Contents

[1. Что такое "вычислительная платформа" с точки зрения пользователя? Каков её состав? 10](#_Toc201708582)

[2. Какие уровни абстракций относятся к дисциплине "Архитектура компьютера"? 10](#_Toc201708583)

[3. В чём заключается отличие Platform-Based Design от проектирования для конкретной платформы? 10](#_Toc201708584)

[4. Почему большинство современных компьютерных систем считаются системами с преобладающей программной составляющей? Приведите примеры. 11](#_Toc201708585)

[5. Что такое информационная и управляющая система? Каковы их отличия (функции, ПО, аппаратура)? 11](#_Toc201708586)

[6. Что такое "требование реального времени"? Примеры систем. 12](#_Toc201708587)

[7. Каковы этапы эволюции управляющих систем: от ИУС до КФС? 12](#_Toc201708588)

[8. Каковы задачи и предмет дисциплины "Системная инженерия"? Какова роль системных инженеров? 13](#_Toc201708589)

[9. Что такое успешная система? Какие точки зрения необходимы для построения успешной системы? 13](#_Toc201708590)

[10. Как можно рассматривать систему с точки зрения структуры? Каковы причины множественности структуры? 14](#_Toc201708591)

[11. Как можно рассматривать систему с точки зрения функционального места? Кто такие заинтересованные стороны (Stakeholders)? Что такое операционное окружение? 14](#_Toc201708592)

[12. Как можно рассматривать систему с точки зрения жизненного цикла? Что такое обеспечивающая система? 14](#_Toc201708593)

[13. Почему плохой менеджмент может увеличить бюджет проекта быстрее, чем другие факторы? 15](#_Toc201708594)

[14. Каковы цели архитектурного проектирования компьютерных систем? 15](#_Toc201708595)

[15. Что такое архитектура по Гради Бучу? Что представляют собой логическая и физическая структура? 15](#_Toc201708596)

[16. Что такое архитектура согласно ISO 42010? Что такое архитектурное описание? 16](#_Toc201708597)

[17. Как архитектурные решения влияют на проектные метрики? Что такое V-диаграмма и какую особенность разработки она демонстрирует? 16](#_Toc201708598)

[18. Сравните подходы к реализации вычислений в арифмометре и на логарифмической линейке. Какие аналогии можно провести с современными компьютерами? 16](#_Toc201708599)

[19. Каков был подход к расчёту артиллерийских таблиц группой людей? Какие аналогии можно провести с современными компьютерами? 17](#_Toc201708600)

[20. Каково устройство электрического реле? Какие виды реле существуют и каковы области их применения? 17](#_Toc201708601)

[21. Как реализуется булев базис на электрических реле? 18](#_Toc201708602)

[22. Что такое программируемые логические контроллеры (ПЛК) и каковы области их применения? 19](#_Toc201708603)

[23. Какова структура программируемых логических контроллеров (ПЛК)? Причины отделения инструментальной составляющей? 19](#_Toc201708604)

[24. Каковы особенности аппаратного обеспечения программируемых логических контроллеров (ПЛК)? 20](#_Toc201708605)

[25. Каковы особенности программного обеспечения программируемых логических контроллеров (ПЛК)? 21](#_Toc201708606)

[26. В чём заключается принцип развития иерархических систем Седова? Иллюстрируйте применительно к вычислительным платформам. 21](#_Toc201708607)

[27. Что такое булев базис и какова его роль в вычислительной технике? Приведите примеры. 22](#_Toc201708608)Как реализуется ввод-вывод с точки зрения системы команд в Memory Mapped IO?

[28. Что такое двоичное кодирование сигналов? Каковы его достоинства и недостатки? Что такое запретная зона? 22](#_Toc201708609)

[29. Какова роль машинного слова в устройстве процессора? Что такое Big- и Little-endian? 23](#_Toc201708610)

[30. Какие существуют способы кодирования целочисленных данных? Что такое позиционное кодирование, код Грея, BCD? 23](#_Toc201708611)

[31. Какие существуют способы кодирования бинарных данных? Что такое Base64, Base58? 23](#_Toc201708612)

[32. Что такое комбинационные схемы? Что такое переходный процесс? С чем связаны задержки и накопление ошибки? Как связаны комбинационные схемы с параллелизмом уровня бит? 24](#_Toc201708613)

[33. Какие состояния существуют в комбинационных схемах (0, 1, x, z) и что они означают? 25](#_Toc201708614)

[34. Каковы особенности реализации "условного оператора" в комбинационных схемах? 25](#_Toc201708615)

[35. Что такое триггеры в цифровых схемах? Каковы варианты их использования? 26](#_Toc201708616)

[36. Что такое D-триггер и RS-триггер? Какие существуют варианты условия изменения состояния? 27](#_Toc201708617)

[37. Что такое пространственные и временные вычисления? Как они могут быть использованы для оптимизации процессоров? 27](#_Toc201708618)

[38. Что называют синхронной схемотехникой? Каковы её достоинства и недостатки? 28](#_Toc201708619)

[39. Почему цифровая схемотехника оперирует уровнями, а не сигналами? Какие возможности это открывает и какие проблемы создаёт? 29](#_Toc201708620)

[40. Каковы ключевые тенденции в производстве радиоэлектронной аппаратуры и связанные с этим проблемы? 30](#_Toc201708621)

[41. Что такое навесной монтаж, монтаж на печатную плату, штырьевой и поверхностный монтаж? Как они обеспечивают "гибкость"? 30](#_Toc201708622)

[42. Какова производственная цепочка поверхностного монтажа? Каковы её этапы? 30](#_Toc201708623)

[43. Какова производственная цепочка кремниевого производства? Каковы особенности формирования цены изделия? 31](#_Toc201708624)

[44. Какие трудности связаны с аппаратным обеспечением на этапах производства и эксплуатации? 32](#_Toc201708625)

[45. Какие подходы к решению проблемы "устаревающей аппаратуры" существуют с точки зрения аппаратуры и ПО? 33](#_Toc201708626)

[46. Что такое концепция 2-этапного производства? Какие существуют варианты этапа "конфигурирования" для разных уровней организации вычислений? 34](#_Toc201708627)

[47. Каково определение программной системы согласно OMG Essence? Каковы её части? 35](#_Toc201708628)

[48. Что такое программное и аппаратное обеспечение? Что означают понятия Hardware и Software? Сопоставьте их. 35](#_Toc201708629)

[49. Какие возможности открывает программное обеспечение (цикл разработки, гибкость, контроль, и т.д.)? 36](#_Toc201708630)

[50. Чем отличается типовое проектирование Hardware/Software от совместного (CoDesign) проектирования Hardware/Software? Каковы их достоинства и недостатки? 36](#_Toc201708631)

[51. Что такое "Модель вычислений"? В чём назначение моделей вычислений? Приведите примеры. 37](#_Toc201708632)

[52. Что такое последовательные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс? 37](#_Toc201708633)

[53. Что такое машина Тьюринга и почему она важна для теории вычислений? 38](#_Toc201708634)

[54. Что такое Random Access Machine? Каково её устройство и место сегодня? Какова её связь с машиной Тьюринга? 38](#_Toc201708635)

[55. Что такое функциональные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс? 39](#_Toc201708636)

[56. Что такое паралелильные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс? 39](#_Toc201708637)

[57. Что такое Model-Driven Engineering (MDE)? Чем цепочка трансформации в MDE отличается от языков высокого уровня? Приведите примеры. 40](#_Toc201708638)

[58. Что такое универсальный процессор? Каковы его особенности и свойства? 40](#_Toc201708639)

[59. В чём заключается противоречие между универсальностью и эффективностью в разных видах процессоров (СБИС, FPGA, CGRA, GPU, DSP, CPU)? 41](#_Toc201708640)

[60. Как соотносятся вычислительные платформы и языки описания вычислительного процесса? Как это связано с эффективностью и областью применения? 41](#_Toc201708641)

[61. Что такое архитектура и микроархитектура процессора? В чём различие между ними? 42](#_Toc201708642)

[62. Что такое система команд и какова её роль в архитектуре процессоров? Что определяет система команд? 43](#_Toc201708643)

[63. Что такое машина фон Неймана? Какова её связь с машиной Тьюринга? Каковы её ключевые принципы? 43](#_Toc201708644)

[64. Что такое микропроцессор и микроконтроллер? В чём их отличия? 44](#_Toc201708645)

[65. Что такое Control Unit и Data Path? Каковы их назначение и принципы взаимодействия? 44](#_Toc201708646)

[66. Какие виды инструкций существуют в машине фон Неймана? Приведите примеры. Каковы принципы кодирования? 45](#_Toc201708647)

[67. Каковы особенности принстонской архитектуры? Каковы её достоинства и недостатки? Какова область применения? 45](#_Toc201708648)

[68. Каковы особенности гарвардской архитектуры? Каковы её достоинства и недостатки? Какова область применения? 46](#_Toc201708649)

[69. Как организовано адресное пространство в гарвардской и принстонской архитектурах? 46](#_Toc201708650)

[70. Какие существуют варианты обхода ограничений гарвардской архитектуры, включая "Модифицированную гарвардскую архитектуру"? 47](#_Toc201708651)

[71. Что такое "аккумуляторная система команд"? Каковы основные элементы? Приведите примеры. Каковы достоинства и недостатки? 47](#_Toc201708652)

[72. Что такое Register-to-Memory системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки? 48](#_Toc201708653)

[73. Что такое Register-to-Register системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки? 49](#_Toc201708654)

[74. Что такое Memory-to-Memory системы команд? Каковы основные элементы? Приведите примеры. Каковы их достоинства и недостатки? 49](#_Toc201708655)

[75. Что такое стековые системы команд? Каковы основные элементы? Что такое неявная адресация аргументов? Каковы их достоинства и недостатки? 50](#_Toc201708656)

[76. Какие виды стеков есть в стековых процессорах? Их количества? 50](#_Toc201708657)

[77. Каковы основные принципы описания алгоритмов для стековых процессоров? Что такое Forth и High-level language computer? 51](#_Toc201708658)

[78. Что такое CISC процессоры? Каковы их ключевые особенности, достоинства и недостатки? 51](#_Toc201708659)

[79. Какие виды адресации аргументов инструкции в CISC процессорах? 52](#_Toc201708660)

[80. В каких случаях CISC инструкции эффективны? Приведите примеры. 52](#_Toc201708661)

[81. В чём выражается комплексность систем команд CISC? 53](#_Toc201708662)

[82. Какие существуют подходы к реализации Control Unit? Достоинства и недостатки Hardwired и микропрограммного управления? 53](#_Toc201708663)

[83. Что такое микропрограммное управление? Что такое микроинструкция? В чём отличия инструкций и микроинструкций? 54](#_Toc201708664)

[84. Как можно оптимизировать инструкции при помощи микропрограммирования? Приведите пример. 55](#_Toc201708665)

[85. Какие существуют подходы к поиску первой микроинструкции для заданной инструкции? 55](#_Toc201708666)

[86. Что такое архитектура NISC? Каковы область её применения, достоинства и недостатки? 56](#_Toc201708667)

[87. Каковы источники роста производительности процессоров? 56](#_Toc201708668)

[88. Какие законы и ограничения влияют на производительность современных процессоров? 56](#_Toc201708669)

[89. Что представляют собой закон Мура и закон Деннарда? 57](#_Toc201708670)

[90. Что представляет собой закон Амдала? На каком уровне параллелизма он работает? 57](#_Toc201708671)

[91. Что такое Power Wall и тёмный кремний? 57](#_Toc201708672)

[92. Какие подходы существуют к решению проблемы Memory Wall в рамках процессоров семейства фон Неймана? 57](#_Toc201708673)

[93. Какие подходы существуют к решению проблемы Memory Wall в рамках ASIC и CGRA? 58](#_Toc201708674)

[94. Что такое RISC? Каковы особенности и предпосылки появления (ПО и аппаратура)? 59](#_Toc201708675)

[95. Как сравниваются CISC и RISC с точки зрения архитектуры? 59](#_Toc201708676)

[96. Как сравниваются CISC и RISC с точки зрения микро-архитектуры? 60](#_Toc201708677)

[97. Каковы особенности кодирования инструкций при сравнении CISC и RISC? 60](#_Toc201708678)

[98. Каковы принципы конвейерного исполнения инструкций? Как это влияет на загрузку и производительность? 61](#_Toc201708679)

[99. Каковы типовые стадии RISC процессора? 61](#_Toc201708680)

[100. Какие проблемы и архитектурные ограничения связаны с конвейеризацией? 62](#_Toc201708681)

[101. Как разрешаются структурные конфликты в конвейере? 62](#_Toc201708682)

[102. Как разрешаются конфликты по данным в конвейере? 63](#_Toc201708683)

[103. Как происходит разрешение конфликтов при помощи пузырька в конвейере? 63](#_Toc201708684)

[104. Как происходит разрешение конфликтов при помощи проброса операндов в конвейере? 63](#_Toc201708685)

[105. Какие существуют способы разрешения конфликтов по управлению во время компиляции? 64](#_Toc201708686)

[106. Какие существуют способы разрешения конфликтов по управлению во время исполнения? 64](#_Toc201708687)

[107. Какие существуют способы предотвращения конфликтов по управлению через branch prediction? Что такое статические предсказатели переходов? 65](#_Toc201708688)

[108. Что представляют собой динамические предсказатели переходов в конвейере? 65](#_Toc201708689)

[109. Как влияют различные виды конфликтов на производительность конвейера? 66](#_Toc201708690)

[110. Как связаны конвейерное исполнение и спекулятивное исполнение инструкций? 66](#_Toc201708691)

[111. Каковы шаги типового взаимодействия с устройством ввода-вывода? Каков интерфейс устройства для процессора? 66](#_Toc201708692)

[112. Как реализуется ввод-вывод с точки зрения системы команд в Port Mapped IO? 67](#_Toc201708693)

[113. Как реализуется ввод-вывод с точки зрения системы команд в Memory Mapped IO? 67](#_Toc201708694)

[114. Как устроен ввод-вывод в Memory Mapped IO с точки зрения устройства процессора? 67](#_Toc201708695)

[115. Как работает программно-управляемый ввод-вывод без прерываний? Каковы его ограничения и типовой сценарий? Как связан с теоремой Котельникова? 67](#_Toc201708696)

[116. Как можно имитировать параллелизм уровня задач для параллельного программно-управляемого ввода-вывода? 68](#_Toc201708697)

[117. Какие существуют уровни параллелизма: бит, инструкций и задач? Приведите примеры. 68](#_Toc201708698)

[118. В чём разница между параллелизмом и конкурентностью (concurrency)? 69](#_Toc201708699)

[119. В чём заключается проблема реализации параллелизма уровня задач в архитектуре фон Неймана? 69](#_Toc201708700)

[120. Каковы достоинства и недостатки кооперативной многозадачности? 70](#_Toc201708701)

[121. Какие механизмы необходимы для реализации кооперативной многозадачности? 70](#_Toc201708702)

[122. Как взаимодействуют кооперативная многозадачность и система прерываний? 70](#_Toc201708703)

[123. Как реализуется кооперативная многозадачность на уровне приложения через Event-loop? 70](#_Toc201708704)

[124. Каковы достоинства и недостатки вытесняющей многозадачности? 71](#_Toc201708705)

[125. Какие механизмы необходимы для реализации вытесняющей многозадачности? 71](#_Toc201708706)

[126. В чём преимущество ввода-вывода с использованием системы прерываний? 72](#_Toc201708707)

[127. Как по шагам происходит обработка прерывания? 72](#_Toc201708708)

[128. Почему появление системы прерываний приводит к автоматическому появлению параллелизма уровня задач? Охарактеризуйте его. 72](#_Toc201708709)

[129. Как классифицируются прерывания по источнику? Их отличительные особенности. 73](#_Toc201708710)

[130. Каковы задачи контроллера прерывания и какие виды прерываний бывают? 73](#_Toc201708711)

[131. Какие виды событий существуют в системе прерываний и как они обрабатываются? 74](#_Toc201708712)

[132. Что такое сторожевой таймер и каков принцип его работы? 74](#_Toc201708713)

[133. Какие подходы существуют к решению проблемы изоляции регистров и инструкций в памяти при многозадачности? 75](#_Toc201708714)

[134. Какие подходы существуют к решению проблемы изоляции данных в памяти (статика, куча, стек) при многозадачности? 75](#_Toc201708715)

[135. Как банки памяти используются для расширения адресного пространства? 75](#_Toc201708716)

[136. Как банки памяти используются для расширения машинного слова? 75](#_Toc201708717)

[137. Что такое сегментная память? Как происходит трансляция адресов? Каковы достоинства и недостатки? 76](#_Toc201708718)

[138. Что такое виртуальная память? Как происходит трансляция адресов? Каковы достоинства и недостатки? 77](#_Toc201708719)

[139. Что такое внутренняя и внешняя фрагментация в контексте сегментной и виртуальной памяти? 77](#_Toc201708720)

[140. Как появление сегментной памяти улучшило пользовательский опыт? 78](#_Toc201708721)

[141. Как появление виртуальной памяти улучшило пользовательский опыт? 78](#_Toc201708722)

[142. Какие уровни (виды) задач существуют: основной поток, прерывание, процессы, потоки, зелёные потоки? 78](#_Toc201708723)

[143. Как могут взаимодействовать процессы через основной поток? Какие бывают разделяемые ресурсы? 78](#_Toc201708724)

[144. Что такое процессоры ввода-вывода? Каковы их назначение, интерфейс, достоинства и недостатки? 78](#_Toc201708725)

[145. Что такое контроллер прямого доступа к памяти (DMA)? Каковы его назначение, интерфейс, достоинства и недостатки? 79](#_Toc201708726)

[146. Какие существуют способы интеграции и взаимодействия контроллера прямого доступа к памяти (DMA) с процессором? 80](#_Toc201708727)

[147. Почему необходима иерархия памяти в современных компьютерных системах? 80](#_Toc201708728)

[148. Какие основные виды памяти входят в иерархию памяти и каковы их характеристики? 80](#_Toc201708729)

[149. В чём различие между явной иерархией памяти и скрытой? Приведите примеры. 81](#_Toc201708730)

[150. Каково устройство памяти с произвольным доступом? Как устроена ROM ячейка? 81](#_Toc201708731)

[151. Каковы технологии реализации SRAM ячеек? Как происходит чтение и запись? 81](#_Toc201708732)

[152. Каковы технологии реализации DRAM ячеек? Как происходит чтение и запись? 82](#_Toc201708733)

[153. Каково устройство многопортовой памяти? Как связана площадь с количеством портов? 82](#_Toc201708734)

[154. Что такое кеш? Каково его назначение, место в иерархии памяти и основные метрики? 83](#_Toc201708735)

[155. Что такое принцип локальности и каковы его виды (временная, пространственная) применительно к механизму кеширования? 83](#_Toc201708736)

[156. Как пошагово происходит процесс чтения данных через кеш-память процессора? 84](#_Toc201708737)

[157. Как пошагово происходит процесс записи данных через кеш-память процессора и какие существуют политики записи? 84](#_Toc201708738)

[158. Что такое кеш-промах и какие существуют типы кеш-промахов? 84](#_Toc201708739)

[159. Что такое ассоциативность кеша? Какова "структура" адреса с точки зрения кеша? 85](#_Toc201708740)

[160. Каковы особенности реализации полностью ассоциативного кеша? 85](#_Toc201708741)

[161. Каковы особенности реализации кеша с прямым отображением? 85](#_Toc201708742)

[162. Что такое множественно-ассоциативный кеш и что определяет уровень множественности? 85](#_Toc201708743)

[163. Как связана ассоциативность кеша с механизмом вытеснения и замещения? 86](#_Toc201708744)

[164. Как работает механизм вытеснения и замещения LRU в кеше? 86](#_Toc201708745)

[165. Как работает механизм вытеснения и замещения PLRU в кеше? 86](#_Toc201708746)

[166. Почему необходимо использование многоуровневой иерархии кеш-памяти в современных процессорах? 86](#_Toc201708747)

[167. В чём разница между разделённым и унифицированным кешем и где они применяются? 87](#_Toc201708748)

[168. В чём разница между включающим и исключающим кешем и где они применяются? 87](#_Toc201708749)

[169. В чём разница между частным и общим кешем и как они применяются в многоядерных процессорах? 87](#_Toc201708750)

[170. Что представляет собой типовой многоуровневый кеш: L1, L2, L3, L4? Каковы их назначение, тип ячеек, классификация? 87](#_Toc201708751)

[171. Что такое когерентность кеш-памяти и каково её значение для многоядерных систем? Какие существуют возможные состояния кеш-линий? 88](#_Toc201708752)

[172. Как происходит обмен информацией о кеш-линиях через справочник? На сколько актуальна проблема масштабирования? 88](#_Toc201708753)

[173. Как происходит обмен информацией о кеш-линиях через отслеживание? На сколько актуальна проблема масштабирования? 88](#_Toc201708754)

[174. Как происходит обмен информацией о кеш-линиях через перехват? На сколько актуальна проблема масштабирования? 89](#_Toc201708755)

[175. Что такое CAP теорема? Какова область её применения и как она применяется к когерентности кешей? 89](#_Toc201708756)

[176. Что означают термины "Consistency", "Availability" и "Partition tolerance" в CAP-теореме? Опишите их. 90](#_Toc201708757)

[177. Как кеш влияет на производительность памяти? Какие существуют способы оптимизации доступа, включая AoS и SoA? 90](#_Toc201708758)

[178. Как кеш работает с виртуальными и физическими адресами? 90](#_Toc201708759)

[179. Что представляет собой концепция уровневой организации компьютерных систем и каковы её преимущества? 91](#_Toc201708760)

[180. Какие существуют примеры уровневой организации компьютерных систем: Lava Flow, Layered Style, OSI Model? 91](#_Toc201708761)

[181. Что такое явление разделения на уровни (disaggregation) и каково его современное положение? Что было до него? 91](#_Toc201708762)

[182. Почему необходимо нарушение уровневой иерархии в современных системах? Приведите примеры. 92](#_Toc201708763)

[183. Какие проблемы возникают при обеспечении реального времени в современных процессорах? 92](#_Toc201708764)

[184. Что такое низкоуровневый параллелизм? Приведите примеры. 92](#_Toc201708765)

[185. Что такое суперскалярные процессоры и каковы принципы их работы, достоинства и недостатки? 93](#_Toc201708766)

[186. Как пошагово выполняются инструкции в суперскалярном процессоре? 93](#_Toc201708767)

[187. Что такое VLIW-процессоры? Какими достоинствами и недостатками они обладают? 94](#_Toc201708768)

[188. Как соотносятся суперскалярные и VLIW процессоры применительно к разным классам задач? 94](#_Toc201708769)

[189. Какие существуют промежуточные варианты между суперскалярным и VLIW процессором? Какие проблемы они решают? 95](#_Toc201708770)

[190. Что такое барьеры памяти и в каких ситуациях они применяются? 95](#_Toc201708771)

[191. Какие ключевые элементы составляют структурное программирование? В чём отличие от кода на ассемблере? 96](#_Toc201708772)

[192. Какие механизмы используются для реализации процедур? 96](#_Toc201708773)

[193. Что такое реентерабельность и почему она важна? Приведите примеры. 96](#_Toc201708774)

[194. Как реализуется механизм рекурсии? 97](#_Toc201708775)

[195. В чём заключается проблема распределения регистров и как работает алгоритм раскраски? 97](#_Toc201708776)

[196. Что такое FPGA и каковы его преимущества и недостатки? 97](#_Toc201708777)

[197. Какие этапы включает в себя синтез для FPGA? 98](#_Toc201708778)

[198. Какие FPGA устроены изнутри? 98](#_Toc201708779)

[199. Как соотносятся CPU и FPGA с точки зрения модели программирования, модели вычислений, модели исполнения и элементов микроархитектуры? 99](#_Toc201708780)

[200. Что такое высокоуровневый синтез (HLS) и какова его роль в современной разработке? 99](#_Toc201708781)

[201. Как высокоуровневый синтез (HLS) автоматизирует процесс выбора микроархитектуры и ускоряет тестирование? 100](#_Toc201708782)

[202. Что представляет собой классификация Флинна и какие классы архитектур она выделяет? Примеры. 100](#_Toc201708783)

[203. Что такое SIMD и SIMT архитектуры? В чём их различия? Что такое lockstep execution? 101](#_Toc201708784)

[204. Что представляет собой классификация Дункана и чем она отличается от классификации Флинна? Какие проблемы она решает? Что включает верхний уровень классификации? 101](#_Toc201708785)

[205. Какие основные типы синхронных архитектур выделяются по классификации Дункана? 102](#_Toc201708786)

[206. Какие принципы лежат в основе работы векторных архитектур и где они применяются? 102](#_Toc201708787)

[207. Как работают ассоциативные массивы и где они применяются? 102](#_Toc201708788)

[208. Как работают систолические архитектуры и где они применяются? 103](#_Toc201708789)

[209. Какие основные типы MIMD-архитектур выделяются по классификации Дункана? 103](#_Toc201708790)

[210. Что из себя представляют архитектуры с распределённой памятью? Каковы их особенности? 103](#_Toc201708791)

[211. Что из себя представляет архитектуры с разделяемой памятью? Каковы их особенности? 104](#_Toc201708792)

[212. Какие основные типы MIMD парадигм выделяются по классификации Дункана? 104](#_Toc201708793)

[213. Что представляет собой MIMD/SIMD парадигма и где она применяется? 104](#_Toc201708794)

[214. Как работает архитектура потоков данных (dataflow)? Подходы к реализации. 104](#_Toc201708795)

[215. Как работает редукционная архитектура? Каким образом она обеспечивает параллелизм? 105](#_Toc201708796)

[216. Как работает wavefront архитектура? В чём её отличия от систолической архитектуры? 105](#_Toc201708797)

[217. Что такое CGRA-процессоры и какие возможности они предоставляют? 106](#_Toc201708798)

[218. В чём различие между пространственными (spatial) и временными (temporal) вычислениями на примере CGRA? 106](#_Toc201708799)

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

**Вычислительная платформа** с точки зрения пользователя — это совокупность аппаратного и программного обеспечения, обеспечивающая выполнение вычислительных задач и взаимодействие с приложениями.

**Состав**:

* **Аппаратное обеспечение**: процессоры, память, устройства хранения, сетевые компоненты.
* **Программное обеспечение**: операционная система, драйверы, библиотеки, API, среды разработки.
* **Интерфейсы**: пользовательские (GUI, CLI) и программные (API, SDK) для взаимодействия.

Коротко: это инфраструктура и софт для работы приложений и задач.

1. Какие уровни абстракций относятся к дисциплине "Архитектура компьютера"?

* Физика
* Устройства
* Аналоговые схемы
* Цифровые схемы
* Логика
* Микро-архитектура
* Архитектура
* ОС
* Application Software

1. В чём заключается отличие Platform-Based Design от проектирования для конкретной платформы?

Platform-Based Design фокусируется на использовании универсальной платформы с переиспользуемыми компонентами для создания различных систем, обеспечивая гибкость и масштабируемость.

Проектирование для конкретной платформы ориентировано на оптимизацию под одну конкретную аппаратно-программную конфигурацию, что снижает гибкость, но повышает эффективность для данной цели

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

Большинство современных компьютерных систем считаются системами с преобладающей программной составляющей, потому что их функциональность и адаптивность определяются программным обеспечением, а не только аппаратным обеспечением. Программы позволяют обновлять системы, добавлять новые функции и оптимизировать производительность без изменения оборудования.

**Примеры**:

* Смартфоны (программное обеспечение управляет интерфейсом, приложениями и обновлениями).
* Персональные компьютеры (операционные системы и приложения определяют их использование).
* Автомобили с автопилотом (программное обеспечение управляет сенсорами и принятием решений).

1. Что такое информационная и управляющая система? Каковы их отличия (функции, ПО, аппаратура)?

**Информационная система (ИС)** — это система, предназначенная для сбора, обработки, хранения и предоставления данных пользователям для поддержки принятия решений.

Особенности:

* Главный приоритет: производительность.
* Спекулятивные вычисления.
* Параллелизм.
* Кластерные и облачные вычисления.

**Управляющая система (УС)** — это система, которая автоматически или полуавтоматически управляет процессами или оборудованием на основе данных и алгоритмов.

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 (см. след. слайд).

**Отличия**:

* **Функции**: ИС фокусируется на анализе и предоставлении информации (например, базы данных), УС — на контроле и управлении процессами (например, автоматизация производства).
* **ПО**: ИС использует базы данных, аналитические программы (SQL, ERP), УС — управляющее ПО (SCADA, PLC).
* **Аппаратура**: ИС опирается на серверы и ПК, УС — на датчики, исполнительные устройства и микроконтроллеры.

1. Что такое "требование реального времени"? Примеры систем.

**Требование реального времени** — это необходимость обработки данных или выполнения задач в пределах строгого временного интервала, где задержка может привести к сбоям (например, не более чем в срок ±).

**Примеры систем**:

* ГСЧ, водооброс (системы управления водоснабжением).
* Пример где один код это ИС или УС в зависимости от задачи: Видеокодек.

1. Каковы этапы эволюции управляющих систем: от ИУС до КФС?

Этапы эволюции управляющих систем от **ИУС (Информационно-управляющие системы - Системы управления информацией)** до **КФС (Компьютеризированные системы - Системы, компьютеризированные)**:

* **ИУС (1960-е)**: Ручное управление с минимальной автоматизацией.
* **ВУС (1970-е)**: Внедрение вычислительных систем (**Вычислительные управляющие системы - Системы управления вычислениями**) (например, **NYC** - без конкретной информации, возможно, название системы).
* **РВУС (1980-е)**: Реальное время с процессорами (**Реальное время управляющие системы - Системы управления в реальном времени**) (например, **ВСс** - возможно, **Вычислительные системы - Вычислительные системы**, требуется уточнение контекста).
* **ПВУС (1990-е)**: Программное обеспечение с микрообработчиками (**Программное обеспечение управляющие системы - Системы управления с программным обеспечением**) (например, **Малые интегральные схемы К155 - Малые интегральные схемы К155**).
* **КФС (2000-е и далее)**: Компьютеризированные системы с высокой степенью автоматизации и интеграции (**Компьютеризированные системы - Системы, компьютеризированные**) (например, **БК** - возможно, **Блок контроля - Блок контроля**, и **КФС**).

1. Каковы задачи и предмет дисциплины "Системная инженерия"? Какова роль системных инженеров?

**Предмет дисциплины "Системная инженерия"**:  
Дисциплина занимается координацией и структурированием процессов, процедур, передачи информации и т.д. для создания и поддержки сложных систем. Она фокусируется на целостном подходе к проектированию, разработке и интеграции систем, обеспечивая их соответствие требованиям заказчика.

**Задачи дисциплины "Системная инженерия"**:

* Определение и документирование требований;
* Управление процессами жизненного цикла системы;
* Проектирование, проверка, приемка и разработка решений для проблем;
* Интеграция и тестирование подсистем;
* Обеспечение взаимодействия всех элементов системы для достижения общих целей.

**Роль системных инженеров**:  
Системные инженеры выступают связующим звеном между заказчиком, разработчиками и командой разработки. Они определяют потребности, разрабатывают спецификации, поддерживают процесс реализации и обеспечивают, чтобы все компоненты системы работали вместе для достижения общих целей проекта.

1. Что такое успешная система? Какие точки зрения необходимы для построения успешной системы?

Успешная система — это система, которая эффективно выполняет свои функции, соответствует заданным требованиям, находится в рамках бюджета и сроков, а также удовлетворяет потребности пользователей и заинтересованных сторон на протяжении всего жизненного цикла.

**Какие точки зрения необходимы для построения успешной системы?**

1. **Точка зрения заказчика**: Учет их потребностей, ожиданий и требований.
2. **Точка зрения разработчиков**: Техническая реализуемость и качество проектирования.
3. **Точка зрения пользователей**: Удобство использования и эффективность взаимодействия.
4. **Точка зрения менеджмента**: Управление ресурсами, бюджетом и сроками.
5. **Точка зрения внешней среды**: Соответствие нормативным требованиям и адаптация к изменениям окружения.
6. Как можно рассматривать систему с точки зрения структуры? Каковы причины множественности структуры?

Система с точки зрения структуры может рассматриваться как комбинация взаимодействующих элементов, организованных для достижения одной или нескольких целей.

Это включает иерархическую организацию подсистем (подсистемы) и элементов (части), где каждая часть выполняет свою функцию, а целое обеспечивает общую цель.

**Каковы причины множественности структуры?**

* Разные точки зрения (например, заказчика, разработчика, пользователя) требуют различных уровней детализации и акцентов.
* Сложность системы, включающая множество подсистем и взаимодействий, приводит к разным структурным представлениям.
* Контекстное использование (например, самолет как система или как продукт) может изменять восприятие структуры.
* Практическая необходимость адаптации к различным аспектам жизненного цикла системы (разработка, эксплуатация, обслуживание).

1. Как можно рассматривать систему с точки зрения функционального места? Кто такие заинтересованные стороны (Stakeholders)? Что такое операционное окружение?

Система может рассматриваться как функциональное место, позволяющее:

* Называть,
* Определять,
* Выделять (select) элементы, процессы или подсистемы в зависимости от их роли и временного контекста (например, использование, обслуживание).

Заинтересованные стороны — это физические лица или организации, имеющие права, доли, претензии или интерес к системе или ее характеристикам, которые удовлетворяют их потребности и ожидания (по ISO/IEC/IEEE 2015).

Операционное окружение — это совокупность внешних и внутренних факторов, влияющих на систему, включая взаимодействие с другими системами, условиями эксплуатации и заинтересованными сторонами, а также техническую среду, в которой система функционирует.

1. Как можно рассматривать систему с точки зрения жизненного цикла? Что такое обеспечивающая система?

Система может рассматриваться как совокупность этапов, которые она проходит от концепции до завершения, включая:

1. Концептуальный этап
2. Этап разработки
3. Этап производства
4. Этап утилизации (англ. utilization)
5. Этап поддержки
6. Этап вывода из эксплуатации

Обеспечивающая система — система, которая дополняет интересующую систему на этапах ее жизненного цикла, но не обязательно вносит непосредственный вклад в ее функционирование во время эксплуатации.

1. Почему плохой менеджмент может увеличить бюджет проекта быстрее, чем другие факторы?

Плохой менеджмент может увеличить бюджет проекта быстрее, чем другие факторы, из-за следующих причин:

* Нерациональное распределение ресурсов (человеческих, финансовых, временных).
* Отсутствие четкого планирования и контроля сроков, что приводит к задержкам и переработкам.
* Плохая коммуникация между командами, вызывающая ошибки и необходимость исправлений.
* Недостаток мониторинга рисков, что приводит к неожиданным затратам.
* Неправильное управление изменениями, увеличивающее стоимость адаптации.

Эти факторы часто имеют кумулятивный эффект, быстро накапливая дополнительные расходы

1. Каковы цели архитектурного проектирования компьютерных систем?

Цели архитектурного проектирования компьютерных систем:

* Обеспечение надежности и производительности системы.
* Оптимизация использования ресурсов (аппаратных и программных).
* Упрощение разработки, интеграции и поддержки системы.
* Обеспечение масштабируемости и адаптивности к будущим изменениям.
* Снижение стоимости жизненного цикла системы за счет эффективной структуры.

1. 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.

1. Что такое архитектура согласно ISO 42010? Что такое архитектурное описание?

Согласно ISO 42010, архитектура — это фундаментальные концепции или свойства системы, воплощенные в ее элементах, отношениях и принципах ее дизайна и эволюции.

Архитектурное описание — это рабочий продукт, используемый для выражения архитектуры.

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

Архитектурные решения напрямую влияют на метрики проекта (например, производительность, масштабируемость, поддерживаемость, безопасность, стоимость). Хороший архитектурный выбор улучшает эти метрики, а плохой может привести к серьезным проблемам.

V-диаграмма - это визуальная модель разработки программного обеспечения, которая показывает взаимосвязь между этапами разработки (development) и соответствующими этапами тестирования (testing). Особенность заключается в том, что она подчеркивает, что тестирование должно планироваться параллельно процессу разработки, а не после его завершения.

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

Арифмометр:

Принцип: Арифмометр - это механическое вычислительное устройство, которое выполняет операции сложения, вычитания, умножения и деления с использованием зубчатых колёс и рычагов. Каждое колесо представляет собой десятичный разряд, и операции выполняются путем физического перемещения этих колёс.

Аналогия с современными компьютерами: Арифмометр можно сравнить с цифровым компьютером. Он оперирует дискретными значениями (цифрами), и операции выполняются последовательно, шаг за шагом, в соответствии с заданным алгоритмом. Это похоже на то, как современный процессор обрабатывает инструкции, закодированные в бинарном виде. Фиксация состояний (положение зубчатых колёс) аналогична хранению данных в памяти компьютера.

Логарифмическая линейка:

Принцип: Логарифмическая линейка - это аналоговое вычислительное устройство, использующее логарифмические шкалы. Умножение и деление сводится к сложению и вычитанию длин на этих шкалах. Результат считывается визуально.

Аналогия с современными компьютерами: Логарифмическую линейку можно сравнить с аналоговым компьютером. Она работает с непрерывными величинами (длинами), представляющими логарифмы чисел. Результаты вычислений представлены в аналоговой форме, и точность ограничена точностью нанесения шкал и точностью считывания. В современных компьютерах, аналоговые вычисления используются, например, в системах обработки сигналов.

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

**Подход к расчёту артиллерийских таблиц группой людей:**

* **Разделение труда:** Работа делилась на множество простых операций (сложение, умножение), каждая выполнялась отдельным человеком.
* **Инструменты:** Использовались арифмометры, логарифмические и тригонометрические таблицы.
* **Организация:** Строгий процесс с многоступенчатой проверкой для минимизации ошибок.

**Аналогии с современными компьютерами:**

* **Параллелизм:** Распределение задач между людьми схоже с параллельной обработкой в многоядерных процессорах.
* **Алгоритм:** Процесс расчёта (шаблон таблицы) аналогичен компьютерной программе.
* **Память:** Справочные таблицы как хранилище данных.
* **Отладка:** Проверка результатов на каждом шаге как отладка программного обеспечения.

1. Каково устройство электрического реле? Какие виды реле существуют и каковы области их применения?

Реле состоит из:

* Катушка (1): Наматывается на ферромагнитный сердечник (2), создавая магнитное поле при подаче тока.
* Якорь (3): Подвижный элемент, который притягивается к сердечнику под действием магнитного поля.
* Контакты (4): Коммутируются якорем. Бывают нормально разомкнутые (НР) и нормально замкнутые (НЗ).
* Основание (5): Конструктивный элемент, на котором закреплены все части реле.
* Пружина (6): Возвращает якорь в исходное положение, когда ток в катушке отсутствует.
* **Принцип работы:**
* При подаче напряжения на катушку (1) создается магнитное поле, которое притягивает якорь (3). Якорь перемещается и замыкает или размыкает контакты (4), тем самым коммутируя цепь. При снятии напряжения с катушки пружина (6) возвращает якорь в исходное положение.

**Виды реле (по слайду):**

Слайд перечисляет:

* Механические
* Пневматические
* Тепловые
* Оптические
* Акустические
* Магнитные

**Области применения (дополнительно, не из слайда):**

* **Автоматика:** Управление различными устройствами и процессами.
* **Защита:** Защита электрических цепей от перегрузок и коротких замыканий.
* **Телекоммуникации:** Коммутация телефонных линий.
* **Электротехника:** Управление мощными цепями с помощью слаботочных сигналов.
* **Автомобильная промышленность:** Управление различными функциями автомобиля (фары, стартер и т.д.).

1. Как реализуется булев базис на электрических реле?

Реализация булева базиса на электрических реле основана на использовании реле в качестве ключей, которые могут замыкать или размыкать электрические цепи. Основные логические операции реализуются следующим образом:

**1. Инверсия (NOT):**

* Используется нормально замкнутый (НЗ) контакт реле.
* Если на катушку реле подано напряжение (вход = 1), якорь переключается, и НЗ контакт размыкается (выход = 0).
* Если на катушку реле не подано напряжение (вход = 0), якорь находится в исходном положении, и НЗ контакт замкнут (выход = 1).

**2. Конъюнкция (AND):**

* Используются два или более реле, соединенные последовательно.
* Ток пойдет на выход только если оба реле (и все реле в цепочке) будут включены, то есть на катушки всех реле должно быть подано напряжение (входы = 1).
* Если хотя бы на одно реле не подано напряжение (хотя бы один вход = 0), цепь разомкнется, и выход будет равен 0.

**3. Дизъюнкция (OR):**

* Используются два или более реле, соединенные параллельно.
* Ток пойдет на выход, если хотя бы одно реле будет включено, то есть хотя бы на одну катушку реле должно быть подано напряжение (хотя бы один вход = 1).
* Только если на все реле не подано напряжение (все входы = 0), цепь будет разомкнута, и выход будет равен 0.

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

Программируемые логические контроллеры (ПЛК) - это специализированная разновидность электронной вычислительной машины, предназначенная для управления различными технологическими процессами и оборудованием.

**Области применения:**

Судя по контексту и общим знаниям, ПЛК применяются в:

* Управляющих системах
* Встраиваемых системах
* Автоматизации производства
* Энергетике
* Транспорте
* И многих других отраслях, где требуется автоматизация и управление процессами.

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

* **Центральный процессор (CPU):** Выполняет логические операции и управляет работой ПЛК. Включает микропроцессор, память (ОЗУ, ПЗУ, EEPROM), таймеры, счетчики и другие функциональные блоки.
* **Модули ввода/вывода (I/O Modules):** Обеспечивают связь ПЛК с внешним миром.
  + **Входные модули:** Получают сигналы от датчиков, кнопок, переключателей и других устройств, преобразуют их в цифровой формат, понятный для CPU.
  + **Выходные модули:** Преобразуют цифровые сигналы от CPU в сигналы, необходимые для управления исполнительными устройствами (двигателями, клапанами, реле, лампами и т.д.).
* **Источник питания (Power Supply):** Обеспечивает ПЛК необходимым напряжением и током.
* **Программирующий интерфейс/Устройство программирования (Programming Device/Interface):** Используется для загрузки, отладки и мониторинга программы, выполняемой ПЛК. Это может быть ПК с специализированным программным обеспечением, портативный программатор или панель оператора.
* **Память (Memory):**
* Отделение инструментальной составляющей (среды разработки и программирования) от самого ПЛК обусловлено несколькими причинами:
* **Удобство разработки и отладки**
* **Совместимость и гибкость**
* **Независимость от аппаратной платформы**
* **Разделение функций**
* **Возможность удалённого доступа**

1. Каковы особенности аппаратного обеспечения программируемых логических контроллеров (ПЛК)?

**Особенности аппаратного обеспечения ПЛК:**

* **Модульность:** Возможность расширения путем добавления модулей ввода/вывода.
* **Прочность:** Устойчивость к вибрациям, температурам, влажности, помехам.
* **Специализированные интерфейсы:** Для подключения датчиков и исполнительных механизмов.
* **Реальное время:** Быстрая и предсказуемая реакция на события.
* **Надежность:** Долгий срок службы.

1. Каковы особенности программного обеспечения программируемых логических контроллеров (ПЛК)?

* **Цикличность:** Программа выполняется циклически (сканирование входов, выполнение логики, обновление выходов).
* **Языки программирования:** Специализированные языки (лестничные диаграммы, функциональные блоки, структурированный текст).
* **Реальное время:** Обеспечение предсказуемого времени выполнения операций.
* **Встроенные библиотеки:** Стандартные функции для управления оборудованием.
* **Конфигурация:** Настройка параметров работы ПЛК и модулей ввода/вывода.

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

Принцип развития иерархических систем Седова заключается в **ограничении разнообразия на нижележащем уровне для получения разнообразия на вышележащем уровне.** Иными словами, на более низком уровне системы вводятся стандарты и ограничения, что позволяет создавать более сложные и разнообразные функциональности на более высоких уровнях.

Рассмотрим архитектуру компьютера, которая является типичной иерархической системой:

* **Нижний уровень: Физический уровень (транзисторы, логические элементы).** Здесь разнообразие сильно ограничено. Используются строго определенные типы транзисторов и логических элементов (AND, OR, NOT и т.д.). Это ограничение необходимо для обеспечения надежной и предсказуемой работы.
* **Следующий уровень: Архитектура процессора (набор инструкций, регистры, кэш-память).** Ограниченное число базовых инструкций (сложение, вычитание, умножение, сравнение, перемещение данных) формирует основу для более сложных операций. Архитектура регламентирует структуру регистров и кэш памяти. Несмотря на это, появляются различные типы процессоров, различающиеся тактовой частотой, количеством ядер, размером кэша и т.д.
* **Уровень операционной системы (ОС):** ОС предоставляет интерфейс для управления аппаратными ресурсами. Ограниченное число системных вызовов позволяет программам взаимодействовать с "железом". ОС стандартизирует доступ к памяти, файловой системе, устройствам ввода/вывода. Это позволяет разрабатывать приложения, которые могут работать на разных типах "железа", поддерживаемых данной ОС.
* **Уровень прикладного программного обеспечения:** Используя ограниченный набор системных вызовов ОС и библиотеки, разработчики могут создавать огромное разнообразие программ: текстовые редакторы, игры, веб-браузеры, системы искусственного интеллекта и т.д. Чем выше уровень, тем больше разнообразия.

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

В вычислительной технике булев базис - это набор базовых логических операций, с помощью которых можно выразить любую другую булеву функцию. Он является основой для построения цифровых схем и компьютерной логики.

Роль булева базиса заключается в том, что он позволяет упростить и стандартизировать разработку цифровых устройств, так как любая сложная логическая функция может быть реализована с помощью комбинации этих базовых операций.

Примеры применения в вычислительной технике:

* **Цифровые схемы:**

Булев базис используется для построения логических вентилей (элементов) в цифровых схемах, таких как процессоры и память.

* **Программирование:**

Булевы переменные и операции используются для управления потоком выполнения программы, в условных операторах (if, else) и циклах.

1. Что такое двоичное кодирование сигналов? Каковы его достоинства и недостатки? Что такое запретная зона?

Двоичное кодирование — это метод представления информации с использованием всего двух символов, обычно 0 и 1, каждый из которых представляет собой двоичный разряд или бит.

#### Достоинства двоичного

1. Надежно и помехоустойчиво.
2. Простая арифметика.
3. Диапазоны и точность наращиваются разрядностью (слева и справа соответственно).
4. Погрешности "by design", а не "by implementation"

#### Недостатки

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

В контексте двоичного кодирования запретная зона - это диапазон значений, которые не могут быть точно представлены в рамках конкретного количества битов.

1. Какова роль машинного слова в устройстве процессора? Что такое Big- и Little-endian?

Машинное слово - это единица данных фиксированного размера, с которой процессор работает как с единым целым, используя команды и аппаратное обеспечение.

Размер машинного слова, измеряемый в битах, является важной характеристикой архитектуры процессора. Определяет, как данные хранятся и обрабатываются в памяти, а также как процессор работает с регистрами и памятью.

**Big-endian и Little-endian:** Это способы хранения многобайтовых данных (например, чисел) в памяти.

* **Big-endian:** Старший байт хранится по младшему адресу. (как чтение слева направо)
* **Little-endian:** Младший байт хранится по младшему адресу.

1. Какие существуют способы кодирования целочисленных данных? Что такое позиционное кодирование, код Грея, BCD?

**Способы кодирования целочисленных данных:**

* **Позиционное кодирование:** Основано на представлении числа как суммы степеней основания системы счисления, умноженных на соответствующие цифры. Наиболее распространенная - двоичная (основание 2), но также используются десятичная (основание 10), восьмеричная (основание 8) и шестнадцатеричная (основание 16).
* **Дополнительный код:** Используется для представления отрицательных чисел. Позволяет выполнять операции сложения и вычитания с отрицательными числами как с положительными.
* **Код Грея:** Две "соседние" кодовые комбинации различаются только цифрой в одном двоичном разряде. Используется для минимизации ошибок при переключении между состояниями.
* **BCD (Binary-Coded Decimal):** Каждый десятичный разряд числа записывается в виде его четырехбитного двоичного кода. Используется для представления чисел, когда важна точность десятичного представления (например, в финансовых приложениях).

1. Какие существуют способы кодирования бинарных данных? Что такое Base64, Base58?

**Способы кодирования бинарных данных:**

* **Base64:** Стандарт кодирования двоичных данных с использованием 64 символов ASCII (латинские буквы A-Z, a-z, цифры 0-9 и 2 дополнительных символа).
* **Base58:** Похож на Base64, но из алфавита исключены не-алфанумерические символы и буквы, которые могут быть неоднозначно интерпретированы (0, I, O, l).

**Base64:** Используется для передачи двоичных данных в текстовом формате, например, в электронных письмах или URL-адресах.

**Base58:** Используется в криптовалютах, таких как Bitcoin, для представления адресов и приватных ключей. Это делает их более удобными для ручного ввода и уменьшает вероятность ошибок.

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

**Комбинационная схема -** схема, составленная из набора логических элементов, в совокупности реализующая заданную таблицу истинности.

**Переходный процесс** – изменение выходных сигналов схемы в ответ на изменение входных сигналов

**Задержки и накопление ошибки связаны с:**

* **Физическими свойствами элементов:** Транзисторы не переключаются мгновенно, у проводов есть емкость и индуктивность, что замедляет распространение сигнала.
* **Схемотехникой:** Чем сложнее схема, тем больше задержек, т.к. сигнал проходит через большее количество логических элементов.
* **Температурой:** Характеристики элементов могут меняться в зависимости от температуры.
* **Технологическим разбросом:** Из-за различий в процессе производства элементы имеют немного отличающиеся характеристики.
* **Связь с параллелизмом уровня бит:**
* Комбинационные схемы лежат в основе арифметико-логических устройств (АЛУ), выполняющих арифметические и логические операции.
* В процессорах, поддерживающих параллелизм на уровне бит, АЛУ способны одновременно обрабатывать несколько битов данных.
* Для реализации параллелизма уровня бит используются комбинационные схемы, выполняющие операции над каждым битом операндов параллельно. Ускорение достигается за счет одновременной работы с несколькими битами, а не последовательной обработки каждого бита.

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

В комбинационных схемах, кроме 0 и 1, могут встречаться состояния X и Z:

* **0 (Логический ноль, Low):** Низкий уровень напряжения, представляющий логическое "ложь".
* **1 (Логическая единица, High):** Высокий уровень напряжения, представляющий логическое "истина".
* **X (Неопределённое состояние, Don't care, Unknown):** Состояние, когда логический уровень сигнала неизвестен или не определен. Это может быть результатом:
  + Неинициализированного сигнала.
  + Гонки сигналов (конфликта) при переключении.
  + Ошибки в схеме.
  + Неопределенного значения в таблице истинности при проектировании.
* **Z (Высокоимпедансное состояние, High impedance, Floating):** Состояние, когда выход схемы фактически отключается от цепи. Схема не выдает ни 0, ни 1, а ведет себя как разомкнутая цепь. Используется для реализации мультиплексоров, тристабильных буферов и других схем, где требуется динамически отключать выход.

1. Каковы особенности реализации "условного оператора" в комбинационных схемах?

Реализация условного оператора (if-then-else) в комбинационных схемах осуществляется с помощью **мультиплексора (MUX)**. Мультиплексор выбирает один из нескольких входных сигналов и перенаправляет его на выход в зависимости от значения управляющего сигнала (сигнала выбора).

**Как это работает:**

1. **Сигнал условия (condition)**: Выступает в качестве управляющего сигнала для мультиплексора.
2. **Входы мультиплексора**:
   * Вход, соответствующий блоку "then" (выход при условии "истина").
   * Вход, соответствующий блоку "else" (выход при условии "ложь").
3. **Выход мультиплексора:** Предоставляет результат условного оператора.

**Особенности реализации:**

* **Комбинационная природа**: Вся схема представляет собой комбинационную схему, т.е. выход определяется только текущими входными значениями. Нет состояния, как в последовательных схемах.
* **Обязательное вычисление обоих блоков**: В отличие от программного кода, где вычисляется только один блок (then или else) в зависимости от условия, комбинационная схема должна иметь аппаратную реализацию для обоих блоков. Мультиплексор выбирает один из уже вычисленных результатов.
* **Скорость**: Скорость работы условного оператора определяется временем задержки распространения сигнала через мультиплексор.
* **Площадь**: Для реализации условного оператора требуется место для реализации мультиплексора и обеих ветвей (then и else). Это может быть существенно больше, чем если бы использовалась последовательная реализация (например, в микропрограмме).
* **Вложенность**: Сложные условные конструкции реализуются путем каскадирования мультиплексоров.
* **Необходимость полной спецификации**: Для каждого возможного значения входных сигналов (включая состояние условия) должен быть определен выход. Неопределенные состояния (X) требуют особого внимания.

1. 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 внешних сигналов.

**Варианты использования триггеров:**

1. **Хранение информации:** Основное применение триггеров - хранение битов информации в регистрах, счетчиках, памяти. Они позволяют запоминать состояние схемы.
2. **Счетчики:** Триггеры, соединенные последовательно, могут формировать счетчики, используемые для подсчета импульсов или событий.
3. **Регистры сдвига:** Триггеры могут быть организованы в регистры сдвига, где данные сдвигаются от одного триггера к другому под управлением тактового сигнала. Используются для последовательной передачи данных.
4. **Синхронизация данных:** Триггеры используются для синхронизации данных, поступающих с разных источников, к общему тактовому сигналу, обеспечивая стабильность и предсказуемость работы цифровой системы.
5. **Делители частоты:** Триггеры могут быть использованы для деления частоты тактового сигнала на два или более.
6. **Формирователи импульсов:** Триггеры могут использоваться для формирования импульсов заданной длительности и формы.
7. **Элементы памяти в конечных автоматах:** Триггеры используются для хранения текущего состояния конечного автомата.
8. Что такое D-триггер и RS-триггер? Какие существуют варианты условия изменения состояния?

* **D-триггер** (от англ. Delay) — запоминает состояние входа и выдаёт его на выход.
* RS-триггер (от англ. Reset/Set) — асинхронный триггер, который сохраняет своё предыдущее состояние при неактивном состоянии обоих входов и изменяет своё состояние при подаче на один из его входов активного уровня.

**Варианты условия изменения состояния (для обоих триггеров):**

Условие изменения состояния триггера определяется **тактирующим сигналом (CLK)**. Существуют следующие варианты:

* 1. **Триггер, управляемый уровнем:**
     + Состояние триггера изменяется, когда тактовый сигнал находится в активном состоянии (высокий или низкий уровень).
     + Проблема: чувствительность к шумам на входе D (для D-триггера) или на входах S и R (для RS-триггера) во время активного уровня тактового сигнала.
  2. **Триггер, управляемый фронтом (импульсом):**
     + Состояние триггера изменяется только в момент перехода тактового сигнала (с низкого уровня на высокий - положительный фронт, или с высокого на низкий - отрицательный фронт).
     + Управляемый фронтом триггер более устойчив к шумам, так как он "защелкивает" значение входного сигнала только в момент фронта тактового импульса.
     + **Положительный фронт (Positive edge-triggered):** Изменение состояния происходит при переходе CLK от 0 к 1.
     + **Отрицательный фронт (Negative edge-triggered):** Изменение состояния происходит при переходе CLK от 1 к 0.

1. Что такое пространственные и временные вычисления? Как они могут быть использованы для оптимизации процессоров?

**Пространственные вычисления:** Распараллеливание задач по разным ресурсам (ядрам, ускорителям) для одновременной обработки данных.

**Временные вычисления:** Распараллеливание во времени через конвейерную обработку, когда каждый этап задачи выполняется последовательно, но одновременно для разных данных.

**Применение для оптимизации процессоров:**

1. **Пространственные вычисления:**
   * **Многоядерные процессоры:** Разные ядра могут выполнять разные части задачи одновременно.
   * **Использование специализированных ускорителей (GPU, TPU, FPGA):** Отдельные части алгоритма, требующие высокой производительности (например, операции с плавающей точкой, машинное обучение), могут быть перенесены на специализированные ускорители, которые работают параллельно с основным процессором.
   * **Масштабирование АЛУ (Арифметико-логического устройства):** Реализация параллельных сумматоров, умножителей и т.д., что позволяет выполнять операции над несколькими битами данных одновременно.
2. **Временные вычисления:**
   * **Конвейерная обработка инструкций:** Разбиение выполнения каждой инструкции на несколько этапов (извлечение, декодирование, выполнение, запись результата) и параллельное выполнение разных этапов для разных инструкций. Это увеличивает пропускную способность процессора.
   * **Суперскалярные процессоры:** Выполнение нескольких инструкций одновременно, если они не зависят друг от друга (инструкции выполняются "не по порядку"). Используются временные вычисления для эффективного использования ресурсов процессора.
   * **Цифровые сигнальные процессоры (DSP):** Используют конвейерную архитектуру для эффективной обработки потоковых данных, таких как аудио- и видеосигналы.
3. Что называют синхронной схемотехникой? Каковы её достоинства и недостатки?

**Синхронная схемотехника** – это подход к проектированию цифровых схем, при котором работа всех элементов схемы (триггеры, регистры, и т.д.) координируется общим **тактовым сигналом**. Изменение состояния элементов происходит только по фронту или уровню этого тактового сигнала.

**Достоинства:**

* **Простота проектирования и анализа:** Наличие тактового сигнала упрощает определение последовательности операций и анализ временных соотношений.
* **Предсказуемость поведения:** Работа схемы строго детерминирована, что упрощает отладку и верификацию.
* **Упрощенное тестирование:** Благодаря предсказуемости, легче разрабатывать тесты для выявления ошибок.
* **Модульность:** Легче создавать сложные системы из отдельных синхронных блоков.

**Недостатки:**

* **Ограничение быстродействия:** Максимальная тактовая частота ограничивается самым медленным элементом схемы (худшим случаем задержки распространения сигнала).
* **Потребление энергии:** Тактовый сигнал постоянно переключается, потребляя энергию, даже если схема не выполняет никаких полезных операций.
* **Синхронный шум (Clock skew):** Неравномерное распространение тактового сигнала по схеме может привести к проблемам с синхронизацией и нарушению работы схемы.
* **Глобальный сигнал:** Тактовый сигнал должен быть доступен всем элементам схемы, что усложняет проектирование и разводку для больших и сложных систем.
* **Неэффективность для асинхронных задач:** Для задач, где данные приходят нерегулярно или с переменной скоростью, синхронная схема может быть неэффективна.

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

Цифровая схемотехника оперирует уровнями (логическими уровнями - 0 и 1, High/Low), а не аналоговыми сигналами из-за следующих причин:

**Почему уровнями, а не сигналами:**

1. **Помехоустойчивость:** Уровни обеспечивают устойчивость к шумам и помехам. Небольшие колебания напряжения в пределах определенного диапазона не влияют на интерпретацию сигнала как "0" или "1". Если бы использовались аналоговые сигналы, даже малые искажения могли бы привести к неверной интерпретации данных.
2. **Абстракция:** Цифровые схемы позволяют абстрагироваться от физических характеристик элементов и оперировать логическими значениями. Это упрощает проектирование и анализ сложных систем.
3. **Возможность воспроизведения и масштабирования:** Цифровые схемы могут быть воспроизведены с высокой точностью, поскольку точное значение напряжения не имеет значения, важен только уровень.
4. **Возможность логической обработки:** Логические элементы (AND, OR, NOT и т.д.) оперируют с уровнями, позволяя реализовать сложные логические функции.

**Возможности, которые это открывает:**

1. **Создание сложных цифровых систем:** Построение процессоров, памяти, и других сложных устройств, используя логические элементы, оперирующие уровнями.
2. **Абстракция от физических деталей:** Упрощение процесса проектирования и верификации.
3. **Надежная передача и хранение данных:** За счет помехоустойчивости и возможности коррекции ошибок.
4. **Алгоритмизация и автоматизация проектирования:** Использование САПР (систем автоматизированного проектирования) для создания цифровых схем.

**Проблемы, которые это создает:**

1. **Дискретизация:** Преобразование аналогового сигнала в цифровой (уровни) неизбежно приводит к потере информации. (квантование)
2. **Задержки распространения сигнала:** Элементы не переключаются мгновенно.
3. **Метастабильность:** Неопределенное состояние, возникающее при нарушении временных соотношений, например, при асинхронной передаче данных.
4. **Переходные процессы:** Переключения между уровнями могут создавать паразитные эффекты (дребезг контактов, звон), которые необходимо учитывать при проектировании.
5. **Потребление электроэнергии.** Переключение уровней, особенно на высоких частотах, приводит к значительному потреблению энергии, особенно в CMOS схемах.
6. 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

1. Что такое навесной монтаж, монтаж на печатную плату, штырьевой и поверхностный монтаж? Как они обеспечивают "гибкость"?

**Навесной монтаж:** Компоненты соединяются проводами непосредственно в воздухе между точками пайки. **Гибкость:** Легкость изменения схемы путем перепайки.

**Монтаж на печатную плату (ПП):** Компоненты паятся на дорожки, выгравированные на печатной плате. **Гибкость:** Модулярность и замена отдельных компонентов.

**Штырьевой монтаж:** Компоненты с штырьками вставляются в отверстия платы и припаиваются с обратной стороны. **Гибкость:** Надежное соединение, легко заменять компоненты.

**Поверхностный монтаж (SMD):** Компоненты устанавливаются и припаиваются прямо на поверхность печатной платы. **Гибкость:** Высокая плотность компоновки, автоматизация монтажа, уменьшение размеров устройств.

1. Какова производственная цепочка поверхностного монтажа? Каковы её этапы?

Производственная цепочка поверхностного монтажа (SMD) состоит из нескольких основных этапов:

1. **Проектирование схемы и печатной платы:** Создание схемы и трассировка дорожек на плате с учетом расположения поверхностных компонентов.
2. **Изготовление печатных плат:** Производство плат с нанесением дорожек. Платы могут быть однослойными или многослойными.
3. **Нанесение паяльной пасты:** С помощью трафарета паяльная паста наносится на контактные площадки, куда будут установлены компоненты.
4. **Установка компонентов:** Автоматизированные машины (pick-and-place) размещают компоненты на плате согласно проекту.
5. **Оплавление паяльной пасты:** Плата проходит через печь, где паяльная паста плавится и создаёт электрическое соединение между компонентами и платой.
6. **Инспекция качества:** Оптический или рентгеновский контроль для проверки качества монтажа и правильности установки компонентов.
7. **Тестирование:** Электрическое тестирование работоспособности собранной платы.
8. Какова производственная цепочка кремниевого производства? Каковы особенности формирования цены изделия?

Производственная цепочка кремниевого производства включает следующие основные этапы:

1. **Выращивание кремниевых слитков:** Используя метод Чохральского, получают монокристаллические силиконовые слитки.
2. **Нарезка пластин:** Слитки нарезаются на тонкие пластины.
3. **Обработка пластин:**
   * **Предварительное шлифование и полировка:** Для получения гладкой поверхности.
   * **Очищение:** Удаление загрязнений.
4. **Фотолитография:** Нанесение светочувствительных слоев для создания схем.
5. **Допирование:** Добавление примесей в кремний для изменения его электрических свойств.
6. **Травление:** Изменение поверхности пластин для формирования компонентов схем.
7. **Осаждение слоев (CVD, PVD):** Нанесение новых материалов для создания слоев устройства.
8. **Металлизация:** Нанесение металлических контактов.
9. **Тестирование и упаковка:** Проверка работоспособности и заключение в корпус.

**Особенности формирования цены изделия:**

1. **Сложность технологического процесса:** Чем сложнее процесс, тем выше стоимость.
2. **Чистота и качество материалов:** Использование высокочистого кремния и других материалов влияет на цену.
3. **Количество и плотность компонентов:** Чем больше компонентов на пластине, тем выше стоимость производства.
4. **Разработка и исследования:** Затраты на R&D для разработки новых технологий и улучшения существующих.
5. **Масштабы производства:** Крупные партии могут уменьшить себестоимость за счет эффекта масштаба.
6. **Отбраковка:** Значительная часть пластин может быть отвернута из-за дефектов, что увеличивает стоимость изделий, прошедших контроль.
7. Какие трудности связаны с аппаратным обеспечением на этапах производства и эксплуатации?

Производство:

Стоимость:

Разработка и производство сложного оборудования требует значительных инвестиций в исследования, разработки, материалы и производственные мощности.

Сложность интеграции:

Интеграция аппаратных компонентов в единую систему может быть сложной задачей, особенно при использовании различных технологий и стандартов.

Качество материалов и компонентов:

Недостаточное качество материалов или компонентов может привести к снижению надежности и долговечности оборудования.

Проблемы масштабирования:

Масштабирование производства для удовлетворения растущего спроса может быть сложным и дорогостоящим процессом.

Эксплуатация:

Необходимость специализированного обслуживания:

Эксплуатация сложного оборудования требует квалифицированного персонала для обслуживания, ремонта и устранения неисправностей.

Риски кибербезопасности:

Оборудование может быть уязвимым для кибератак, что может привести к остановке производства, потере данных или другим серьезным последствиям.

Устаревание:

Технологии быстро развиваются, и оборудование может устаревать, что требует дорогостоящих обновлений или замены.

Сложности с поставками запчастей:

В случае поломки или неисправности оборудования, поиск и доставка необходимых запчастей может занять много времени и средств.

Влияние окружающей среды:

Оборудование может быть чувствительно к условиям окружающей среды, таким как температура, влажность, вибрация, что может повлиять на его работу и срок службы.

Проблемы совместимости:

Аппаратное и программное обеспечение могут быть несовместимы, что требует дополнительных усилий для обеспечения их бесперебойной работы.

1. Какие подходы к решению проблемы "устаревающей аппаратуры" существуют с точки зрения аппаратуры и ПО?

Аппаратные подходы:

1. **Модульность и расширяемость:**
   * Использование модульного дизайна, где возможно обновление отдельных компонентов без замены всей системы (например, замена процессора, добавление оперативной памяти).
2. **Обновление прошивки:**
   * Возможность обновления встроенного ПО (прошивки) для улучшения функциональности или исправления ошибок без физической замены компонентов.
3. **Использование стандартных интерфейсов:**
   * Применение общепринятых стандартов и интерфейсов позволяет легко заменить устаревшие части системы.
4. **Производство по технологии "forever young":**
   * Создание "устойчивых к устареванию" аппаратных платформ с длительным жизненным циклом, например, для промышленных систем.

Программные подходы:

1. **Эмуляция:**
   * Использование ПО для эмуляции устаревшего оборудования, что позволяет запускать старые приложения на новой аппаратуре.
2. **Виртуализация:**
   * Применение виртуальных машин для хранения и запуска ПО, требующего устаревшее оборудование, на современных системах.
3. **Обновление и поддержка ПО:**
   * Регулярные обновления операционных систем и приложений для их адаптации к новым аппаратным средствам.
4. **Интерпретаторы и трансляторы:**
   * Использование языков программирования и интерпретаторов, которые позволяют программам работать на различных платформах без модификации исходного кода.
5. **Открытые стандарты и системы:**
   * Применение открытых систем и стандартов, которые легче поддерживаются и обновляются сообществом разработчиков.
6. Что такое концепция 2-этапного производства? Какие существуют варианты этапа "конфигурирования" для разных уровней организации вычислений?

**Концепция 2-этапного производства** предполагает создание универсальной аппаратной платформы на первом этапе и настройку её приложений на втором этапе.

Этапы:

1. **Производство универсальной системы:** Создание аппаратных платформ, которые могут быть адаптированы для различных задач и приложений.
2. **Конфигурирование и настройка:** Подстройка платформы под конкретные нужды с помощью специфического программного обеспечения.

Варианты конфигурирования:

1. **Для уровня архитектуры:**
   * Изменение на уровне микропрограммы процессора.
   * Перепрограммируемые логические матрицы (FPGA) для создания уникальных процессов.
2. **Для уровня системной интеграции:**
   * Использование программных интерфейсов и драйверов для интеграции с внешними устройствами.
   * Настройка операционных систем для поддержки специфических функций.
3. **Для уровня приложений:**
   * Использование API и SDK для разработки пользовательских приложений.
   * Настройка приложений и сервисов для лучшего использования ресурсов платформы.
4. Каково определение программной системы согласно OMG Essence? Каковы её части?

Согласно OMG Essence, программная система - это система, состоящая из программного обеспечения, аппаратного обеспечения и данных, которая предоставляет свою основную ценность посредством выполнения программного обеспечения.

Её части:

* software (программное обеспечение)
* hardware (аппаратное обеспечение)
* data (данные, необходимые для работы и являющиеся частью системы)

1. 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 вычислительная система хранит и обрабатывает).

Аппаратное обеспечение включает: компьютеры и логические устройства, внешние устройства и диагностическую аппаратуру, энергетическое оборудование, батареи и аккумуляторы.

**Программное обеспечение (ПО) -** совокупность программ, системы обработки информации и программных документов, необходимых для эксплуатации.

Позволяет аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления

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.

1. Какие возможности открывает программное обеспечение (цикл разработки, гибкость, контроль, и т.д.)?
2. Быстрый цикл разработки.
3. Легко заменяется прямо у пользователя.
4. Пользователь как Beta-тестер.
5. Возможно удалённое обновление, в том числе без информированного согласия.
6. Сервис — вершина владения компьютерной системой.
7. Процесс создания и внедрения ПО автоматизируется (CI/CD).
8. Высокая сложность программ
9. Чем отличается типовое проектирование Hardware/Software от совместного (CoDesign) проектирования Hardware/Software? Каковы их достоинства и недостатки?

**Типовое проектирование Hardware/Software**:

* **Описание**: Раздельная разработка аппаратного (Hardware) и программного (Software) обеспечения, где каждая часть проектируется независимо, с последующей интеграцией.
* **Достоинства**:
  + Простота управления проектом, так как команды работают независимо.
  + Возможность использования готовых аппаратных платформ.
  + Меньшая сложность координации на ранних этапах.
* **Недостатки**:
  + Позднее обнаружение проблем интеграции (например, несоответствие требований).
  + Ограниченная оптимизация производительности из-за слабого взаимодействия.
  + Длительный цикл разработки при необходимости доработок.

**Совместное проектирование (CoDesign) Hardware/Software**:

* **Описание**: Одновременная разработка аппаратного и программного обеспечения с тесным взаимодействием команд для оптимизации системы.
* **Достоинства**:
  + Высокая оптимизация производительности и энергоэффективности.
  + Раннее выявление и устранение проблем интеграции.
  + Ускорение цикла разработки за счет параллельной работы.
* **Недостатки**:
  + Высокая сложность управления и координации между командами.
  + Требуются специалисты с междисциплинарными знаниями.
  + Увеличенные затраты на начальных этапах из-за интенсивного взаимодействия.

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

**Что такое "Модель вычислений"?** Модель вычислений (Model of Computation — MoC) — это формальная структура или абстракция, описывающая, как выполняются вычисления, включая алгоритмы, входные и выходные данные, а также способ реализации (электронный, клеточный и т.д.).

**Назначение моделей вычислений:**

* Определение языка для описания алгоритмов и программ.
* Описание возможностей вычислительной машины.
* Характеризация исполнения программ (возможные состояния, их последовательность).
* Обеспечение единого подхода к анализу и сравнению вычислительных систем.

**Примеры:**

1. Машина Тьюринга — теоретическая модель для универсальных вычислений.
2. Регистровая машина — модель с конечным числом регистров для арифметических операций.
3. Клеточные автоматы — модель для параллельных вычислений (например, "Игра Жизнь").
4. Что такое последовательные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс?

**Что такое последовательные модели вычислений?** Последовательные модели вычислений (Sequential Models of Computation) — это формальные системы, описывающие вычислительный процесс как последовательность состояний или переходов, выполняемых шаг за шагом, где каждое состояние зависит от предыдущего. Они предполагают однопоточный порядок выполнения инструкций.

**Примеры:**

1. Finite state machines (Конечные автоматы).
2. Pushdown automata (Стековые автоматы).
3. Turing machines (Машины Тьюринга).
4. Random Access Machines / von Neumann Machine (Машины с произвольным доступом / Архитектура фон Неймана).

**Как представляется вычислительный процесс?** Вычислительный процесс представляется как преобразование входных данных (проблемы) в выходные данные через последовательное выполнение инструкций (t1, t2, t3, ..., tn) процессором (CPU). Каждая инструкция изменяет состояние системы, что отражено в цепочке переходов состояний, управляемых алгоритмом.

1. Что такое машина Тьюринга и почему она важна для теории вычислений?

**Что такое машина Тьюринга?** Машина Тьюринга — это теоретическая модель вычислений, состоящая из бесконечной ленты, головы чтения/записи и конечного набора состояний. Она выполняет операции (чтение, запись, перемещение) по заданным инструкциям, представляя алгоритм как последовательность переходов состояний.

**Почему важна для теории вычислений?**

* Определяет границы вычислимости, показывая, что может и не может быть вычислено алгоритмически.
* Лежит в основе концепции универсальной машины, которая может эмулировать любые другие вычислительные устройства.
* Является основой для формализации алгоритмов и доказательства теорем, таких как неразрешимость проблемы останова.

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

**Что такое Random Access Machine (RAM)?** Random Access Machine — это теоретическая модель вычислений, представляющая компьютер с произвольным доступом к памяти. Она включает процессор, выполняющий инструкции, и память, где данные и программы хранятся по адресам и доступны в произвольном порядке.

**Устройство:**

* Процессор с набором регистров и арифметико-логическим устройством.
* Память, разделённая на ячейки с уникальными адресами.
* Инструкции (например, загрузка, хранение, арифметические операции) выполняются последовательно.

**Место сегодня:** RAM используется как практическая модель для анализа алгоритмов и оценки их сложности (временной и пространственной). Она близка к архитектуре современных процессоров (например, фон Неймана), но упрощена для теоретических целей.

**Связь с машиной Тьюринга:** RAM является расширением машины Тьюринга, добавляя случайный доступ к памяти вместо последовательного сканирования ленты. Обе модели эквивалентны по вычислительной мощности: любая вычислимая функция на машине Тьюринга может быть реализована на RAM и наоборот, при условии бесконечной памяти.

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

Функциональные модели вычислений (Functional Models of Computation) — это подход к описанию вычислений через применение функций к данным, где процесс представлен как последовательное преобразование входных данных в выходные через функции, без явного состояния или последовательности инструкций. Они основаны на математических объектах и рекурсивных вычислениях.

**Примеры:**

1. Арифметика.
2. Лямбда исчисление (полнота по Тьюрингу).
3. Комбинаторная логика.
4. Общие рекурсивные функции.
5. Абстрактные системы переписывания.

**Как представляется вычислительный процесс?** Вычислительный процесс представляется как иерархическое применение функций к данным. Например, выражение (11 + 9) × (2 + 4) разбивается на подвыражения (eval left и eval right), которые вычисляются независимо (20 × 6), и результат (120) получается через рекурсивное преобразование, избегая явных состояний или циклов, что близко к математическому объекту и программированию в функциональном стиле.

1. Что такое паралелильные модели вычислений? Приведите примеры. Как в них представляется вычислительный процесс?

Параллельные модели вычислений (Concurrent Models of Computation) — это формальные системы, описывающие вычисления, выполняемые одновременно несколькими независимыми процессами или потоками, взаимодействующими через обмен данными или синхронизацию.

**Примеры:**

* Kahn process networks — сети процессов с неограниченными буферами.
* Actor model — отправка сообщений между независимыми актерами.
* Discrete Event — события в заданные моменты времени.
* Multi-Thread Model — общая память для потоков.
* Synchronous Data Flow — синхронизированные потоки данных.

**Как представляется вычислительный процесс?** Вычислительный процесс представляется как одновременное выполнение нескольких подпроцессов (например, multiply, add, compare), взаимодействующих через каналы или общую память. На диаграмме видно, как входные данные (in1, in2, in3) параллельно обрабатываются в узлах (sum, scan, vel), а результаты объединяются для получения итогового вывода (out), с учетом синхронизации (carry-propagate, carry-save).

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

**Что такое Model-Driven Engineering (MDE)?** Model-Driven Engineering — это подход к разработке программного обеспечения, где модели используются как основные артефакты. Он включает создание независимых от вычислений (CIM), платформ (PIM) и специфичных для платформ (PSM) моделей, которые автоматически трансформируются в код (Specific Code) с использованием инструментов и правил.

**Отличие цепочки трансформации в MDE от языков высокого уровня:**

* **MDE**: Основан на последовательных трансформациях моделей (CIM → PIM → PSM → Code) с автоматизацией, где каждая стадия абстрагирует детали, фокусируясь на архитектуре и платформе.
* **Языки высокого уровня**: Прямая разработка кода (например, на Python или Java) без промежуточных моделей, где программист управляет всеми деталями реализации.

**Примеры:**

* **MDE**: Трансформация UML-модели (PIM) в код для Android (PSM) с использованием инструмента вроде Eclipse Modeling Framework.
* **Язык высокого уровня**: Написание приложения на C++ без предварительных моделей, с ручным управлением платформой.

1. Что такое универсальный процессор? Каковы его особенности и свойства?

**Что такое универсальный процессор?** Универсальный процессор — это вычислительное устройство, которое не зависит от конкретной машины Тьюринга, не ограничивается практической универсальностью и способно выполнять программы в реальном времени (run-time), включая последовательное исполнение инструкций.

**Особенности и свойства:**

1. **Независимость от машины Тьюринга**: Не привязан к конкретной модели Тьюринга, что обеспечивает гибкость.
2. **Теоретическая универсальность**: Может выполнять любые алгоритмы, которые теоретически вычислимы.
3. **Практическая универсальность**: Поддерживает как аппаратное (Hardware), так и программное (Software) обеспечение.
4. **Отсутствие "серьезных" ограничений**: Гибкость в исполнении программ, адаптируемость под разные задачи.
5. **Изменяемость ПО**: Позволяет модифицировать программное обеспечение для разных целей.
6. В чём заключается противоречие между универсальностью и эффективностью в разных видах процессоров (СБИС, FPGA, CGRA, GPU, DSP, CPU)?

**Противоречие между универсальностью и эффективностью:**

* **ASIC (Application-Specific Integrated Circuit)**: Высокая энергоэффективность (1000 MOPS/mW) и производительность за счёт узкой специализации, но низкая универсальность из-за фиксированного назначения.
* **FPGA (Field-Programmable Gate Array)**: Средняя энергоэффективность (100 MOPS/mW) и гибкость благодаря перепрограммируемости, но менее эффективна, чем ASIC, из-за накладных расходов на конфигурацию.
* **CGRA (Coarse-Grained Reconfigurable Arrays)**: Баланс между энергоэффективностью (100-1000 MOPS/mW) и универсальностью, но сложность программирования снижает практическую гибкость.
* **GPU (Graphics Processing Unit)**: Высокая производительность (GOPS) для параллельных задач, но ограниченная универсальность из-за оптимизации под графику и вычисления.
* **DSP (Digital Signal Processor)**: Энергоэффективность (10-100 MOPS/mW) для сигналов, но узкая специализация ограничивает универсальность.
* **CPU (Central Processing Unit)**: Высокая универсальность благодаря гибкому программированию, но низкая энергоэффективность (1-10 MOPS/mW) из-за общего назначения.

**Суть противоречия**: Чем выше универсальность (гибкость программирования), тем ниже энергоэффективность и производительность, так как специализированные процессоры (ASIC, DSP) оптимизированы для конкретных задач, тогда как универсальные (CPU) теряют эффективность из-за универсальности архитектуры.

1. Как соотносятся вычислительные платформы и языки описания вычислительного процесса? Как это связано с эффективностью и областью применения?

**Соотношение вычислительных платформ и языков описания вычислительного процесса:** Вычислительные платформы (например, ASIC, FPGA, CPU, GPU) определяют аппаратные возможности выполнения задач, тогда как языки описания вычислительного процесса (например, VHDL, Verilog для FPGA, C/C++ для CPU, CUDA для GPU) предоставляют инструменты для программирования этих платформ. Язык должен соответствовать архитектуре платформы: низкоуровневые языки (VHDL) подходят для специализированных платформ (FPGA), а высокоуровневые (Python) — для универсальных (CPU).

**Связь с эффективностью:**

* **Специализированные платформы (ASIC, FPGA)** с низкоуровневыми языками (VHDL) обеспечивают высокую энергоэффективность (до 1000 MOPS/mW) за счёт точной оптимизации, но требуют больше усилий на разработку.
* **Универсальные платформы (CPU)** с высокоуровневыми языками (C++) дают гибкость, но снижают эффективность (1-10 MOPS/mW) из-за обобщённого подхода.
* **Параллельные платформы (GPU)** с языками вроде CUDA достигают высокой производительности (GOPS) для специфических задач, но теряют универсальность.

**Связь с областью применения:**

* ASIC и FPGA с соответствующими языками используются в телекоммуникациях и IoT, где важна энергоэффективность.
* CPU с высокоуровневыми языками подходят для общего назначения (офисные приложения).
* GPU с CUDA применяются в машинном обучении и играх, где требуется параллельная обработка данных.

1. Что такое архитектура и микроархитектура процессора? В чём различие между ними?

**Что такое архитектура и микроархитектура процессора?**

* **Архитектура процессора**: Определяет интерфейс между оборудованием и программным обеспечением, включая набор инструкций (ISA), регистры, модели памяти и основные принципы работы, такие как управление автоматами и типы памяти.
* **Микроархитектура процессора**: Описывает конкретную реализацию архитектуры на уровне оборудования, включая схемы, технологии и детали выполнения инструкций (например, конвейеры, кэши, многократные архитектуры).

**В чём различие между ними?**

* **Уровень абстракции**: Архитектура — это общий стандарт (например, x86), а микроархитектура — конкретная реализация (например, Intel Core).
* **Гибкость**: Архитектура позволяет многим микроархитектурам реализовать один и тот же ISA, тогда как микроархитектура фиксирует детали производительности и энергоэффективности.
* **Связь с ISA**: Архитектура определяет ISA, а микроархитектура реализует её через оптимизации (конвейеры, транзисторы).

1. Что такое система команд и какова её роль в архитектуре процессоров? Что определяет система команд?

Система команд (Instruction Set Architecture, ISA) — это интерфейс между программным обеспечением и оборудованием процессора, определяющий набор инструкций, которые процессор может выполнять, а также форматы данных, регистры и модели памяти.

**Роль в архитектуре процессоров:**

* Обеспечивает абстракцию для программирования, позволяя компиляторам переводить высокоуровневый код в машинные инструкции.
* Определяет взаимодействие между приложением, компилятором и аппаратной частью (датapath, управление, функции).
* Влияет на производительность, энергоэффективность и совместимость процессора.

**Что определяет система команд?**

1. Типы данных и их операции (например, арифметические, логические).
2. Форматы инструкций и их кодирование.
3. Набор регистров и их использование.
4. Механизмы адресации и доступа к памяти.
5. Методы ввода-вывода и прерывания.
6. Что такое машина фон Неймана? Какова её связь с машиной Тьюринга? Каковы её ключевые принципы?

Машина фон Неймана — это архитектура вычислительной системы, где программы и данные хранятся в одной и той же памяти с произвольным доступом (RAM), а процессор (CPU) состоит из блока управления и арифметико-логического блока (ALU) для выполнения инструкций.

**Связь с машиной Тьюринга:** Машина фон Неймана является практической реализацией концепции машины Тьюринга. Обе модели эквивалентны по вычислительной мощности: машина Тьюринга теоретически описывает вычислимость, а машина фон Неймана реализует это через хранилище программ и последовательное выполнение инструкций, хотя у Тьюринга лента бесконечна, а у фон Неймана память конечна.

**Ключевые принципы:**

1. **Хранение программ и данных**: Программы и данные находятся в общей памяти.
2. **Последовательное выполнение**: Инструкции выполняются шаг за шагом.
3. **Разделение на блоки**: CPU (управление и ALU) взаимодействует с памятью.
4. **Адресный доступ**: Память доступна по адресам (Random-Access Memory).
5. **Возможность изменения**: Программы могут быть модифицированы во время работы.
6. Что такое микропроцессор и микроконтроллер? В чём их отличия?

**Что такое микропроцессор?** Микропроцессор — это центральный процессор (CPU) на одном чипе, выполняющий вычисления по заданным инструкциям. Он требует внешней памяти и периферии (например, RAM, ввода-вывода) для работы.

**Что такое микроконтроллер?** Микроконтроллер — это интегрированная система на чипе, включающая процессор, память (RAM, ROM/Flash) и периферийные устройства (таймеры, АЦП) для автономной работы.

**Отличия:**

* **Интеграция**: Микропроцессор — только CPU, микроконтроллер — полный системный блок.
* **Применение**: Микропроцессоры используются в ПК и серверах, микроконтроллеры — в встраиваемых системах (IoT, автомобили).
* **Сложность**: Микропроцессоры требуют внешних компонентов, микроконтроллеры самодостаточны.
* **Мощность**: Микропроцессоры мощнее, микроконтроллеры оптимизированы для энергоэффективности.

1. Что такое Control Unit и Data Path? Каковы их назначение и принципы взаимодействия?

* **Control Unit (CU)**: Компонент процессора, управляющий выполнением инструкций, генерирующий сигналы для синхронизации и координации работы других блоков.
* **Data Path**: Набор функциональных блоков (ALU, регистры, шины), выполняющих арифметические, логические и перемещения данных в соответствии с инструкциями.

**Назначение:**

* **CU**: Обеспечивает последовательность операций (fetch, decode, execute) и управляет потоком данных.
* **Data Path**: Выполняет вычисления и передачу данных между регистрами и памятью.

**Принципы взаимодействия:**

* CU декодирует инструкцию из памяти и отправляет сигналы управления (control signals) в Data Path.
* Data Path выполняет указанные операции (например, сложение в ALU) и возвращает результаты.
* Взаимодействие синхронизируется через тактовый сигнал, обеспечивая циклический процесс (fetch → decode → execute).

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

В машине фон Неймана инструкции классифицируются по их назначению:

1. **Арифметические (Arithmetic)**: Выполняют математические операции (сложение, вычитание).
   * Пример: ADD R1, R2 (R1 = R1 + R2).
2. **Логические (Logical)**: Операции с битами (AND, OR, XOR).
   * Пример: AND R1, R2 (битовая операция AND).
3. **Передачи данных (Data Transfer)**: Перемещение данных между регистрами и памятью.
   * Пример: LOAD R1, [100] (загрузка из адреса 100 в R1).
4. **Управления (Control)**: Изменение потока выполнения (переходы, вызовы).
   * Пример: JMP 200 (переход к адресу 200).
5. **Ввода-вывода (I/O)**: Инструкции для работы с периферией.
   * Пример: OUT R1 (вывод содержимого R1).

**Принципы кодирования:**

* **Фиксированная длина**: Инструкции имеют одинаковую длину (например, 32 бита), что упрощает декодирование.
* **Формат**: Обычно включает опкод (opcode, определяет тип инструкции) и операнды (регистры или адреса).
  + Пример: [Opcode: 6 бит] [Регистр 1: 5 бит] [Регистр 2: 5 бит] [Адрес/Константа: 16 бит].
* **Двоичный код**: Инструкции кодируются в бинарной форме для обработки процессором.
* **Модульность**: Опкоды стандартизированы для совместимости с ISA.
* **Адресация**: Поддержка различных режимов (прямой, косвенный, относительный).

1. Каковы особенности принстонской архитектуры? Каковы её достоинства и недостатки? Какова область применения?

Принстонская архитектура (или архитектура фон Неймана) характеризуется единой памятью для хранения программ и данных, доступной через одну шину. Процессор (CPU) последовательно извлекает инструкции и данные, используя общий адресный и управляющий сигналы.

**Достоинства:**

* Простота конструкции и реализации из-за единой памяти.
* Гибкость в использовании памяти для программ и данных.
* Удобство программирования и отладки.

**Недостатки:**

* Бутылочное горлышко из-за конкуренции между извлечением инструкций и доступом к данным (von Neumann bottleneck).
* Ограниченная пропускная способность шины, что снижает производительность при интенсивных вычислениях.
* Уязвимость к ошибкам из-за единого пространства памяти.

**Область применения:**

* Общие вычислительные системы (ПК, серверы).
* Встраиваемые системы с умеренными требованиями к производительности (бытовая электроника, микроконтроллеры).

1. Каковы особенности гарвардской архитектуры? Каковы её достоинства и недостатки? Какова область применения?

Гарвардская архитектура использует отдельные памяти и шины для программ (инструкций) и данных, с независимыми адресными и управляющими сигналами. Это позволяет одновременно извлекать инструкции и обрабатывать данные.

**Достоинства:**

* Высокая пропускная способность благодаря параллельному доступу к памяти.
* Отсутствие "бутылочного горлышка" фон Неймана, что повышает производительность.
* Улучшенная надежность за счёт разделения памяти.

**Недостатки:**

* Сложность конструкции из-за наличия двух систем памяти.
* Высокие затраты на разработку и производство.
* Ограниченная гибкость при использовании одной памяти для разных задач.

**Область применения:**

* Цифровые сигнальные процессоры (DSP) для аудио- и видеопереработки.
* Встраиваемые системы с высокими требованиями к скорости (например, в телекоммуникациях).
* Микроконтроллеры (например, AVR, PIC) для реального времени.

1. Как организовано адресное пространство в гарвардской и принстонской архитектурах?

**Адресное пространство в гарвардской архитектуре:**

* **Отдельные пространства**: Имеются две независимые системы адресации — одна для памяти программ (инструкций) и другая для памяти данных.
* **Разные шины**: Каждая память (программная и данных) имеет свою адресную шину, что позволяет одновременно обращаться к разным адресам.
* **Фиксация размеров**: Размер адресного пространства для инструкций и данных может различаться (например, 16 бит для данных и 32 бита для программ).
* **Пример**: В микроконтроллере PIC адрес программной памяти (ROM) отделён от адресной памяти данных (RAM).

**Адресное пространство в принстонской архитектуре:**

* **Единое пространство**: Одна общая память для программ и данных с единой системой адресации.
* **Общая шина**: Адресная и управляющая шины используются для доступа к любому адресу в памяти.
* **Линейная организация**: Все адреса (инструкции и данные) размещаются в одном непрерывном диапазоне (например, 32-битное адресное пространство).
* **Пример**: В типичном CPU (x86) данные и инструкции хранятся в одной RAM с общим адресным пространством.

1. Какие существуют варианты обхода ограничений гарвардской архитектуры, включая "Модифицированную гарвардскую архитектуру"?

Варианты обхода ограничений гарвардской архитектуры включают:

1. Архитектура "Память инструкций как данные": использование памяти инструкций как данных для программирования и выполнения кода.
2. Архитектура "Память данных как инструкции": использование памяти данных как инструкций для выполнения кода.
3. Модифицированная гарвардская архитектура (основной поток): позволяет прямой доступ к памяти данных и программ, обеспечивая более эффективное выполнение кода.
4. Что такое "аккумуляторная система команд"? Каковы основные элементы? Приведите примеры. Каковы достоинства и недостатки?

Аккумуляторная система команд — это архитектура процессора, где операции выполняются с использованием единственного регистра (аккумулятора), который служит для временного хранения данных и результатов вычислений.

**Основные элементы:**

* **Аккумулятор (ACC):** Регистр для хранения данных и результатов.
* **Арифметико-логическое устройство (ALU):** Выполняет арифметические и логические операции.
* **Память (Memory):** Хранит данные и инструкции.
* **Программный счетчик (PC):** Указывает на текущую инструкцию в памяти.

**Примеры:**

* Ранние машины: IBM 7090, DEC PDP-8.
* Современные: Архитектуры DSP (цифровые сигнальные процессоры).

**Достоинства:**

* Простота конструкции и минимальное внутреннее состояние.
* Эффективность для задач с большим количеством загрузок и сохранений.

**Недостатки:**

* Ограниченная производительность из-за зависимости от одного аккумулятора.
* Сложность при выполнении сложных операций, требующих нескольких регистров.

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

Register-to-Memory системы команд — это архитектура процессора, где одна операция ALU включает один адрес памяти и один регистр, а операции обычно используют два операнда.

**Основные элементы:**

* **Регистры (R₀, R₁, ...):** Хранят данные для операций.
* **Арифметико-логическое устройство (ALU):** Выполняет вычисления.
* **Память (Memory):** Хранит данные и инструкции.
* **Программный счетчик (PC):** Указывает на текущую инструкцию.

**Примеры:**

* IBM 360/370, VAX.

**Достоинства:**

* Малое количество инструкций.
* Плотное кодирование.

**Недостатки:**

* Результат уничтожает один из операндов.
* Длина инструкции варьируется.
* Количество тактов на инструкцию варьируется.
* Сложнее организовать конвейеризацию.

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

Register-to-Register системы команд (Load-Store) — это архитектура процессора, где операции ALU выполняются только между регистрами, без прямого доступа к памяти, а данные загружаются и сохраняются отдельно.

**Основные элементы:**

* **Регистры (R₀, R₁, ...):** Хранят данные для операций.
* **Арифметико-логическое устройство (ALU):** Выполняет вычисления между регистрами.
* **Память (Memory):** Хранит данные и инструкции.
* **Программный счетчик (PC):** Указывает на текущую инструкцию.

**Примеры:**

* CDC 6600, CRAY-1, большинство RISC.

**Достоинства:**

* Простые инструкции фиксированной длины.
* Легко реализуемая конвейеризация.
* Высокий счет инструкций.

**Недостатки:**

* Большее кодирование, но упрощает распределение регистров.
* Высокое количество инструкций.

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

Memory-to-Memory системы команд — это архитектура процессора, где все операнды ALU берутся напрямую из памяти, а операции выполняются без использования регистров.

**Основные элементы:**

* **Память (Memory):** Хранит данные и инструкции, служит источником и приемником операндов.
* **Арифметико-логическое устройство (ALU):** Выполняет вычисления между данными из памяти.
* **Программный счетчик (PC):** Указывает на текущую инструкцию.

**Примеры:**

* VAX.

**Достоинства:**

* Отсутствие необходимости в регистрах.
* Наименьшее количество инструкций.

**Недостатки:**

* Большая вариация длины инструкций.
* Большая вариация количества тактов на инструкцию.
* Высокий трафик памяти.

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

Стековые системы команд — это архитектура процессора, где данные обрабатываются через стек (First-In Last-Out, FILO), а операции ALU выполняются с использованием вершины стека (TOS).

**Основные элементы:**

* **Стек (Stack):** Структура данных FILO с вершиной стека (TOS).
* **Арифметико-логическое устройство (ALU):** Выполняет операции над данными на вершине стека.
* **Память (Memory):** Хранит данные и инструкции.
* **Программный счетчик (PC):** Указывает на текущую инструкцию.

**Что такое неявная адресация аргументов?** Неявная адресация аргументов означает, что операнды для ALU-операций не указываются явно, а берутся автоматически с вершины стека (TOS), что упрощает кодирование инструкций.

**Достоинства:**

* Короткие инструкции.
* Простота написания компилятора.

**Недостатки:**

* Низкая эффективность кода из-за работы со стеком.
* Фиксированный доступ к вершине стека.
* Ограниченная скорость и кэш для верхних элементов стека.

1. Какие виды стеков есть в стековых процессорах? Их количества?

В стековых процессорах используются следующие виды стеков:

1. **Операционный стек (Execution Stack)** — основной стек для выполнения инструкций, содержит операнды и промежуточные результаты.
2. **Стек возвратов (Return Stack)** — используется для хранения адресов возврата при вызове подпрограмм.
3. **Стек данных (Data Stack)** — отдельный стек для работы с данными, если архитектура поддерживает разделение стеков.

Количество стеков зависит от архитектуры процессора:

* Минимально — 1 стек (обычно операционный).
* В более сложных системах (например, Forth-процессорах) — 2 стека (операционный и стек возвратов).
* В редких случаях — 3 стека, если поддерживается отдельный стек данных.

Точное количество определяется конкретной реализацией процессора.

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

**Основные принципы описания алгоритмов для стековых процессоров:**

* Использование стека как основного хранилища операндов и промежуточных результатов.
* Операции выполняются с вершины стека (TOS), где аргументы извлекаются неявно.
* Алгоритмы строятся на последовательности команд push (загрузка данных), pop (извлечение данных) и операций ALU (например, add, mul).
* Минимизация количества регистров, акцент на стековую адресацию.
* Разделение логики на подпрограммы с использованием стека возвратов.

**Что такое Forth?** Forth — это стековый язык программирования, разработанный для эффективной работы на стековых процессорах. Он использует обратную польскую нотацию (Reverse Polish Notation, RPN), где операции выполняются сразу после помещения операндов на стек. Forth прост, модулен и оптимизирован для встраиваемых систем.

**Что такое High-level language computer?** High-level language computer (HLMC) — это концепция архитектуры процессора, оптимизированной для прямого выполнения кода на высокоуровневых языках (например, Forth или Pascal) без значительных компиляций. Такие системы, как Burrough B5000, используют стековые механизмы для упрощения интерпретации и повышения производительности при работе с высокоуровневыми конструкциями.

1. Что такое CISC процессоры? Каковы их ключевые особенности, достоинства и недостатки?

CISC (Complex Instruction Set Computer) — это архитектура процессора, где отдельные инструкции могут выполнять несколько низкоуровневых операций (загрузка из памяти, арифметические действия, сохранение в память) или поддерживать многоступенчатые операции и различные режимы адресации в одной инструкции.

**Ключевые особенности:**

* Низкоуровневые языки.
* Разнообразие архитектур.
* Неразвитость компиляторов.
* Удобство программирования.
* Высокая производительность.
* Минимум обмена программами.
* Минимум накладных расходов.

**Достоинства:**

* Сложная система команд (использование анализа).
* Сложное устройство процессора и Control Unit.
* Сложно генерировать эффективный машинный код.

**Недостатки:**

* Высокая сложность реализации.
* Длительные циклы выполнения инструкций.
* Трудности с оптимизацией и конвейеризацией.

1. Какие виды адресации аргументов инструкции в CISC процессорах?

В CISC процессорах используются следующие виды адресации аргументов инструкции:

* **Прямая адресация (Direct Addressing):** Адрес операнда указан явно в инструкции.
* **Непрямая адресация (Indirect Addressing):** Адрес операнда находится по адресу, указанному в инструкции.
* **Регистровая адресация (Register Addressing):** Операнд находится в регистре.
* **Адресация со смещением (Indexed Addressing):** Адрес операнда вычисляется как сумма базового адреса и смещения.
* **Базовая адресация (Base Addressing):** Адрес операнда вычисляется относительно базового регистра.
* **Относительная адресация (Relative Addressing):** Адрес операнда вычисляется относительно текущего значения счетчика команд (PC).
* **Смешанная адресация (Mixed Addressing):** Комбинация нескольких методов, например, регистровая с индексом.

1. В каких случаях CISC инструкции эффективны? Приведите примеры.

CISC инструкции эффективны в следующих случаях:

* При выполнении сложных операций, требующих нескольких шагов (например, умножение с накоплением или обработка строк), где одна инструкция может заменить серию простых команд.
* В задачах, где важна плотность кода, например, в системах с ограниченной памятью (встроенные системы, старые компьютеры).
* При программировании на низкоуровневых языках, где разработчик может напрямую использовать богатый набор инструкций.
* В приложениях, где минимизация обмена данными между памятью и процессором критична.

**Примеры:**

* **IBM System/360 и 370:** Эффективны для коммерческих приложений, таких как обработка данных и транзакций.
* **VAX:** Хорошо справляется с научными вычислениями и обработкой больших объемов данных.
* **Intel x86 (ранние модели):** Использовались в ПК для задач общего назначения, где важна совместимость и компактность кода.

1. В чём выражается комплексность систем команд CISC?

Комплексность систем команд CISC выражается в следующем:

* **Большое количество инструкций:** Сотни различных команд, включая сложные многооперационные инструкции.
* **Разнообразие режимов адресации:** Поддержка множества способов доступа к операндам (прямая, непрямая, регистровая, индексная и др.).
* **Многоступенчатые операции:** Одна инструкция может выполнять несколько низкоуровневых действий (загрузка, вычисление, сохранение).
* **Переменная длина инструкций:** Разная длина команд, что усложняет декодирование и выполнение.
* **Сложная микроархитектура:** Необходимость сложного блока управления (Control Unit) для интерпретации и выполнения инструкций.
* **Высокая плотность кода:** Компактность программ за счет сложных инструкций, но с увеличением сложности процессора.

1. Какие существуют подходы к реализации Control Unit? Достоинства и недостатки Hardwired и микропрограммного управления?

* **Hardwired (Жестко запрограммированное управление):** Управление реализовано через фиксированную комбинационную логику и регистры состояния.
* **Microprogrammed (Микропрограммное управление):** Управление осуществляется с помощью микропрограмм, хранящихся в памяти управления (Control Storage).

**Достоинства и недостатки Hardwired управления:**

* **Достоинства:**
  + Высокая скорость выполнения благодаря прямой логике.
  + Простота для фиксированных инструкций.
* **Недостатки:**
  + Сложность изменения или добавления новых инструкций.
  + Высокая стоимость разработки и модификации.

**Достоинства и недостатки микропрограммного управления:**

* **Достоинства:**
  + Гибкость: легко добавлять или изменять инструкции через обновление микропрограмм.
  + Упрощает разработку сложных систем команд.
* **Недостатки:**
  + Более низкая скорость из-за необходимости извлечения микрокода.
  + Увеличение объема памяти для хранения микропрограмм.

1. Что такое микропрограммное управление? Что такое микроинструкция? В чём отличия инструкций и микроинструкций?

Микропрограммное управление — это подход к реализации блока управления (Control Unit), где последовательность микрокоманд (микропрограмм) хранится в специальной памяти (Control Storage) и используется для управления выполнением инструкций процессора. Каждая микропрограмма разбивает сложные инструкции на элементарные шаги.

**Что такое микроинструкция?** Микроинструкция — это базовая единица микропрограммы, содержащая набор сигналов управления, которые определяют действия процессора на одном такте. Она включает поля для управления регистрами, ALU, мультиплексорами и другими компонентами.

**В чём отличия инструкций и микроинструкций?**

* **Уровень абстракции:** Инструкции — это команды высокого уровня, видимые программисту или компилятору, тогда как микроинструкции — низкоуровневые команды, управляющие оборудованием.
* **Состав:** Инструкция содержит опкод и операнды, а микроинструкция — детали управления (сигналы для регистров, ALU и т.д.).
* **Исполнение:** Одна инструкция может быть разбита на несколько микроинструкций, выполняемых последовательно.
* **Хранение:** Инструкции хранятся в основной памяти, микроинструкции — в памяти управления.

1. Как можно оптимизировать инструкции при помощи микропрограммирования? Приведите пример.

Оптимизация инструкций с использованием микропрограммирования достигается за счет:

* Разбиения сложных инструкций на более простые микрооперации, что позволяет ускорить выполнение.
* Повторного использования общих последовательностей микрокода для различных инструкций.
* Упрощения декодирования за счет создания специализированных микропрограмм.
* Адаптации микрокода под конкретные задачи или архитектуру процессора.
* Снижения количества тактов за счет параллельного выполнения микроопераций.

**Пример:** Рассмотрим инструкцию умножения с накоплением (например, MULACC R1, R2), которая умножает содержимое регистра R1 на R2 и добавляет результат к аккумулятору.

* Без микропрограммирования: Инструкция может требовать нескольких отдельных шагов (загрузка R1, загрузка R2, умножение, добавление к ACC), что занимает больше тактов.
* С микропрограммированием: Микропрограмма может быть оптимизирована следующим образом:
  1. Микроинструкция 1: Загрузить R1 и R2 в ALU.
  2. Микроинструкция 2: Выполнить умножение в ALU.
  3. Микроинструкция 3: Добавить результат к ACC и сохранить. Эту последовательность можно закодировать как одну микропрограмму, уменьшив общее время выполнения и улучшив эффективность за счет параллельных операций внутри ALU.

1. Какие существуют подходы к поиску первой микроинструкции для заданной инструкции?

Существуют следующие подходы к поиску первой микроинструкции для заданной инструкции:

1. **Алгоритмический подход (сопоставление инструкции с условием):**
   * Используется сопоставление кода опкода инструкции с условием в таблице переходов.
   * Пример: Опкод 0101 сопоставляется с адресом первой микроинструкции.
2. **Отображение кода операции на адрес микроинструкции:**
   * Пример: Код операции 0101 напрямую отображается на адрес в памяти управления (например, 0x05).
   * Используется для простых систем с фиксированным соответствием.
3. **Непрямое отображение (через таблицу look-up-table):**
   * Требует использования таблицы соответствия (look-up-table), где для каждого опкода хранится адрес первой микроинструкции.
   * Пример: Для опкода 0101 таблица указывает адрес 4 в памяти управления.

Эти подходы выбираются в зависимости от сложности архитектуры и требований к скорости выполнения.

1. Что такое архитектура NISC? Каковы область её применения, достоинства и недостатки?

NISC (No-Instruction-Set-Computer) — это архитектура процессора, не использующая фиксированный набор инструкций (ISA). Вместо этого она основана на прямом отображении алгоритмов на аппаратные микрооперации, управляемые высокоуровневым синтезом (HLS), где программа компилируется в специализированный поток микроопераций.

**Область применения:**

* Применяется в ускорителях вычислений, в высокоуровневом синтезе (HLS).
* Проект NITTA — CGRA процессор, где вычислительные блоки управляются в стиле NISC.

**Достоинства:**

* Упрощение аппаратуры за счет отсутствия фиксированной ISA.
* Высокая производительность для специфических задач благодаря прямому синтезу.
* Гибкость адаптации под конкретные алгоритмы.

**Недостатки:**

* Необходимость сложного высокоуровневого синтеза (HLS).
* Отсутствие стандартной ISA затрудняет совместимость и переносимость кода.
* Высокая сложность разработки и оптимизации.

1. Каковы источники роста производительности процессоров?
2. Частота.
3. Специализация системы команд, аппаратуры.
4. Параллелизм уровня бит, инструкций, задач.
5. Адаптация структуры вычислителя под задачу и параллелизм.
6. Динамическая адаптация.
7. Какие законы и ограничения влияют на производительность современных процессоров?

На производительность современных процессоров влияют следующие законы и ограничения:

1. **Закон Мура:** Увеличение числа транзисторов на чипе удваивается примерно каждые 2 года, что исторически обеспечивало рост производительности.
2. **Закон Амдала:** Ограничение ускорения параллельных вычислений из-за последовательной части программы.
3. **Закон Деннарда:** Эффективность энергопотребления растет с уменьшением размера транзисторов, но этот закон стал менее применим с ростом тепловыделения.
4. **Power Wall:** Ограничение роста производительности из-за ограничений энергопотребления и тепловыделения.
5. **Memory Wall:** Разрыв между скоростью процессора и памятью, что ограничивает доступ к данным.
6. Что представляют собой закон Мура и закон Деннарда?

**Закон Мура**: Количество транзисторов в микропроцессорах удваивается примерно каждые 2 года, что приводит к экспоненциальному росту вычислительной мощности.

**Закон Деннарда**: При уменьшении размеров транзисторов (масштабировании) их энергопотребление снижается пропорционально квадрату уменьшения размеров, позволяя увеличивать плотность и производительность без роста энергозатрат.

1. Что представляет собой закон Амдала? На каком уровне параллелизма он работает?

**Закон Амдала**: Описывает ускорение, которое можно получить при параллельной обработке задачи. Ускорение ограничено долей последовательной части программы: S=1(1−P)+PN S = \frac{1}{(1-P) + \frac{P}{N}} S=(1−P)+NP​1​, где P P P — доля параллельной части, N N N — число процессоров.

**Уровень параллелизма**: Работает на уровне **задач** или **программ**, оценивая, как распараллеливание задач влияет на общее время выполнения.

1. Что такое Power Wall и тёмный кремний?

**Power Wall**: Ограничение роста производительности процессоров из-за высокой плотности энергопотребления, когда дальнейшее увеличение частоты или числа транзисторов приводит к чрезмерному тепловыделению, которое сложно рассеять.

**Тёмный кремний (Dark Silicon)**: Часть чипа, которая остаётся неактивной (выключенной) в определённый момент времени из-за ограничений по энергопотреблению и тепловыделению, чтобы предотвратить перегрев и сохранить энергоэффективность.

1. Какие подходы существуют к решению проблемы Memory Wall в рамках процессоров семейства фон Неймана?

**Memory Wall** — проблема, связанная с разрывом в скорости между процессором и памятью. Для её решения в процессорах архитектуры фон Неймана применяются следующие подходы:

1. **Кэш-память**: Использование многоуровневой иерархии кэш-памяти (L1, L2, L3) для хранения часто используемых данных ближе к процессору.
2. **Предвыборка данных**: Алгоритмы предвыборки (prefetching), предугадывающие, какие данные понадобятся, и заранее загружающие их в кэш.
3. **Увеличение пропускной способности памяти**: Использование более быстрых типов памяти (например, DDR5, HBM) и многоканальных контроллеров памяти.
4. **Сжатие данных**: Сжатие данных в памяти для уменьшения объёма передаваемой информации и ускорения доступа.
5. **Оптимизация памяти**: Технологии, такие как NUMA (Non-Uniform Memory Access) и локализация данных для минимизации задержек.
6. **Интеграция памяти на кристалле**: Использование технологий вроде eDRAM или 3D-стекирования памяти (HBM, 3D XPoint) для сокращения задержек.
7. Какие подходы существуют к решению проблемы Memory Wall в рамках ASIC и CGRA?

Для решения проблемы **Memory Wall** (разрыва в скорости между вычислениями и доступом к памяти) в **ASIC** (Application-Specific Integrated Circuits) и **CGRA** (Coarse-Grained Reconfigurable Arrays) применяются следующие подходы:

**ASIC**:

1. **Встроенная память**: Интеграция высокоскоростной памяти (SRAM, eDRAM) непосредственно на кристалле для минимизации задержек.
2. **Специализированные контроллеры памяти**: Оптимизированные под конкретное приложение контроллеры для эффективного управления потоками данных.
3. **Иерархия памяти**: Использование локальных буферов и регистров для хранения промежуточных данных, сокращающих обращения к внешней памяти.
4. **Параллельный доступ**: Широкие шины данных и многопортовые структуры памяти для увеличения пропускной способности.
5. **Оптимизация данных**: Сжатие или реорганизация данных для уменьшения объёма передаваемой информации.

**CGRA**:

1. **Распределённая память**: Размещение небольших локальных банков памяти рядом с вычислительными блоками для быстрого доступа.
2. **Реконфигурируемые маршруты данных**: Динамическая настройка путей передачи данных между вычислительными элементами и памятью для минимизации задержек.
3. **Предвыборка и буферизация**: Локальные буферы и механизмы предвыборки для прогнозирования и кэширования необходимых данных.
4. **Снижение объёма данных**: Использование алгоритмов, оптимизированных для работы с компактными данными, и локальных вычислений.
5. **Параллельная обработка памяти**: Многопотоковый доступ к памяти для одновременной работы с несколькими потоками данных.
6. Что такое RISC? Каковы особенности и предпосылки появления (ПО и аппаратура)?

**RISC (Reduced Instruction Set Computer)** — архитектура компьютеров с сокращённым набором инструкций, ориентированная на простоту и скорость выполнения.

**Особенности**:

* Простые и фиксированной длины инструкции (например, 32 бита).
* Оптимизация для конвейерной обработки (pipelining).
* Высокая скорость выполнения за счёт минимизации сложных операций.
* Меньше зависимостей между инструкциями, что упрощает параллелизм.

**Предпосылки появления**:

* **Программное обеспечение**: Сложные инструкции в CISC-архитектурах (Complex Instruction Set Computer) редко использовались компиляторами, снижая эффективность.
* **Аппаратура**: Рост сложности декодирования и выполнения сложных инструкций в CISC, а также необходимость повышения производительности при ограниченных ресурсах привели к идее упрощения процессоров и акценте на конвейерной обработке.

1. Как сравниваются CISC и RISC с точки зрения архитектуры?

**CISC (Complex Instruction Set Computer)** и **RISC (Reduced Instruction Set Computer)** различаются по архитектуре следующим образом:

* **Набор инструкций**: CISC использует сложный набор инструкций переменной длины, включая многоадресные операции, тогда как RISC применяет простой набор фиксированной длины с минимальными операциями.
* **Декодирование**: CISC требует сложного декодера из-за разнообразия инструкций, в то время как RISC имеет простой декодер, что ускоряет выполнение.
* **Конвейеризация**: RISC оптимизирован для конвейерной обработки (pipelining) благодаря однотипным инструкциям, тогда как CISC сложнее адаптировать к конвейеру.
* **Использование микрокода**: CISC часто использует микрокод для реализации сложных инструкций, в то время как RISC полагается на прямое выполнение аппаратных операций.
* **Производительность**: RISC обеспечивает более высокую скорость за счёт простоты и параллелизма, тогда как CISC может быть эффективнее для специфических задач с минимальным числом инструкций.
* **Память**: CISC требует больше памяти для хранения сложных инструкций, в то время как RISC использует компактные и однотипные команды.

RISC фокусируется на скорости и энергоэффективности, а CISC — на универсальности и плотности кода.

1. Как сравниваются CISC и RISC с точки зрения микро-архитектуры?

**CISC (Complex Instruction Set Computer)** и **RISC (Reduced Instruction Set Computer)** различаются по микроархитектуре следующим образом:

* **Декодирование инструкций**: CISC использует сложный декодер и микрокод для обработки переменной длины инструкций, тогда как RISC применяет простой декодер для фиксированных инструкций, упрощая конвейер.
* **Конвейеризация**: RISC оптимизирован для эффективного конвейерного выполнения благодаря однородным стадиям (fetch, decode, execute), в то время как CISC требует дополнительных стадий из-за сложных инструкций, что затрудняет pipelining.
* **Использование регистров**: RISC использует больше регистров и меньшую зависимость от памяти, ускоряя доступ к данным, тогда как CISC полагается на меньшее число регистров и более частые обращения к памяти.
* **АЛУ (арифметико-логическое устройство)**: В RISC АЛУ выполняет простые операции, а сложные задачи разбиваются на последовательность инструкций; в CISC АЛУ часто интегрирует сложные многооперационные команды.
* **Контроль и управление**: RISC имеет более простую схему управления сигналами благодаря унифицированным инструкциям, тогда как CISC требует сложной логики управления из-за разнообразия команд.
* **Энергоэффективность**: RISC более энергоэффективен за счёт простоты и параллелизма, тогда как CISC может быть менее эффективным из-за высокой нагрузки на декодирование.

RISC фокусируется на упрощении и скорости на уровне микроархитектуры, в то время как CISC оптимизирован для плотности кода и универсальности.

1. Каковы особенности кодирования инструкций при сравнении CISC и RISC?

**Особенности кодирования инструкций при сравнении CISC и RISC:**

* **Длина инструкций**: CISC использует инструкции переменной длины (обычно 8–120 бит), что позволяет компактно кодировать сложные операции, тогда как RISC применяет фиксированную длину (обычно 32 бита), упрощая декодирование и конвейеризацию.
* **Формат**: CISC включает сложные многоадресные форматы с переменным количеством операндов и режимов адресации, в то время как RISC использует простые форматы с фиксированным числом операндов (обычно 2–3), упрощая обработку.
* **Микрокод**: CISC часто кодирует инструкции с использованием микрокода для реализации сложных операций, тогда как RISC избегает микрокода, полагаясь на простые команды, выполняемые аппаратно.
* **Плотность кода**: CISC обеспечивает высокую плотность кода за счёт компактных сложных инструкций, тогда как RISC требует больше инструкций для той же задачи, увеличивая размер кода.
* **Адресация**: CISC поддерживает множество режимов адресации (например, косвенную, базовую), закодированных в инструкциях, в то время как RISC ограничивается простыми режимами (регистровая, непосредственная), снижая сложность декодера.

1. Каковы принципы конвейерного исполнения инструкций? Как это влияет на загрузку и производительность?

**Принципы конвейерного исполнения инструкций:**

1. **Разделение стадий**: Инструкции разделяются на последовательные этапы (забор, декодирование, выполнение, запись результата), выполняемые параллельно для разных инструкций.
2. **Одновременная обработка**: Несколько инструкций находятся в конвейере одновременно, каждая на своей стадии.
3. **Синхронизация по тактам**: Каждый этап завершается за один цикл такта, что позволяет запускать новую инструкцию на каждом такте.
4. **Оптимизация задержек**: Стадии проектируются для равномерной длительности, минимизируя простои.

**Влияние на загрузку и производительность:**

* **Увеличение производительности**: Теоретически позволяет выполнить одну инструкцию за цикл после заполнения конвейера, многократно ускоряя выполнение программ.
* **Увеличение загрузки**: Конвейер повышает использование процессора, так как все его части (ALU, регистры, память) работают параллельно.
* **Ограничения**: Зависимости между инструкциями (data hazards) или ветвления могут вызвать простои (stalls), снижая эффективность. Требуется оптимизация компилятора и аппаратных решений (например, предсказание ветвлений).

Конвейеризация улучшает производительность при условии минимизации задержек и корректной организации кода.

1. Каковы типовые стадии RISC процессора?

Типовые стадии RISC-процессора:

1. **Fetch (извлечение)**: Извлечение инструкции из памяти в регистр инструкций.
2. **Decode (декодирование)**: Декодирование инструкции и определение операндов.
3. **Execute (выполнение)**: Выполнение операции в арифметико-логическом устройстве (ALU).
4. **Memory Access (доступ к памяти)**: Чтение/запись данных в память, если требуется.
5. **Write Back (запись результата)**: Запись результата в регистр.

Эти стадии оптимизированы для конвейерной обработки.

1. Какие проблемы и архитектурные ограничения связаны с конвейеризацией?

**Проблемы и архитектурные ограничения конвейеризации:**

1. **Зависимости данных (Data Hazards)**: Конфликты, когда одна инструкция зависит от результата предыдущей, не успевшей записаться, вызывая задержки (stalls).
2. **Зависимости управления (Control Hazards)**: Ветвления (например, условные переходы) создают неопределенность следующей инструкции, требуя предсказания или буферизации.
3. **Зависимости структур (Structural Hazards)**: Конкуренция за общие ресурсы (например, память или ALU), если несколько стадий пытаются использовать их одновременно.
4. **Ограничения длины конвейера**: Увеличение числа стадий повышает задержки из-за сложных зависимостей и потребности в сбросе при ошибках.
5. **Сложность реализации**: Требуются дополнительные механизмы (предсказание ветвлений, переупорядочивание инструкций), увеличивающие сложность и энергопотребление.
6. **Неравномерность стадий**: Разные инструкции могут требовать разного времени на стадиях, что снижает эффективность.
7. Как разрешаются структурные конфликты в конвейере?

* Варианты полного решения проблемы:
  + Гарвардская архитектура.
  + Двухпортовая память.

**Разрешение структурных конфликтов в конвейере:**

1. **Дублирование ресурсов**: Использование нескольких экземпляров критических блоков (например, ALU или памяти), чтобы избежать конкуренции.
2. **Задержки (Stalls)**: Временная приостановка конвейера до освобождения ресурса, хотя это снижает производительность.
3. **Планирование инструкций**: Переупорядочивание инструкций компилятором или процессором для минимизации одновременного доступа к одному ресурсу.
4. **Буферизация**: Использование буферов (например, очереди записи) для временного хранения данных, пока ресурс занят.
5. **Оптимизация архитектуры**: Разделение ресурсов (например, отдельные шины для чтения и записи) для параллельного доступа.

Эти методы минимизируют простои и повышают эффективность конвейера.

1. Как разрешаются конфликты по данным в конвейере?

Исполнения не по порядку (out-of-order). Компилятор/процессор.

Переименования регистров. Если зависимость по данным ложная. Запись может быть переназначена на другой регистр (пример WAW).

Вставка пузырька.

Проброс операндов (bypassing, operand forwarding) между стадиями процессора, минуя регистровый файл.

1. Как происходит разрешение конфликтов при помощи пузырька в конвейере?

**Разрешение конфликтов при помощи пузырька (bubble) в конвейере:**

* **Пузырьк (Bubble)**: Это пустой цикл или "нулевой" этап, вставляемый в конвейер, чтобы создать задержку и предотвратить конфликты (например, data hazards или control hazards).
* **Процесс**: Когда обнаруживается зависимость (например, инструкция зависит от результата предыдущей), конвейер вставляет пузырьк, пропуская выполнение на одной или нескольких стадиях, пока данные не станут доступны.
* **Эффект**: Это позволяет синхронизировать выполнение инструкций, избегая перезаписи или неправильного порядка, но снижает производительность из-за простоев.
* **Пример**: Если результат ALU нужен на следующей инструкции, пузырьк вставляется после стадии Write Back, чтобы дать время на запись.

1. Как происходит разрешение конфликтов при помощи проброса операндов в конвейере?

**Разрешение конфликтов при помощи проброса операндов (Data Forwarding) в конвейере:**

* **Процесс**: Результат инструкции, ещё не записанный в регистр, передаётся напрямую из стадии выполнения (Execute) или памяти (Memory Access) в стадию декодирования (Decode) или выполнения следующей инструкции, где он нужен.
* **Механизм**: Специальные пути (forwarding paths) в микроархитектуре соединяют выходы стадий с входами, позволяя обойти задержку записи/чтения из регистра.
* **Эффект**: Устраняет необходимость вставки пузырьков (stalls), сохраняя непрерывность конвейера и повышая производительность.
* **Пример**: Если инструкция 2 зависит от результата инструкции 1, результат ALU передаётся в декодер инструкции 2 вместо ожидания записи в регистр.

1. Какие существуют способы разрешения конфликтов по управлению во время компиляции?

**Способы разрешения конфликтов по управлению (Control Hazards) во время компиляции:**

1. **Предсказание ветвлений (Branch Prediction)**: Компилятор анализирует код и добавляет подсказки о вероятном направлении ветвления (например, "вероятно, переход"), что помогает оптимизировать порядок инструкций.
2. **Переупорядочивание инструкций (Instruction Scheduling)**: Расположение инструкций так, чтобы минимизировать влияние ветвлений, перемещая независимые инструкции перед точками ветвления.
3. **Слияние ветвей (Branch Folding)**: Преобразование условных переходов в последовательный код, если условия могут быть вычислены заранее.
4. **Вставка предвыборки (Prefetching)**: Загрузка инструкций для обоих путей ветвления, позволяя конвейеру продолжать работу до разрешения условия.
5. **Оптимизация циклов**: Разворачивание циклов (loop unrolling) для уменьшения числа ветвлений за счёт повторения кода.
6. Какие существуют способы разрешения конфликтов по управлению во время исполнения?

**Способы разрешения конфликтов по управлению (Control Hazards) во время исполнения:**

1. **Предсказание ветвлений (Branch Prediction)**: Аппаратные механизмы (например, ветвевые предсказатели) угадывают направление перехода на основе истории, позволяя конвейеру продолжать выполнение.
2. **Задержка перехода (Branch Delay Slot)**: Исполнение следующей инструкции после ветвления независимо от исхода перехода, если она безопасна.
3. **Пузырьки (Stalls)**: Вставка пустых циклов (bubbles) в конвейер до разрешения ветвления, чтобы избежать неправильного выполнения.
4. **Динамическое переупорядочивание (Out-of-Order Execution)**: Процессор выполняет инструкции в оптимальном порядке, пропуская зависимые от ветвления, пока условие не разрешено.
5. **Сброс конвейера (Pipeline Flush)**: Отмена инструкций после неправильного предсказания ветвления и возврат к правильному пути.
6. Какие существуют способы предотвращения конфликтов по управлению через branch prediction? Что такое статические предсказатели переходов?

**Способы предотвращения конфликтов по управлению через branch prediction:**

1. **Статическое предсказание**: Использование фиксированных правил (например, всегда предсказывать переход или не переходить) на основе структуры кода.
2. **Динамическое предсказание**: Адаптивное предсказание на основе истории ветвлений с помощью ветвевых предсказателей (например, таблиц ветвлений).
3. **Профилирование кода**: Анализ поведения программы для выбора наиболее вероятного пути ветвления.
4. **Предсказание на основе направления**: Учет, что обратные переходы (например, в циклах) чаще выполняются (90%), а прямые — реже (50%).
5. **Комбинированные подходы**: Использование статических и динамических методов для повышения точности.

**Статические предсказатели переходов:**

* Это простые алгоритмы, встроенные в компилятор или процессор, которые используют фиксированные предположения о поведении ветвлений без учёта их истории.
* Примеры:
  + **Всегда переходить**: Предсказывает выполнение ветвления (например, для циклов).
  + **Никогда не переходить**: Предсказывает пропуск ветвления (для прямых переходов).
  + **На основе направления**: Предсказывает переход для обратных ветвей и не переход для прямых (как показано на слайде: 90% для backward, 50% для forward).

1. Что представляют собой динамические предсказатели переходов в конвейере?

**Динамические предсказатели переходов в конвейере** — это аппаратные механизмы, адаптирующиеся к поведению программы на основе истории выполнения ветвлений. Они улучшают производительность конвейера, минимизируя конфликты управления.

* **Принцип работы**: Предсказывают, будет ли ветвление выполнено (taken) или пропущено (not taken), используя данные о прошлых исходах. Например, таблица ветвлений (Branch History Table) хранит состояние для каждого адреса ветвления.
* **Схема с накоплением (2-bit predictor)**: Использует 2-битное состояние (например, "strongly not taken", "weakly not taken", "weakly taken", "strongly taken"), как показано на слайде. Состояние обновляется:
  + Переход выполнен → смещение к "taken" (например, от "weakly not taken" к "weakly taken").
  + Переход не выполнен → смещение к "not taken".
  + Требуется два противоположных исхода для смены "strongly" состояния.
* **Преимущества**: Высокая точность за счёт адаптации к реальному поведению кода (например, циклы или условия).
* **Пример**: Если ветвление часто выполняется, предсказатель переходит в "strongly taken", и конвейер заранее загружает следующий PC.

1. Как влияют различные виды конфликтов на производительность конвейера?
2. **Конфликт данных (Data Hazard)**
   * **Влияние**: Вызывает **приостановку конвейера (pipeline stall)**, пока данные не станут доступны. Это увеличивает время выполнения и снижает пропускную способность.
3. **Конфликт управления (Control Hazard)**
   * **Влияние**: Приводит к приостановке конвейера, так как процессор должен ждать завершения ветвления, что снижает эффективность (потеря циклов).
4. **Структурный конфликт (Structural Hazard)**
   * **Влияние**: Вызывает задержки, так как инструкции вынуждены ждать освобождения ресурса, что снижает производительность.
5. **Конфликт памяти (Memory Hazard)**
   * **Влияние**: Замедляет выполнение, особенно если доступ к памяти медленный (например, при промахе кэша).

**Общее влияние на производительность**:

* Конфликты увеличивают **CPI (Cycles Per Instruction)**, снижая эффективность конвейера.
* В худшем случае производительность может упасть до уровня неконвейерной архитектуры из-за частых приостановок.
* Современные процессоры используют комбинацию методов (предсказание ветвлений, пересылка данных, суперскалярные архитектуры) для минимизации влияния конфликтов.

1. Как связаны конвейерное исполнение и спекулятивное исполнение инструкций?

Конвейерное исполнение и спекулятивное исполнение связаны тем, что оба метода используются для повышения производительности процессора. Конвейерное исполнение разделяет выполнение инструкций на этапы, позволяя обрабатывать несколько инструкций одновременно. Спекулятивное исполнение дополняет это, предсказывая и выполняя инструкции заранее (например, при условных переходах), чтобы минимизировать простои конвейера, хотя неверные предсказания могут требовать отката.

1. Каковы шаги типового взаимодействия с устройством ввода-вывода? Каков интерфейс устройства для процессора?

**Шаги типового взаимодействия с устройством ввода-вывода:**

1. Конфигурация устройства;
2. Проверка состояния;
3. Отправка данных;
4. Проверка статуса ответа;
5. Чтение ответа.

**Интерфейс устройства для процессора:**  
Интерфейс включает регистры данных и статус, через которые процессор обменивается данными и контролирует состояние устройства.

1. Как реализуется ввод-вывод с точки зрения системы команд в Port Mapped IO?

В Port-Mapped I/O (PMIO) ввод-вывод реализуется с точки зрения системы команд через специальные инструкции, выделенные для работы с портами ввода-вывода. Процессор использует отдельные команды (например, IN и OUT для ISA) для чтения данных из портов или записи в них, взаимодействуя с устройствами через выделенные портовые адреса. Это изолировано от адресации памяти, обеспечивая прямой доступ к устройствам через их собственные регистры

1. Как реализуется ввод-вывод с точки зрения системы команд в Memory Mapped IO?

В Memory-Mapped I/O (MMIO) ввод-вывод реализуется через систему команд, где устройства ввода-вывода адресуются как часть памяти. Процессор использует стандартные инструкции чтения/записи памяти (например, LOAD и STORE) для доступа к адресам, отведенным устройствам, что позволяет интегрировать их в общую адресную область памяти системы.

1. Как устроен ввод-вывод в Memory Mapped IO с точки зрения устройства процессора?

В Memory-Mapped I/O с точки зрения устройства процессора ввод-вывод организован через интеграцию устройств в адресное пространство памяти. Процессор использует регистры данных и статус, доступные по определенным адресам памяти, для чтения или записи данных. Аппаратное обеспечение (например, шина данных и адресов) обеспечивает прямое взаимодействие процессора с устройствами, а управляющие сигналы определяют операции ввода-вывода, как если бы это была обычная память.

1. Как работает программно-управляемый ввод-вывод без прерываний? Каковы его ограничения и типовой сценарий? Как связан с теоремой Котельникова?

**Как работает:** Программно-управляемый ввод-вывод без прерываний (polling) работает через постоянное опрос процессором состояния устройства ввода-вывода. Процессор периодически проверяет статус (например, готовность данных) и выполняет операции чтения или записи.

**Ограничения:**

1. Высокая нагрузка на процессор из-за постоянного опроса.
2. Низкая эффективность при медленных устройствах, так как процессор ожидает ответа.
3. Ограниченная масштабируемость для множества устройств.

**Типовой сценарий:**

1. Работа с ключом.
2. Использование SPI.
3. Имитация параллелизма.

**Связь с теоремой Котельникова:** Теорема Котельникова определяет минимальную частоту дискретизации сигнала. В контексте polling, частота опроса должна быть как минимум вдвое выше частоты изменения данных устройства, чтобы избежать потерь информации, что напрямую связано с этой теоремой.

1. Как можно имитировать параллелизм уровня задач для параллельного программно-управляемого ввода-вывода?

Имитация параллелизма уровня задач для параллельного программно-управляемого ввода-вывода может быть реализована следующим образом:

1. **Использование цикла с ключами:** Создается цикл, который опрашивает состояние нескольких устройств (например, с помощью switch для каждого устройства).
2. **Разделение потоков управления:** Каждый поток управляет отдельным устройством, имитируя параллельную обработку (например, с использованием process для каждого устройства).
3. **Использование прерываний (к ним мы вернёмся позже):** Имитация через многократное переключение между задачами в цикле, чтобы создать иллюзию одновременной работы.
4. **Многопоточность:** Использование многопоточности (например, в Python с threading или в C с pthread) для одновременного опроса устройств.
5. **Кооперативная многозадачность:** Реализация через планировщик задач, где каждая задача уступает управление после завершения своей операции (например, с помощью yield).

Это позволяет эффективно управлять несколькими устройствами, хотя истинный параллелизм зависит от оборудования.

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

 **Уровень битов (Bit-level Parallelism):** Параллельная обработка битов внутри одного слова.

* Пример: Использование 32-битных или 64-битных операций в процессорах.

 **Уровень команд (Instruction-level Parallelism):** Параллельное выполнение нескольких инструкций.

* Пример: Пайплайнинг или суперскалярная архитектура в современных процессорах.

 **Уровень задач (Task/Thread-level Parallelism):** Параллельное выполнение нескольких задач или потоков.

* Пример: Многопоточная обработка в программах с использованием библиотек вроде threading в Python.

1. В чём разница между параллелизмом и конкурентностью (concurrency)?

Разница между параллелизмом и конкурентностью:

* **Параллелизм** — одновременное выполнение нескольких задач на разных процессорах или ядрах, что требует аппаратной поддержки (например, многопоточность на многоядерных CPU).
* **Конкурентность** — организация выполнения задач так, чтобы они казались выполняющимися одновременно, даже если используются один процессор (например, через переключение контекста, как в многозадачности).

Параллелизм физически одновременен, а конкурентность — логически организована

1. В чём заключается проблема реализации параллелизма уровня задач в архитектуре фон Неймана?

Проблема реализации параллелизма уровня задач в архитектуре фон Неймана заключается в ее последовательной природе: данные и инструкции хранятся в единой памяти, а процессор обрабатывает их поочередно. Это приводит к следующим ограничениям:

1. **Конфликт доступа к памяти:** Одновременный доступ нескольких задач к общей памяти вызывает конкуренцию и требует синхронизации.
2. **Отсутствие встроенной поддержки:** Архитектура не предусматривает встроенных механизмов для параллельного выполнения задач, что усложняет многозадачность.
3. **Зависимость от прерываний:** Реализация требует прерываний или кооперативной многозадачности, что увеличивает накладные расходы и сложность.

Эти факторы затрудняют эффективное масштабирование параллелизма без дополнительных решений, таких как многопоточность или специализированные архитектуры

1. Каковы достоинства и недостатки кооперативной многозадачности?

**Достоинства кооперативной многозадачности:**

1. Контроль за ресурсами со стороны задач.
2. Известные точки остановки.
3. Легкость и эффективность (при программной реализации).

**Недостатки кооперативной многозадачности:**

1. Контроль за ресурсами осуществляется задачей.
2. Сбой задачи может быть глобальным.
3. Низкая эффективность при трудоемких задачах ввода-вывода.
4. Сложность разработки прерываний при сложных системах
5. Какие механизмы необходимы для реализации кооперативной многозадачности?

Для реализации кооперативной многозадачности необходимы следующие механизмы:

1. **Механизмы остановки выполнения задачи:** Добровольная передача управления ("диспетчеру").
2. **Механизмы сохранения состояния задачи:** Регистры, стек, состояние сопроцессоров, памяти, т.д., для переключения между задачами.
3. **Механизмы планирования:** Какой задаче отдать процессорное время следующей.
4. **Механизмы восстановления состояния:** Восстановление состояния и передача управления.
5. Механизмы **изоляции** задач: независимое выполнение, безопасность.
6. Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.

Эти механизмы обеспечивают координацию задач и передачу управления между ними.

1. Как взаимодействуют кооперативная многозадачность и система прерываний?

Кооперативная многозадачность и система прерываний взаимодействуют ограниченно, так как кооперативная модель полагается на добровольную передачу управления задачами, а прерывания нарушают этот принцип. В кооперативной многозадачности задачи сами уступают управление, тогда как прерывания принудительно прерывают выполнение для обработки событий (например, от устройств ввода-вывода). Это может привести к конфликтам, если прерывание нарушит последовательность передачи управления, что требует дополнительной синхронизации или ограничения использования прерываний в такой системе.

1. Как реализуется кооперативная многозадачность на уровне приложения через Event-loop?

Кооперативная многозадачность на уровне приложения через Event-loop реализуется следующим образом:

1. **Цикл событий (Event-loop):** Основной цикл приложения постоянно проверяет очередь событий (например, входные данные, таймеры) и вызывает соответствующие обработчики.
2. **Добровольная уступка:** Каждая задача (обработчик события) выполняется до завершения или явной точки уступки (yield), передавая управление обратно в цикл.
3. **Очередь задач:** События и задачи добавляются в очередь, где Event-loop выбирает следующую для обработки.
4. **Состояние задач:** Состояние сохраняется между вызовами обработчиков, обеспечивая непрерывность выполнения.

Пример: В JavaScript используется Event-loop в среде Node.js, где асинхронные операции (I/O, таймеры) обрабатываются через коллбеки или промисы, обеспечивая кооперативное переключение

1. Каковы достоинства и недостатки вытесняющей многозадачности?

**Достоинства вытесняющей многозадачности:**

1. Простота разработки ПО с одним процессом.
2. Контроль за ресурсами со стороны ОС.
3. Интерактивность системы (быстрый отклик на события).

**Недостатки вытесняющей многозадачности:**

1. "Лишние" переключения (избыточная нагрузка на процессор).
2. "Тяжесть" процессоров (перерывания, состояния, т.д.).
3. Разделяемые состояния и непредсказуемые переключения.
4. Непредсказуемая длительность работы. Реальное время.
5. Какие механизмы необходимы для реализации вытесняющей многозадачности?
6. Механизм **прерывания** процесса — забрать процессор у задачи независимо от её желания.
7. Механизм **сохранения** состояния задачи: регистры, стек, состояния сопроцессоров, память, ввод-вывод, кеш, состояние предсказателя переходов, и т.п.
8. Механизм **планирования** — какой задаче отдать процессорное время следующей.
9. Механизм **возобновления** остановленного процесса: восстановление состояния и передача управления.
10. Механизмы **изоляции** задач: независимое выполнение, безопасность.
11. Механизмы **взаимодействия** между задачами: передача данных и сигналов, общие ресурсы.
12. В чём преимущество ввода-вывода с использованием системы прерываний?

Преимущество ввода-вывода с использованием системы прерываний заключается в том, что процессор освобождается для выполнения других задач, пока устройство обрабатывает данные. Это повышает эффективность, так как процессор не тратит время на опрос состояния устройства (как в polling), а реагирует только при возникновении события (прерывании), что особенно полезно для медленных операций ввода-вывода.

1. Как по шагам происходит обработка прерывания?

 **Обнаружение прерывания:** Устройство сигнализирует процессору о событии через линию прерывания.

 **Сохранение состояния:** Процессор сохраняет текущий контекст (регистры, флаги, адрес возврата) в стеке.

 **Определение типа прерывания:** Процессор определяет источник прерывания (например, через вектор прерываний).

 **Переход к обработчику:** Управление передается соответствующему обработчику прерываний (ISR).

 **Обработка прерывания:** Обработчик выполняет необходимую логику (например, чтение данных с устройства).

 **Восстановление состояния:** После завершения обработчик восстанавливает сохраненный контекст.

 **Возврат к выполнению:** Процессор возвращается к прерванной программе.

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

Появление системы прерываний приводит к автоматическому появлению параллелизма уровня задач, так как прерывания позволяют процессору переключаться между выполнением основной программы и обработкой событий от устройств или других задач. Это создает иллюзию одновременного выполнения нескольких задач.

**Характеристика:**

* **Параллелизм уровня задач (Task-level Parallelism):** Происходит за счет прерываний, которые принудительно прерывают текущую задачу и передают управление обработчику прерывания (ISR), имитируя параллельное выполнение.
* **Непредсказуемость:** Переключения зависят от внешних событий, что делает их асинхронными.
* **Эффективность:** Освобождает процессор от ожидания, улучшая использование ресурсов.
* **Ограничения:** Требует синхронизации, чтобы избежать конфликтов (например, гонок данных)

1. Как классифицируются прерывания по источнику? Их отличительные особенности.
2. **Аппаратные**.
   1. **Внешние** (асинхронные для внутренних циклов процессора): переполнение таймера, нажатие клавиши, сетевой пакет.
   2. **Внутренние** (синхронные): деление на 0, ошибка доступа к памяти, и т.п.
3. **Программные** (вызывается инструкцией): взаимодействие программы и ОС.

**Отличительные особенности:**

* Внешние прерывания асинхронны и управляются оборудованием, внутренние — синхронны и связаны с программной логикой.
* Внешние требуют проверки состояния устройств, внутренние — анализа ошибок процессора.

1. Каковы задачи контроллера прерывания и какие виды прерываний бывают?

**Задачи контроллера прерывания:** Контроллер прерывания управляет обработкой прерываний в процессоре, выполняя следующие задачи:

* Определяет приоритет и источник прерывания.
* Сообщает процессору о необходимости прерывания работы текущей задачи.
* Сохраняет контекст процессора (регистры) перед переключением.
* Выполняет переход к обработчику прерывания (ISR).
* Восстанавливает контекст после обработки.

**Виды прерываний:**

1. **Аппаратные (Hardware Interrupts)**: Инициируются внешними устройствами (например, кнопка, таймер).
   * Пример: IRQ от клавиатуры.
2. **Программные (Software Interrupts)**: Вызываются инструкциями в коде (например, системные вызовы).
   * Пример: INT 0x80 в x86 для системного вызова.
3. **Исключения (Exceptions)**: Возникают из-за ошибок выполнения (деление на ноль, доступ к недопустимому адресу).
   * Пример: Page Fault.
4. **Сброс (Reset)**: Инициирует перезапуск системы.
   * Пример: Сброс при включении питания.
5. Какие виды событий существуют в системе прерываний и как они обрабатываются?

В системе прерываний выделяются следующие виды событий:

1. **Внешние события (External Events)**: Инициируются устройствами (например, нажатие кнопки, сигнал таймера).
2. **Внутренние события (Internal Events)**: Происходят из-за ошибок процессора (например, деление на ноль, переполнение).
3. **Программные события (Software-Triggered Events)**: Вызываются программно (например, системный вызов).
4. **Сбросовые события (Reset Events)**: Инициируют перезапуск системы (например, при питании).

**Как они обрабатываются?**

* **Обнаружение**: Контроллер прерываний или процессор фиксирует событие и определяет его тип и приоритет.
* **Сохранение контекста**: Сохраняются текущие значения регистров и состояние процессора.
* **Переход к обработчику**: Процессор переходит к соответствующему обработчику прерываний (ISR) по вектору прерывания.
* **Обработка**: ISR выполняет действия (например, чтение данных с устройства или коррекция ошибки).
* **Восстановление**: Контекст восстанавливается, и процессор возвращается к прерванной задаче (с помощью команды RETI).
* **Приоритет**: Выполняется по иерархии (например, сброс выше аппаратных прерываний).

1. Что такое сторожевой таймер и каков принцип его работы?

Сторожевой таймер (Watchdog Timer, WDT) — это аппаратный или программный механизм, предназначенный для автоматического перезапуска системы при её зависании или некорректной работе.

**Принцип работы:**

* Таймер инициализируется с заданным интервалом (например, 1 секунда).
* Программа регулярно (до истечения интервала) сбрасывает таймер (feed the watchdog) через специальную команду.
* Если сбрасывание не происходит (например, из-за зависания), таймер истекает и генерирует сигнал сброса (reset), перезапуская систему.
* Это обеспечивает самодиагностику и восстановление в реальном времени, особенно в встраиваемых системах.

1. Какие подходы существуют к решению проблемы изоляции регистров и инструкций в памяти при многозадачности?
2. Инструкции:
   * **Размещаем** инструкции в разных областях памяти.
   * **Корректируем** все абсолютные адреса переходов.
3. Данные:
   * **Размещаем** данные в разных областях памяти.
   * **Корректируем** все абсолютные адреса.
4. Регистры:
   * **Формируем** соглашение о вызове процедур внутри задач.
   * **Сохраненяем** и **загружаем** регистры между зачами.
5. Автоматическая память:
   * **Оцениваем** потребность в памяти.
   * **Выделяем** подходящие диапазоны (фрагментация).
6. Динамическая память: **выделяем** диапазон.
7. **Верим** в "добросовестное использование".
8. Какие подходы существуют к решению проблемы изоляции данных в памяти (статика, куча, стек) при многозадачности?
9. Банки памяти (Memory Bank), отчасти
10. Сегментация (Segmentation)
11. Виртуальная память (Virtual Memory)
12. Как банки памяти используются для расширения адресного пространства?

Банки памяти расширяют адресное пространство, разделяя физическую память на несколько независимых блоков (банков), каждый из которых имеет свой диапазон адресов. Это достигается следующим образом:

* **Разделение адресного пространства**: Адресная шина делится на части — часть адреса определяет банк (bank select), а остальная часть указывает адрес внутри банка.
* **Многобанковая организация**: Каждый банк подключается к отдельной шине данных или управляющим сигналам, что позволяет одновременно обращаться к разным банкам.
* **Расширение через адресацию**: Использование дополнительных бит (например, через регистр выбора банка) увеличивает общий объем доступной памяти за пределы стандартного адресного пространства процессора.

1. Как банки памяти используются для расширения машинного слова?

Банки памяти расширяют машинное слово, объединяя несколько банков для параллельного доступа к данным, что увеличивает ширину слова процессора. Это происходит следующим образом:

* **Параллельный доступ**: Каждый банк предоставляет часть данных машинного слова (например, 8 бит из 32-битного слова), и все банки читаются одновременно через отдельные шины данных.
* **Синхронизация адресов**: Один адрес применяется ко всем банкам, но каждый банк выдает свою порцию данных, формируя более широкое слово.
* **Конфигурация**: Ширина слова определяется количеством банков (например, 4 банка по 8 бит = 32 бита).
* **Пример**: В процессоре с 16-битным словом добавление двух банков (по 16 бит каждый) позволяет обработать 32-битное слово за один цикл.

1. Что такое сегментная память? Как происходит трансляция адресов? Каковы достоинства и недостатки?

Сегментная память — это метод организации памяти, при котором адресное пространство делится на переменной длины сегменты (например, код, данные, стек), каждый из которых имеет уникальный базовый адрес и длину. Адреса состоят из сегментного регистра (указатель на сегмент) и смещения внутри сегмента.

**Как происходит трансляция адресов?**

* **Формирование адреса**: Логический адрес состоит из номера сегмента (загружается в сегментный регистр, например, CS, DS) и смещения (offset).
* **Преобразование**: Аппаратная часть (сегментный механизм или MMU) добавляет базовый адрес сегмента к смещению, образуя физический адрес.
* **Пример**: Если базовый адрес сегмента = 1000, а смещение = 50, то физический адрес = 1050.

**Достоинства:**

1. Таблицы сегментов относительно малы.
2. Таблицы сегментов просты в обработке и перемещении.
3. Средние размеры сегментов больше, чем размеры большинства страниц, что позволяет хранить в сегментах больше данных процесса.
4. 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 (Linux).

1. Что такое виртуальная память? Как происходит трансляция адресов? Каковы достоинства и недостатки?

Виртуальная память (Virtual Memory) — это механизм операционной системы, который предоставляет каждому процессу иллюзию работы в отдельной адресной пространстве, используя комбинацию физической оперативной памяти и дискового пространства (страничный файл).

Процесс трансляции адресов:

1. **Режим виртуальные пространства на страницы.**
2. **Неопределённое физическое размещение данных.**
3. **Не фиксировать объём памяти, используемый задачей.**
4. **Использование таблицы страниц, чем есть физически. Выгрузка на диск.**

Достоинства:

* **Нагрузка неиспользуемые страницы лобой задачи.**
* **Выгрузка их на диск.**
* **Права доступа. Отображение страниц на разные адресные пространства.**

Недостатки:

* **Больший объём таблицы страниц (вызов-вывод, перенос страниц).**
* **Необходимость длительности доступа к памяти.**
* **Высокая сложность реализации.**

1. Что такое внутренняя и внешняя фрагментация в контексте сегментной и виртуальной памяти?

**Внутренняя фрагментация** — это ситуация, когда выделенный блок памяти больше, чем требуется процессу (например, P1 требует 3 МБ, а выделено 4 МБ). Неиспользованная часть (1 МБ) становится внутренней фрагментацией, что снижает эффективность использования памяти.

**Внешняя фрагментация** — это ситуация, когда свободная память разбита на мелкие несмежные блоки (например, после выделения памяти процессам P1-P4), которые не могут быть использованы для размещения нового процесса из-за отсутствия непрерывного свободного пространства

1. Как появление сегментной памяти улучшило пользовательский опыт?

Сегментная память улучшила пользовательский опыт, позволив более гибкое управление памятью, разделение кода и данных, а также защиту процессов друг от друга, что повысило стабильность и удобство работы.

1. Как появление виртуальной памяти улучшило пользовательский опыт?

Виртуальная память улучшила пользовательский опыт, позволив запускать больше процессов одновременно, изолируя их адресные пространства, и обеспечивая эффективное использование памяти за счет подкачки на диск, что повысило стабильность и многозадачность.

1. Какие уровни (виды) задач существуют: основной поток, прерывание, процессы, потоки, зелёные потоки?
2. **Main**/Kernel/Основной поток — исходный поток инструкций.
3. **Прерывания**. Особый исходный код обработчиков прерываний.
4. **Процессы**. Изолированные адресные пространства. Нет прямого доступа.
5. **Потоки**. Работают в адресном пространстве процесса. Прямой доступ ко всем его данным.
6. **Зелёные потоки**. В рамках Run-Time или виртуальной машины
7. Как могут взаимодействовать процессы через основной поток? Какие бывают разделяемые ресурсы?

Процессы могут взаимодействовать через основной поток с помощью механизмов, таких как:

* **Shared Memory** (разделяемая память),
* **Signals** (сигналы),
* **I/O: Network, Files, Pipes** (ввод-вывод: сеть, файлы, каналы).

Разделяемые ресурсы включают:

* Разделяемую память (Shared Memory),
* Файлы,
* Сетевые соединения,
* Каналы (Pipes).

1. Что такое процессоры ввода-вывода? Каковы их назначение, интерфейс, достоинства и недостатки?

Процессоры ввода-вывода (Channel I/O) — это специализированные устройства или подсистемы, управляющие передачей данных между памятью и внешними устройствами.

Назначение:

- Позволяют программе централизованно управлять взаимодействием с внешними устройствами, снижая нагрузку на основной процессор.

Интерфейс:

- Пример: команды вроде `SEEK`, `SEARCH KEY EQUAL`, `READ DATA` с параметрами (например, номер цилиндра, буфер).

Достоинства:

- Автоматизация конвертации форматов.

- Применение в многопроцессорных системах.

- Использование DMA (прямого доступа к памяти) для ускорения передачи данных.

Недостатки:

- Сложность реализации и настройки.

- Ограниченная гибкость в сравнении с универсальными процессорами.

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

Контроллер прямого доступа к памяти (DMA) — это устройство, которое позволяет передавать данные между памятью и внешними устройствами без участия основного процессора.

Назначение:

- Скорость и эффективность контроллера DSA.

- Интерактивность, как у процессоров разряжен от "рутины".

- Параллелизм: DMA может иметь несколько каналов для параллельной работы.

Интерфейс:

- Команды: `SEEK`, `SEARCH KEY EQUAL`, `READ DATA` с параметрами (например, номер цилиндра, буфер).

Достоинства:

1. Скорость и эффективность контроллера DSA.
2. Интерактивность, так как процессор разгружен от "рутины".
3. Параллелизм: DMA может иметь несколько каналов для параллельной работы.

Недостатки:

1. Проблемы совместимости.
2. Сложность при непоследовательном доступе к памяти.
3. Ограниченный контроль за системной шиной. Синхронизация работы процессора и DMA.
4. Конфликты использования DMA разными устройствами ввода-вывода.
5. Какие существуют способы интеграции и взаимодействия контроллера прямого доступа к памяти (DMA) с процессором?

Существуют следующие способы интеграции и взаимодействия контроллера DMA с процессором:

* **Third-party, Bus mastering**: Управление DMA осуществляется процессором, а DMA может осуществляться и устройствами ввода-вывода.
* **Burst Mode**: Передача DMA обеспечивает единую операцию, которая не может быть прервана процессором.
* **Cycle stealing mode**: Процессору и DMA выделяется фиксированный слот времени в рамках цикла.
* **Transparent Mode**: Процессору передается управление без вмешательства DMA

1. Почему необходима иерархия памяти в современных компьютерных системах?

Иерархия памяти необходима в современных компьютерных системах для баланса между скоростью доступа (регистры, кэш), объемом хранения (RAM, SSD, HDD) и стоимостью. Она позволяет оптимизировать производительность, предоставляя быструю память для частых операций и более медленную, но дешевую и емкую память для долгосрочного хранения данных.

1. Какие основные виды памяти входят в иерархию памяти и каковы их характеристики?

Основные виды памяти в иерархии и их характеристики:

* **Регистры**: Сверхбыстрые, малой емкости, дорогие, встроены в процессор.
* **Кэш (L1, L2, L3)**: Быстрый, небольшой объем, дорогой, уменьшает задержки доступа.
* **RAM (SDRAM, DDR)**: Средняя скорость, средняя емкость, разумная цена, волатильна.
* **SSD**: Высокая скорость, средняя емкость, умеренная цена, неволатильна.
* **HDD**: Медленный, большой объем, дешевый, неволатильный.

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

**Явная иерархия памяти**: Пользователь или программа явно управляет перемещением данных между уровнями (например, с HDD на RAM). Пример: ручное кэширование файлов в оперативную память.

**Скрытая иерархия памяти**: Управление происходит автоматически системой (например, процессором или ОС) без участия пользователя. Пример: аппаратное кэширование данных в L1/L2 кэше процессора.

1. Каково устройство памяти с произвольным доступом? Как устроена ROM ячейка?
2. C **произвольным доступом**, Random-Access Memory (RAM). Задержка доступа не зависит от истории запросов.

ROM (Read-Only Memory) ячейка — это постоянная память, где данные записаны на этапе производства:

* **Транзисторная структура**: Ячейка состоит из транзистора, состояние которого (включен/выключен) определяет бит (0 или 1).
* **Предварительная запись**: Данные "записаны" путем маскирования или программирования (например, в PROM/EPROM) на фабрике или с помощью высокого напряжения.
* **Отсутствие перезаписи**: Ячейка не изменяется во время работы, обеспечивая неизменность данных.
* **Сетка адресов**: Подключена к адресным и выходным линиям для чтения содержимого.

1. Каковы технологии реализации SRAM ячеек? Как происходит чтение и запись?

SRAM (Static Random Access Memory) ячейки обычно реализуются с использованием 6 транзисторов (6T SRAM Cell):

* **Бистабильный элемент**: Два инвертора, соединенные обратной связью, хранят 1 бит (0 или 1) в двух стабильных состояниях.
* **Доступные транзисторы (Access FETs)**: Два дополнительных транзистора управляют доступом к ячейке через wordline и bitlines.

Процесс чтения:

1. **Предзарядка**: Драйверы заряжают все bitlines до Vdd (1) и оставляют их в подвешенном состоянии.
2. **Активация wordline**: Декодер адреса активирует одну wordline.
3. **Опускание bitline**: Активная ячейка медленно опускает один из bitlines до GND (0).
4. **Чувствительность**: Sense amplifiers обнаруживают изменение напряжения и формируют выходные данные.

Процесс записи:

1. **Установка значений**: Драйверы устанавливают bitlines на желаемые значения (Vdd и GND для 1, GND и Vdd для 0).
2. **Активация wordline**: Декодер адреса активирует одну wordline.
3. **Перезапись**: Драйверы переписывают ячейку, заставляя её принять новое значение
4. Каковы технологии реализации DRAM ячеек? Как происходит чтение и запись?

DRAM (Dynamic Random Access Memory) ячейки обычно реализуются с использованием 1 транзистора (1T) и 1 конденсатора (1C):

* **Конденсатор**: Хранит заряд, представляющий бит (заряд — 1, разряд — 0).
* **Транзистор доступа (Access FET)**: Управляет доступом к конденсатору через wordline и bitline.
* **Динамическая природа**: Требует периодической перезарядки (рефреша) из-за утечки заряда.

Процесс чтения:

1. **Предзарядка**: Bitline заряжается до Vdd/2.
2. **Активация wordline**: Включает транзистор доступа.
3. **Обмен зарядом**: Конденсатор и bitline обмениваются зарядом:
   * Если конденсатор разряжен, напряжение bitline слегка падает.
   * Если заряжен, напряжение слегка повышается.
4. **Чувствительность**: Sense amplifier определяет 0 или 1.
   * Проблема: Чтение разрушительное (заряд теряется), поэтому данные переписываются в конец цикла.

Процесс записи:

* **Установка значений**: Bitline заряжается до Vdd (для 1) или GND (для 0).
* **Активация wordline**: Включает транзистор, заряжая или разряжая конденсатор.
* **Фиксация**: Конденсатор принимает новое значение, после чего wordline отключается.

1. Каково устройство многопортовой памяти? Как связана площадь с количеством портов?

Многопортовая SRAM позволяет выполнять несколько операций чтения или записи одновременно. Она реализуется путем добавления:

* **Наборов wordlines и bitlines** для каждого порта.
* **Дополнительных транзисторов доступа (Access FETs)** для управления каждым портом.
* Структуры ячеек, поддерживающей параллельный доступ (например, 6T SRAM с расширением).

Связь площади с количеством портов:

* **Стоимость на бит**: Для N портов:
  + Wordlines: N,
  + Bitlines: 2^N,
  + Access FETs: 2^N.
* **Площадь**: Провода (wires) доминируют, и площадь растет пропорционально O(N^2), так как увеличивается количество линий и транзисторов с ростом числа портов.

1. Что такое кеш? Каково его назначение, место в иерархии памяти и основные метрики?

Кэш — промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью.

**Назначение:**

* Ускорение выполнения программ за счет снижения задержек при доступе к данным.
* Снижение нагрузки на основную память (RAM).

**Место в иерархии памяти:**

* Расположен между регистрами процессора (самый быстрый уровень) и оперативной памятью (RAM), обычно представлен уровнями L1, L2, L3.

**Основные метрики:**

* **Размер**: Обычно от нескольких КБ (L1) до нескольких МБ (L3).
* **Скорость**: Задержка доступа в наносекундах (быстрее, чем у RAM).
* **Кэш-мисс (Cache Miss)**: Процент запросов, не найденных в кэше.
* **Кэш-хит (Cache Hit)**: Процент успешных обращений к кэшу.
* **Пропускная способность**: Объем данных, обрабатываемых за единицу времени.

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

**Принцип локальности** — это предположение, что данные, к которым недавно обращались, или данные, расположенные рядом, будут использоваться в ближайшем будущем.

Виды локальности при кешировании:

* **Временная локальность**: Данные, недавно загруженные в кэш, с высокой вероятностью будут запрошены снова в ближайшее время. Пример: повторное чтение одного и того же участка кода.
* **Пространственная локальность**: Данные, расположенные близко к недавно загруженным, также с высокой вероятностью будут запрошены. Пример: последовательное чтение массива данных.

1. Как пошагово происходит процесс чтения данных через кеш-память процессора?
2. **Тег найден** →→ **кеш-попадание (cache hit)**. Данные читаются в процессор из кеш-линии.
3. **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 получения данных — произвольна, т.к.:
     + возможен многоуровневый кеш;
     + возможна блокировка памяти (восстановление памяти, DMA).
4. Как пошагово происходит процесс записи данных через кеш-память процессора и какие существуют политики записи?
5. **Немедленная запись** (write-through). Изменение вызывает синхронное обновление памяти. Иногда медленнее, чем без кеша.
6. **Отложенная запись** (write-back). Обновление памяти при вытеснении кеш-линии, периодически, или по запросу.
   * Требует хранения признака модификации ("грязный").
   * Группировка изменений, сокрытие промежуточных состояний.
   * Возможно неконсистентное состояние кеша и памяти. Для процессора — невидимое, для других устройств (DMA) — требуется принудительная запись.
7. **Гибридные варианты**. Пример: немедленная запись с буферизацией.
8. Что такое кеш-промах и какие существуют типы кеш-промахов?

**Кеш-промах (Cache Miss)** — это ситуация, когда запрашиваемые данные отсутствуют в кэше, и процессору приходится обращаться к более медленной памяти.

Типы кеш-промахов:

1. **Запись с размещением (Write Allocate, Fetch on Write)**:
   * Данные загружаются в кэш после записи, если их там нет.
   * Примеры: Загрузка блока, обновление блока, повторная загрузка при необходимости.
   * Поведение: Аналогично промаху по чтению.
2. **Запись без размещения (No-Write Allocate, Write Around)**:
   * Данные записываются напрямую в основную память, минуя кэш.
   * Примеры: Пропуск кэша при записи (write-through), турбо-изменения.
   * Поведение: Минимизирует нагрузку на кэш, но может замедлить последующие
3. Что такое ассоциативность кеша? Какова "структура" адреса с точки зрения кеша?

**Ассоциативность кеша** — это способ организации данных в кэше, определяющий, сколько блоков памяти может соответствовать одной строке (или набору) кэша. Основные типы:

1. **Прямой адрес**: Один адрес соответствует одному блоку в кэше, много памяти, мало блоков компараторов.
2. **Множество битов**: Не нужны, если работать блоками, а не байтами.
3. **Средние биты**: Циклически повторяются в адресах памяти.
4. **Старинные биты**: Относительно уникальны (если используются).

**Структура адреса с точки зрения кеша**:

* **Тег (Tag)**: Высокий бит адреса, идентифицирует блок в кэше.
* **Индекс (Index)**: Средние биты, определяют строку в кэше.
* **Смещение (Offset)**: Низкие биты, указывают байт внутри блока.

1. Каковы особенности реализации полностью ассоциативного кеша?

Полностью ассоциативный кеш характеризуется следующими особенностями:

1. Любой блок памяти может быть размещён в любой строке кеша.
2. Используется много логик:
   * Компараторы для сравнения тегов.
   * Вытеснение для управления заменой блоков.
3. Каковы особенности реализации кеша с прямым отображением?

Кеш с прямым отображением характеризуется следующими особенностями:

1. Каждый блок памяти отображается только в одну строку кеша.
2. Один меньший компаратор.
3. Нет вытеснения (замены блоков).
4. Коллизия и техническая выгрузка.
5. Что такое множественно-ассоциативный кеш и что определяет уровень множественности?

М множественно-ассоциативный кеш (N-Way Set Associative Cache) — это кеш, где каждый блок памяти может быть отображен в один из N банков в наборе.

Уровень множественности (N) определяется количеством банков в наборе, что влияет на производительность и сложность вытеснения.

1. Как связана ассоциативность кеша с механизмом вытеснения и замещения?

Ассоциативность кеша определяет, как блоки памяти отображаются в кеш, а механизм вытеснения и замещения (например, LRU или Pseudo-LRU) управляет выбором блока для замены при переполнении. Высокая ассоциативность (например, N-Way) увеличивает сложность и стоимость вытеснения, требуя более сложных алгоритмов, таких как Victim Cache или Zapret кеширования, для оптимизации производительности.

1. Как работает механизм вытеснения и замещения LRU в кеше?

Механизм вытеснения LRU (Least Recently Used) в кеше удаляет блок, который наименее недавно использовался. При каждом доступе к блоку он помечается как "недавно использованный", и при переполнении кеша выбирается для вытеснения блок с наименьшим временем последнего использования.

1. Как работает механизм вытеснения и замещения PLRU в кеше?

Механизм вытеснения PLRU (Pseudo-Least Recently Used) в кеше использует упрощённый алгоритм для приближённого определения наименее недавно использованного блока. Вместо точного отслеживания порядка использования, PLRU использует биты дерева или таблицы для оценки, что снижает сложность и затраты по сравнению с LRU, сохраняя при этом разумную эффективность вытеснения.

1. Почему необходимо использование многоуровневой иерархии кеш-памяти в современных процессорах?

Использование многоуровневой иерархии кеш-памяти необходимо для:

1. Повышения скорости доступа.
2. Синхронизации пропускной способности процессора и памяти.
3. Оптимизации интерфейсов.

Способы организации многоуровневых кешей:

1. Разделённый/унифицированный.
2. Включение/исключение.
3. Частные/общие.
4. В чём разница между разделённым и унифицированным кешем и где они применяются?

Разница между разделённым и унифицированным кешем:

* **Разделённый кеш** (Banked): Память разделена на банки (например, для инструкций и данных), что повышает скорость доступа. Применяется в процессорах с отдельными кешами инструкций (Trace cache) и данных (по процессорному ядру).
* **Унифицированный кеш** (Unified): Один кеш используется для инструкций и данных, что упрощает управление и позволяет адаптировать пропорции. Применяется в процессорах с общей памятью для повышения эффективности (например, в конфигурациях с несколькими банками кеш-памяти).

1. В чём разница между включающим и исключающим кешем и где они применяются?

Разница между включающим (Inclusive) и исключающим (Exclusive) кешем:

* **Включающий кеш**: Все данные в L1 также присутствуют в L2/L3, упрощая согласованность. Применяется в системах с высокой надёжностью кеширования (например, серверы).
* **Исключающий кеш**: Данные в L1 и L2/L3 могут быть разными, что увеличивает общий объём кеша. Применяется в высокопроизводительных процессорах (например, с Non-Inclusive, Non-Exclusive — NINE — вариантами) для оптимизации варьирования.

1. В чём разница между частным и общим кешем и как они применяются в многоядерных процессорах?

Разница между частным и общим кешем:

* **Частный кеш** (Private): Каждый процессорный ядро (например, CPU 0, CPU 1) имеет собственный кеш (L1), обеспечивая быструю скорость доступа (1 цикл). Применяется в многоядерных процессорах для локальной оптимизации производительности.
* **Общий кеш** (Shared): Кеш (L2) разделяется между ядрами, что увеличивает время доступа (4 цикла), но улучшает синхронизацию данных. Применяется для общего доступа к данным и повышения эффективности в многоядерных системах.

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

Типовой многоуровневый кеш (L1, L2, L3, L4) представляет собой иерархию кешей в процессорах:

* **L0 (опционально)**: Специальный кеш для стека, int/float чисел, обычно доступен за 1 такт.
* **L1**: Быстрый, неотъемлемая часть процессора, разделён (Banked), размер 32 Кбайт, используется для инструкций и данных.
* **L2**: Общий доступ, размер 128 Кбайт до 1-12 Мбайт, используется для расширения L1.
* **L3**: До 24 Мбайт и более, синхронизация данных в многоядерных процессорах.
* **L4 (экзотика)**: До серверов и интерфейсов, оптимизация доступа к памяти.

**Назначение**: Ускорение доступа к данным.  
**Тип ячеек**: SRAM.  
**Классификация**: Иерархическая (по уровню доступа и объёму)

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

Когерентность кеш-памяти — это согласованность данных между кешами разных ядер (например, L1, L2, L3) и основной памятью в многоядерных системах, чтобы все процессоры видели одинаковые данные. Она важна для корректной работы многопоточности и DMA.

Возможные состояния кеш-линий (например, в протоколе MESI):

* **Modified (M)**: Данные изменены, отличаются от основной памяти.
* **Exclusive (E)**: Данные совпадают с памятью, доступны только одному ядру.
* **Shared (S)**: Данные совпадают, доступны нескольким ядрам.
* **Invalid (I)**: Данные недействительны, требуют обновления.

1. Как происходит обмен информацией о кеш-линиях через справочник? На сколько актуальна проблема масштабирования?

Обмен информацией о кеш-линиях через справочник (Directory) происходит следующим образом:

* Передача запросов записи осуществляется через справочник.
* При изменении каталога либо обновляется, либо аннулирует другие кеш-линии с этой записью.

Проблема масштабирования актуальна: масштабирование, количество процессоров может быть велико. Недостаток: задержка запроса к справочнику.

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

Обмен информацией о кеш-линиях через отслеживание (Snooping) происходит следующим образом:

* Кеш-память отслеживает транзакции шины, проверяя адреса записей.
* При изменении данных в кеше другого ядра, соответствующие кеш-линии обновляются или помечаются как "неактуальные".

Проблема масштабирования актуальна: при увеличении числа процессоров эффективность снижается из-за перегрузки шины, что ограничивает масштабируемость.

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

Обмен информацией о кеш-линиях через перехват (Snarfing) происходит следующим образом:

* Кеш перехватывает адреса записей и линии данных на предмет обращения к данным своих кеш-линий.
* Если наблюдается запись, кеш обновляет свои данные.

Проблема масштабирования менее актуальна: перехват эффективен в небольших системах, но при увеличении числа процессоров может возникнуть задержка из-за конфликтов доступа.

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

CAP-теорема утверждает, что в распределённых системах нельзя одновременно обеспечить:

1. **Согласованность (Consistency)** — каждое чтение получает последнюю запись или ошибку.
2. **Доступность (Availability)** — каждый запрос получает ответ (без ошибок), но без гарантии.
3. **Толерантность к разделению (Partition Tolerance)** — система продолжает работать при разрыве связи.

**Область применения**: Распределённые системы хранения данных (например, MongoDB, Redis, Cassandra, CouchDB).

**Применение к когерентности кешей**: В многоядерных системах CAP влияет на выбор стратегии когерентности. Например, при разрыве (Partition) можно пожертвовать согласованностью (Consistency) ради доступности (Availability), что может привести к временной неактуальности кеш-линий, требуя механизмов синхронизации (например, Directory или Snooping).

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

В CAP-теореме термины означают:

* **Consistency (Согласованность)**: Все узлы системы видят одинаковые данные одновременно, обеспечивая последнюю версию записи при каждом чтении или ошибку.
* **Availability (Доступность)**: Каждый запрос получает ответ (успешный или ошибочный) от доступного узла, даже если данные могут быть неактуальными.
* **Partition Tolerance (Толерантность к разделению)**: Система продолжает функционировать, несмотря на разрывы связи между узлами, допуская возможные несогласованности.

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

Кеш влияет на производительность памяти, ускоряя доступ к часто используемым данным за счёт их хранения ближе к процессору (L1, L2, L3), уменьшая задержки по сравнению с DRAM.

Способы оптимизации доступа:

* **AoS (Array of Structures)**: Данные хранятся как структуры подряд (например, {x, y, z} для каждой сущности), удобны для последовательного доступа, но могут вызывать кеш-промахи при выборочном чтении.
* **SoA (Structure of Arrays)**: Данные разделены по типам (массивы x[], y[], z[]), оптимизируют кеш при обработке одного поля, снижая промахи и улучшая векторизацию.

Дополнительно: использование prefetching, оптимизация размеров блоков и когерентности (например, MESI) также повышают эффективность.

1. Как кеш работает с виртуальными и физическими адресами?

Кеш работает с виртуальными и физическими адресами следующим образом:

* **Виртуальные адреса**: Используются в кеше L1 для быстрого доступа (TLB ускоряет трансляцию), но требуют согласованности с MMU.
* **Физические адреса**: Применяются в L2 и выше после трансляции через страничную таблицу, обеспечивая точное отображение в физическую память.

Переключение между ними происходит через TLB (Translation Lookaside Buffer), что минимизирует задержки.

1. Что представляет собой концепция уровневой организации компьютерных систем и каковы её преимущества?

Концепция уровневой организации компьютерных систем предполагает разделение работы системы на уровни (например, оборудование, операционная система, приложения), где каждый уровень выполняет специфические функции и взаимодействует с соседними уровнями через интерфейсы.

Преимущества:

* Модульность и независимость уровней упрощают разработку и отладку.
* Повышение переносимости за счёт абстракции.
* Упрощение тестирования и обновления отдельных компонентов.

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

Примеры уровневой организации компьютерных систем включают:

1. **Layered Style (Слоистая архитектура)** — разделение системы на последовательные уровни (например, оборудование, операционная система, приложения), где каждый слой предоставляет услуги верхнему и использует функции нижнего. Пример: архитектура операционных систем (ядро, драйверы, пользовательский интерфейс).
2. **OSI Model (Модель OSI)** — семиярусная модель сетевой архитектуры (физический, канальный, сетевой, транспортный, сеансовый, представительский, прикладной уровни), стандартизирующая взаимодействие сетевых протоколов.
3. **Lava Flow** — неформальная концепция, где уровни формируются эволюционно, часто с накоплением устаревшего кода, что отличается от строгой слоистой структуры. Пример: старые системы с наследием технического долга.

Эти подходы различаются по степени формализации и управляемости.

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

Явление разделения на уровни (disaggregation) — это процесс разделения вертикально интегрированных отраслей на независимые специализированные уровни (например, hardware, OS, ПО). До этого (1980-е) доминировали вертикальные силосы (IBM, DEC), где компании контролировали все уровни.

Современное положение:

С 1995 года преобладают модульные кластеры — уровни разделены, развиваются независимо (например, Intel + Dell + Microsoft). Это ускоряет инновации и снижает затраты благодаря эффекту масштаба и законам Мура.

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

Нарушение уровневой иерархии в современных системах необходимо для повышения производительности, упрощения разработки и адаптации к специфическим требованиям. Причины:

* **Производительность**: Прямой доступ между уровнями (например, драйверов к оборудованию) ускоряет выполнение задач, минуя промежуточные слои.
* **Гибкость**: Ускорение разработки за счёт обхода строгих интерфейсов.
* **Оптимизация**: Адаптация под конкретные задачи, где строгая иерархия может быть неэффективной.

Примеры:

1. **Байпас кэша в CPU**: Прямой доступ к оперативной памяти, минуя кэш-уровни, для ускорения работы в реальном времени.
2. **Direct Memory Access (DMA)**: Устройства обмениваются данными с памятью, обходя процессор и операционную систему.
3. **Виртуализация**: Гипервизоры напрямую управляют оборудованием, нарушая строгую иерархию ОС.

Нарушение требует осторожности, так как может привести к снижению модульности и увеличению сложности отладки.

1. Какие проблемы возникают при обеспечении реального времени в современных процессорах?

Проблемы при обеспечении реального времени в современных процессорах включают:

* **Оптимизация кэшей**: Неопределённость времени доступа из-за кэширования данных.
* **Многозадачность**: Конкуренция потоков и прерывания нарушают предсказуемость.
* **Энергопотребление**: Динамическое управление частотой (DVFS) усложняет тайминги.
* **Виртуализация**: Дополнительные слои увеличивают задержки.
* **Сложность архитектур**: Современные многоядерные системы затрудняют синхронизацию.

1. Что такое низкоуровневый параллелизм? Приведите примеры.

Низкоуровневый параллелизм (Low-Level Parallelism) — это подход к выполнению вычислений, при котором параллельная обработка данных или инструкций осуществляется на уровне аппаратного обеспечения или низкоуровневого программирования, без явного управления со стороны программиста.

Примеры:

* **Суперскалярные процессоры** — выполнение нескольких инструкций одновременно за счёт динамического планирования.
* **Very Long Instruction Word (VLIW)** — процессоры, где одна длинная инструкция содержит несколько операций для параллельного выполнения (например, в некоторых DSP).

1. Что такое суперскалярные процессоры и каковы принципы их работы, достоинства и недостатки?

Суперскалярные процессоры — это архитектуры, которые выполняют несколько инструкций одновременно за счёт параллельного выполнения независимых операций в одной тактовой цикле.

Принципы работы:

* Инструкции читаются в очередь команд по порядку.
* Декодируются и перемещаются на станции резервирования.
* Станции резервирования выполняют переупорядочивание:
  1. по мере доступности данных;
  2. по мере доступности вычислительных ресурсов.

Достоинства:

* Рост производительности. Сглаживание длительности выполнения инструкций.
* Повышение уровня загрузки ресурсов.
* Совместимость с существующим машинным кодом.
* Компилятор может устранить значительное количество конфликтов за счёт сортировки инструкций.

Недостатки:

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

1. Как пошагово выполняются инструкции в суперскалярном процессоре?

Выполнение инструкций в суперскалярном процессоре происходит пошагово в конвейерном режиме с параллельной обработкой. Основные этапы:

1. **Извлечение (Fetch)**: Извлекаются несколько инструкций из памяти в один цикл, используя расширенный декодирующий блок.
2. **Декодирование (Decode)**: Инструкции анализируются на зависимости и распределяются по функциональным блокам (ALU, FPU и т.д.).
3. **Планирование (Dispatch)**: Независимые инструкции отправляются на выполнение в доступные исполнительные модули.
4. **Выполнение (Execute)**: Несколько инструкций выполняются параллельно, если нет конфликтов данных.
5. **Запись результата (Write-back)**: Результаты записываются в регистры, сохраняя порядок при необходимости.

Процесс оптимизируется динамическим планированием и распараллеливанием.

1. Что такое VLIW-процессоры? Какими достоинствами и недостатками они обладают?

VLIW (Very Long Instruction Word) процессоры — это архитектуры, где одна длинная инструкция содержит несколько операций, предназначенных для параллельного выполнения на разных функциональных блоках процессора, с компилятором, управляющим этим параллелизмом.

Достоинства:

1. Упрощение процессора, снижение энергопотребления.
2. Упрощение декодера. Рост частоты.
3. Компилятор имеет больше информации о коде, он лучше знает, что параллельно!

Недостатки:

1. Сложность компилятора.
2. Высокая нагрузка на каналы данных и регистровые файлы.
3. Конфликты конвейера приводят к простою всех узлов.
4. Проблемы условных переходов.
5. Низкая плотность кода.
6. Ширина команды — огр. микроархитектуры.
7. Как соотносятся суперскалярные и VLIW процессоры применительно к разным классам задач?

Суперскалярные и VLIW процессоры отличаются подходами к параллелизму, что определяет их применимость к разным классам задач:

* **Суперскалярные процессоры**:
  + Подход: Динамическое распараллеливание инструкций в реальном времени.
  + Применение: Универсальные задачи (общая обработка, ОС, приложения), где код не оптимизирован, а динамическая адаптация важна (например, настольные компьютеры, серверы).
  + Преимущество: Гибкость при непредсказуемых зависимостях.
* **VLIW процессоры**:
  + Подход: Статическое распараллеливание компилятором в длинные инструкции.
  + Применение: Специализированные задачи с предсказуемым потоком (DSP, обработка сигналов, мультимедиа), где код оптимизирован заранее.
  + Преимущество: Высокая эффективность при статическом анализе.

Суперскалярные подходят для общих вычислений, VLIW — для узких, оптимизированных приложений.

1. Какие существуют промежуточные варианты между суперскалярным и VLIW процессором? Какие проблемы они решают?

Промежуточные варианты между суперскалярными и VLIW процессорами включают:

1. **Explicitly Parallel Instruction Computing (EPIC)** — комбинирует статическое распараллеливание (как в VLIW) с некоторыми динамическими возможностями (как в суперскалярных). Использует расширенные инструкции с подсказками для компилятора (например, Intel Itanium).
2. **Hybrid Architectures** — интегрируют VLIW-ядра с суперскалярными элементами, адаптируясь к нагрузке (например, некоторые DSP с переключаемым режимом).

Проблемы, которые они решают:

* Снижение зависимости от качества компилятора (EPIC добавляет динамическую гибкость).
* Баланс между производительностью и сложностью оборудования (гибридные архитектуры оптимизируют ресурсы).
* Улучшение адаптации к смешанным задачам, где статический и динамический подходы требуются одновременно.

1. Что такое барьеры памяти и в каких ситуациях они применяются?

Барьеры памяти (memory barriers) — это синхронизационные механизмы, которые гарантируют порядок выполнения операций с памятью между процессорами или ядрами, предотвращая переупорядочивание операций компилятором или оборудованием.

(вид барьерной инструкции, которая приказывает компилятору (при генерации инструкций) и процессору (при исполнении инструкций) устанавливать строгую последовательность между обращениями к памяти до и после барьера.)

Применение:

* В многопоточных системах для обеспечения согласованности данных (например, при использовании lock-free алгоритмов).
* В архитектурах с кэш-когерентностью для синхронизации доступа к общим ресурсам.
* В реальном времени, где предсказуемость порядка операций критична.

1. Какие ключевые элементы составляют структурное программирование? В чём отличие от кода на ассемблере?

Ключевые элементы структурного программирования:

* Последовательность (последовательное выполнение).
* Выбор (if-then-else, switch).
* Циклы (for, while, do-while).

Отличие от кода на ассемблере:

* Структурное программирование использует высокоуровневые конструкции с ясной логикой, тогда как ассемблер оперирует низкоуровневыми инструкциями (jmp, mov) без встроенной структуры.
* Ассемблер требует ручного управления потоками, в то время как структурное программирование обеспечивает модульность и читаемость.

1. Какие механизмы используются для реализации процедур?

Механизмы реализации процедур включают:

1. **Стек вызовов** — используется для хранения локальных переменных, параметров и адреса возврата (например, в большинстве современных процессоров).
2. **Передача параметров** — через регистры или стек (call by value, call by reference).
3. **Возврат управления** — через инструкции возврата (RET в ассемблере), восстанавливающие точку продолжения.
4. **Активационная запись** — структура в стеке, содержащая контекст процедуры (например, адрес возврата и сохранённые регистры).

Эти механизмы обеспечивают управление и передачу данных между вызывающей и вызываемой процедурой.

1. Что такое реентерабельность и почему она важна? Приведите примеры.

Реентерабельность (или реэнтерентность) — это свойство кода, которое позволяет ему безопасно выполняться одновременно несколькими потоками или процессами без риска повреждения данных или неправильного поведения. Это важно в многозадачных системах, таких как операционные системы или серверные приложения, где несколько вызовов функции могут перекрываться.

Примеры:

1. **Статическое выделение памяти**: Функция, использующая статические переменные, не является реентерабельной, так как данные могут быть перезаписаны другим вызовом.
   * Пример: swap с base->tmp может конфликтовать при параллельных вызовах.
2. **Автоматическая память, стек**: Функции, использующие стек (например, рекурсивный fact), обычно реентерабельны, так как каждый вызов имеет собственный контекст.
3. **Проблемы с данными**: Если функция изменяет глобальные переменные, она не реентерабельна. Использование локальных переменных или аргументов делает код безопасным.
4. Как реализуется механизм рекурсии?

Механизм рекурсии реализуется следующим образом:

1. **Реализация через call, return**:
   * Статическое выделение памяти для каждого входа.
   * Автоматическая память, стек (push, pop с рекурсией).
2. **Зачем**:
   * Параллелизм уровня задач.
   * Рекурсивные алгоритмы.
3. **Проблемы**: Автоматическая память, утечки данных, перезапись адреса возврата.
4. В чём заключается проблема распределения регистров и как работает алгоритм раскраски?

Проблема распределения регистров заключается в назначении ограниченного числа регистров переменным программы так, чтобы минимизировать конфликты и временные затраты.

Алгоритм раскраски графов решает это, представляя переменные как вершины графа, а конфликты (одновременное использование) — как рёбра. Вершины окрашиваются в цвета (регистры), так что соседние вершины имеют разные цвета. Если цветов не хватает, используется "спилл" (временное хранение в памяти).

1. Что такое FPGA и каковы его преимущества и недостатки?

FPGA (Field-Programmable Gate Array) — это программируемая логическая интегральная схема, позволяющая пользователю настраивать аппаратную логику после производства.

**Преимущества:**

* Гибкость: можно перепрограммировать для разных задач.
* Высокая производительность: параллельная обработка задач.
* Быстрая разработка: прототипирование без производства чипов.
* Адаптивность: подходит для специфических приложений.

**Недостатки:**

* Высокая стоимость по сравнению с ASIC для массового производства.
* Сложность проектирования: требует знаний в области аппаратного программирования.
* Ограниченная энергоэффективность: потребляет больше энергии, чем специализированные чипы.
* Ограниченные ресурсы: меньшая плотность логики по сравнению с ASIC.

1. Какие этапы включает в себя синтез для FPGA?

Синтез для FPGA включает следующие этапы:

1. **Анализ и трансляция**: Исходный код (VHDL/Verilog) анализируется, проверяется на синтаксические ошибки и преобразуется в промежуточное представление (RTL).
2. **Оптимизация RTL**: Логическая оптимизация для упрощения схемы, минимизации задержек и ресурсов.
3. **Сопоставление (Mapping)**: Преобразование RTL в логические элементы, доступные на FPGA (LUT, регистры).
4. **Размещение (Placement)**: Распределение логических элементов по физическим ресурсам FPGA.
5. **Трассировка (Routing)**: Настройка соединений между элементами с учетом временных и физических ограничений.
6. **Генерация битового потока**: Создание конфигурационного файла для программирования FPGA.
7. **Проверка тайминга**: Анализ временных характеристик для обеспечения корректной работы схемы.
8. Какие FPGA устроены изнутри?

FPGA (Field-Programmable Gate Array) состоит из следующих основных компонентов:

1. **Логические блоки (CLB)**: Основные элементы для реализации логики, включают:
   * **LUT (Look-Up Tables)**: Таблицы для выполнения логических функций.
   * **Триггеры**: Для хранения состояния (регистры).
   * **Мультиплексоры**: Для маршрутизации сигналов.
2. **Программируемая межсоединительная сеть**: Система проводников и коммутаторов для связи между логическими блоками, вводами/выводами и другими ресурсами.
3. **Блоки ввода-вывода (IOB)**: Интерфейсы для связи с внешними устройствами, поддерживают различные стандарты сигналов.
4. **Специализированные блоки**:
   * **DSP-блоки**: Для выполнения операций цифровой обработки сигналов (умножение, сложение).
   * **Блоки памяти (BRAM)**: Встроенная память для хранения данных.
   * **Часовые ресурсы**: Генераторы тактовых сигналов и PLL/DCM для управления синхронизацией.
5. **Конфигурационная память**: Хранит данные для настройки логики и соединений, программируется пользователем.
6. Как соотносятся CPU и FPGA с точки зрения модели программирования, модели вычислений, модели исполнения и элементов микроархитектуры?

 **Модель программирования**:

* **CPU**: Высокоуровневые языки (C, Python) или ассемблер, компилируемые в последовательные инструкции. Программирование ориентировано на временную последовательность операций.
* **FPGA**: Языки описания аппаратуры (VHDL, Verilog), описывающие параллельную логику и схемы. Программирование задаёт структуру и поведение на уровне оборудования.

 **Модель вычислений**:

* **CPU**: Последовательная модель (фон Неймана), выполнение инструкций по одной (или с ограниченным параллелизмом через конвейеры/SIMD).
* **FPGA**: Параллельная модель, где множество логических блоков работают одновременно, реализуя специализированные схемы для конкретных задач.

 **Модель исполнения**:

* **CPU**: Выполнение инструкций в фиксированной микроархитектуре (регистры, ALU, кэш) с временным разделением ресурсов. Исполнение универсально и управляется программным кодом.
* **FPGA**: Аппаратная реализация задач, где логика "встраивается" в конфигурируемые блоки (LUT, триггеры). Исполнение полностью параллельное, без разделения времени.

 **Элементы микроархитектуры**:

* **CPU**: Фиксированная структура — ALU, регистры, кэш-память, контроллеры прерываний, конвейеры, предсказатели ветвлений. Оптимизирована для универсальных вычислений.
* **FPGA**: Программируемые элементы — логические блоки (CLB с LUT и триггерами), межсоединения, блоки ввода-вывода (IOB), DSP-блоки, встроенная память (BRAM). Гибкость в настройке под конкретные задачи.

1. Что такое высокоуровневый синтез (HLS) и какова его роль в современной разработке?

Высокоуровневый синтез (HLS, High-Level Synthesis) — это процесс автоматического преобразования кода на высокоуровневых языках программирования (например, C, C++, или SystemC) в описание оборудования на языке описания оборудования (HDL), таком как VHDL или Verilog.

Роль в современной разработке:

* Ускоряет проектирование сложных систем на чипах (SoC) и полевых программируемых вентильных массивах (FPGA).
* Сокращает время разработки за счёт абстракции от низкоуровневых деталей.
* Позволяет инженерам сосредотачиваться на алгоритмах, а не на оборудовании.

1. Как высокоуровневый синтез (HLS) автоматизирует процесс выбора микроархитектуры и ускоряет тестирование?

Высокоуровневый синтез (HLS) автоматизирует процесс выбора микроархитектуры, анализируя код на высоком уровне (например, C/C++) и автоматически генерируя оптимальные варианты аппаратного описания (HDL), включая выбор типов блоков, регистров и конвейеров, основываясь на заданных ограничениях (скорость, площадь, энергопотребление).

Ускоряет тестирование за счёт:

* Генерации симуляций и верификации на ранних этапах.
* Автоматического создания тестовых бенчмарков.
* Повторного использования проверенных модулей, что снижает ручной труд.

1. Что представляет собой классификация Флинна и какие классы архитектур она выделяет? Примеры.

Классификация Флинна представляет собой систему классификации архитектур вычислительных систем по количеству потоков инструкций и данных. Она выделяет следующие классы:

1. **SISD (Single Instruction, Single Data)** — однопоточная архитектура с одной инструкцией и одним потоком данных. Пример: традиционные процессоры (например, ранние CPU).
2. **SIMD (Single Instruction, Multiple Data)** — одна инструкция выполняется над несколькими данными. Пример: GPU, векторные процессоры.
3. **MISD (Multiple Instruction, Single Data)** — несколько инструкций применяются к одному потоку данных (редко встречается). Пример: теоретические системы с параллельной обработкой.
4. **MIMD (Multiple Instruction, Multiple Data)** — несколько инструкций и данных обрабатываются параллельно. Пример: многоядерные процессоры, кластеры.

Особенности: простота и понятность, не применима к не фон Неймановским архитектурам, переусложнённость класса MIMD.

1. Что такое SIMD и SIMT архитектуры? В чём их различия? Что такое lockstep execution?

SIMD (Single Instruction, Multiple Data) — архитектура, где одна инструкция выполняется одновременно над несколькими данными с использованием нескольких процессоров (PU), что упрощает вычисления (например, в Lockstep execution).

SIMT (Single Instruction, Multiple Threads) — расширение SIMD, где одна инструкция выполняется над несколькими потоками данных, как в современных GPU и GPGPU (CUDA), с возможностью ветвления (divergence).

Различия:

* SIMD работает с фиксированными данными, SIMT — с потоками, поддерживая ветвление.
* SIMT более гибок за счёт управления потоками.

Lockstep execution — синхронное выполнение инструкций всеми процессорами, шаг за шагом, что характерно для SIMD.

1. Что представляет собой классификация Дункана и чем она отличается от классификации Флинна? Какие проблемы она решает? Что включает верхний уровень классификации?

Классификация Дункана представляет собой иерархическую систему, расширяющую классификацию Флинна, разделяя архитектуры по синхронности и типам памяти, включая MIMD парадигмы. Отличия от классификации Флинна:

* Флинн классифицирует по потокам инструкций и данных (SISD, SIMD, MISD, MIMD).
* Дункан добавляет детализированные подтипы, такие как Vector, Systolic, Dataflow, Reduction, Wavefront, учитывая синхронность и память.

Проблемы, которые решает:

* Подробное описание сложных архитектур.
* Учет современных (на 1990 год) подходов, включая MoC (Models of Computation).
* Абстрагироваться от низкоуровневого параллелизма (конвейер, суперскаляр, VLIW).
* Переиспользовать элементы таксономии Флинна от 1966.
* Актуализировать и включить новинки на 1990 год.
* Навести порядок в MIMD.

Верхний уровень классификации:

* Синхронные (Synchronous): Vector, SIMD, Systolic, Associative memory.
* MIMD: Distributed memory, Shared memory, MIMD/SIMD, Dataflow, Reduction, Wavefront.

1. Какие основные типы синхронных архитектур выделяются по классификации Дункана?

Векторные архитектуры.

SIMD архитектуры:

* Процессорные массивы.
* Ассоциативные массивы.

Систолические массивы.

Совмещение памяти и вычислений\*.

1. Какие принципы лежат в основе работы векторных архитектур и где они применяются?

Принципы работы векторных архитектур:

1. Сокращение количества обрабатываемых инструкций.
2. Конвейерное исполнение векторных операций.
3. Множество функциональных узлов (суперскалярность).

Применение:

* Векторные операции — операции, выполняемые на последовательностях данных.
* Применяются в научных вычислениях, обработке изображений и других вычислительно-интенсивных задачах.

1. Как работают ассоциативные массивы и где они применяются?

Ассоциативные массивы (Content-Addressable Memory, CAM) работают, сравнивая входной поисковый шаблон с содержимым памяти параллельно, возвращая адрес или результат совпадения. Принципы:

1. Позволяют быстро искать информацию в массивах данных.
2. Используются в сетевых устройствах.

Применение:

* В сетевых маршрутизаторах для быстрого поиска адресов.
* В базах данных и системах искусственного интеллекта для ускоренного поиска.

1. Как работают систолические архитектуры и где они применяются?

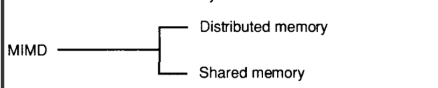
Систолические архитектуры работают как конвейерная система, где данные циркулируют между процессорами (P) и общей памятью (Memory) в синхронном ритме, подобно пульсу. Принципы:

1. Данные ритмично поступают из памяти.
2. Проходят через сеть процессоров, возвращаются в память.
3. Синхронизация — через глобальные часы (не путать с тактовым сигналом).

Применение:

* Процессоры объединены локальными связями и каждый пик процессора выполняют короткие неизменные операции.

1. Какие основные типы MIMD-архитектур выделяются по классификации Дункана?



Аналогия из современных веб приложений:

* синхронизация через единую базу данных;
* микросервисная архитектура.

1. Что из себя представляют архитектуры с распределённой памятью? Каковы их особенности?

Архитектуры с распределённой памятью (Distributed Memory Architectures) представляют собой системы, где каждый процессор имеет собственную локальную память (P и M на схеме), а доступ к данным других процессоров осуществляется через механизм обмена сообщениями (message passing).

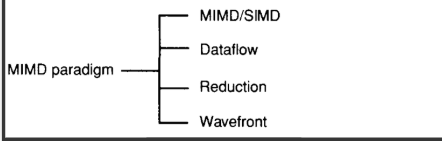
Особенности:

1. Координация совместной работы через прямое взаимодействие процессоров.
2. Данные/сигналы явно передаются между процессорами.
3. Мотивация: обеспечить масштабируемость многопроцессорной архитектуры и доступ к локальным данным процессора.
4. Что из себя представляет архитектуры с разделяемой памятью? Каковы их особенности?

Архитектуры с разделяемой памятью (Shared Memory Architectures) представляют собой системы, где несколько процессоров (P) имеют доступ к общей памяти (M), что позволяет им совместно использовать данные.

Особенности:

1. Координация совместной работы через общую память.
2. Кэш-когерентность данных для согласования кэшей процессоров.
3. Примеры подходов: (а) шинное соединение; (б) перекрестная шина 2x2; (с) сеть на кристалле, запрос маршрутизируется от P к M.
4. Широко распространены, так как применимы и прозрачны для программистов.
5. Какие основные типы MIMD парадигм выделяются по классификации Дункана?



1. Что представляет собой MIMD/SIMD парадигма и где она применяется?

MIMD/SIMD парадигма сочетает в себе две модели параллельных вычислений: MIMD (Multiple Instruction, Multiple Data), где разные процессоры выполняют разные инструкции над разными данными, и SIMD (Single Instruction, Multiple Data), где один набор инструкций применяется к нескольким данным одновременно.

В этой архитектуре используется мастер-слейв подход: мастер (контроллер) распределяет задачи как SIMD, а слейвы (процессоры) могут работать как MIMD с локальной памятью.

Применяется в задачах, требующих высокой параллелизации, таких как обработка графики, научные вычисления и анализ данных (например, через fork/join модели).

1. Как работает архитектура потоков данных (dataflow)? Подходы к реализации.

Архитектура потоков данных (dataflow) работает на основе активации вычислений, когда данные и инструкции представлены в виде токенов, а вычисления выполняются по мере их доступности.

Основные принципы:

* Распределённая память. Токены (данные и инструкции) циркулируют между узлами.
* По кольцу посылаются токены (метка + значение).
* Получатель пересылает токен дальше и сохраняет значение, если является адресатом.
* Если задача разблокирована (все необходимые токены получены), запускается работа.
* Результат работы отправляется токеном по кольцу.

Подходы к реализации:

* Распределённая память (распределение данных между узлами).
* Отсутствует память и канал доступа как "бутылочное горлышко".
* Аналогия: микросервисные архитектуры.

1. Как работает редукционная архитектура? Каким образом она обеспечивает параллелизм?

Редукционная архитектура (Reduction) работает на основе графового представления вычислений, где данные и операции представлены узлами, а их выполнение активируется по мере доступности всех необходимых входных данных (demand-driven подход). Процесс:

* Разрабатываются для параллелизма и функционального программирования.
* Выполнение: выборка инструкций, заменяемых инструкциями на её результат в графе.
* Проблемы: ссылокная прозрачность, разворачивание.

Параллелизм обеспечивается:

* Независимыми подграфами, которые могут выполняться одновременно.
* Распределением вычислений между узлами (например, Node 2 и Node 5), где каждый узел обрабатывает свои данные (d, e, f, g) параллельно.

1. Как работает wavefront архитектура? В чём её отличия от систолической архитектуры?

Wavefront архитектура работает как синхронная система, где вычисления выполняются волнообразно (wavefront), распространяясь по массиву процессоров в зависимости от доступности данных. Основные принципы:

1. Комбинация систолической архитектуры с асинхронным потоком данных.
2. Управление процессом динамически, за счёт распространения сигнала по сети.
3. Синхронизация и режим работы выстраиваются динамически.
4. Процесс в программировании. Лучше масштабируется.
5. Риск переполнения буферов B Run-Time.

Отличия от систолической архитектуры:

* Wavefront использует динамическую синхронизацию и асинхронный поток данных, в то время как систолическая архитектура полагается на строгую синхронизацию и регулярный поток данных.
* Wavefront лучше масштабируется, тогда как систолическая архитектура оптимизирована для фиксированных массивов процессоров.

1. Что такое CGRA-процессоры и какие возможности они предоставляют?

Архитектура потоков данных (dataflow) работает на основе активации вычислений, когда данные и инструкции представлены в виде токенов, а вычисления выполняются по мере их доступности.

Основные принципы:

* Распределённая память. Токены (данные и инструкции) циркулируют между узлами.
* По кольцу посылаются токены (метка + значение).
* Получатель пересылает токен дальше и сохраняет значение, если является адресатом.
* Если задача разблокирована (все необходимые токены получены), запускается работа.
* Результат работы отправляется токеном по кольцу.

Подходы к реализации:

* Распределённая память (распределение данных между узлами).
* Отсутствует память и канал доступа как "бутылочное горлышко".
* Аналогия: микросервисные архитектуры.

1. В чём различие между пространственными (spatial) и временными (temporal) вычислениями на примере CGRA?

Различие между **пространственными (spatial)** и **временными (temporal)** вычислениями на примере CGRA (Coarse-Grained Reconfigurable Architecture):

1. **Пространственные вычисления**:
   * **Сущность**: Вычисления распределяются по множеству физических вычислительных элементов (PE — Processing Elements) в CGRA, работающих параллельно. Каждый элемент выполняет свою операцию одновременно с другими.
   * **Пример в CGRA**: Операции (например, сложение, умножение) назначаются на разные PE, соединённые программируемой сетью. Данные передаются между PE напрямую, минимизируя зависимость от времени.
   * **Особенности**: Высокая параллельность, низкая задержка, но требует больше аппаратных ресурсов.
2. **Временные вычисления**:
   * **Сущность**: Вычисления выполняются последовательно на одном или ограниченном числе вычислительных элементов, где ресурсы переиспользуются во времени.
   * **Пример в CGRA**: Один PE выполняет несколько операций поочерёдно, используя временное планирование (scheduling). Данные сохраняются в регистрах или памяти между шагами.
   * **Особенности**: Экономия ресурсов, но увеличение времени выполнения из-за последовательности.

**Ключевое различие**:

* Пространственные вычисления используют **параллелизм** за счёт распределения задач по множеству PE, что ускоряет выполнение, но требует больше аппаратного пространства.
* Временные вычисления используют **переиспользование** ресурсов во времени, что экономит место, но увеличивает задержки.
* В CGRA оба подхода могут комбинироваться: пространственный параллелизм для независимых операций и временное планирование для оптимизации использования PE.