**Словарь терминов и определений**

**Архитектура набора команд** ( instruction set architecture, **ISA**) — часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора.

На этом уровне определяются реализованные в микропроцессоре конкретного типа:

* архитектура памяти;
* взаимодействие с внешними устройствами ввода/вывода;
* режимы адресации;
* регистры;
* машинные команды;
* различные типы внутренних данных (например, с плавающей запятой, целочисленные типы и т. д.),;

обработчики прерываний и исключительных состояний.

**Микроархитектура -** описывает модель, топологию и реализацию ISA на микросхеме микропроцессора.

На этом уровне определяется:

- конструкция и взаимосвязь основных блоков ЦП,

- структура ядер, исполнительных устройств, АЛУ, а также их взаимодействия,

- блоков предсказания переходов,

- организация конвейеров,

- организация кэш-памяти,

- взаимодействие с внешними устройствами.

**Микроархитектура** описывает, как именно в процессоре расположены и соединены друг с другом регистры, АЛУ, конечные автоматы, блоки памяти и другие блоки, необходимые для реализации архитектуры процессора. У каждой архитектуры может быть много различных микроархитектур, обеспечивающих разное соотношение производительности, цены и сложности. Все они смогут выполнять одни и те же программы, но их внутреннее устройство может очень сильно отличаться.

**Компьютерная архитектура** определяется набором команд и архитектурным состоянием. Архитектурное состояние процессора RISC-V определяется содержимым счетчика команд (program counter,PC) и 32 видимых программисту регистров, поэтому любой процессор, реализующий архитектуру RISC-V, вне зависимости от его микроархитектуры обязан иметь счетчик команд и ровно 32 регистра. Зная текущее архитектурное состояние, процессор точно знает, какую операцию над какими данными надо выполнить для получения нового архитектурного состояния. У некоторых микроархитектур есть также и неархитектурное (т. е. невидимое программисту) состояние, которое используется или для упрощения логики, или для улучшения производительности.

**Однотактная микроархитектура** подразумевает выполнение команды процессора за один такт. Чаще всего Ее принцип работы легко объяснить, а устройство управления довольно простое. Из-за того, что все действия выполняются за один такт, эта микроархитектура не требует никакого неархитектурного состояния – нет, к примеру, внутренних регистров промежуточного хранения данных или команд которые нельзя изменить исполняемым программным кодом. При этом подходе все исполнительные узлы процессорного ядра (за исключением, конечно же его регистров, определенных или других элементов с памятью состояния) архитектурой представлены комбинационно-логическими схемами. Минимальная длительность такта при этом определяется временем выполнения самой медленной команды (например, временем распространения ). Кроме того, процессору требуются отдельные запоминающие устройства для команд и данных, что на практике, как правило, избыточно.

**Многотактная микроархитектура** выполняет команду за несколько более коротких тактов. Простым командам нужно меньше тактов, чем сложным. Вдобавок многотактная микроархитектура уменьшает количество необходимой аппаратуры путем повторного использования таких «дорогих» блоков, как сумматоры и блоки памяти. Например, при выполнении команды один и тот же сумматор на разных тактах может быть использован для разных целей. Повторное использование блоков достигается путем добавления в многотактный процессор нескольких не архитектурных регистров для записи в память промежуточных результатов. Многотактный процессор выполняет только одну команду за раз, и каждая команда занимает несколько тактов. Такой процессор обходится единственным запоминающим устройством, поскольку обращается к нему в одном цикле для получения команды, а в другом – для чтения или записи данных. Экономия на оборудовании стала определяющим фактором применения многотактных процессоров в недорогих системах.

**Конвейерная микроархитектура** – результат применения принципа конвейерной обработки к однотактной микроархитектуре. Вследствие этого она позволяет выполнять несколько команд одновременно, значительно улучшая пропускную способность процессора. Конвейерная микроархитектура требует дополнительной логики для разрешения конфликтов между одновременно выполняемыми в конвейере командами. Она также требует несколько не архитектурных регистров, расположенных между стадиями конвейера. Тем не менее эта дополнительная логика и регистры того стоят – в наши дни все коммерческие высокопроизводительные процессоры используют конвейеры. [Харрисы].

**Архитектура фон Неймана** (Принстонская архитектура) характеризуется использованием общей оперативной памяти для хранения программ, данных, а также для организации стека. Для обращения к этой памяти используется общая системная шина, по которой в процессор поступают и команды, и данные.

**Гарвардская архитектура** была разработана с целью увеличить скорость выполнения вычислительных операций и оптимизировать работу памяти. Она характеризуется физическим разделением памяти программ и памяти данных. Каждая память соединяется с процессором отдельной шиной, что позволяет одновременно с чтением-записью данных при выполнении текущей команды производить выборку и декодирование следующей команды.

**Модифицированная гарвардская архитектура** используется общая шина данных и шина адреса для всех внешних данных, а внутри процессора применяется шина данных, шина команд и две шины адреса.

**Super Harvard Architecture (SHARC) — расширенная гарвардская архитектура** инструкции выбираются из кэш-памяти обе шины остаются свободными и появляется возможность передать два операнда одновременно.

**CISC (Complete Instruction Set Computing)** - тип архитектуры процессора с полным набором команд. При этом подходе выполнение любой сколь угодно сложной команды из системы команд процессора реализовывается аппаратно внутри самого процессора.

**RISC (Restricted (Reduced) Instruction Set Computer** - компьютер с сокращенным набором команд) - архитектура процессора, в которой быстродействие увеличивается за счет упрощения инструкций, чтобы их декодирование было более простым, а время выполнения - короче. В процессорах с RISC-архитектурой используется ограниченный набор быстрых команд. Каждая команда RISC-процессора должна выполняться за один машинный такт.

**MISC-архитектура (Minimal Instruction Set Computer)** является процессором с минимальным набором команд. Она отличается ещё большей простотой и используется для ещё большего снижения энергопотребления и итоговой стоимости процессора. MISC-архитектура применяется в IoT-сегменте и компьютерах малой стоимости вроде роутеров.

**Hart (Аппаратный поток)** — под потоком (hart) подразумевается аппаратный поток – выполнение которого (или точнее «хранение архитектурного состояния») обеспечивается дублирующими регистрами (теневыми регистрами).

**Trap (Ловушка)** — ловушка это совокупное объедение смысла таких слов, как прерывание и исключение.

**Ловушка исключения (exception)** — это понятие означает синхронное событие, которое прерывает исполнения кода (наверное, самое банальное – попытка деления на ноль). Исключение может прерываться другим исключением, или NMI.

**Ловушка прерывания (interrupt)** — внешнее асинхронное событие, которое может привести к тому, что поток неожиданно может передать управление. Прерывание может прерываться другим прерыванием, NMI, или исключением.

**Ловушка немаскируемого прерывания(NMI)** — немаскируемое прерывание. NMI не может прерываться другим NMI, но может перейти из обработчика NMI в режим обработки исключения, если в момент обработки NMI произойдет исключение.

**IP-ядра** (англ. IP cores), IP-блоки (IP — англ. intellectual property), СФ-блоки (СФ — сложные функциональные), VC (англ. virtual components — виртуальные компоненты) — готовые блоки для проектирования микросхем (например, для построения систем на кристалле).

Различают три основных класса блоков:

- программные IP-блоки (англ. soft blocks) — блоки, специфицированные на языке описания аппаратуры;

- схемотехнические блоки (англ. firm blocks) — блоки, специфицированные на схемотехническом уровне, без привязки к конкретной топологической реализации;

- физические (топологические) блоки (англ. hard blocks) — блоки, специфицированные на физическом уровне реализации СБИС (например, GDSII для ASIC).

Hard IP-Core — сложнофункциональный блок, передаваемый потребителю в виде законченной схемотехнической конструкции, разработанной на основе базы данных и оптимизированной по размерам, потребляемой мощности и электрическим характеристикам.

В FPGA (ПЛИС) под Hard IP-Core понимают специализированные области кристалла, выделенные для определённых функций. В этих областях реализованы блоки неизменной структуры, спроектированные по методологии ASIC (как области типа БМК или схемы со стандартными ячейками), оптимизированные для заданной функции и не имеющие средств её программирования[1]. В случае использования данного вида ядер размер площади, используемой на кристалле, сокращается, улучшаются характеристики быстродействия, но происходит потеря универсальности.

**Soft-микропроцессор (софт-процессор**, **также «микропроцессор с программным ядром»**) — микропроцессорное ядро, которое может быть полностью создано с использованием только логического синтеза. Реализуется с использованием различных полупроводниковых устройств, содержащих программируемую логику (ПЛИС — например, ASIC, FPGA, CPLD). Большинство систем, если они используют soft-процессор, использует единственный soft-процессор. Однако максимальное число ядер фактически ограничено лишь ресурсами конкретной ПЛИС, что может быть полезным в специфических проектах.

**ABI**. (Application Binary Interface) - стандарт, который определяет взаимосвязь между низкоуровневыми концепциями выполнения кода и высокоуровневыми языками программирования, а также возможности машинного кода определенных аппаратных / операционных платформ. Сюда относятся такие вещи:

Как типы данных языков C, C++, Fortran и т. д. лежат в памяти (размеры данных, выравнивание, endianness).

Как работают вложенные вызовы функций - где и как сохраняется информация о том, куда нужно сделать возврат в вызывающую функцию, где сохраняются передаваемые в функцию переменные (в каких регистрах CPU и/или ячейках памяти), как передаются возвращаемые из функции значения.

Как работает первоначальная инициализация программы (код startup), какой должен быть формат исполняемого кода (executable), как и где хранится код и данные в памяти, как работает загружаемый код DLL, и т. п.

Таким образом, ABI специфичен для определенного языка (следовательно, есть C ABI, C++ ABI, Fortran ABI, Pascal ABI, ..., даже спецификация байт-кода Java, хотя и нацелена на некий "виртуальный" процессор вместо реального, это тоже ABI), а также для определенной операционной системы (MS Windows и Linux на одном и том же оборудовании используют разный ABI) и аппаратных особенностей CPU (API для процессоров ARM и x86 отличаются друг от друга). [https://microsin.net/programming/arm/what-is-abi-and-eabi.html]