Вы наверняка знаете, что мир процессоров разбит на два лагеря. Это ARM и x86. Приоритет процессоров x86 — максимальная производительность, а у ARM — высокая энергоэффективность. Совместить оба этих свойства в одном виде процессоров пока не получается. Поэтому первые чаще всего используют в настольных ПК, а вторые — в мобильных устройствах.

Также важно помнить, что эти два типа процессоров сильно отличаются, поэтому одно и то же ПО не сможет работать с обоими одновременно. Каждая программа должна быть разработана под конкретный тип процессора. А теперь рассмотрим каждый тип подробнее.

**x86**

Этот тип архитектуры был создан в 1978 году и относится к разновидности CISC (Complex Instruction Set Computing). Если упрощенно, он предполагает, что в процессоре есть инструкции для большинства задач. Он применяется в настольных ПК, ноутбуках и других устройствах, которым нужна максимальная производительность без экономии на питании.

Так как история развития этих процессоров дольше, чем у ARM, они имеют больший набор команд. Это делает их очень сложными и продвинутыми, позволяющими выполнять множество сложных вычислений за короткое время. В случае с ARM-процессором, командный ассортимент пока более скудный, но разрыв постепенно сокращается. Причина — уменьшение техпроцесса и развитие производственных технологий. К минусам же, кроме повышенных энергопотребления и тепловыделения, можно отнести сложность и запутанность команд из-за долгой истории развития.

**ARM**

ARM был создан в 1985 году британской компанией Acorn и принадлежит к типу RISC (Reduced Instruction Set Computing). Здесь другой подход — процессор содержит минимальное количество необходимых для работы команд. За счет этого инструкции здесь проще и меньше.

Перед разработчиками ARM стояла цель создать архитектуру, свободную от недостатков x86. Можно точно сказать, что им удалось сделать крайне эффективную и недорогую архитектуру. Популярности ARM-процессоров на рынке мобильных устройств способствовало не только это. Вот ряд дополнительных преимуществ:

1. они дешевле в производстве и развертывании
2. архитектура ARM позволяет крупным поставщикам создать свои собственные решения на ARM-архитектуре для разных ниш

Кроме того, эффективное потребление энергии и отсутствие перегрева делают такие процессоры оптимальным выбором для серверных задач, а также для использования в маршрутизаторах и высокопроизводительных решениях хранения данных.

Если же говорить о недостатках, то они заключаются в следующем — из-за минимизации размеров и фокуса на автономности, данные обрабатываются менее производительно, чем x86. Кроме того, если какая-то существующая часть ИТ-инфраструктуры построена на решениях от x86, следует учитывать это при выборе ARM. Одно и то же ПО на обоих решениях не запустится, для одного из них его точно придется адаптировать.

**CISC и RISC**

Разработка CISC-процессоров началась в конце 70-х годов, когда память была очень дорогой. Компиляторы тоже были плохие, а люди писали на ассемблере.

Так как память была дорогой, люди искали способ минимизировать использование памяти. Одно из таких решений — использовать сложные инструкции процессора, которые делают много действий.

Это также помогло программистам на ассемблере, так как они смогли писать более простые программы, ведь всегда найдется инструкция, которая выполняет то, что нужно.

Через некоторое время это стало сложным. Проектирование декодеров для таких команд стало существенной проблемой. Изначально ее решили с помощью микрокода.

В программировании повторяющийся код выносится в отдельные подпрограммы (функции), которые можно вызывать множество раз.

Идея микрокода очень близка к этому. Для каждой инструкции из набора создается подпрограмма, которая состоит из простых инструкций.

Таким образом, процессор содержит небольшой набор простых инструкций. На их основе можно создать множество сложных инструкций из набора команд с помощью добавления подпрограмм в микрокод.

Если говорить более подробно, то *для внутреннего языка типичны следующие черты:*

1. большое число различных машинных команд,
2. реализация некоторыми командами сложных преобразований;
3. разнообразие способов адресации операндов;
4. множество форматов команд различной разрядности;
5. наличие команд, в которых обработка совмещается с обращением к памяти.

*Программистская структура характеризуется следующей особенностью.*

* В состав процессора входит сравнительно небольшое число регистров общего назначения.

**Достоинства CISC-процессоров:**

* возможность создания более коротких программ за счет большой «вычислительной мощности» отдельных команд;
* меньший (по сравнению с RISC-процессорами) семантический разрыв между внутренним языком машины и языком программирования высокого уровня.

**Недостатки CISC-процессоров:**

* усложнение структуры процессора, обусловленное достаточно сложным внутренним языком;
* необходимость использования устройства управления с программируемой логикой, характеризуемого значительными аппаратурными затратами;
* сложные компиляторы, преобразующие программы с языка ассемблера в машинные коды из-за сложности внутреннего языка машины;
* значительное время машинного такта, из-за сложности выполняемых микроопераций;
* частое обращение к ЗУ, обусловленное небольшим числом регистров общего назначения;
* сложность реализации эффективного конвейера команд.

Шло время. Оперативная память стала дешеветь, компиляторы стали лучше, а большинство разработчиков перестало писать на ассемблере.

Эти технологические изменения спровоцировали появление философии RISC.

Сперва люди анализировали программы и заметили, что некоторые сложные инструкций CISC не используются большинством программистов.

Разработчики компиляторов затруднялись в выборе правильной сложной инструкции. Вместо этого они предпочли использовать комбинацию нескольких простых инструкций для решения проблемы.

Идея RISC заключается в замене сложных инструкций на комбинацию простых. Так не придется заниматься сложной отладкой микрокода.

Одним из аргументов RISC было то, что люди перестали писать ассемблерный код, поэтому необходимо создать набор инструкций, удобный для компиляторов. Архитектура RISC оптимизирована для компиляторов, но не для людей.

Опять же, если говорить более подробно, то *для внутреннего языка типичны следующие черты:*

* малое число различных команд (<=128);
* реализация командами только простых преобразований, выполнение всех (75%) команд за один цикл (такт);
* стандартная однословная длина всех команд, равная естественной длине слова и ширине шины данных;
* малое количество форматов команд (<=4);
* малое число способов адресации (<=4);
* доступ к памяти осуществляется только командами «Чтение» и «Запись»; все команды, кроме «Чтение» и «Запись», имеют тип «регистр-регистр».

*Программистская структура характеризуется следующими особенностями.*

* В состав процессора входит достаточно большой регистровый файл, содержащий не менее 32 РОН.
* Регистровый файл организован в виде частично перекрывающих друг друга «окон» регистров.

**Достоинства RISC-процессоров:**

* организация большого регистрового файла в виде перекрывающих друг друга «окон» регистров позволяет при вызовах подпрограмм осуществлять быстрое переключение процессора на подпрограмму (и возвращаться к вызвавшей программе) без сохранения внутренних регистров процессора в стеке, а для передачи параметров использовать общие области смежных перекрывающихся окон.
* сокращение числа различных команд, форматов команд и данных, видов адресации позволяет разместить на кристалле процессора простое устройство управления с «жесткой» логикой, занимающее малую площадь кристалла (примерно 6%, обычно около 50%);
* Благодаря простому внутреннему языку процессора возможно создание эффективных оптимизирующих компиляторов, преобразующих программу с языка высокого уровня в машинные команды процессора, при этом часть вычислений, предусмотренных в программе на языке высокого уровня выполняется компилятором при преобразовании программы до запуска ее на выполнение.

**Недостатки RISC-процессоров:**

* неразвитость внутреннего языка процессора, приводящая к более длинным программам, по сравнению с программами для CISC-процессоров, прежде всего, из-за малой «вычислительной мощности» команд;
* увеличение семантического разрыва между простым внутренним языком процессора и языками высокого уровня. Это приводит к усложнению разработки трансляторов, однако простота внутреннего языка «смягчает» недостаток;
* необходимость программной реализации сложных операций (как в первых микропроцессорах).

**ARM1 и Intel 386**

Для примера вот два процессора одного поколения. ARM1 и Intel 386. При схожей производительности ARM почти вдвое меньше по площади. А транзисторов на нем в 11 раз меньше: 25 тысяч против 275 тысяч. Энергопотребление тоже отличается на порядок: 0.1 Ватт против 2 Ватт у Intel.

Поэтому наши смартфоны, которые работают на ARM процессорах с архитектурой RISC, долго живут, не требуют активного охлаждения и такие быстрые.

**Что сейчас?**

Допустим мы решили, что архитектура ARM более эффективная и универсальная, но это не значит, что x86 похоронен.

На самом деле, в Intel и AMD не дураки сидят. И сейчас под капотом современные CISC-процессоры очень похожи на RISC. Постепенно разработчики CISC-процессоров все-таки пришли к этому и начали делать гибридные процессоры. Уже достаточно давно процессоры Intel и AMD разбивают входные инструкции на более мелкие микроинструкции. Получается, что разница между RISC и CISC процессорами уже сейчас минимальна.

**Итог**

В результате мы видим две крайности: x86 являются мощными решениями, обвешанными инструкциями, которые могут выполнять абсолютно любые задачи с хорошей скоростью. Но за это приходится платить увеличенным тепловыделением. ARM же — простые процессоры, у которых набор инструкций ощутимо меньше, поэтому выполнение многих серьезных задач на них не имеет особого смысла из-за медлительности процесса. Но при этом и тепловыделение низкое.

Таким образом, позволим себе сделать осторожный прогноз: похоже, все идет к победе ARM-архитектуры, но утверждать это с вероятностью в 100 % пока нельзя. В любом случае, на сегодняшний день ARM точно не выглядит универсальным решением для любых задач.