Вопросы к зачету КСИС:

1. Что такое микропроцессорная система? Особенности традиционной цифровой системы.

Микропроцессорная система - электронная система, предназначенная для обработки входных сигналов и выдачи выходных сигналов. В качестве входных и выходных сигналов при этом могут использоваться

* аналоговые сигналы
* одиночные цифровые сигналы,
* цифровые коды,
* последовательности цифровых кодов.

1. Что такое микропроцессорная система? Особенности системы на гибкой логике?

Ответ: **Микропроцессорные системы** ­– электронные системы, предназначенные для автоматизации обработки информации и управления различными процессами.

Преодолеть этот недостаток позволяют электронные системы **гибкой логики**, которые могут легко адаптироваться под любую задачу, перестраиваясь с одного алгоритма на другой без изменения электронной схемы. В таких системах изменение условий влияет только на изменение программы, в соответствии с которой работает система

1. Преимущества и недостатки систем на жесткой и гибкой логике?

Характерной особенностью традиционных цифровых систем является тот факт, что алгоритм обработки и хранения информации в ней жёстко связан со схемотехникой системы, то есть для конкретно поставленной задачи разрабатывается и реализуется конкретная электронная схема. Любое изменение исходных условий задачи повлечет за собой и изменение её схемотехнического решения, т.е. изменение алгоритма функционирования системы возможно только путём изменения её структуры. Такие схемы называют **схемами жёсткой логики**.

Преимуществом систем жёсткой логики является их высокое быстродействие, так как такие системы никогда не имеют аппаратной избыточности, а скорость выполнения алгоритмов определяется в ней только быстродействием отдельных логических элементов.

Самым большим недостатком цифровой системы на жёсткой логике является тот факт, что при изменении условий задачи схему нужно проектировать и изготавливать заново.

Преодолеть этот недостаток позволяют электронные системы **гибкой логики**, которые могут легко адаптироваться под любую за-дачу, перестраиваясь с одного алгоритма на другой без изменения электронной схемы. В таких системах изменение условий влияет только на изменение программы, в соответствии с которой работает система

Такая система является программируемой (перепрограммируемой). Именно к системе гибкой логики и относятся микропроцессорные системе.

Конечно, аппаратно такая схема может быть избыточна, так как должна функционировать и для решения самой простой, и для решения самой сложной задачи. А решение трудной задачи требует гораздо больше аппаратных средств, чем решение простой.

1. Что такое Большая интегральная схема? Какие бывают процессоры по количеству БИС?

**Больша́я интегра́льная схе́ма** (БИС) — **интегральная схема** соответствующая большой степени интеграции (плотности упаковки (размещения) на одном кристалле (чипе) элементов **интегральных схем**). Число схемных элементов в БИС от 103 до 10**4**.

По числу **больших интегральных схем** (**БИС**) в микропроцессорном комплекте различают микропроцессоры однокристальные, многокристальные и многокристальные секционные. **Процессоры** даже самых простых ЭВМ имеют сложную функциональную структуру, содержат большое **количество** электронных элементов и множество разветвленных связей

1. Виды процессоров по назначению? Преимущества и недостатки?

**арифметический процессор** , **буферный процессор** , **процессор данных** , **процессор баз данных** , **текстовый процессор** , **процессор ввода-вывода** , **интерфейсный процессор** , **лингвистический процессор** , **сетевой процессор** , **межсетевой процессор** , **процессор передачи данных** , **терминальный процессор** , **специализированный процессор** и др. ;

Этот класс процессоров предназ­начен для решения широкого круга задач обработки разнообразной информации и находит применение в персональных компьютерах, рабочих станциях, серверах и других цифровых системах массового применения. К универсальным процессо­рам относят 32–разрядные микропроцессоры (хотя некоторые микропроцессоры этого класса имеют 64–разрядную или 128–разрядную структуру), которые изго­тавливаются по самой современной промышленной технологии, обеспечивающей максимальную частоту функционирования.

1. Виды процессоров по виду обрабатываемых сигналов? Преимущества и недостатки?

По виду обрабатываемых входных сигналов различают цифровые и аналоговые микропроцессоры. Сами микропроцессоры цифровые устройства, однако могут иметь встроенные аналого-цифровые и цифро-аналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются и после обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе времени и т.д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т.д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной "настройки" цифровой части микропроцессора на различные алгоритмы обработки сигналов

1. Виды процессоров по временной организации работы? Преимущества и недостатки?

По характеру **временной организации работы** микропроцессоры делят на синхронные и асинхронные. Синхронные микропроцессоры - микропроцессоры, в которых начало и конец выполнения операций задаются устройством управления (время выполнения операций в этом случае не зависит от **вида** выполняемых команд и величин операндов).

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции. Для более эффективного использования каждого устройства микропроцессорной системы в состав асинхронно работающих устройств вводят электронные цепи, обеспечивающие автономное функционирование устройств. Закончив работу над какой-либо операцией, устройство вырабатывает сигнал запроса, означающий его готовность к выполнению следующей операции. При этом роль естественного распределителя работ принимает на себя память, которая в соответствии с заранее установленным приоритетом выполняет запросы остальных устройств по обеспечению их командной информацией и данными.

1. Структура процессора?

Центральный процессор – электронный блок либо интегральная схема (микропроцессор), исполняющая машинные инструкции (код программ), главная часть аппаратного обеспечения компьютера или программируемого логического контроллера.

Самое главное в процессоре это регистры. Регистры состоят из триггеров. Триггер может иметь 2 значения 0 или 1. Регистры бывают 8, 16, 32, 64 разрядные, понятно, что если 8 разрядов, то в регистре 8 триггеров.

Регистры – это специальные ячейки памяти, физически расположенные внутри процессора. В отличие от ОЗУ, где для обращения к данным требуется использовать шину адреса, к регистрам процессор может обращаться напрямую. Это существенно ускорят работу с данными.

* АЛУ – арифметико-логическое устройство выполняет арифметические операции, такие как сложение, вычитание, а также логические операции.
* БУ – блок управления определяет последовательность микрокоманд, выполняемых при обработке машинных кодов (команд).
* ТГ – тактовый генератор, или генератор тактовых импульсов, задаёт рабочую частоту процессора. С помощью тактовых импульсов выполняется синхронизация для внутренних команд процессора и остальных устройств. Тактовый генератор вырабатывает (генерирует) прямоугольные импульсы, которые следуют с определённой частотой.

1. Типы архитектур процессора? По набору команд? По организации адресного пространства?

CISC

CISC (англ. Complex Instruction Set Computer — «компьютер с полным набором команд») — тип процессорной архитектуры, в первую очередь, с нефиксированной длиной команд, а также с кодированием арифметических действий в одной команде и небольшим числом регистров, многие из которых выполняют строго определенную функцию.

Самый яркий пример CISC архитектуры — это x86 (он же IA-32) и x86\_64 (он же AMD64).

В CISC процессорах одна команда может быть заменена ей аналогичной, либо группой команд, выполняющих ту же функцию. Отсюда вытекают плюсы и минусы архитектуры: высокая производительность благодаря тому, что несколько команд могут быть заменены одной аналогичной, но большая цена по сравнению с RISC процессорами из-за более сложной архитектуры, в которой многие команды сложнее раскодировать.

### RISC

RISC (англ. Reduced Instruction Set Computer — «компьютер с сокращённым набором команд») — архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций: их декодирование становится более простым, а время выполнения — меньшим. Первые RISC-процессоры не имели даже инструкций умножения и деления и не поддерживали работу с числами с плавающей запятой.

По сравнению с CISC эта архитектура имеет константную длину команды, а также меньшее количество схожих инструкций, позволяя уменьшить итоговую цену процессора и энергопотребление, что критично для мобильного сегмента. У RISC также большее количество регистров.

Примеры RISC-архитектур: PowerPC, серия архитектур ARM (ARM7, ARM9, ARM11, Cortex).

В общем случае RISC быстрее CISC. Даже если системе RISC приходится выполнять 4 или 5 команд вместо одной, которую выполняет CISC, RISC все равно выигрывает в скорости, так как RISC-команды выполняются в 10 раз быстрее.

Отсюда возникает закономерный вопрос: почему многие всё ещё используют CISC, когда есть RISC? Всё дело в совместимости. x86\_64 всё ещё лидер в desktop-сегменте только по историческим причинам. Так как старые программы работают только на x86, то и новые desktop-системы должны быть x86(\_64), чтобы все старые программы и игры могли работать на новой машине.

Для Open Source это по большей части не является проблемой, так как пользователь может найти в интернете версию программы под другую архитектуру. Сделать же версию проприетарной программы под другую архитектуру может только владелец исходного кода программы.

### MISC

MISC (англ. Minimal Instruction Set Computer — «компьютер с минимальным набором команд»).

Ещё более простая архитектура, используемая в первую очередь для ещё большего уменьшения итоговой цены и энергопотребления процессора. Используется в IoT-сегменте и недорогих компьютерах, например, роутерах.

Для увеличения производительности во всех вышеперечисленных архитектурах может использоваться “спекулятивное исполнение команд”. Это выполнение команды до того, как станет известно, понадобится эта команда или нет.

### VLIW

VLIW (англ. Very Long Instruction Word — «очень длинная машинная команда») — архитектура процессоров с несколькими вычислительными устройствами. Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно.

По сути является архитектурой CISC со своим аналогом спекулятивного исполнения команд, только сама спекуляция выполняется во время компиляции, а не во время работы программы, из-за чего уязвимости Meltdown и Spectre невозможны для этих процессоров. Компиляторы для процессоров этой архитектуры сильно привязаны к конкретным процессорам. Например, в следующем поколении максимальная длина «очень длинной команды» может из условных 256 бит стать 512 бит, и тут приходится выбирать между увеличением производительности путём компиляции под новый процессор и обратной совместимостью со старым процессором. Опять же, Open Sourсe позволяет простой перекомпиляцией получить программу под конкретный процессор.

1. Особенности шинной структуры связей? Преимущества и недостатки?

При шинной структуре связей (рис. 1.6) все сигналы между устройствами передаются по одним и тем же линиям связи, но в разное время (это называется мультиплексированной передачей). Причем передача по всем линиям связи может осуществляться в обоих направлениях (так называемая двунаправленная передача). В результате количество линий связи существенно сокращается, а правила обмена (протоколы) упрощаются. Группа линий связи, по которым передаются сигналы или коды как раз и называется шиной (англ. bus).

при шинной структуре связей легко осуществляется пересылка всех информационных потоков в нужном направлении,

Однако при шинной структуре связей вся информация передается по линиям связи последовательно во времени, по очереди, что снижает быстродействие системы по сравнению с классической структурой связей.

1. Что такое системная магистраль? Что в нее входит?

Системная магистраль обеспечивает коммуникацию компонентов МПС(микропроцессорная система) друг с другом. Конструктивно системная магистраль представляет набор печатных проводников, образующих линии связи, посредством которых соединяются контакты розеток (слоты) модулей МПС.

Системная магистраль содержит 3 различные шины:

- шина адреса – набор однонаправленных сигнальных линий, по которым от ЦП к памяти или к устройствам ввода/вывода передаются коды адреса, количество линий шины адреса равно максимальной разрядности кода адреса;

- шина данных – набор двунаправленных сигнальных линий, по которым передаются данные между процессором и памятью либо устройствами ввода-вывода, разрядность шины данных определяет разрядность всей системной шины;

- шина управления – набор сигнальных линий, которые используются для стробирования передаваемых или принимаемых данных (то есть для определения момента времени, когда информационный код выставлен на шину данных), для подтверждения приема данных, для тактирования (синхронизации работы) модулей МПС и т. д.

1. Режимы обмена информацией на системной магистрали?

Циклы обмена информацией делятся на два основных типа:

- цикл записи (вывода), в котором процессор записывает (выводит) информацию;

- цикл чтения (ввода), в котором процессор читает (вводит) информацию.

Во время каждого цикла устройства, участвующие в обмене информации, передают друг другу по системной магистрали информационные и управляющие сигналы в строго установленном порядке или, как еще говорят, в соответствии с принятым протоколом обмена информацией.

1. Регистры общего назначения? Как организованы?

Регистры EAX, EBX, ECX, EDX – это регистры общего назначения. Они имеют определённое назначение (так уж сложилось исторически), однако в них можно хранить любую информацию.

Регистры EBP, ESP, ESI, EDI – это также регистры общего назначения. Они имеют уже более конкретное назначение. В них также можно хранить пользовательские данные, но делать это нужно уже более осторожно, чтобы не получить «неожиданный» результат.

1. Классификация команд?

По архитектуре команд процессоры можно подразделить на

- процессоры с расширенной системой команд (CISC-процессоры) и

- процессоры с ограниченной системой команд (RISC- процессоры)

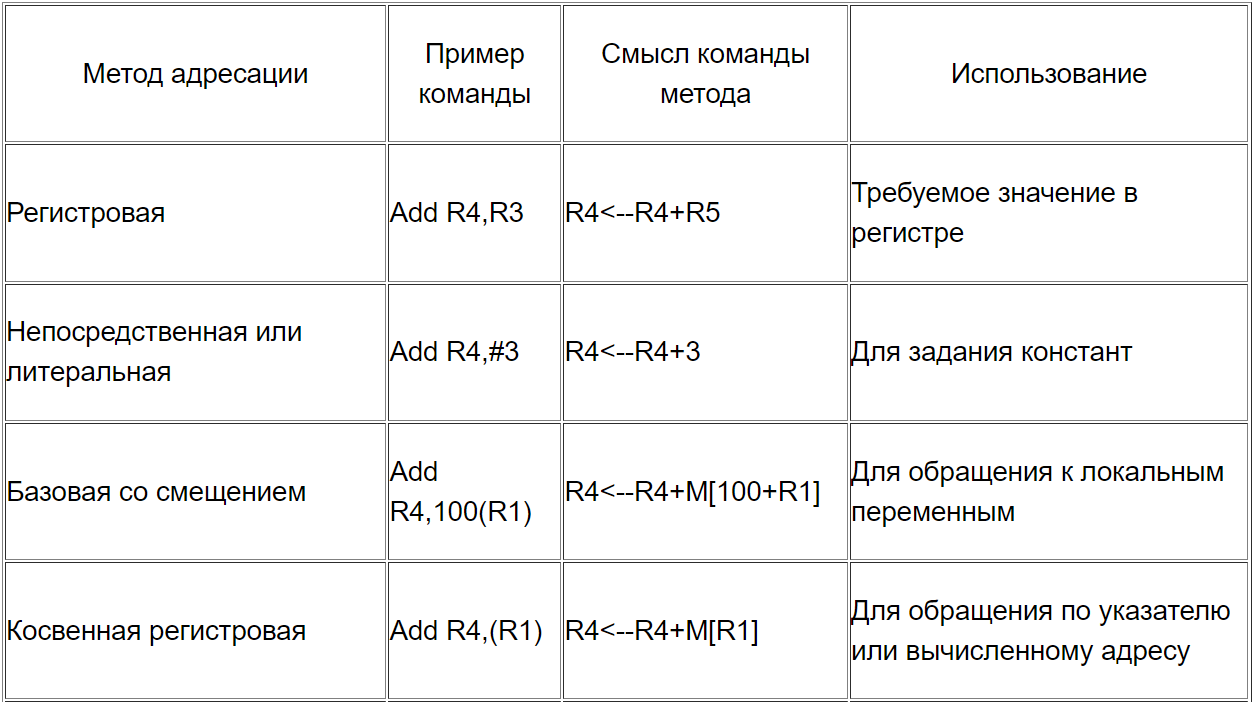
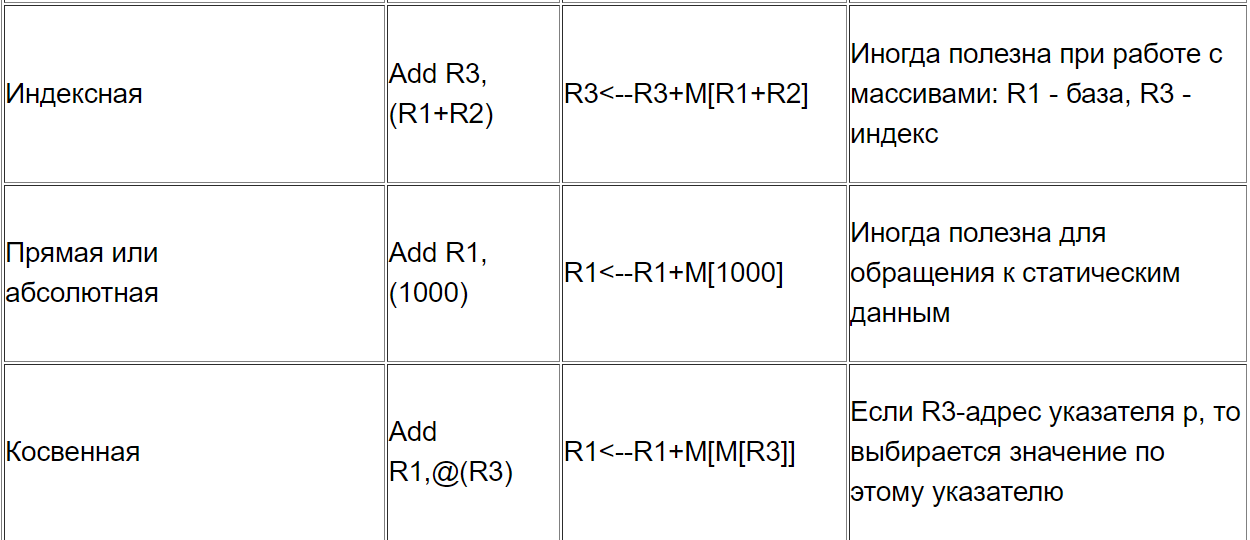
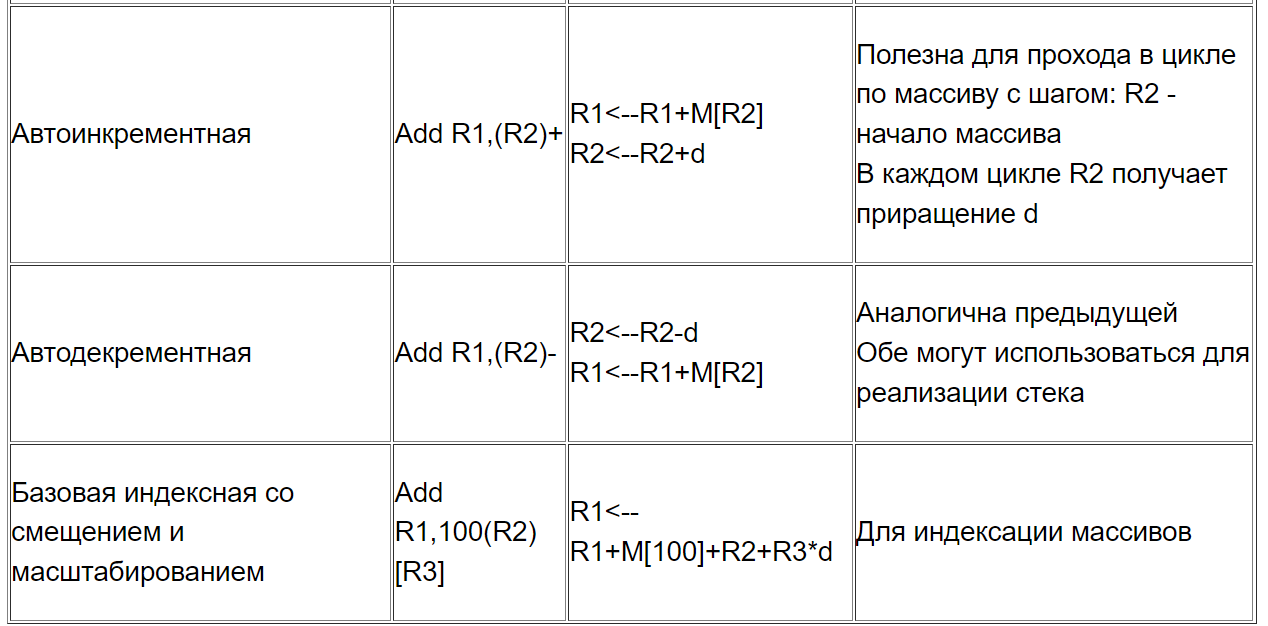
CISC- процессоры (Complex Instruction Set Computer) характеризуется широким набором команд, позволяющим обрабатывать данные различного формата с использованием различных сложных способов адресации. При этом многие процедуры реализации команд выносятся на аппаратный уровень. Такие процессоры ориентированы на использование в ЭВМ общего назначения, и им свойственно следующее:

* большинство команд имеют обращение к ОЗУ,
* длительность цикла исполнения каждой команды индивидуальна,
* имеет место ограниченный объем регистровой памяти.

RISC – процессоры (Reduced Instruction Set Computer) характеризуются не только ограничением количества команд, но и использованием самых простых способов адресации. Все команды разделены на две группы: к первой группе отнесены команды обработки данных, причем их состав таков, что все они короткие. Ко второй группе относятся все остальные команды, в том числе, команды обращения к памяти. Команды первой группы работают только с регистровой памятью, просты по своим функциям и имеют одинаковую длительность циклов. Выполнение длинных операций вынесено на программный уровень.

Кроме того, регистровая память в RISC - процессорах имеет существенно большую емкость; последнее необходимо для уменьшения в теле программы количества команд обмена с целью экономии процессорного времени на обмен между ОЗУ и РОН.

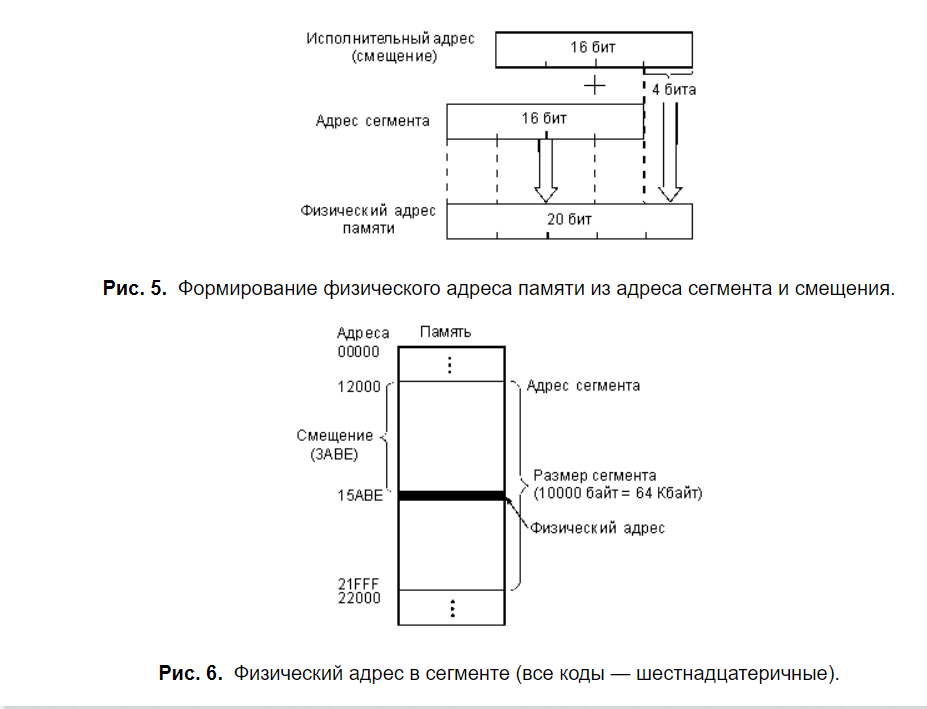
1. Методы адресации?

1. Сегментирование памяти? Для чего нужно?

Для хранения кодов адресов памяти используются не отдельные регистры, а пары регистров:

* сегментный регистр определяет адрес начала сегмента (то есть положение сегмента в памяти);
* регистр указателя (регистр смещения) определяет положение рабочего адреса внутри сегмента.



1. Архитектура параллельных вычислительных систем? Преимущества и недостатки?

Параллельные вычислительные системы — это физические компьютерные, а также программные системы, реализующие тем или иным способом параллельную обработку данных на многих вычислительных узлах.[1] Идея распараллеливания вычислений основана на том, что большинство задач может быть разделено на набор меньших задач, которые могут быть решены одновременно.

Писать программы для параллельных систем сложнее, чем для последовательных[[3]](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B#cite_note-3), так как конкуренция за ресурсы представляет новый класс потенциальных ошибок в программном обеспечении ([багов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B0)), среди которых [состояние гонки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8) является самой распространённой. Взаимодействие и [синхронизация](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D0%B2&action=edit&redlink=1) между процессами представляют большой барьер для получения высокой производительности параллельных систем.

1. Классификация параллельных вычислительных систем?

Общая классификация архитектур ЭВМ по признакам наличия [параллелизма](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B7%D0%BC_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в потоках команд и данных была предложена [Майклом Флинном](https://ru.wikipedia.org/wiki/%D0%A4%D0%BB%D0%B8%D0%BD%D0%BD,_%D0%9C%D0%B0%D0%B9%D0%BA%D0%BB_(%D1%83%D1%87%D1%91%D0%BD%D1%8B%D0%B9)) в 1966 году[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC#cite_note-1) и расширена в 1972 году[[2]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC#cite_note-2). Все разнообразие архитектур ЭВМ в этой таксономии сводится к четырём классам:

* ОКОД — Вычислительная система с **о**диночным потоком **к**оманд и **о**диночным потоком **д**анных  
  ([SISD](https://ru.wikipedia.org/wiki/SISD), **S**ingle **I**nstruction stream over a **S**ingle **D**ata stream).
* ОКМД — Вычислительная система с **о**диночным потоком **к**оманд и **м**ножественным потоком **д**анных  
  ([SIMD](https://ru.wikipedia.org/wiki/SIMD), **S**ingle **I**nstruction, **M**ultiple **D**ata).
* МКОД — Вычислительная система со **м**ножественным потоком **к**оманд и **о**диночным потоком **д**анных  
  ([MISD](https://ru.wikipedia.org/wiki/MISD), **M**ultiple **I**nstruction **S**ingle **D**ata).
* МКМД — Вычислительная система со **м**ножественным потоком **к**оманд и **м**ножественным потоком **д**анных  
  ([MIMD](https://ru.wikipedia.org/wiki/MIMD), **M**ultiple **I**nstruction **M**ultiple **D**ata).

1. Системы с общей и распределенной памятью? Преимущества и недостатки?

В системах с общей памятью все процессоры имеют равные возможности по доступу к единому адресному пространству. Единая память может быть построена как одноблочная или по модульному принципу, но обычно практикуется второй вариант. В системе с распределенной памятью каждый процессор обладает собственной памятью и способен адресоваться только к ней. Некоторые авторы называют этот тип систем многомашинными ВС или мульткомпьютерами, подчеркивая тот факт, что блоки, из которых строится система, сами по себе являются небольшими вычислительными системами с процессором и памятью. Модели архитектур с распределенной памятью принято обозначать как архитектуры без прямого доступа к удаленной памяти (NORMA, No Remote Memory Access). Такое название следует из того факта, что каждый процессор имеет доступ только к своей локальной памяти. Доступ к удаленной памяти (локальной памяти другого процессора) возможен только путем обмена сообщениями с процессором, которому принадлежит адресуемая память.

1. Способы распараллеливания?

Распараллеливание программ - процесс адаптации алгоритмов, записанных в виде программ, для их эффективного исполнения на вычислительной системе параллельной архитектуры.

Распараллеливание программ проводят двумя способами:

применение специальных библиотек программ некоторого стандартного языка программирования путем переписывания кода программ на специальный язык, описывающий параллелизм и понятный трансляторам целевой вычислительной системы (например, Windows API - многопоточные приложения для C++);

вставка некоторой специальной разметки (например, в соответствии со стандартом систем передачи сообщений между параллельно исполняемыми процессами инструкций Message Passing Interface (MPI) или стандартом интерфейса приложений для параллельных систем с общей памятью ОрепМР). Распараллеливание может быть ручным, автоматизированным и полуавтоматизированным. Для оценки эффективности его качества применяются вышеописанные характеристики - ускорение и загруженность.

1. Dataflow архитектура? Преимущества и недостатки?

В архитектурах с управлением потоком данных (Dataflow) [[01]](https://habr.com/ru/post/122479/#ref-01) отсутствует понятие «последовательность инструкций», нет Instruction Pointer'а, отсутствует даже адресуемая память в привычном нам смысле. Программа в потоковой системе — это не набор команд, а вычислительный граф. Каждый узел графа представляют собой оператор или набор операторов, а ветви отражают зависимости узлов по данным. Достоинства и недостатки DFD нотации

У каждого инструментария предназначенного для бизнес-моделирования, есть свои плюсы и минусы. Остановимся по подробнее на Data Flow Diagrams.

К главным преимуществам можно отнести:

* Способность нотации точно определить внешние сущности, при этом используя анализ потоков информации внутри и за пределами системы
* Способность проектирование сверху вниз.
* Описание процессов нижнего уровня. Это нужно для преодоления логической незавершенности модели и построении полностью функциональной спецификации для разрабатываемой системы.

1. Аппаратная реализация Dataflow?

В потоковых машинах данные передаются и хранятся в виде т.н. *токенов* (token). Токен — это структура, содержащая собственно передаваемое значение и *метку* — указатель узла назначения. Простейшая потоковая вычислительная система состоит из двух устройств: исполнительного (execution unit) и устройства сопоставления (matching unit) [[11]](https://habr.com/ru/post/122479/#ref-11). Исполнительное устройство служит для выполнения инструкций и формирования токенов с результатами операций. Как правило, оно включает в себя память команд, доступную только для чтения. Готовность входных данных узла определяется по наличию набора токенов с одинаковыми метками.

1. Статическая Dataflow?

В ней каждый вычислительный узел представлен в единственном экземпляре, число узлов заранее известно, также заранее известно число токенов, циркулирующих в системе. Роль устройства сопоставления здесь выполняла память взаимодействий (activity store). В ней хранились пары токенов вместе с адресом узла назначения, флагами готовности и кодом операции. Любой вычислительный узел в этой архитектуре имел только два входа и состоял из одного оператора. При обнаружении готовности обоих операндов устройство выборки (fetch unit) считывало код операции, и данные отправлялись на обработку в исполнительное устройство (operation unit).

1. Динамическая Dataflow с помеченными токенами?

Системы с помеченными токенами (tagged-token architecture) в известной мере свободны от основного недостатка статической модели. В них число токенов, одновременно присутствующих на дуге, не ограничивается. Это открывает возможность выполнения итераций в произвольной последовательности, но требует учета принадлежности токенов к одной и той же итерации. С этой целью токен должен содержать информацию о вычислительном контексте, в котором он используется, например о номере итерации цикла. Этот контекст называют «цветом значения», а токен соответственно называют «окрашенным», в силу чего метод имеет еще одно называние - метод окрашенных токенов.

В модели с помеченными токенами структура токена сложнее, чем в статической модели: <v, <f, n, c, i>, a>, где c определяет фрагмент кода или тело цикла в составе реализуемой функции f, а i (индекс) представляет цвет токена. Каждая дуга потокового графа может рассматриваться как вместилище, способное содержать произвольное число токенов с различными тегами. Правило активирования вершины в модели с помеченными токенами имеет вид: вершина активируется, когда на всех ее входных дугах присутствуют токены с идентичным цветом.

Значительным шагом в архитектуре потоковых ВС стало изобретение механизма явной адресации токенов (explicit token-store), имеющего и другое название - непосредственное согласование (direct matching). В основе этого механизма лежит наблюдение: все токены в одной и той же итерации цикла и в одном и том же вхождении в реентерабельную процедуру имеют идентичный тег (цвет). При инициализации очередной итерации цикла или очередном обращении к процедуре формируется так называемый кадр токенов, содержащий токены, относящиеся к данной итерации или данному обращению, то есть токены с одинаковыми тегами. Использование конкретных ячеек внутри кадра задается на этапе компиляции. Каждому кадру выделяется отдельная область в специальной памяти кадров (frame memory), причем раздача памяти под каждый кадр происходит уже на этапе выполнения программы.

1. Динамическая Dataflow с явно адресуемыми токенами?

В схеме с явной адресацией токенов любое вычисление полностью описывается указателем команды (IP, Instruction Pointer) и указателем кадра (FP, Frame Pointer). Токен выглядит следующим образом: <v, <FP,IP>>.

Команды, реализующие потоковый граф, хранятся в памяти команд и имеют формат: орс•i•dis. Здесь i (индекс в памяти кадров) определяет положение ячейки с нужным токеном внутри кадра, то есть какое число нужно добавить к FP, чтобы получить адрес интересующего токена. Поле dis указывает на местоположение команды, которой должен быть передан результат обработки данного токена. Адрес в этом поле также задан в виде смещения - числа, которое следует прибавить к текущему значению IP, чтобы получить исполнительный адрес команды назначенияв памяти команд. Если потребителей токена несколько, в поле dis заносится несколько значений смещения.

Каждому слову в памяти кадров придан бит наличия, единичное значение которого удостоверяет, что в ячейке находится токен, ждущий согласования, то есть что одно из искомых значений операндов уже имеется. Как и в архитектуре с окрашенными токенами, определено, что вершины могут иметь максимум две входных дуги. Когда на входную дугу вершины поступает токен <v1, <FP,IP>>, в ячейке памяти кадров с адресом FP+(IP.i) проверяется бит наличия (здесь IP.i означает содержимое поля i в команде, хранящейся по адресу, указанному в IP). Если бит наличия сброшен (ни один из пары токенов еще не поступал), поле значения пришедшего токена (v1) заносится в анализируемую ячейку памяти кадров, а бит наличия в этой ячейке устанавливается в единицу, фиксируя факт, что первый токен из пары уже доступен.

Если токен <v2,<FR,IP>> приходит в вершину, для которой уже хранится значение v1, команда, представляющая данную вершину, может быть активирована и выполнена с операндами v1 и v2. В этот момент значение v1 извлекается из памяти кадров, бит наличия сбрасывается, и на функциональный блок, предназначенный для выполнения операции, передается пакет команды <v1,v2, FP, IP, IP.opc, IP.dis>, содержащий операнды (v1 и v2), код операции (IP.opc) и адресат ее результата (IP. dis). Входящие в этот пакет значения FP и IP нужны, чтобы вместе с IP.dis вычислить исполнительный адрес адресата. После выполнения операции функциональный блок пересылает результат в блок формирования токенов.

1. Гибридные Dataflow?

«Чистые» потоковые архитектуры, подобные описанным MIT Static Dataflow Machine и Manchester Dataflow Machine, к сожалению, имели много слабых мест:

Dataflow-машины давали огромные возможности для параллелизма выполнения. Обратной стороной этого преимущества было то, что на последовательных участках вычислительного графа они показывали резкое падение производительности.

Загрузка исполнительных устройств была далека от максимально возможной. Большая часть машинного времени тратилась на поиск соответствия операндов, выборку инструкций, а исполнительное устройство все это время простаивало, выполняя лишь по одной инструкции на каждую пару токенов.

Трудным было конструирование устройств сопоставления. Ассоциативная память сложнее, дороже, медленнее, занимает больше места и потребляет больше энергии, по сравнению с обычной оперативной памятью такого же объема.

Сам принцип управления потоком данных не позволял организовать эффективный конвейер. Почти все устройства работали асинхронно, требовались буферы и очереди в линиях связи.

По сравнению с классической многопроцессорной архитектурой, в dataflow-машинах значительно выше была нагрузка на коммутационную сеть. Ведь фактически, каждая операция требовала пересылки двух токенов.

В попытках решить перечисленные проблемы стали появляться гибридные архитектуры, сочетающие в себе элементы как архитектур потока данных, так и потока управления.

1. Реконфигурируемые системы? Преимущества и недостатки

Реконфигурируемые архитектуры являются некоторой золотой серединой для тех, кто не может или не готов платить за разработку специлизированного вычислительного элемента, но тем не менее требует очень высокая производительность и сложные конвейеры. В этом случае можно порекомендовать воспользоваться вычислительными элементами, позволяющими менять свою функцию в очень широких пределах. Зачем нужны реконфигурируемые вычисления?

Для повышения производительности (включая предсказуемость) и вычислительно-энергетической эффективности по сравнению с программной реализацией.

например, приложения обработки сигналов в виде конфигурируемого аппаратного обеспечения.

Для того, чтобы иметь мощные операции, специфичные для приложения.

Для улучшения гибкости продукта и стоимости/времени разработки по сравнению с аппаратным решением (ASIC)

например, шифрование, сжатие или обработка сетевых протоколов в конфигурируемом аппаратном обеспечении

Для использования того же самого аппаратного обеспечения в различных целях на разных стадиях вычислений (снижает стоимость). Преимущества устройств конфигурируемой логики

Нет жесткой неизменяемой настройки и возможна разработка новых приложений после производства

“Позднее связывание”

Экономия от масштаба (большая отдача от вложения, фиксированная стоимость разработки)

Время выпуска на рынок короче, чем для ASIC (возможна работа с изменяющимися требованиями и стандартами, новыми идеями)

Потенциальные недостатки:

Гораздо меньшая эффективность (площадь чипа, производительность, потребление) по сравнению с ASIC

Необходимость верификации корректности

(общее для всех аппаратных и программных решений)