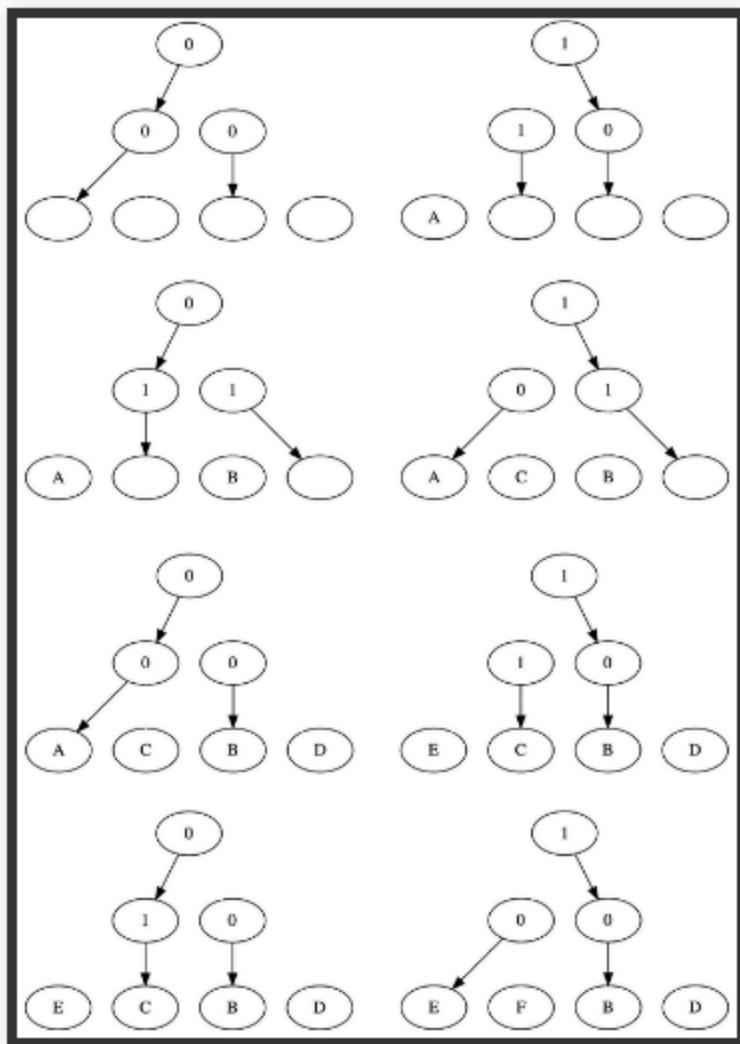
Аппаратно геморная история, реализовать сложно для больших ассоциативностей.

165)Как работает механизм вытеснения и замещения PLRU в кеше?

PLRU (Pseudo Least Recently Used)

на практике он не работает также хорошо, как настоящий LRU, но +- так же, но с точки зрения реализации проще. В нем не надо реализовывать счетчики всякие. Грубо говоря строятся бинарные деревья, маршруты по которым мы доступались. В итоге количество схемотеха у нас пропорционален логарифму от количества кеш линий, куда лучше чем

линейный рост в LRU.



A B C _D_ E _D_ F

- Бинарное дерево:
 - вершины хранят путь: `left(0) / right(1)`;
 - листья: кеш-линии.
- Поиск записи для вставки или вытеснения:
 - спуск по дереву согласно значениям в вершинах;
 - инверсия всех пройдённых вершин.
- Кеш попадание — инверсия пути от корня до листа.
- Значительно "легче" LRU

166)Почему необходимо использование многоуровневой иерархии кеш-памяти в современных процессорах?

Задачи кеша:

- Повышение скорости доступа
- Синхронизация проц. ядер
- Оптимизация интерфейсов

Свойства кеша определяют:

- технологией ячеек памяти
- устройством кеша(размер, ассоциативность и тд)

Способы организации многоуровневых кешей:

- Разделенный/унифицированный
- Включающие/исключающие
- Частные/общие

Выбирая разные характеристики кешей, мы будем получать огромную разницу в их размере.

Для реальной эффективной работы нам нужно выстраивать несколько кешей с разной специализацией(для данных и инструкций, специализированный для работы одного процессора или группы процессоров, для оптимизацией с работой памяти).

В итоге многоуровневая иерархия кеш памяти - это компромисс между скоростью и энергопотреблением и стоимостью реализации.

В итоге получаем:

- Чем быстрее память, тем она дороже. Без иерархии, если бы захуячили L1 кеши, то участились бы промахи. Если бы захуячили L3 кеши, то упала бы общая производительность
- Опять же временная и пространственная локальность

167)В чём разница между разделённым и унифицированным кешем и где они применяются?

Разделенный кеш позволяет выделить один кеш специально для данных, один кеш специально для инструкций. С точки зрения процессора: это дает два канала доступа к памяти, позволяет специализировать кеш для определенного паттерна использования.

В итоге:

кеш память может быть реализована одним или несколькими банками памяти. Иметь один канал доступа или несколько.

Принцип разделения:

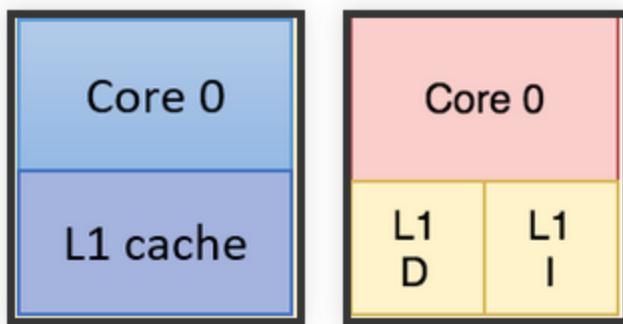
- По типу данных:
 - Кеш данных
 - Кеш инструкций
 - микрокод
- По процессорному ядру

Достоинства разделенного кеша:

- Повысить скорость доступа(больше каналов)
- Адаптировать устройство кеша под специфику использования

Недостатки:

- Усложняет процессора
- Конфликты между банками кеш памяти

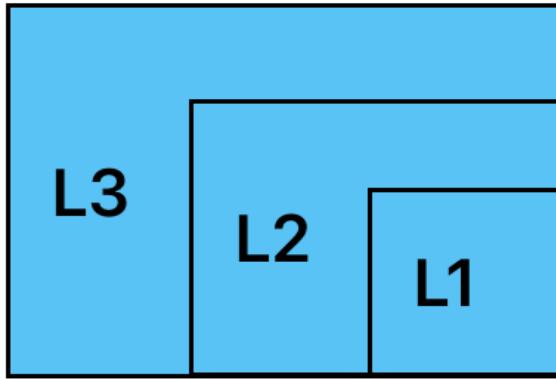
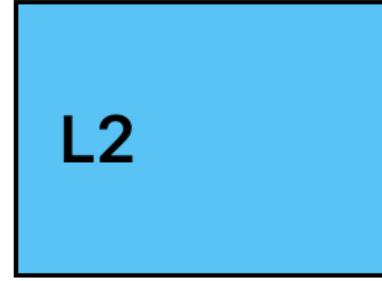
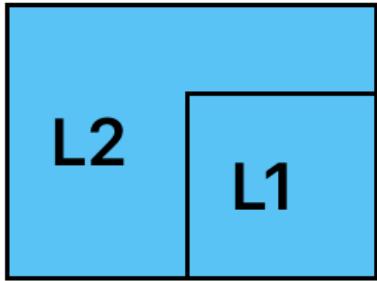
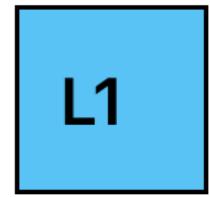
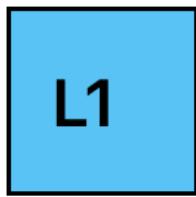


вот на картинке:

кеши самого низкого уровня, которые не шарятся между процессорными ядрами, они часто раздельными, L1D(data) и L1I(instruction), просто потому что это позволяет получить достоинства гарвардской архитектуры и не заплатить за это огромную стоимость.

168) В чём разница между включающим и исключающим кешем и где они применяются?

Вопрос тут один, должны ли кеши более высокого уровня иметь в себе копию данных кеша более низкого уровня



Inclusive

Exclusive

- Инклюзивная - данные кеш-линий дублируются. Доступ происходит быстрее(выгрузка L1 не может привести к выгрузке L2), но часть памяти теряется
- Эксклюзивная - данные не дублируются. Память используется эффективнее.
- Есть гибридные варианты, но я хз чо по ним сказать.

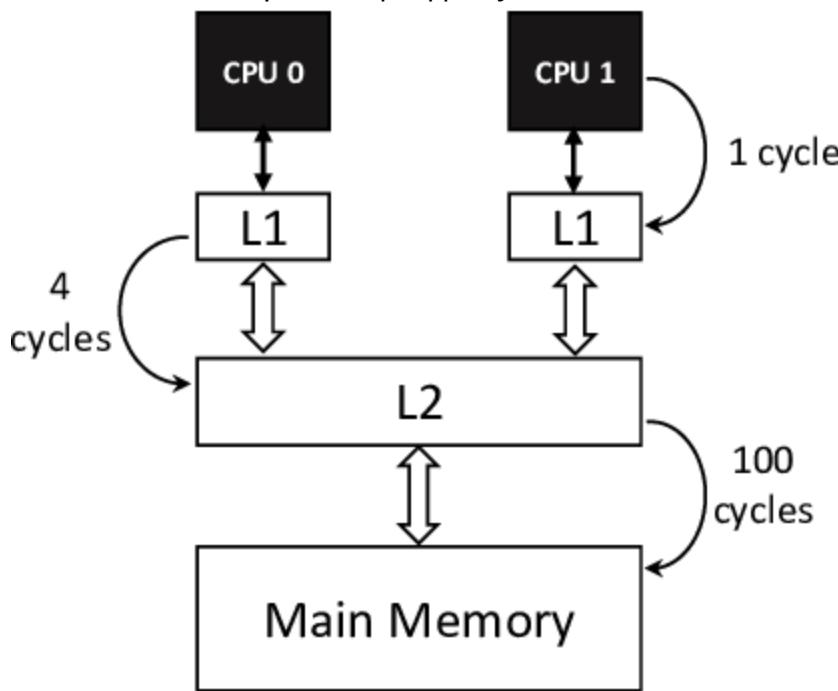
169) В чём разница между частным и общим кешем и как они применяются в многоядерных процессорах?

Частный кеш - ядро имеет эксклюзивный доступ к кешу.

- Скорость доступа ядра
- Дублирование и конфликты

Общий кеш - несколько ядер имеют доступ к кешу.

- Меньше кеш промахов
- Нет дублирования
- Постоянная синхронизация доступа

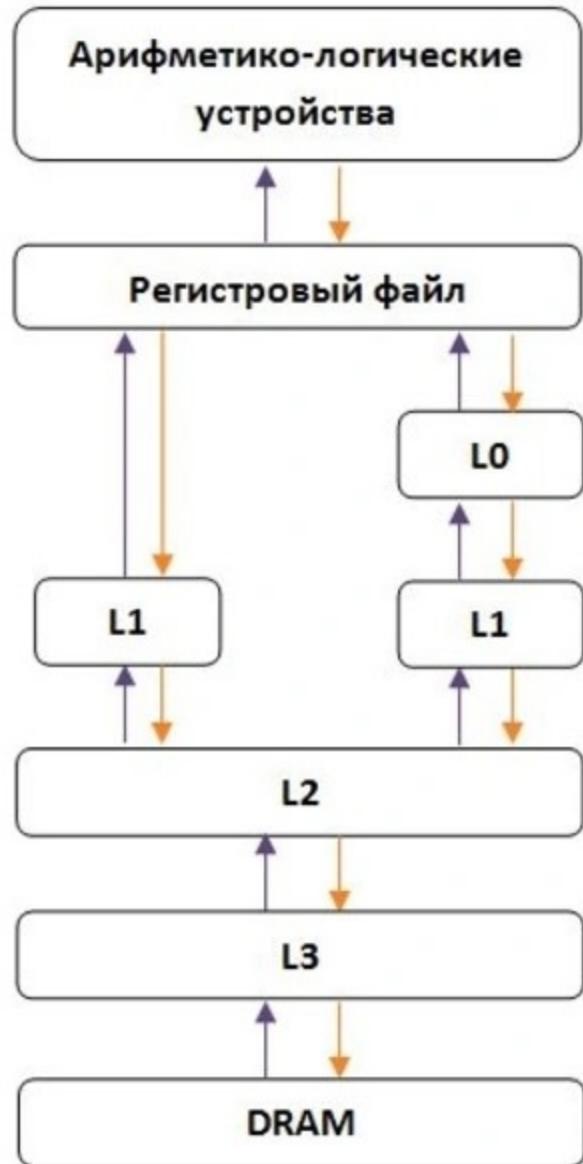


как применяются, можно привести пример перемножения матриц, и про частые обращения к памяти и плясать от достоинств и недостатков.

170)Что представляет собой типовой многоуровневый кеш: L1, L2, L3, L4? Каковы их назначение, тип ячеек, классификация?

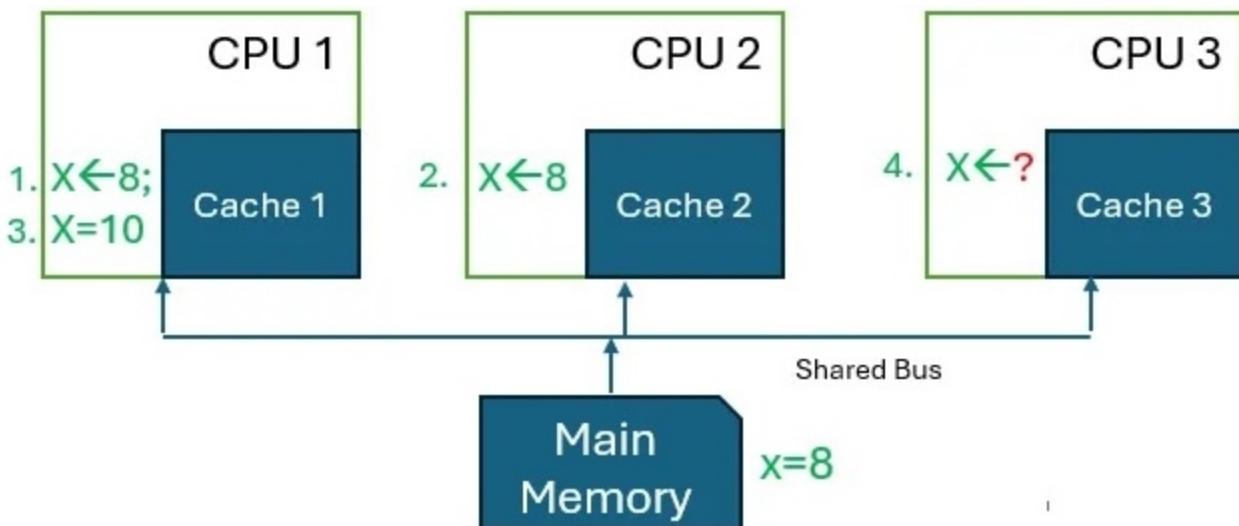
- L0 (опция) - специальный кеш для: стека, int/float чисел. Обычный доступен ща такт. (специальный кеш, для длинных операций умножения/деления, является микроархитектурой процессора, прибит гвоздями условно)
- L1 - быстрый, неотъемлемая часть процессора. Обычно разделен (Banked)(раздельный кеш, который существует на уровне отдельного процессорного ядра, задача которого обеспечить эффективный доступ к памяти)
- L2 - обычно часть процессора. От 128 Кбайт до 1-12 Мбайт. Обычно общий(shared) (часто бывает специализирован под конкретное ядро, но может и быть расшарен. Его основная задача - чтобы несколько ядер могли эффективно подступаться к памяти)
- L3 - до 24 Мбайт и более. Синхронизация данных в многоядерных процессорах. (направлен на многопроцессорные истории, когда физически на процессоре будет стоять два процессорных чипа, которые будут подступаться к 1 памяти)

- L4 (экзотика для серверов и мейнфреймов). Оптимизация интерфейса доступа к памяти.(ваще редка история)



171)Что такое когерентность кеш-памяти и каково её значение для многоядерных систем? Какие существуют возможные состояния кеш-линий?

Когерентность кеш памяти - согласованность данных между кешами разных ядер, гарантирующая, что все процессоры видят актуальную версию информации. В многопроцессорных системах без когерентности возникли бы ошибки, когда одно ядро изменяет данные, а другое продолжает использовать устаревшую копию из своего кеша.



вот на картинке показана сама проблема эта.

Для многоядерных систем когерентность:

- предотвращает конфликты
- без когерентности невозможны нормальные блокировки
- ну и синхронизация общих ресурсов

Состояния кеш-линий:

MOESI - Modified; Owned; Exclusive; Shared, Invalid

1. Модифицировано(Modified) - свежие уникальные данные(Значит в этой кэш линии будут лежать уникальные данные, которые еще не были скопированы в память. Если есть модифицированная кэш запись, значит только там, где она была модифицирована может писать и читать, остальные не могут)
2. Владелец (Owned) - Только одна запись в состоянии (Кто владеет данными, кто продолжает с ними работать и делать с ними что угодно, не надо перехватывать управление)
3. Эксклюзивное (Exclusive) - когда есть ячейка в памяти и существует только в одном из кешей внутри всего процессора. Если она там лежит, то можем перевести в состояние owned. Состояние это хорошее тем, что мы можем писать в эти данные и никого не оповещать об этом.
4. Разделяемое (Shared) - содержит свежие данные, могут быть копии. Свежие данные, их никто не редактировал и они разделяемые.
5. Не актуальное (Invalid) - не содержит корректных данных.

172)Как происходит обмен информацией о кеш-линиях через справочник? На сколько актуальна

проблема масштабирования?

Справочник в данном случае есть ничто иное как общий справочник с информацией о разделяемых данных. Грубо говоря, формируем наш справочник как отдельную память для кешей, с которой нужно сверяться при работе с кешем.

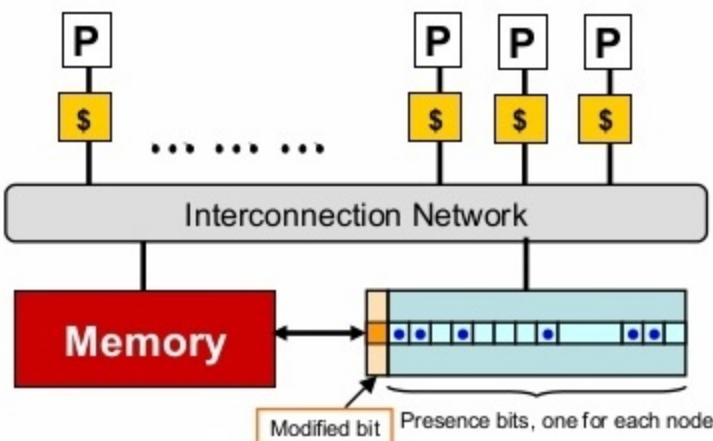
- Перед загрузкой записи осуществляется проверка через справочник
- При изменении каталог либо обновляется, либо аннулирует другие кеши с этой записью

Достоинства:

- Масштабирование, количество процессоров может быть велико

Недостаток:

- задержка запроса к справочнику



карочи проблема в том, что такой общий справочник это ужасный bottleneck. Чем больше процессорных ядер - тем чаще к нему будут ходить. (вот и вся проблема масштабирования)

173)Как происходит обмен информацией о кеш-линиях через отслеживание? На сколько актуальна проблема масштабирования?

Отслеживание

идея вот в чем, берем все наши кеши, вешаем на общую шину доступа к памяти. И каждый кеш все время слушает, что у нас там происходит: кто у нас доступается к данным, кто по каким адресам пишет и читает. Это можно делать параллельно

или другими словами:

Отслеживание операций других процессоров с памятью.

- Кеши отслеживают адресные линии на предмет обращений к данным своих кеш линий
- Если наблюдается запись - кеш помечает линию как неактуальную.

Достоинства: скорость

Недостаток: ограниченная масштабируемость

проблема вот в чем: каждый процессор, обращаясь к кешу, имеет свой адрес => если два процессора, то один кеш должен слушать соседний, если их 4, то кеш будет слушать 3 и так далее. В общем история которая практически не масштабируется. Количество логики и площади процессора просто неприличное, это пиздец как неприлично

174)Как происходит обмен информацией о кеш-линиях через перехват? На сколько актуальна проблема масштабирования?

Перехват - модификация отслеживания

которая более эффективная, но и более дорогая с точки зрения реализации.

идея вот в чем: а что если мы будем слушать не только кто куда писал и читал, но будем слушать еще и данные. Это позволит автоматически актуализировать кеш линии в кеше.

- Кеши отслеживают адресные линии и линии данных на предмет обращения к данным своих кеш линий
- Если наблюдается запись - кеш обновляет кеш линию
В целом быстро, эффективно, но требует безумное количество логики для многоядерных систем. В целом и проблема масштабирования та же, раз логика усложняется, так и площадь процессора больше требовать будет

175)Что такое CAP теорема? Какова область её применения и как она применяется к когерентности кешей?

CAP теорема:

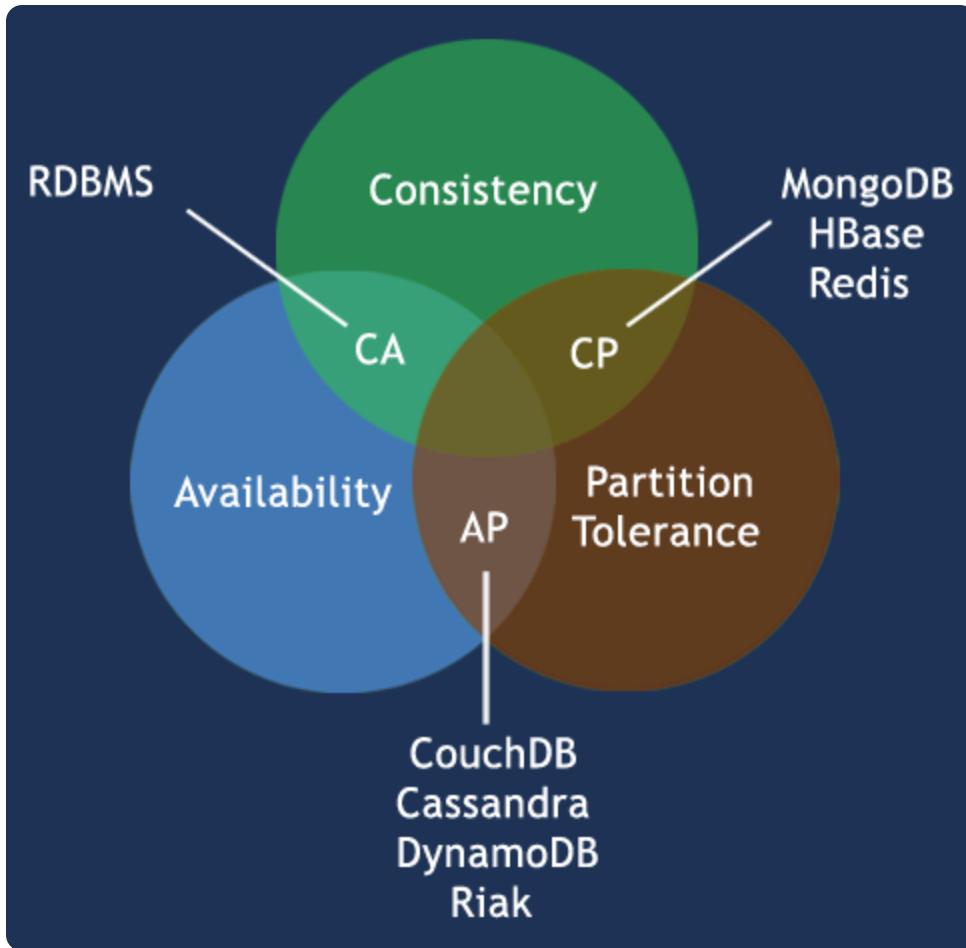
область применения: распределенные базы данных, облачные вычисления и микросервисы

В распределенных системах нельзя одновременно обеспечить:

1. Согласованность(Consistency) - каждое чтение получает последнюю запись или ошибку (Согласованность в кешах: если допустим в L1 ядра 1 записать данные и то

же самое сделать с другим ядром и не синхронизировать их между собой, то в один и тот же момент времени ядра прочитают разные данные по одному адресу)

2. Доступность(Availability) - каждый запрос получает ответ(без ошибок), но и без гарантии, что он содержит последнюю запись (Ситуация такая же, если мы можем прочитать те данные, которые записали это говорит о том, что мы можем дойти до памяти и прочитать их)
3. Устойчивость к разбиению (Partition Tolerance | brainsplit) - система продолжает работать, несмотря на произвольное количество потерь в сети между узлами.(хз чо тут сказать)



176)Что означают термины "Consistency", "Availability" и "Partition tolerance" в CAP-теореме? Опишите их.

описано в билете выше

177)Как кеш влияет на производительность памяти? Какие существуют способы оптимизации доступа, включая AoS и SoA?

В теории, кеш память прозрачна для программиста. Он не должен ожидать от нее неприятностей.

на практике имеем:

- meltdown(уязвимость которая затрагивает спекулятивные вычисления и кеш памяти.
Она позволяет программе читать чужие данные из защищенных блоков памяти, используя анализ времени доступа к кешу)
- Кеш и перемножение матриц
- Array of Structures(AoS) или Structures of Arrays(SoA)

Сначала про кеш и перемножение матриц:

идея в том, что для каждой ячейки памяти из матрицы А надо пройти по всей строке, а в матрице В по всему столбцу. И так обходя обе матрицы, мы должны перемножать их попарно. Эту историю можно реализовать либо через 2 цикла с перебором всех элементов матрицы + третий цикл, внутри которого будем считать. Проблема такой истории вот в чем: память складывается в кеш линейно, по адресам, по строкам доступ идет хорошо, как раз с принципом пространственной локальности, потому что идем по строчкам, постоянно ее повторяем. Со второй же матрицей ситуация куда хуже, там мы идем по столбикам, мы постоянно прыгаем по адресам. В итоге, для матрицы А всё будет хорошо, а вот для матрицы В(если мы попадем в шаг прямоотображаемого кеша), то у нас будет кеш промах при каждом доступе к памяти матрицы В. Классическим решением является - транспонирование матрицы. Отсюда получаем сильное повышение производительности.

Второй прикол уже связан с AoS и SoA, а именно с массивом структур или структурой массивов.

```
struct point3D {  
    float x;  
    float y;  
    float z;  
};  
  
struct point3D points[N];  
  
float get_point_x(int i) {
```

```
    return points[i].x;  
}
```

```
struct pointlist3D {  
    float x[N];  
    float y[N];  
    float z[N];  
};  
  
struct pointlist3D points;  
  
float get_point_x(int i) {  
    return points.x[i];  
}
```

Вопросы тут из разряда, какой вариант лучше, с точки зрения программиста, 1 вариант куда приятнее(хотя в целом многое зависит от алгоритма). Карочи очень многое зависит от задачи, которая от нас требуется.

Если хотим что то общее для информационной системы, то нам подойдет AoS.

Если хотим считать статистику, то вариант с SoA будет лучше.

178)Как кеш работает с виртуальными и физическими адресами?

Кеш-память с виртуальными адресами

Преимущества (ВА)

- Быстрота доступа.
Доступ к памяти минуя трансляцию адреса.
- Простота реализации.
Простота кеш памяти. Не требуется трансляция адресов перед проверкой кеш попадания.

Недостатки (ВА)

- Проблемы совместимости процессов. Требуется различать контексты из-за коллизий виртуальных адресов.
- Неэффективность при смене контекста. Смена контекста может потребовать очистку кеша для предотвращения конфликтов данных.

Кеш-память с физическими адресами

Преимущества (ФА)	Недостатки (ФА)
<ul style="list-style-type: none">• Универсальность. Физические адреса уникальны в рамках системы.	<ul style="list-style-type: none">• Задержки из-за трансляции адресов: Доступ к данным в кеше по физическому адресу требует предварительной трансляции виртуального адреса.
<ul style="list-style-type: none">• Консистентность данных: Физические адреса упрощают поддержание консистентности кешей и памяти.	<ul style="list-style-type: none">• Сложность реализации: Необходимость поддержки трансляции адресов увеличивает сложность управления кешем.

Возможны смешанные варианты:

- L1 – виртуальные адреса для минимизации задержек,
- L2, L3 – физические для универсальности и консистентности.

179)Что представляет собой концепция уровневой организации компьютерных систем и каковы её преимущества

она позволяет:

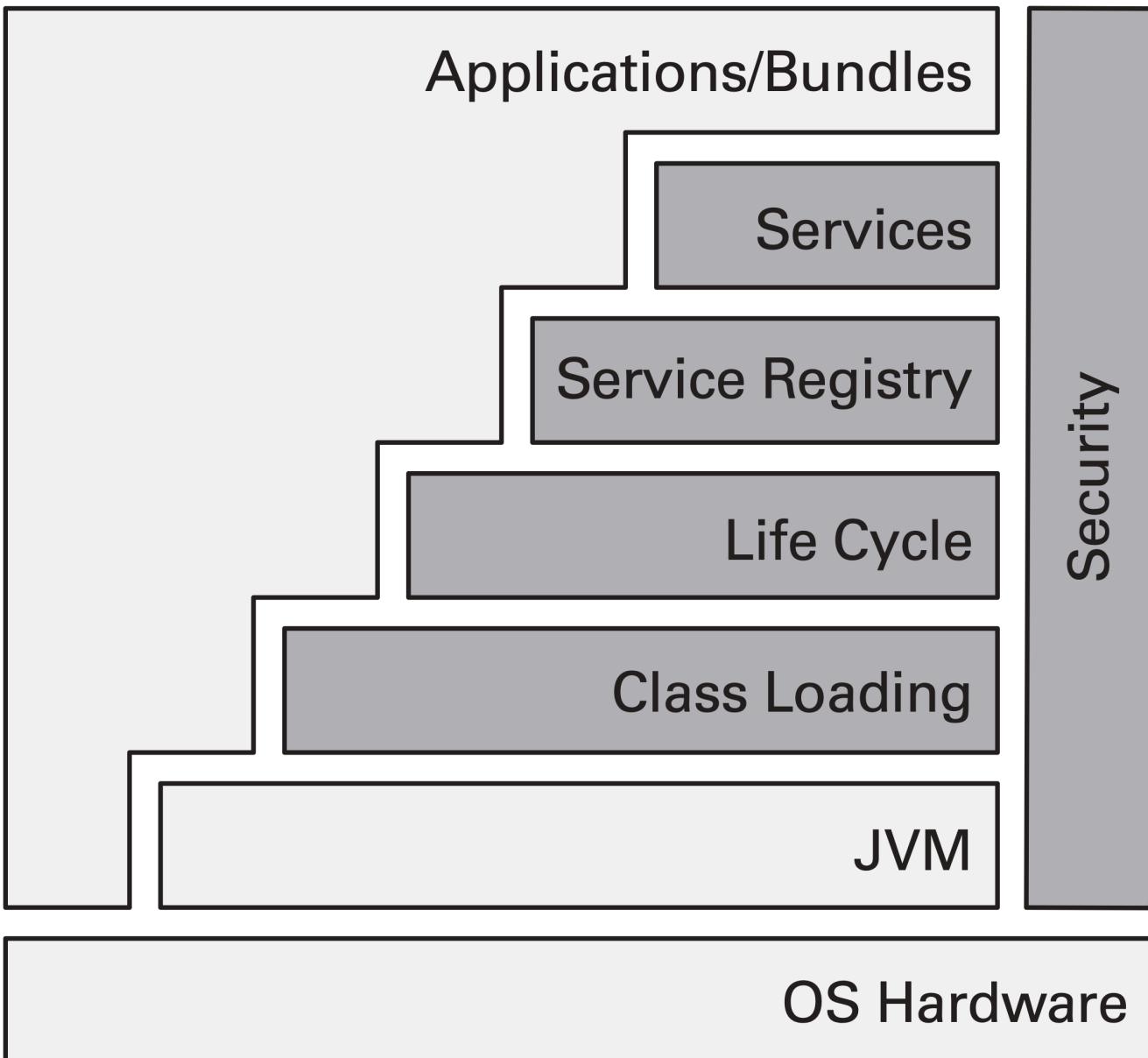
- упростить проектирование и понимание системы
- модульность и независимость(можно менять уровни не затрагивая другие)
- оптимизация конкретных уровней(можно оптимизировать уровни локально)
- Уровни ограничивают доступ к критически важным ресурсам

180)Какие существуют примеры уровневой организации компьютерных систем: Lava Flow, Layered Style, OSI Model?

Lava Flow - это плохой пример, антипаттерн или уже уровневая организация курильщика, карочи плохо так делать НЕЛЬЗЯ

Идея его в том, что развитие большинства информационных систем, а именно их кодовой базы можно описать по аналогии с вулканом. Карочи итеративно говнокод накапливается. Пришел разраб, написал код и ливнул. Пришел другой, исправил, изменил либы и всё подобное. И так дальше, и дальше и получаем нечитаемую кодовую базу.

Layered style(Уровневый архитектурный стиль) - такой подход к организации компьютерной системы, который строится на том, что мы выделяем в нашей системе отдельные слои. Каждый слой предоставляет набор сервисов, модулей, набор компонентов, которые можно вызывать на вышестоящем уровне. Позволяет строить структуры как на картинках:



Всё это позволяет довольно гибко формировать компьютерную систему из разных объемных частей и связывать их, а также понимать их уровни взаимодействия.

Вот еще пара приколов про Уровневый архитектурный стиль:

- Изоляция, модифицируемость и переносимость
- Управление сложностью и структура кода разработчикам
- Разделение интересов

[OSI Model \(Open System Interconnection\)](#) - идея в том, что есть 7 уровней, через которые должен пройти любой сетевой пакет, от приложения до физического уровня. Уровни эти чтобы обеспечить модульность и вариативность вот сами уровни:

1. Application Layer (прикладной)
2. Presentation Layer (уровень представления)
3. Session Layer (сеансовый)

4. Transport Layer (транспортный)
5. Network Layer (сетевой)
6. Data Link Layer (канальный)
7. Physical Layer (физический)

181)Что такое явление разделения на уровни (disaggregation) и каково его современное положение? Что было до него?

disaggregation(разделение на уровни) - историческая явление, связанное с тем, раньше старые филиалы, с их организацией вычислительной техникой: каждая компания имела весь стек технологий как свой продукт(как пример IBM сейчас). Весь стек от железки до ПО. Это круто с одной стороны, потому что можно сделать модули идеально адаптированными друг к другу, нет проблем с интеграцией и относительно легко ломать обратную совместимость.

Со временем, каждый кусок компьютера сильно усложнялся, поэтому каждый уровень и фрагмент компьютерной системы сильно усложнился и рынок перестроился. В каждой компании стало сложно разрабатывать условно свою собственную ОСь, поэтому стали появляться компании, которые специализируются конкретно на ОСах и тд. В общем случилось расслоение индустрии по уровням, где на каждом уровне существуют свои экосистемы, уже в рамках которых идет наслаждение новых технологий и уровней.

Опять же вспоминаем принцип Седова, который тут очень хорошо заходит.

182)Почему необходимо нарушение уровневой иерархии в современных системах? Приведите примеры.

Так как мы расслоили нашу архитектуру по уровням, мы получили большую проблему, связанную с тем, что стало гораздо сложнее обеспечить какие то сквозные свойства через всю нашу систему(как пример: производительность. Или дырки на уровне аппаратуры не позволяют обеспечить безопасность на вышестоящем уровне).

Требование реально времени для систем управления. Фактически, находясь на высоком уровне абстракции у нас нет инструментария, которым мы могли бы контролировать что происходит в машине. И всякие случайные временные задержки могут руинить наши задачи(но сами эти задержки мы никак не пофиксим к сожалению)

Если же мы хотим обеспечить реальное время, то у нас есть два исхода: первый накинуть еще больше мощности(буквально берем вычислитель который кратно превосходит наши требования, это пиздец идея. Но из за запасов производительности мы переплачиваем за железо и + энергопотребление)

второй вариант: производительность. Пишем программчику на питончике, у нас всё круто, но если нам нужна производительность мы начинаем писать какие нибудь утилиты на си, которые собираются в виртуальную машину, которая их вызывает, чтобы просто снизить затраты интерпретацию и добиться максимальной производительности.

Карочи вывод: уровни есть, но мы вынуждены часто нарушать их или размывать, просто чтобы обеспечить системные свойства.

183)Какие проблемы возникают при обеспечении реального времени в современных процессорах?

из билета выше, там кроется ответ в примерах.

184)Что такое низкоуровневый параллелизм? Приведите примеры.

Вспоминаем уровневую организацию, а именно уровень executable, на котором исполняются наши программы.

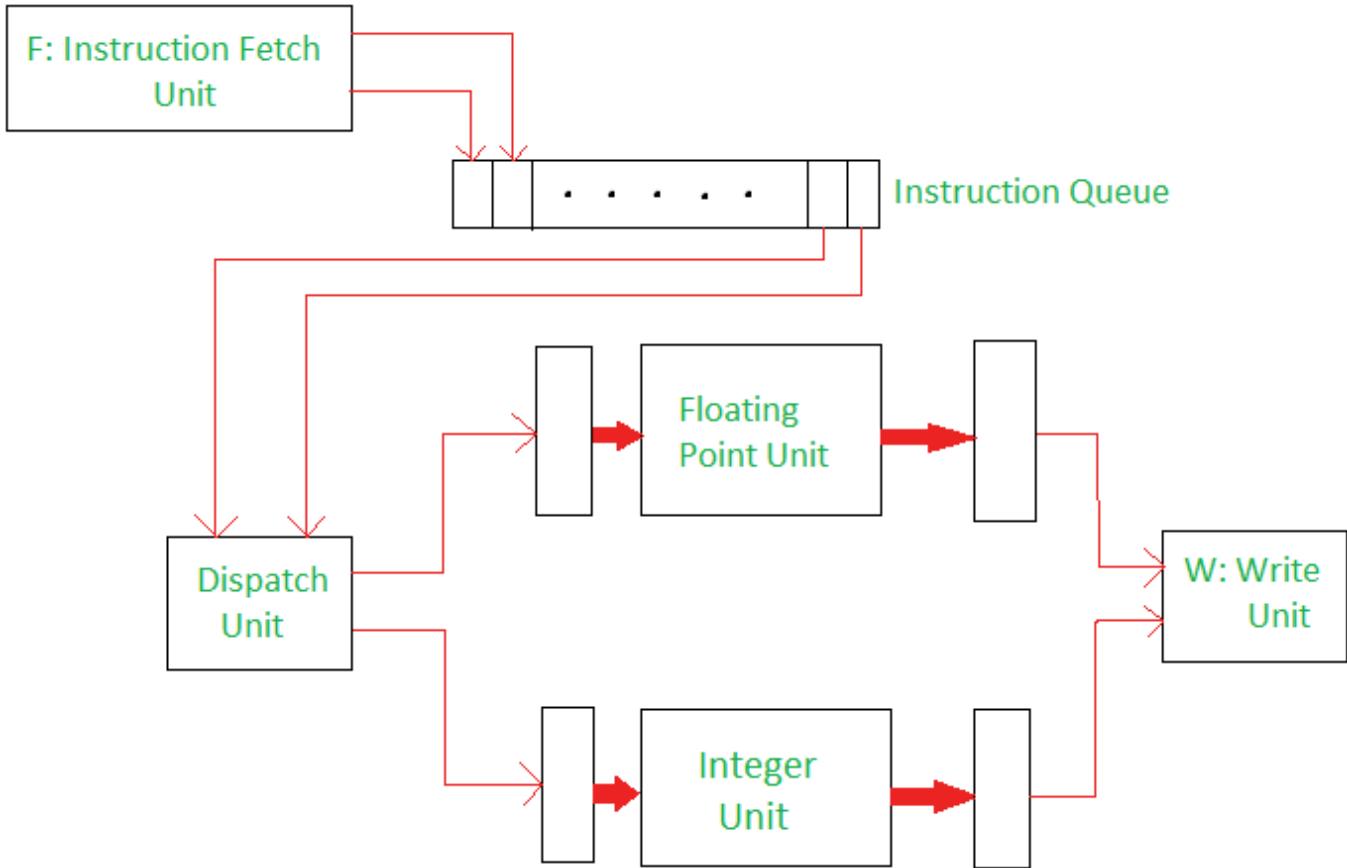
под низкоуровневым параллелизмом мы имеем в виду те виды параллелизма, которые являются прозрачными для программиста. Те не тогда, когда программист огромным количеством усилий обеспечить параллельные вычисления, а когда это можно получить относительно "бесплатно".

Из таких вариантов, мы уже рассматривали конвейеризацию(в билетах выше хорошо расписано). Но есть другой подход. А что вдруг мы не будем пытаться вытягивать инструкции на конвейер, а воткнем два сумматора, несколько АЛУ и тп. То есть буквально установим множественные функциональные узлы в ЦПУ(независимые функциональные блоки для арифметических и булевых операций, выполняемых одновременно) карочи, основная идея, в процессор толкаем много вычислительных узлов и там, где возможно будет делать параллельные вычисления.

185)Что такое суперскалярные процессоры и каковы

принципы их работы, достоинства и недостатки?

Суперскалярный процессор - это ЦПУ, способный выполнять несколько инструкций за счет параллельной работы нескольких исполнительных устройств, по типу алу, фпу, load/store блоков.



на картинке, есть два блока для флоат и инт чисел, соответственно мы можем выполнять их параллельно, так как для обоих типов чисел есть два отдельных блока с алу.

А как враще сделать так, чтобы такой процессор работал нормально. Ответ таков, что идет сочетание конвейерной организации и дополнительных особенной суперскалярных процессоров. В основе этого принципа лежит то, что фактическая работа некоторых операций будет занимать заметно больше чем 1 такт. У нас есть instruction fetch unit, из которого мы положили эту инструкцию в очередь. Из этой очереди мы можем анализировать налету все инструкции что там лежат. После dispatch unit, смотрит какие инструкции у нас лежат в очереди, смотрит какой блок свободен и отдает инструкцию в нужный юнит. Идея простая - реализуется почти всегда.

Вот еще более подобный вариант работы:

Сам процессор строится по конвейерному типу, его задача просто быстро подгружать инструкции за такт.

Инструкции читаются в очередь команд по порядку, декодируются и перемещаются на станции резервирования. Станции резервирования выполняют переупорядочивание(по мере доступности данных, по мере доступности вычислительных ресурсов.) Потом

собираем результаты воедино, и сохраняем результаты.

Достоинства:

- Рост производительности и сглаживание длительности выполнения инструкций.
- Повышение уровня загрузки ресурсов
- Совместимость с существующим машинным кодом
- Компилятор может устранить значительное количество конфликтов за счет сортировки инструкций

Недостатки:

- Конфликты по данным оказывают значительное влияние на производительность и сложность процессора.
- Высокое энергопотребление
- Проблемы детерминированности работы многоядерных процессоров.

186)Как пошагово выполняются инструкции в суперскалярном процессоре?

в билете выше описано

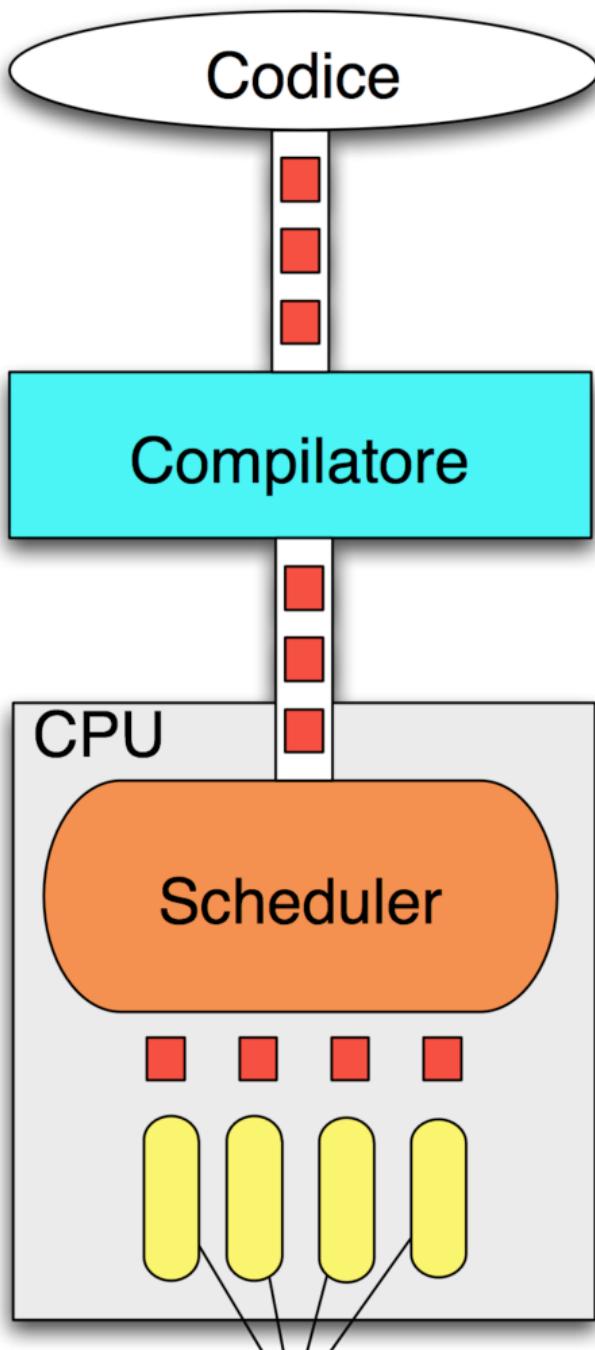
187)Что такое VLIW-процессоры? Какими достоинствами и недостатками они обладают?

VLIW (Very Long Instruction Word)

Вспомним RISC = упростим процессор за счет языков высокого уровня

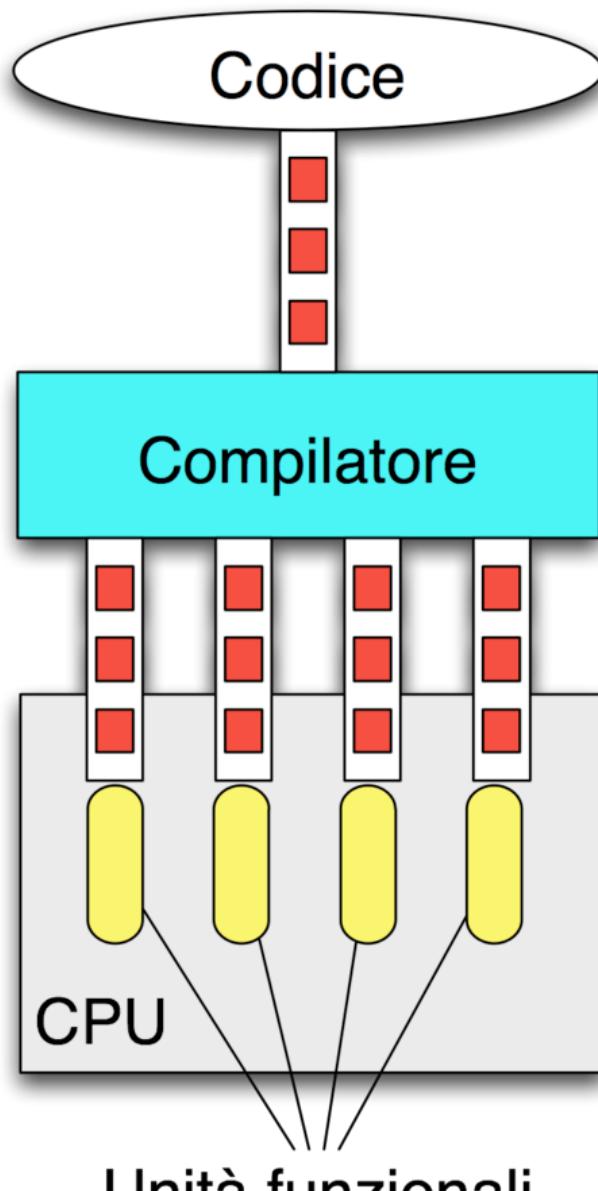
VLIW же = упростим процессор за счет переноса параллелизма инструкций в compile time.

Карочи тоже мем какой то, типа го сделаем огромные инструкции, внутри которой будет заложена вся логика для блоков из суперскалярных процессоров(то есть буквально зашьем логику из аппаратной в программу).



Unità funzionali

CPU Classica



Unità funzionali

CPU VLIW

на картинке должно быть понятно:

в суперскалярных: последовательность инструкций линейная и компилятор выдает ту же последовательность и в рантайме процессора есть нужна аппаратура, которая раскидывает их выполнение как надо.

в vliw же: пускай компилятор раскидывает один раз инструкции как надо

в итоге:

- У нас много ALU
- Объединены несколько инструкций в одну
- Убраны механизмы суперскаляра
- Компилятор знает всё -> максимальная оптимизация и параллелизм.

Достоинства

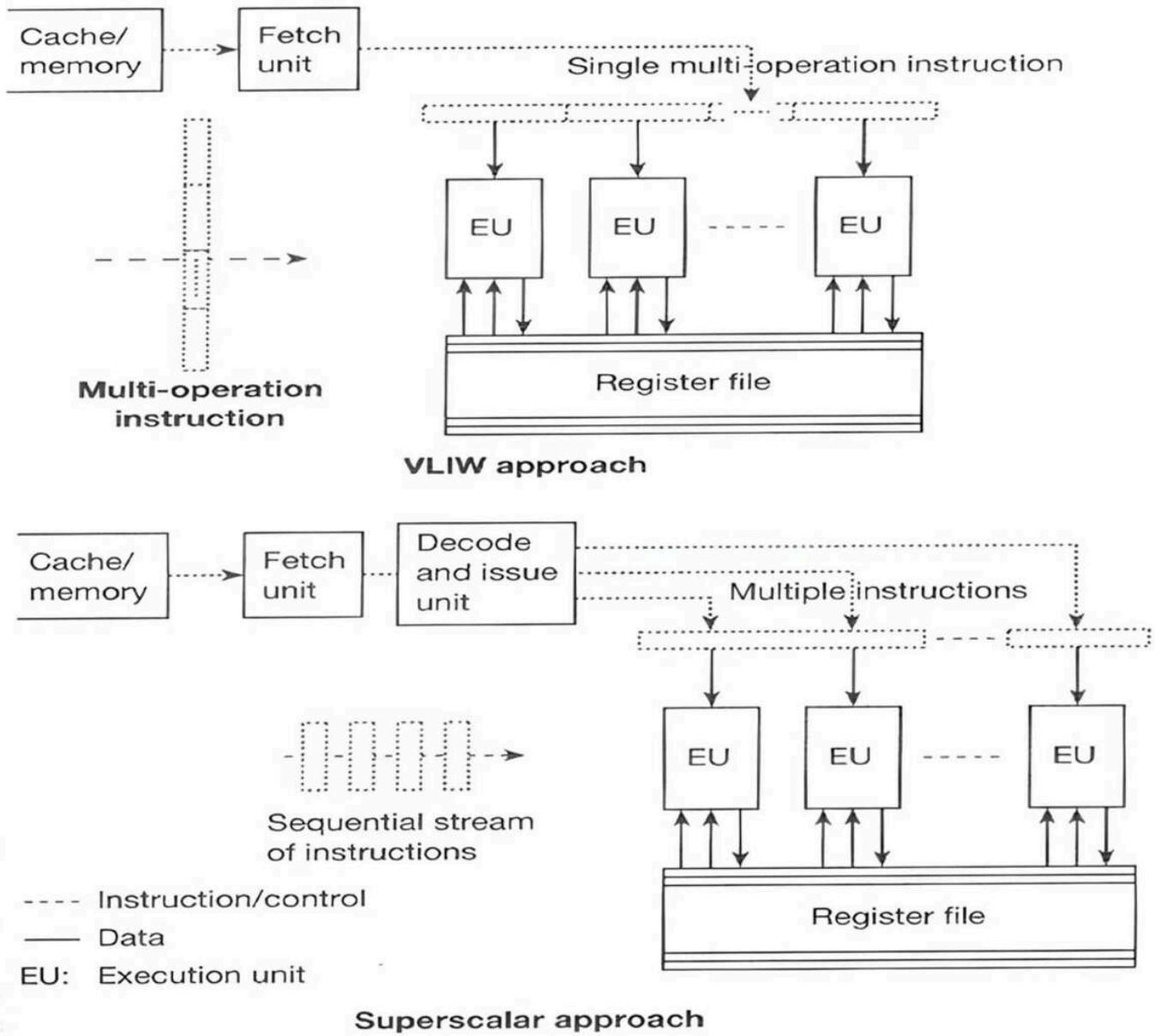
- Упрощение процессора, снижение энергопотребления
- Упрощение декодера, рост частоты
- Компилятор имеет больше информации о коде, он лучше знает, что параллельно

Недостатки

- Сложность компилятора
- Высокая нагрузка на каналы данных и регистровые файлы.
- Конфликты конвейера приводят к простою всех узлов.
- Проблемы условных переходов.
- Низкая плотность кода.
- Ширина команды — ограничена микроархитектурой

**188)Как соотносятся суперскалярные и VLIW
процессоры применительно к разным классам
задач?**

часть приведена в билете выше



в целом на схемке почти всё показано, что надо понимать:

влив в компайлтайме делаем распараллеливание, в суперскаляре в рантайме.

Есть еще прикол такой:

если мы хотим сделать параллельную среду исполнения, в случае с влив мы этого сделать не сможем, так как переключение с одной задачи на другую затратит все инструкции изначальной задачи, в суперскалярном можем перемешивать, так как ему всё равно откуда прилетела инструкция. В итоге в случае переключения потоков: влив будет строго разграничено, в суперскалярном они могут перемешаться в случае независимости по памяти и данным и у нас будет небольшой выигрыш по тaktам.

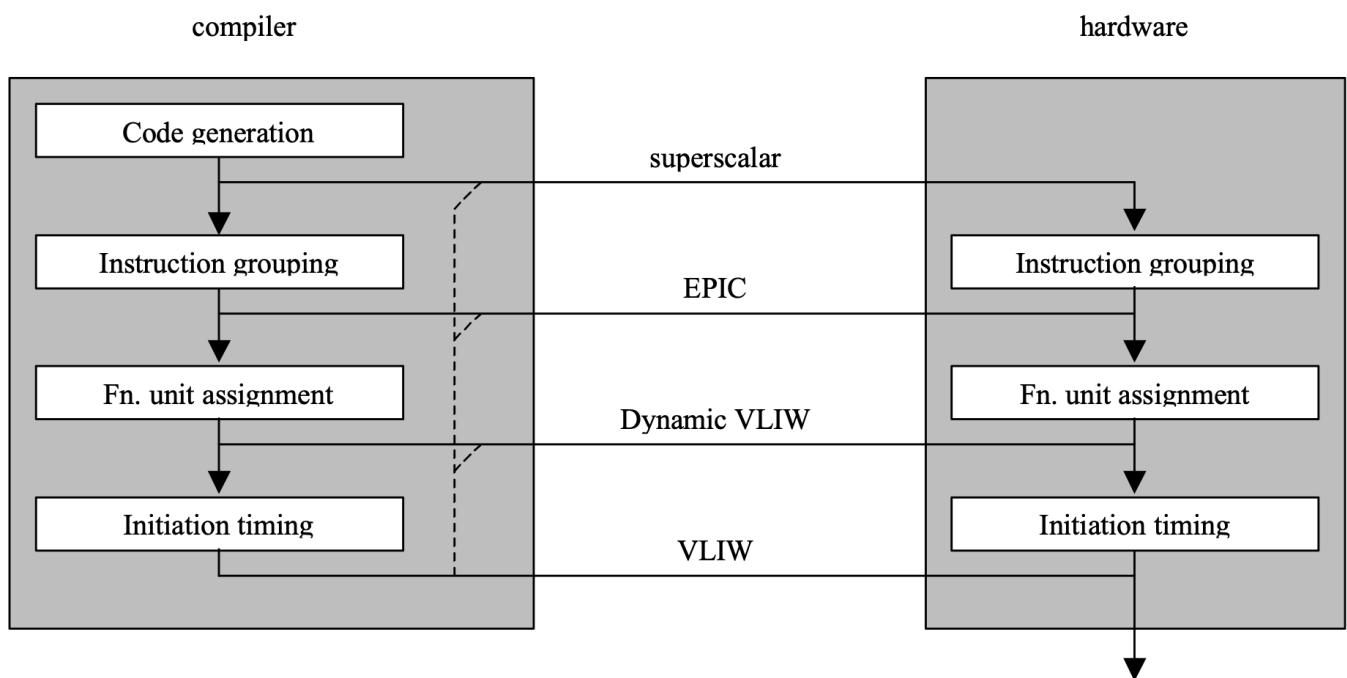
Есть и обратная сторона:

влив процессора выигрывают в сигнальной обработке данных и прочих встроенных системах. Так как мы четко знаем какую задачу наш процессор будет решать, это значит, мы можем сгенерировать довольно линейный код(с малым количеством условных

переходов и процедур) и тут влив сияет. (так как влив компилятор может распараллелить всё что угодно и мы тут выигрываем в производительности и энергопотреблении)

В итоге влив процессора используются в сигнальной обработке данных и встроенных системах. А суперскалярные побеждают там, где сложный код, с большим количеством ifов, процедур, ветвлений и постоянным переключением задач.

189)Какие существуют промежуточные варианты между суперскалярным и VLIW процессором? Какие проблемы они решают?



этапы трансляции и этапы интерпретации.

Два семейства промежуточных архитектур:

EPIC и Dynamic VLIW

EPIC (Explicitly Parallel Instruction Computing) - идея заключается в умном компиляторе, но процессор сохраняет часть динамической логики.(как сказал пенской: компилятор дает советы процессору что можно параллелить, а что нельзя)

Dynamic VLIW - карочи тут я ничего не понял, но что то очень похожее как EPIC.

190)Что такое барьеры памяти и в каких ситуациях они применяются?

В виду того, что инструкции могут переставляться случайным образом(либо это сделает компилятор, или в рантайме это сделает суперскалярный процессор), то у нас появляется дополнительный класс инструкций, которые могут интерпретироваться как на уровне компиляции, так и на уровне самого процессора, задача которых ограничить возможность реордеринга инструкций.

Барьер памяти - вид барьера инструкции, которая приказывает компилятору (при генерации инструкций) и процессору (при исполнении инструкций) устанавливать строгую последовательность между обращениями к памяти до и после барьера. Все обращения к памяти перед барьером будут гарантированы выполнены до первого обращения к памяти после барьера.

Карочи на япах уже рассказывали.

191)Какие ключевые элементы составляют структурное программирование? В чём отличие от кода на ассемблере?

Классическое структурное программирование строится на том, что мы спрямляем тот программный код, который у нас есть, те заключаем в определенную структуру. А именно: структурный блок, 1 точка входа, 1 точка выхода, между ними произвольное количество инструкций, которые должны выполняться строго последовательно. Сами эти структурные блоки организуются 4 способами:

последовательное, условное, конструкция цикла и возможность вызова(попадаем во внешний блок).

в итоге можно описать так: от инструкций и goto -> к структурным блокам и их последовательности.

192)Какие механизмы используются для реализации процедур?

зачем нам нужны процедуры:

- Переиспользование машинного кода
- Оптимизация работы кеша инструкций.

Работу с процедурами можно реализовать через:

- inline(не является процедурами с точки зрения рантайма, но является процедурами с точки зрения компиляции) - грубо говоря, как макроопределение в си, раскрываем определение и инлайним туда, где вызывалось
- goto - скорее искусственный вариант, в каком то месте кода располагаем набор инструкций, к которым можем перейти через goto столько раз сколько это нужно. Чтобы это стало настоящей процедурой, то нужно управлять как мы будем выходить
- call, return - база(тут в целом можно добавить про автоматическую память и стек) для любой процедуры, помимо того, что надо организовать инструкции(переход к их исполнению и выход). Надо еще решить вопрос с тем, а где лежат данные для вызова этих инструкций. Есть вариант сделать в лоб - это выделить рабочую память статически(но в таком случае наши процедуры нереентерабельные)

193)Что такое реентерабельность и почему она важна? Приведите примеры.

Реентерабельность функции - это повторно используемая процедура, которую могут использовать несколько программ одновременно.(Возникает когда есть параллельные процессы или система прерываний и вторая ситуация это рекурсивная функция) Сложности возникают с кэшированием, если размещаем данные в фиксированном пространстве, то это фиксированное место в пространстве не соответствует принципу локальности с данными. Вторая проблема со сбросами регистров(вопрос а как мы разделяем регистры между разными уровнями)

194)Как реализуется механизм рекурсии?

Рекурсия - это вызов функцией самой себя. Ее работа обеспечивается комбинацией стекового фрейма, управлением потоком выполнения и соглашения о вызовах.

Аппаратная основа рекурсии:

стек вызова. Обычно содержит аргументы функции, адрес возврата, локальные переменные, состояния регистров

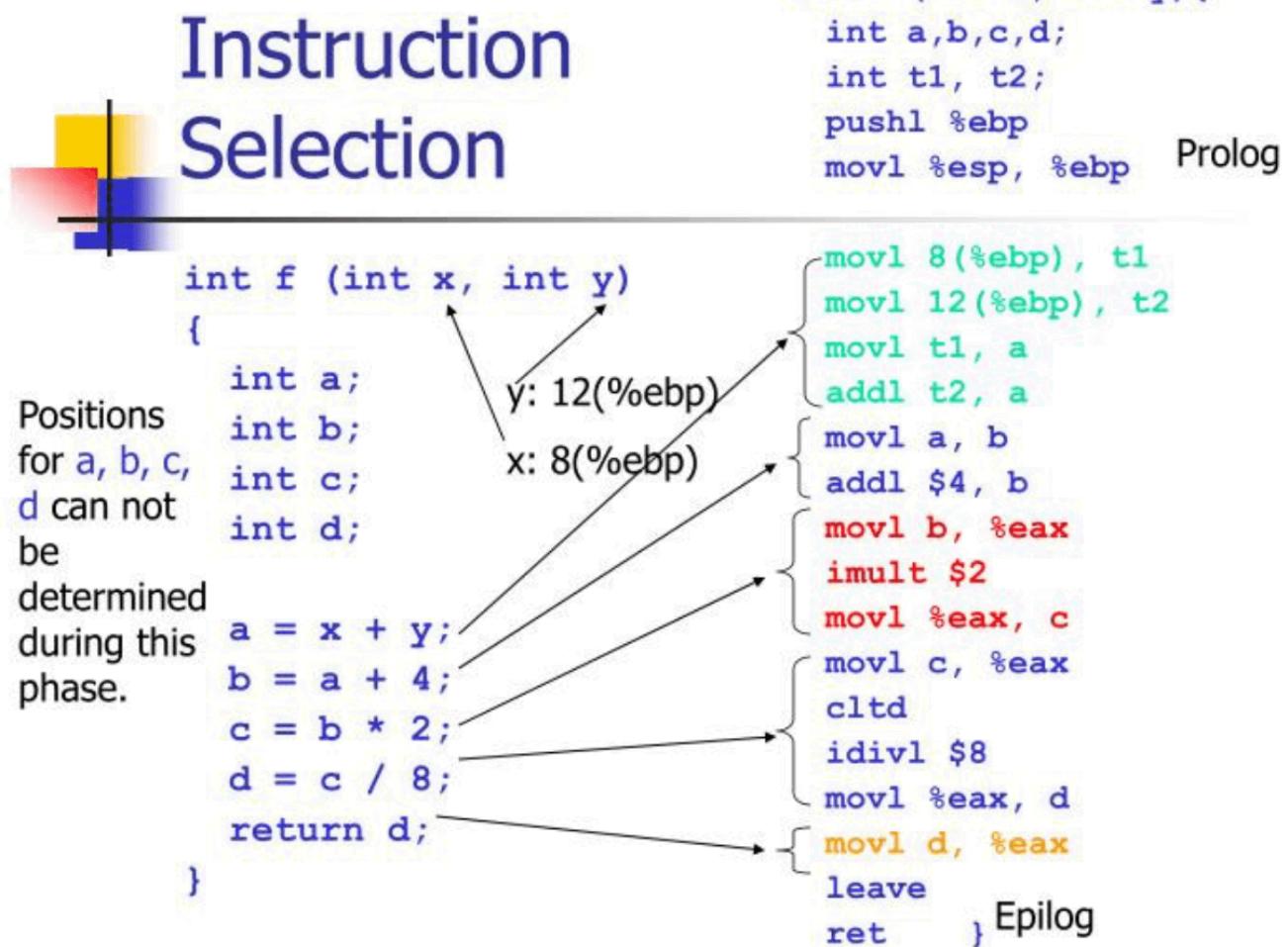
Каждый рекурсивный вызов создает новый стековый фрейм, который содержит аргументы и локальные переменные, изолированные для каждого вызова.

195)В чём заключается проблема распределения

регистров и как работает алгоритм раскраски?

Проблемы следующие:

- количество регистров конечно
- не все регистры одинаковы(особенно в CISC)



первая проблема это как их замапить их в наши регистры.

Один из возможных подходов это раскраска регистров. Идея вот в чем:

наша задача отобразить в виде графа все регистры, которые есть в нашей программе, после этого наша задача ребрами связать те регистры которые должны присутствовать в памяти одновременно и после этого предложить такую раскраску этих вершин, чтобы ни один цвет не был в соседних вершинах. Если удалось добиться такой раскраски, то если количество цветов \leq количество регистров, то наша команда может эффективно работать.

196)Что такое FPGA и каковы его преимущества и недостатки?

FPGA (Field-Programmable Gate Array) / ПЛИС (Программируемые логические интегральные схемы) - интегральная микросхема, используемая для создания конфигурируемых цифровых электронных схем.

Логика данных задается данными, а не конструкцией.

Для формирования конфигурации используется специальное ПО и Hardware Description Language (HDL): Verilog, VHDL

Он реконфигурируется, а не программируется.

то же самое, но другими словами:

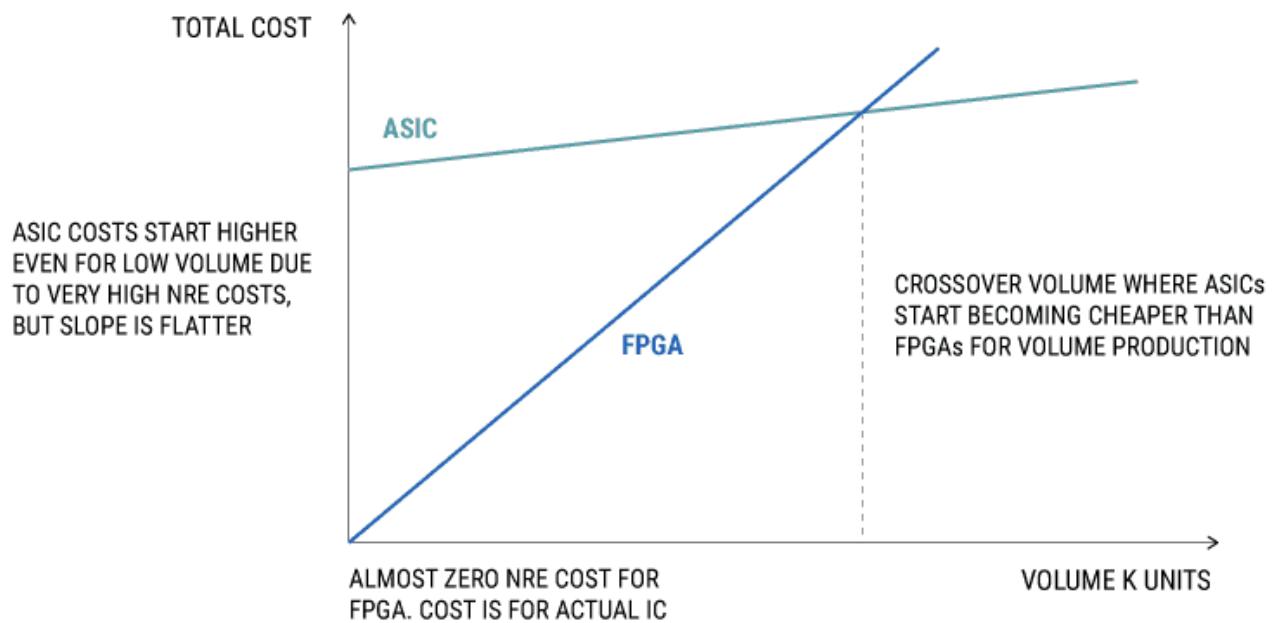
их идея и суть заключается в том, что они программным образом позволяют сформировать цифровую схему. Это не симуляция на уровне процессора, это интегральная схема, которой уже после производства можно задать ту логическую функцию/схему которую она будет реализовывать(задать таблицу истинности, указать как данные будут попадать в регистр и тп).

Преимущества:

- гибкость(можно реализовать любую цифровую схему)
- низкие задержки
- аппаратная изоляция

Недостатки:

- Ограниченные частоты
- нет универсальности(перепрограммирование под новую задачу требует полной перекомпиляции)



FPGA vs ASIC COST ANALYSIS

на графике стоимостьasic и fpga:

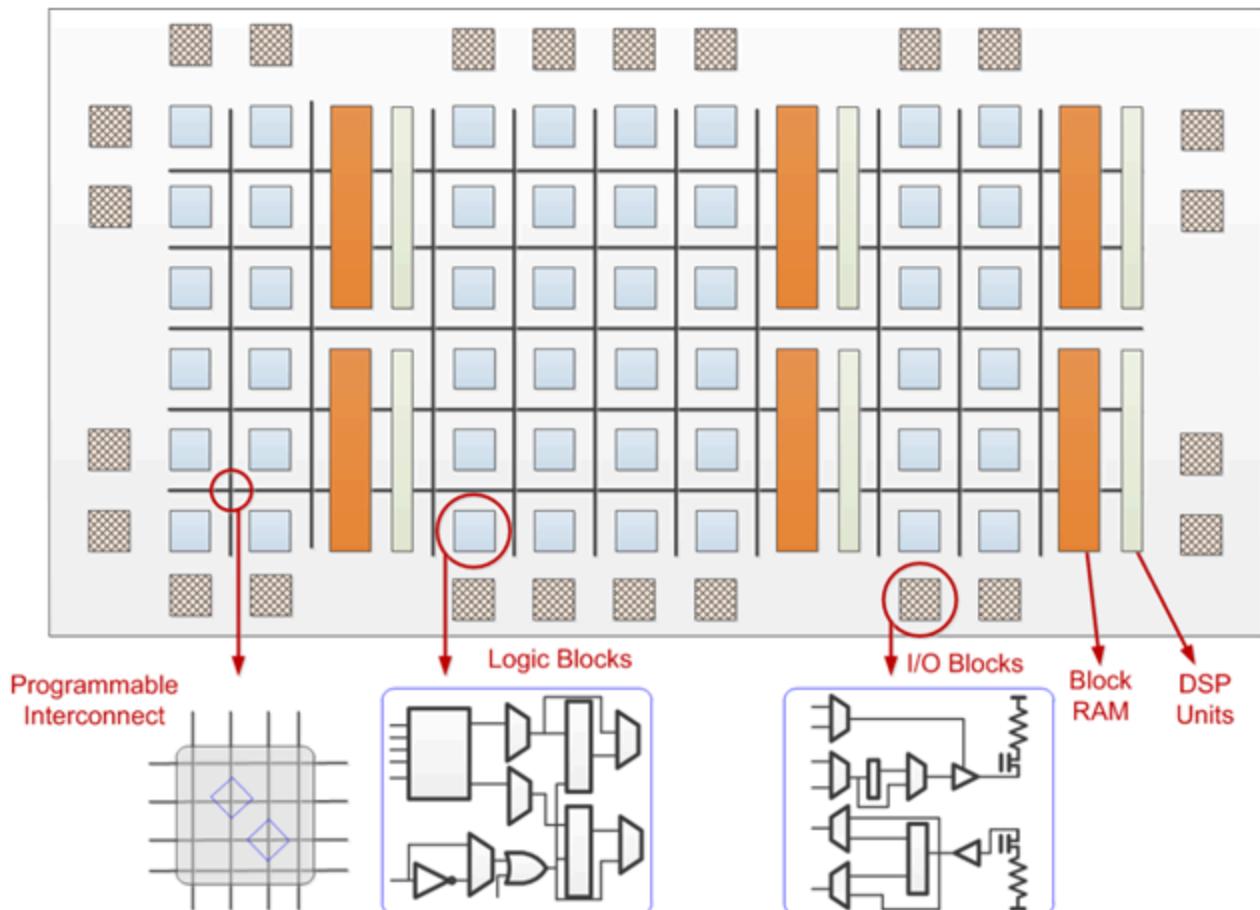
fpga потребляет больше энергии, работает медленнее, но в конечном итоге позволяет создать настоящую цифровую схему и заплатить за нее фиксированную стоимость.

197)Какие этапы включает в себя синтез для FPGA?

пишем на языке описания аппаратуры, где описываем картинку нашей схемы.

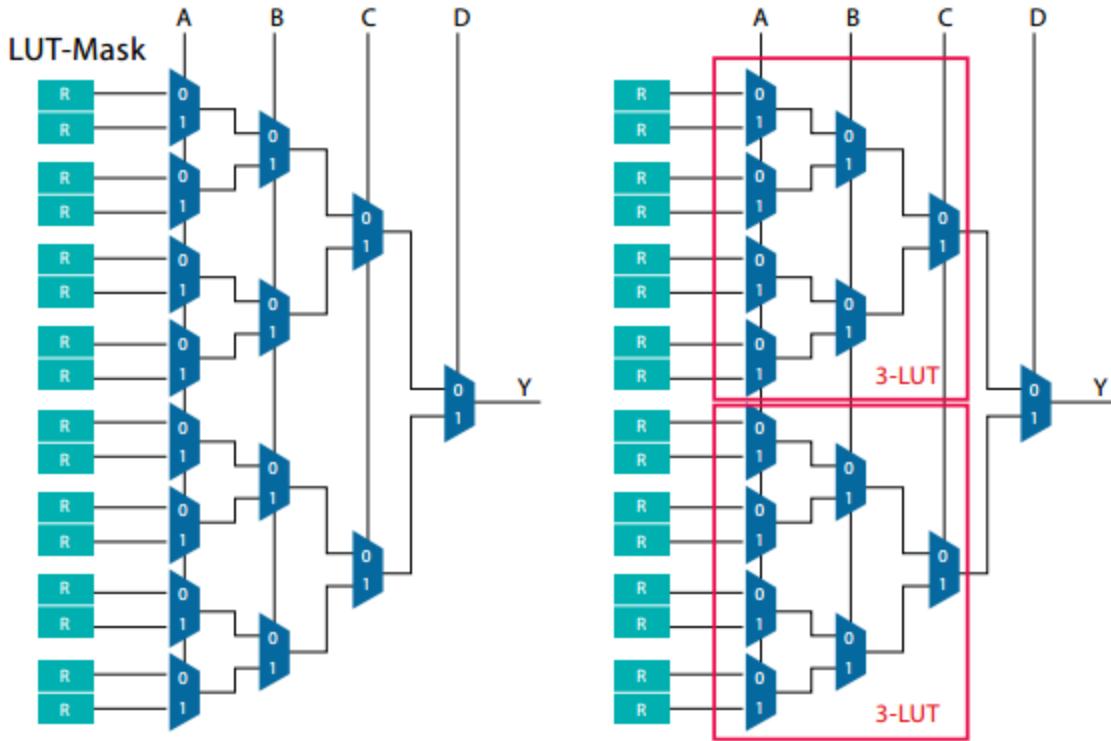
потом она транслируется в принципиальную электрическую схему(где нас не волнуют тайминги, длины проводов, нас интересует только то, что мы описали корректную схему) после чего вся эта конструкция транслируется в конкретный вычислительный базис аппаратуры, на которой может быть запущен.

198)Какие FPGA устроены изнутри?



1 картинка устройство плис

2 картинка Lookup tables.

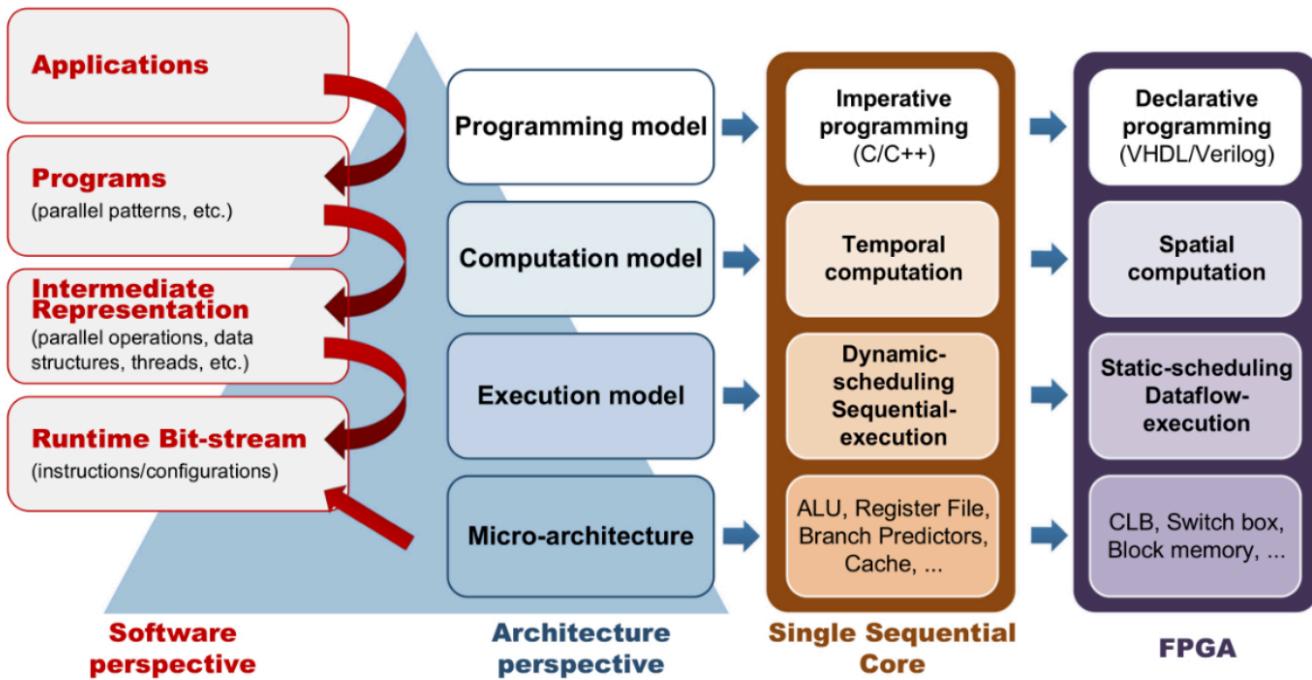


В простейшем случае можно представить ПЛИС, как матрицу, составленную из универсальных логических элементов, которые мы можем соединять между собой по нашему усмотрению

Состоит аппаратно он из:

1. Программируемого интерконнектора(буквально сетка из проводочек, программным образом определяем связаны или нет эти проводочки)
2. Логические блоки(в простейшем случае - кусочек памяти, который представляет из себя маленькую таблицу истинности)
имея интерконект и логические блоки можно уже собрать любую комбинационную схему.
3. Дальше появляются блоки ввода-вывода
4. Есть еще большие чипы памяти, которые позволяют реализовывать память, регистры, хранение каких либо данных
5. ДСП блоки: специализированные маленькие asic процессора, которые могут реализовывать аппаратные умножители/делители

199)Как соотносятся CPU и FPGA с точки зрения модели программирования, модели вычислений, модели исполнения и элементов микроархитектуры?



есть у нас 4 уровня рассмотрения нашей системы:

applications (уровень алгоритма с каким мы хотим работать)

programs (структура программы)

intermediate representation(то как мы будем представлять это внутри)

runtime bit-stream(по сути наш CU и datapath и как его элементы взаимодействуют)

модель программирования:

для CPU - императивное программирование(C/C++)

для FPGA - декларативное программирование

Модель вычислений:

Для CPU - временные вычисления

Для FPGA - пространственные вычисления

Модель исполнения(как мы понимаем какой блок какую функцию вычисляет):

FPGA - статически до запуска системы фиксируем предназначение каждого элемента, только появляется какой нибудь рантайм

CPU - полностью динамическое планирование

Элементы микроархитектуры(сильно завязан на внутренней детали реализации):

FPGA - логические блоки, кусочки памяти

CPU - алу, память, регистры...

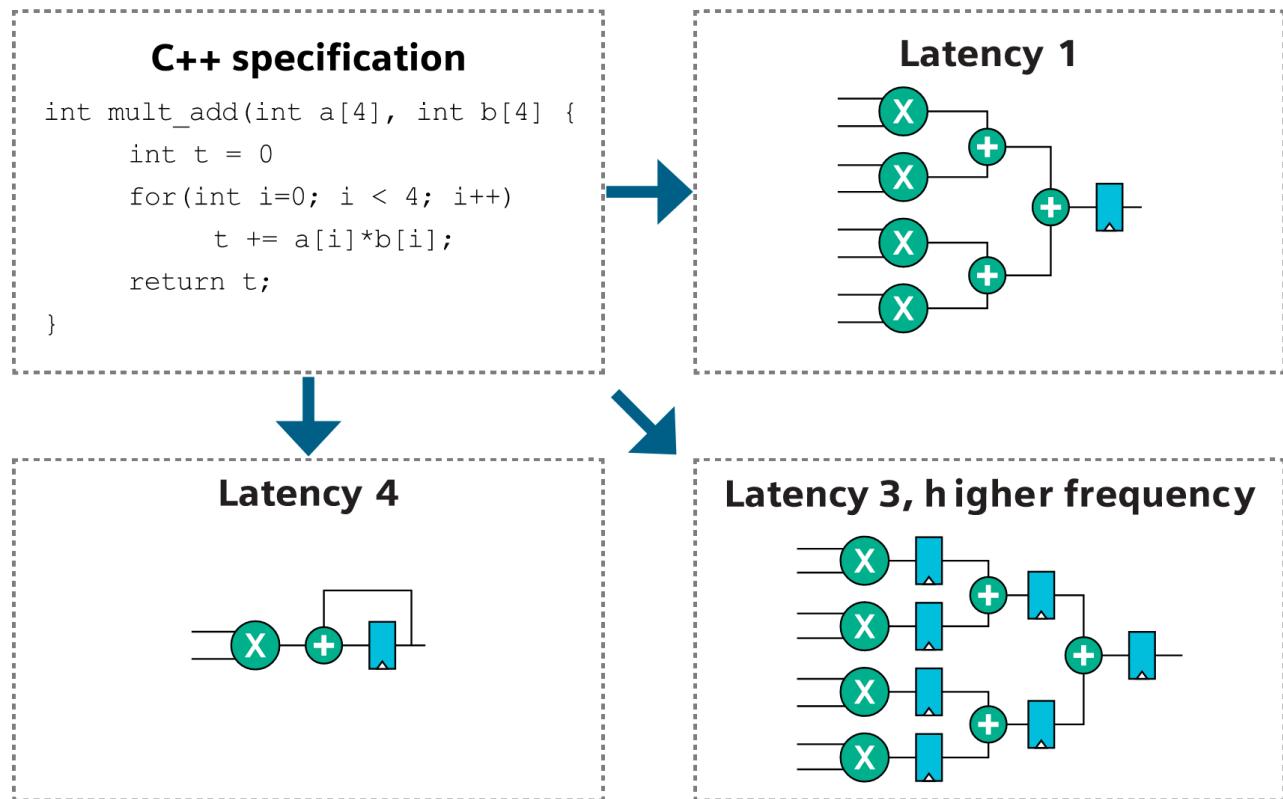
200)Что такое высокоуровневый синтез (HLS) и какова его роль в современной разработке?

HLS (High Level Synthesis)

У FPGA есть множество проблем:

- Множество вариантов отображения алгоритма в цифровую схему
- Выбор зависит от требований
- Изменение требований может вызвать перепроектирование
- Разработка схем трудоемка и требует специалистов
- Медленные тесты

Идея вот в чем, а давайте дадим программистам привычный им инструментарий, который позволит им написать проанализированную схему. С точки зрения аппаратуры у нас есть несколько вариантов реализации(на картинке)



Высокоуровневый синтез позволяет нас абстрагировать от деталей аппаратной реализации.

В итоге HLS это инструментарий, в который мы загружаем программку на си и она дает нам выбор аппаратной реализации.

Высокоуровневый синтез. Достоинства

1. Скорость проектирования. Быстрое прототипирование.
2. Переносимость между разными целевыми платформами.
3. Адаптируемость микроархитектуры под новые условия.
4. Автоматизация процесса оптимизации схемы.

Недостатки

1. Ограниченный контроль за результатом синтеза. "Чудеса"
2. Кривая обучения (специальность HLS Engineer)

3. Уровень зрелости технологии сильно варьируется:

- верификация результата;
- стабильность работы.

201) Как высокоуровневый синтез (HLS) автоматизирует процесс выбора микроархитектуры и ускоряет тестирование?

Высокоуровневый синтез (HLS) кардинально меняет процесс разработки аппаратного обеспечения, автоматизируя ключевые этапы — от выбора микроархитектуры до верификации. Вот как это работает:

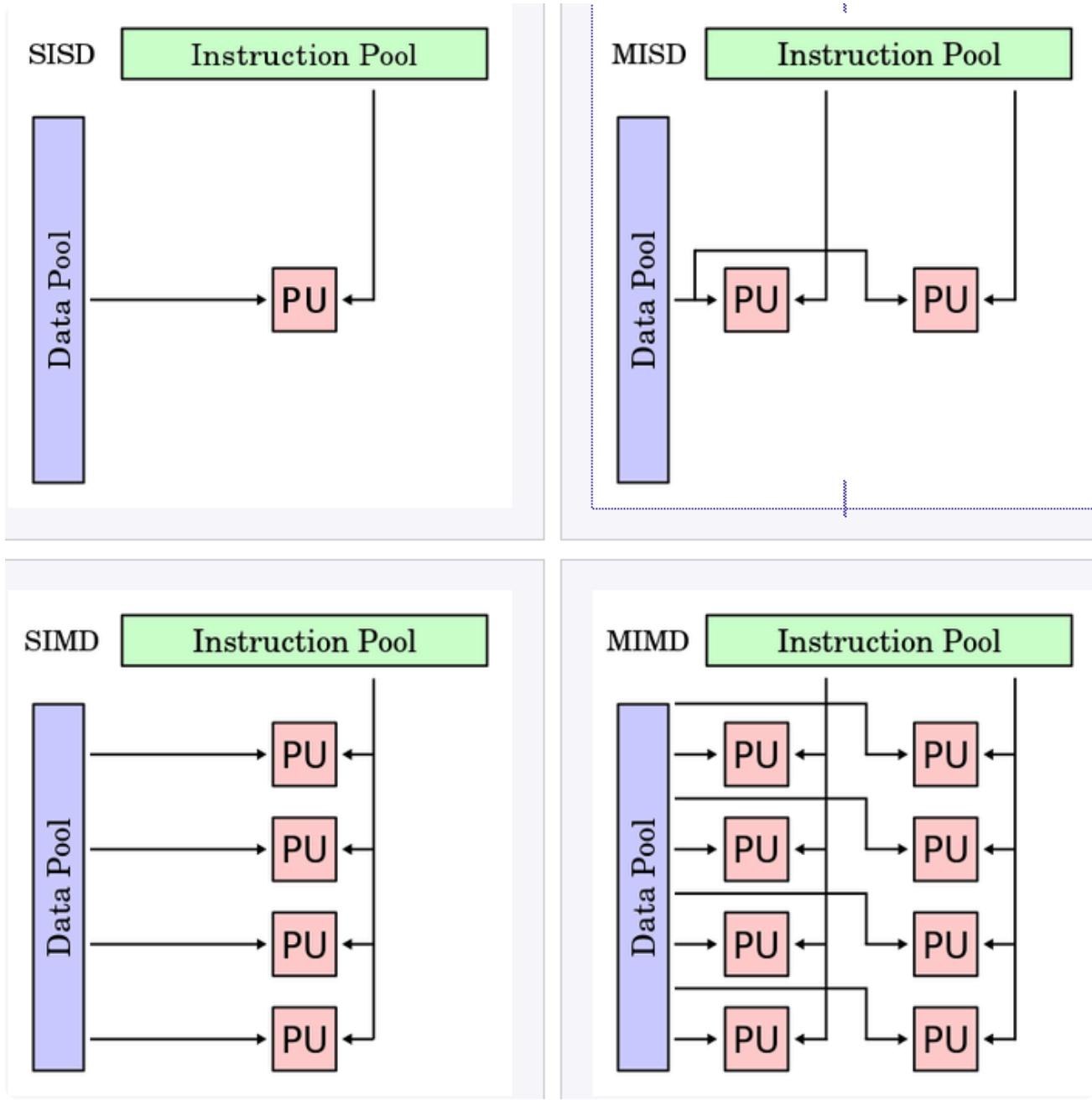
1. анализ и оптимизация кода (HLS инструменты анализируют высокоуровневый код и автоматически принимают решения по поводу распараллеливания операций, выбора между ресурсами и оптимизацией интерфейсов)
2. HLS позволяет быстро исследовать дизайн пространство(букально предоставляя выбор между максимальной производительностью или минимальной площадью)
3. Ну и плюс, hls позволяет тестировать алгоритм до синтеза.

202)Что представляет собой классификация Флинна и какие классы архитектур она выделяет? Примеры.

Классификация Флинна - классификация параллельных процессоров на основе:

- количества потоков инструкций
 - количества потоков данных
- Достоинства: простая и понятная
Недостатки:
- не применима к фон Неймановским архитектурам

- Перегруженность класса MIMD



Классы архитектур:

1. **SISD(Single Instruction, Single Data):** простой последовательный процессор
2. **SIMD (Single Instruction, Multiply Data):** одновременное выполнение несколькими процессорами одной инструкции(Lockstep execution - много потоков данных, они работают строго синхронно и выполняют одно действие); Назначение - однообразная обработка множества наборов данных. Пример: GPU, моделирование. Ключевые ограничения: операции ветвления
3. **SIMT(Single Instruction Multiply Threads):** расширение SIMD, позволяющее работать с оператором ветвления. Широко применяется в современных GPU. Включает механизмы синхронизации потоков между собой для достижения lockstep execution.

4. MISD (Multiply Instruction Single Data): один поток данных обрабатывается разными наборами инструкций. Обычно выделяется для полноты классификации. На практике - повышение надежности работы системы. Защита от ошибок проектирования и алгоритмов. Параллельная разработка и эксплуатация решений задачи.
5. MIMD (Multiply Instructions Multiply Data): множество процессоров автономно выполняют различные инструкции над различными данными. Самый разнообразный класс процессоров по классификации Флинна. Не выделяет подвиды.

203)Что такое SIMD и SIMT архитектуры? В чём их различия? Что такое lockstep execution?

см билет выше.

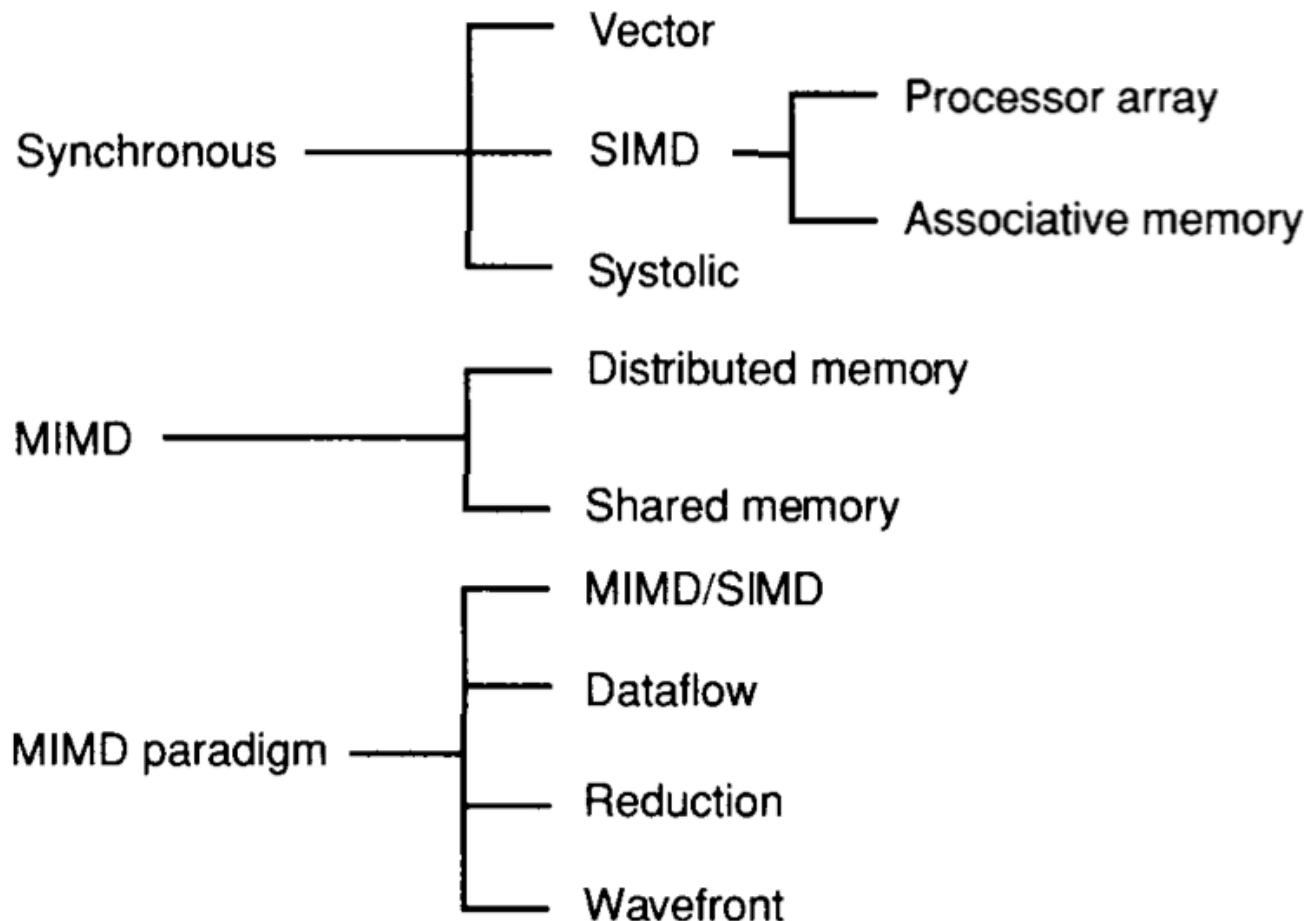
204)Что представляет собой классификация Дункана и чем она отличается от классификации Флинна? Какие проблемы она решает? Что включает верхний уровень классификации?

Классификация Дункана - расширение классификация Флинна, она добавляет новые критерии для более точного описания современных параллельных архитектур.

Решает проблемы:

- Невыразимость огромного количества процессорных архитектур
 - Перегруженность класса MIMD
- Задачи:
- Абстрагироваться от низкоуровневого параллелизма(конвейер, суперскаляр, vliw)
 - Переиспользоваться элементы таксономии флинна
 - Навести порядок в MIMD
- Достоинства:
- относительная полнота
- Недостатки:

- сложность, "искусственная"



Верхний уровень классификации:

1. Синхронные архитектуры

Параллельные архитектуры с единым механизмом управления (глобальные часы, центральный блок управления и т.п.). **Различаются механизмами синхронизации.**

2. MIMD архитектуры

Параллельные архитектуры, которые могут исполнять независимые потоки инструкций. Потоки инструкций исполняются автономно, требуется динамическая синхронизация. **Различаются структурой организации памяти.**

3. MIMD парадигма (MIMD paradigm architectures)

Архитектура MIMD процессора определяется моделью вычислений (МоС). МоС определяет характер потоков данных, принципы взаимодействия и синхронизации.

Различаются МоС.

205)Какие основные типы синхронных архитектур выделяются по классификации Дункана?

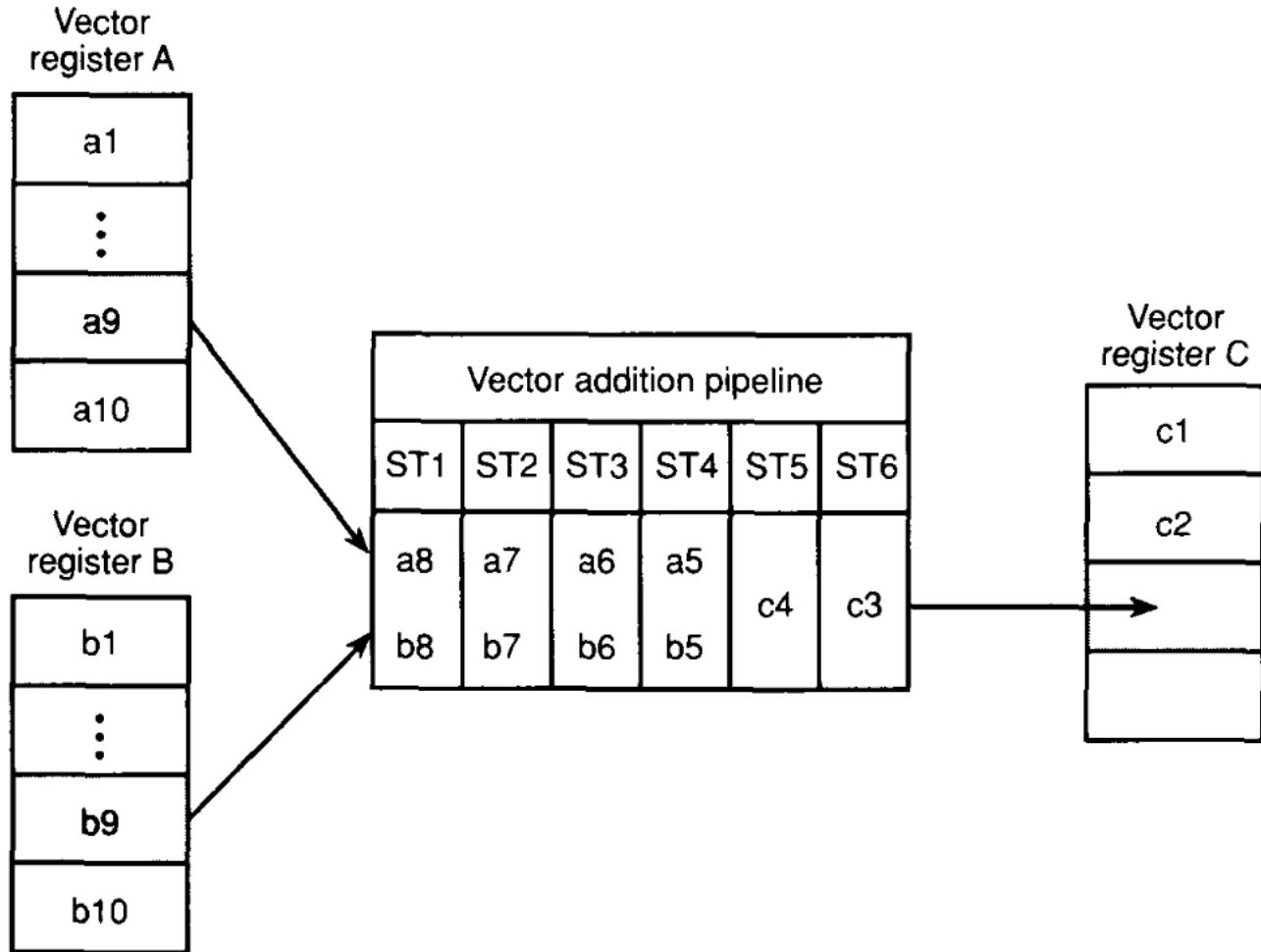
Синхронные архитектуры - Класс процессорных архитектур, обладающих единым/централизованным/синхронным управлением, определяющим поведение всего процессора в целом.

- Векторные архитектуры.
- SIMD архитектуры:
 - Процессорные массивы.
 - Ассоциативные массивы.
- Систолические массивы.

из картинки выше:

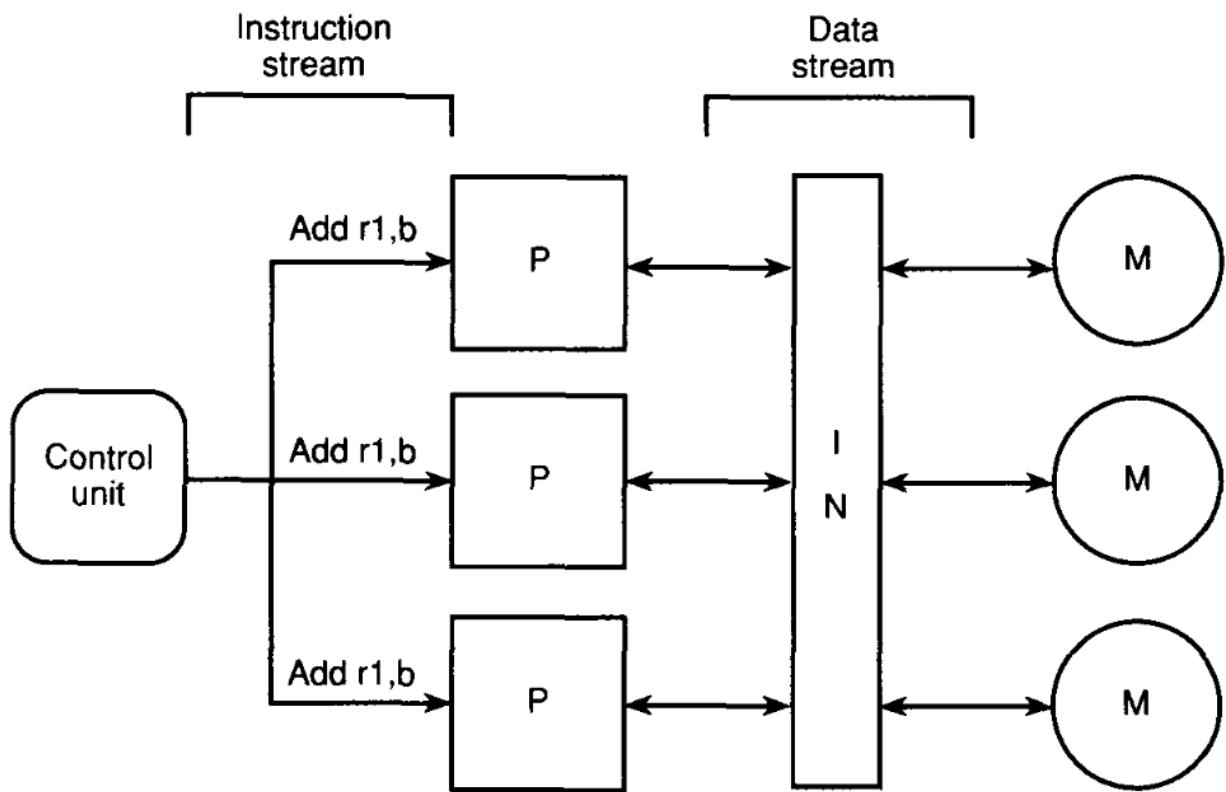
1. Синхронные векторные:

- Сокращение количества обрабатываемых инструкций.
- Конвейерное исполнение векторных операций.
- Множество функциональных узлов (суперскалярность).



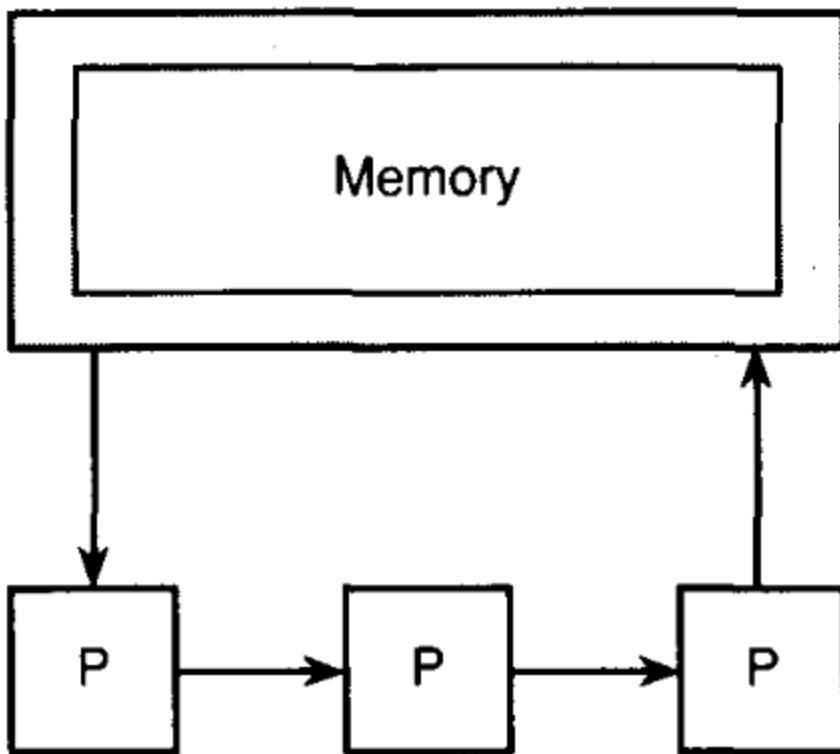
2. Синхронные SIMD системы:

- все процессорные элементы выполняют одну инструкцию одновременно в тактовом режиме



3. синхронные систолические:

- Решетка процессорных элементов синхронизированна по тактам
- Данные перекачиваются между элементами



- данные ритмично поступают из памяти, проходят через сеть процессоров, и возвращаются в память.

- Синхронизация — через глобальные часы, такт вычислений (не путать с тактовым сигналом).
- Процессоры объединены локальными связями, и каждый цикл процессоры выполняют короткие неизменные операции.

206)Какие принципы лежат в основе работы векторных архитектур и где они применяются?

выше

210)Что из себя представляют архитектуры с распределённой памятью? Каковы их особенности?

211)Что из себя представляет архитектуры с разделяемой памятью? Каковы их особенности?

выше

212)Какие основные типы MIMD парадигм выделяются по классификации Дункана?

213)Что представляет собой MIMD/SIMD парадигма и где она применяется?

214)Как работает архитектура потоков данных (dataflow)? Подходы к реализации.

215)Как работает редукционная архитектура? Каким образом она обеспечивает параллелизм?

216)Как работает *wavefront* архитектура? В чём её отличия от систолической архитектуры?

217)Что такое CGRA-процессоры и какие возможности они предоставляют?

218)В чём различие между пространственными (spatial) и временными (temporal) вычислениями на примере CGRA?

