Микроконтроллеры

КУРС ЛЕКЦИЙ

ЧУ ПО «СОЦИАЛЬНО-ТЕХНОЛОГИЧЕСКИЙ КОЛЛЕДЖ»

ПРЕПОДАВАТЕЛЬ: **БОРИСОВ АЛЕКСЕЙ АЛЬБЕРТОВИЧ**

RAZUMDOM

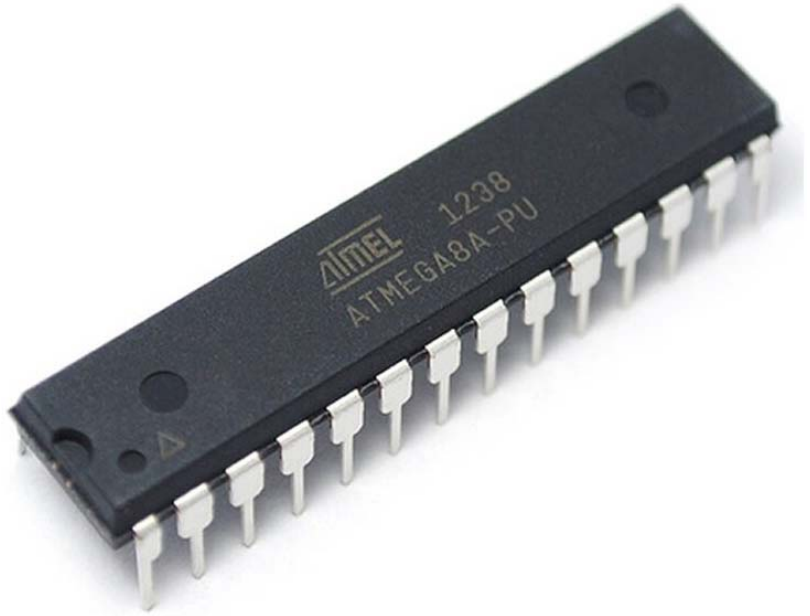
СОЦИАЛЬНО ТЕХНОЛОГИЧЕСКИЙ КОЛЛЕДЖ

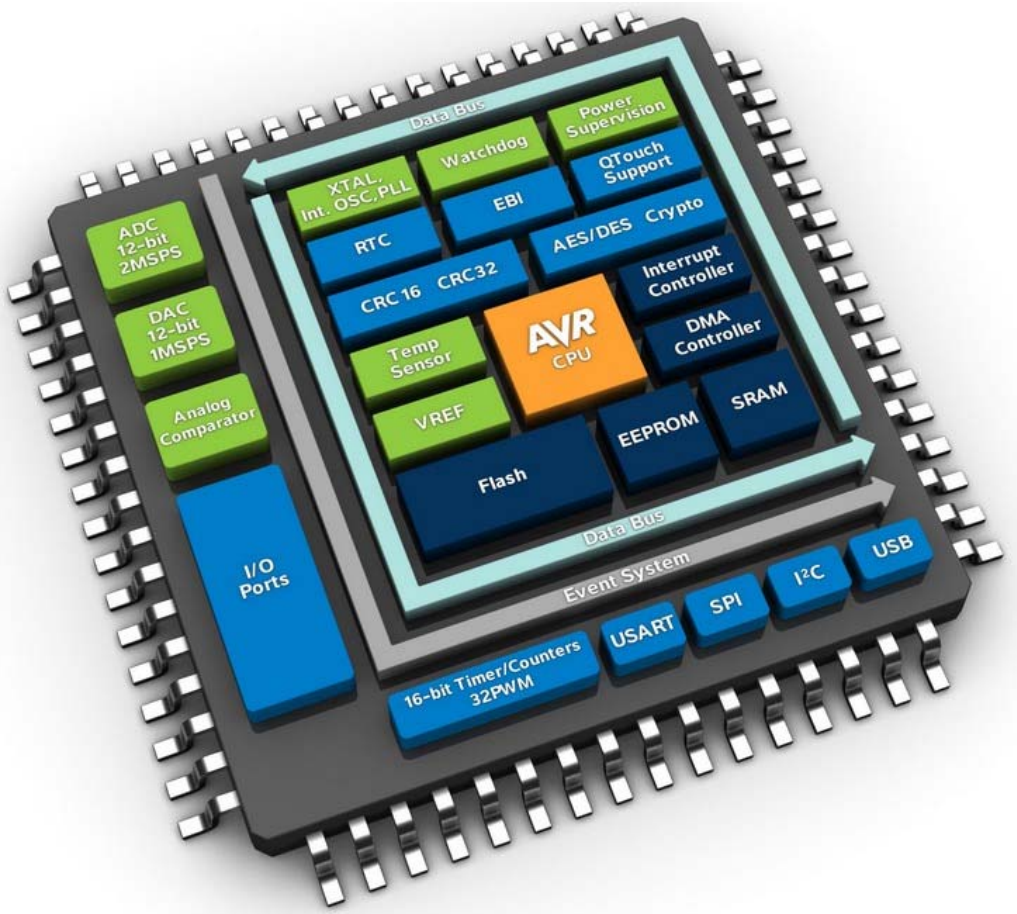
Программа занятий

1. Основные понятия и термины
2. Отличия микроконтроллеров
3. Семейства микроконтроллеров
4. Программирование
5. Классификация
6. Разрядность
7. Система команд
8. Конвейеризация: инновация RISC
9. Архитектура памяти
10. Регистры
11. Архитектура ARM
12. Производители микроконтроллеров

Понятие Микроконтроллер

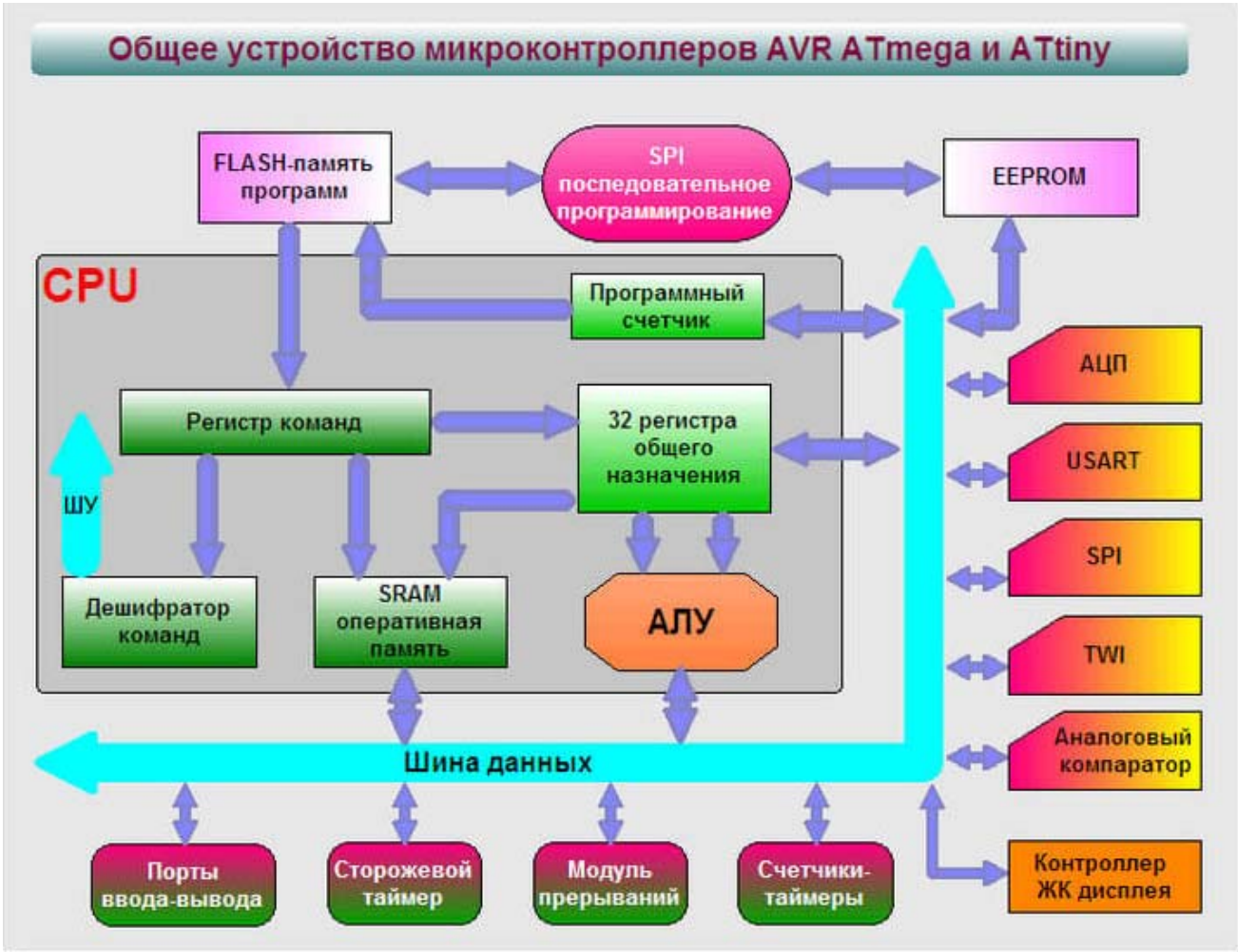
Микроконтро́ллер (англ. Micro Controller Unit, MCU) — микросхема для программного управления электронными устройствами. Обычно изготавливается в виде единого кристалла с функциями ядра микропроцессора, шин команд и данных, периферийных устройств, ОЗУ и ПЗУ. По сути, это компьютер в чипе, выполняющий роль периферийного процессора.





Термин микроконтроллер

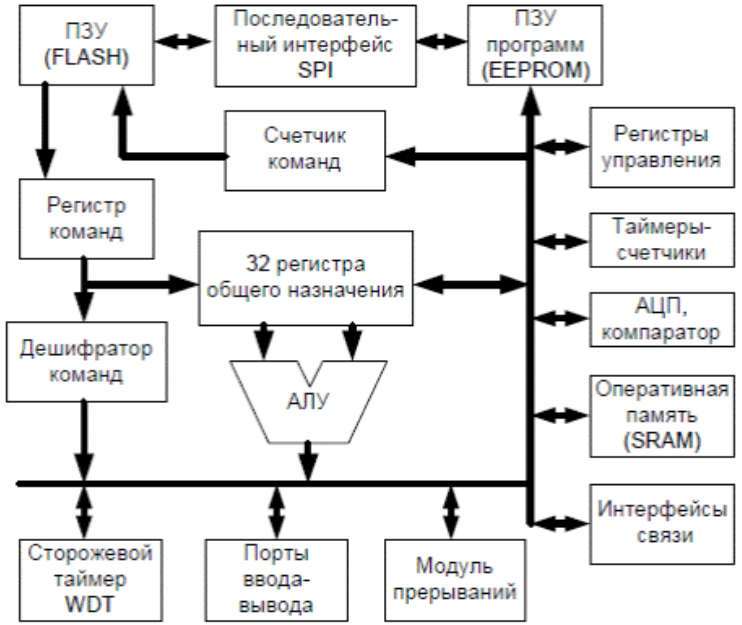
С появлением ***однокристальных микроЭВМ*** началась эра массового применения компьютерной автоматизации в области управления. Это обстоятельство и определило термин ***«контроллер»*** (англ. controller — регулятор, управляющее устройство).
Термин **«микроконтроллер» (МК)** вытеснил из употребления ранее использовавшийся термин **«однокристальная микроЭВМ»**.
Первый патент на однокристальную микроЭВМ был выдан в 1971 году инженерам Майклу Кокрэну и Гэри Буну, сотрудникам американской Texas Instruments. Именно они предложили на одном кристалле разместить не только процессор, но и память с устройствами ввода-вывода.



Отличие микроконтроллеров от микропроцессоров

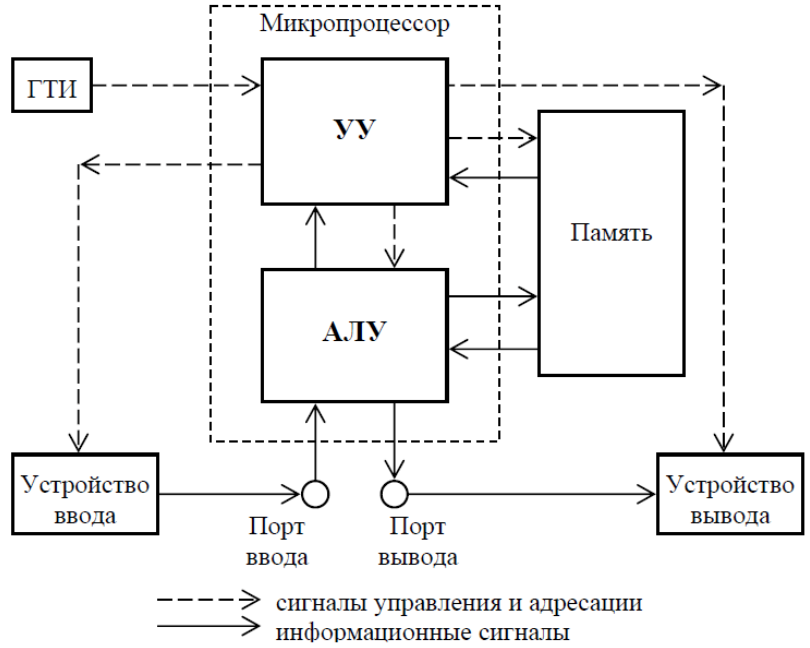
Микроконтроллер

На одном кристалле расположено как вычислительное устройство, так и ПЗУ, ОЗУ, порты ввода/вывода, таймеры, АЦП, последовательные и параллельные интерфейсы. Практически все необходимые элементы.
Но вычислительной мощности у МК чаще всего очень мало. Её хватает только на процесс управления каким либо устройством.



Микропроцессор

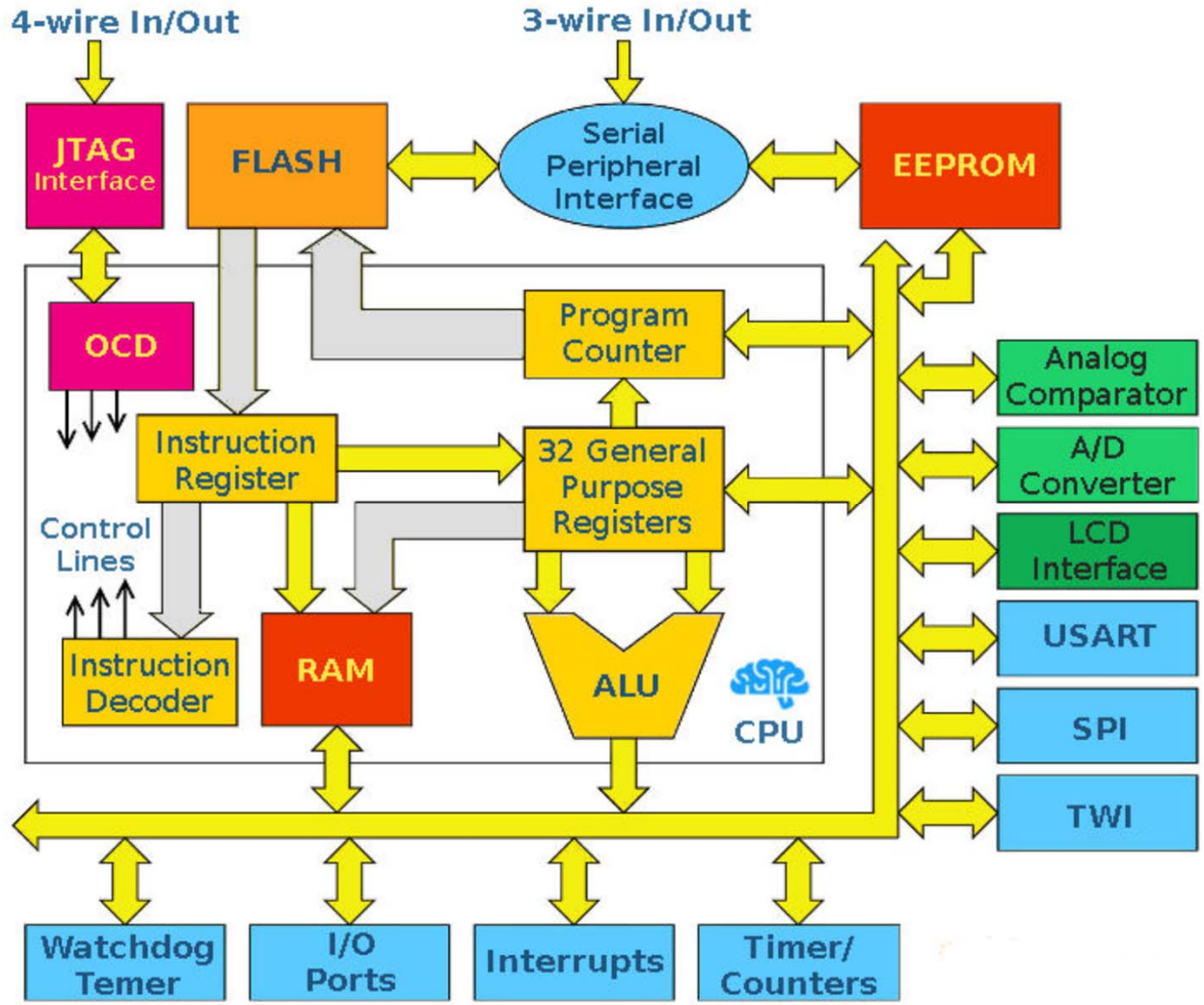
Содержит в себе арифметико-логическое устройство, блок синхронизации и управления, запоминающие устройство, регистры и шину. То есть МП содержит в себе только то, что непосредственно понадобится для выполнения арифметический и логических операций. Все остальные комплектующие (ОЗУ, ПЗУ, устройства ввода/вывода, интерфейсы) нужно подключать извне.



Список периферийных устройств

Список периферийных устройств микроконтроллеров:

- **универсальные цифровые порты**, которые можно настраивать как на ввод, так и на вывод;
- различные интерфейсы ввода-вывода, такие, **как UART, I²C, SPI, CAN, USB, IEEE 1394, Ethernet;**
- **аналого-цифровые и цифро-аналоговые преобразователи;**
- **компараторы;**
- широтно-импульсные модуляторы (**ШИМ-контроллер**);
- **таймеры;**
- контроллеры бесколлекторных двигателей, в том числе шаговых;
- контроллеры дисплеев и клавиатур;
- радиочастотные приемники и передатчики;
- массивы встроенной флеш-памяти;
- встроенные тактовый генератор и сторожевой таймер;



Семейства микроконтроллеров

Семейства микроконтроллеров включает:

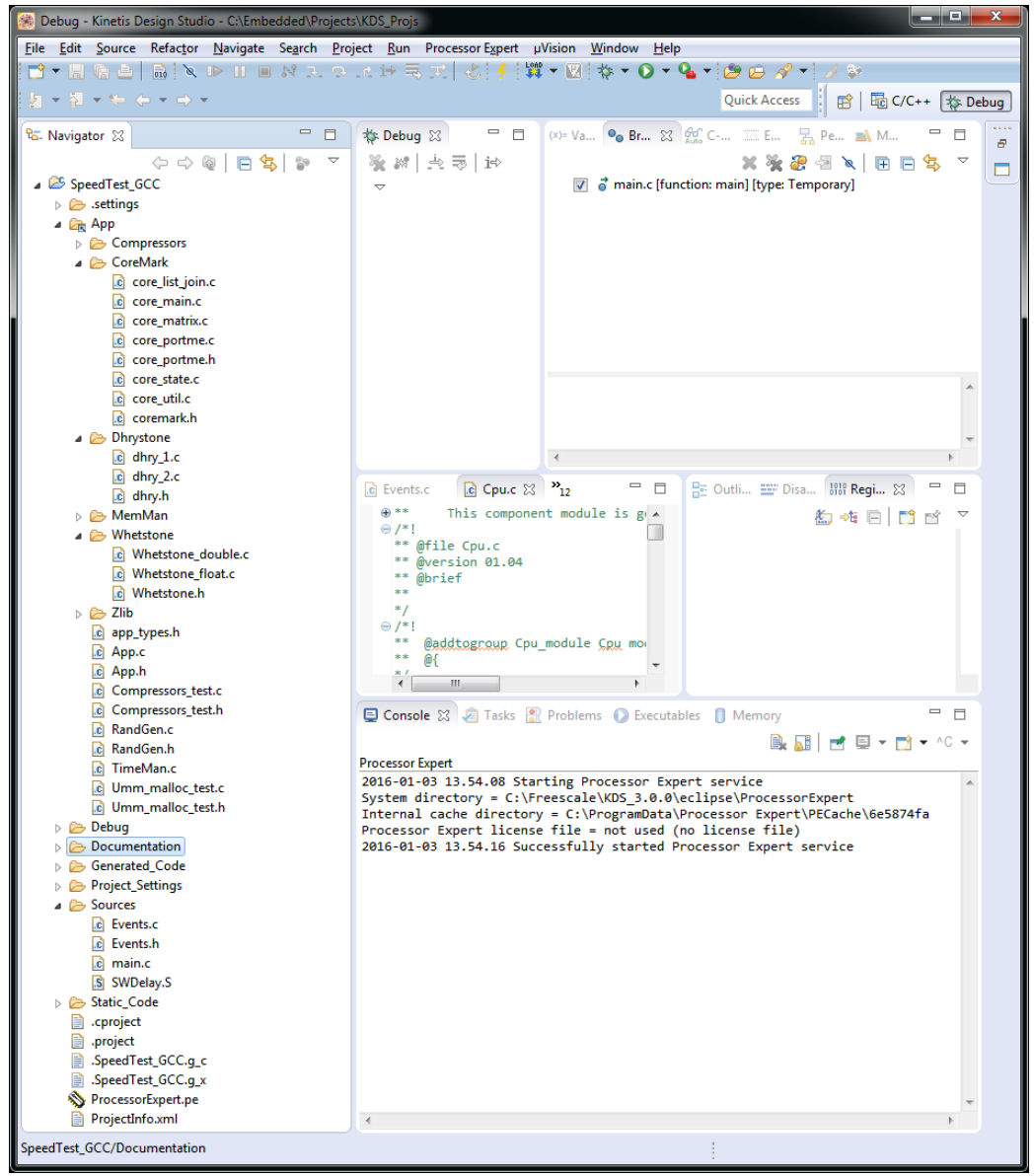
- **ARM** (ARM Limited)
  - ST Microelectronics STM32 ARM-based MCUs
  - ARM Cortex, ARM7 и ARM9-based MCUs
  - Texas Instruments Stellaris MCUs
  - NXP ARM-based LPC MCUs
  - Toshiba ARM-based MCUs
  - Analog Devices ARM7-based MCUs
  - Cirrus Logic ARM7-based MCUs
  - Freescale Semiconductor ARM9-based MCUs
  - Silicon Labs EFM32 ARM-based MCUs
- **AVR** (Atmel)
  - ATmega
  - ATtiny
  - XMega
- **MCS 51** (Intel)
- **C8051**F34x
- **MSP430** (TI)
- **PIC** (Microchip)
- **STM8** (STMicroelectronics)
- **RL78** (Renesas Electronics)
- **Espressif**
  - ESP8266
  - ESP32
- **RISC-V** (открытая расширяемая система команд)
  - Espressif ESP32-C3 ... ESP32-P4
  - SiFive FE310
  - WCH CH32V003 ... CH32V307
  - Миландр Счётчик К1986ВК025
  - Микрон/Прогресс АМУР К1948ВК018
  - НИИЭТ линейка чипов с К1921ВГ015

Программирование микропроцессоров

Программирование микроконтроллеров осуществляется на языке ассемблера или Си.

Компиляторы Си для МК:

- **GNU Compiler Collection** — поддерживает ARM, AVR, MSP430 и многие другие архитектуры
- ECLIPSE
- **Small Device C Compiler** — поддерживает множество архитектур
- **CodeVisionAVR** (для AVR)
- **IAR** (для любых МК)
- **WinAVR** (для AVR и AVR32)
- **Keil** (для архитектуры 8051 и ARM)
- **HiTECH** (для архитектуры 8051 и PIC от Microchip)

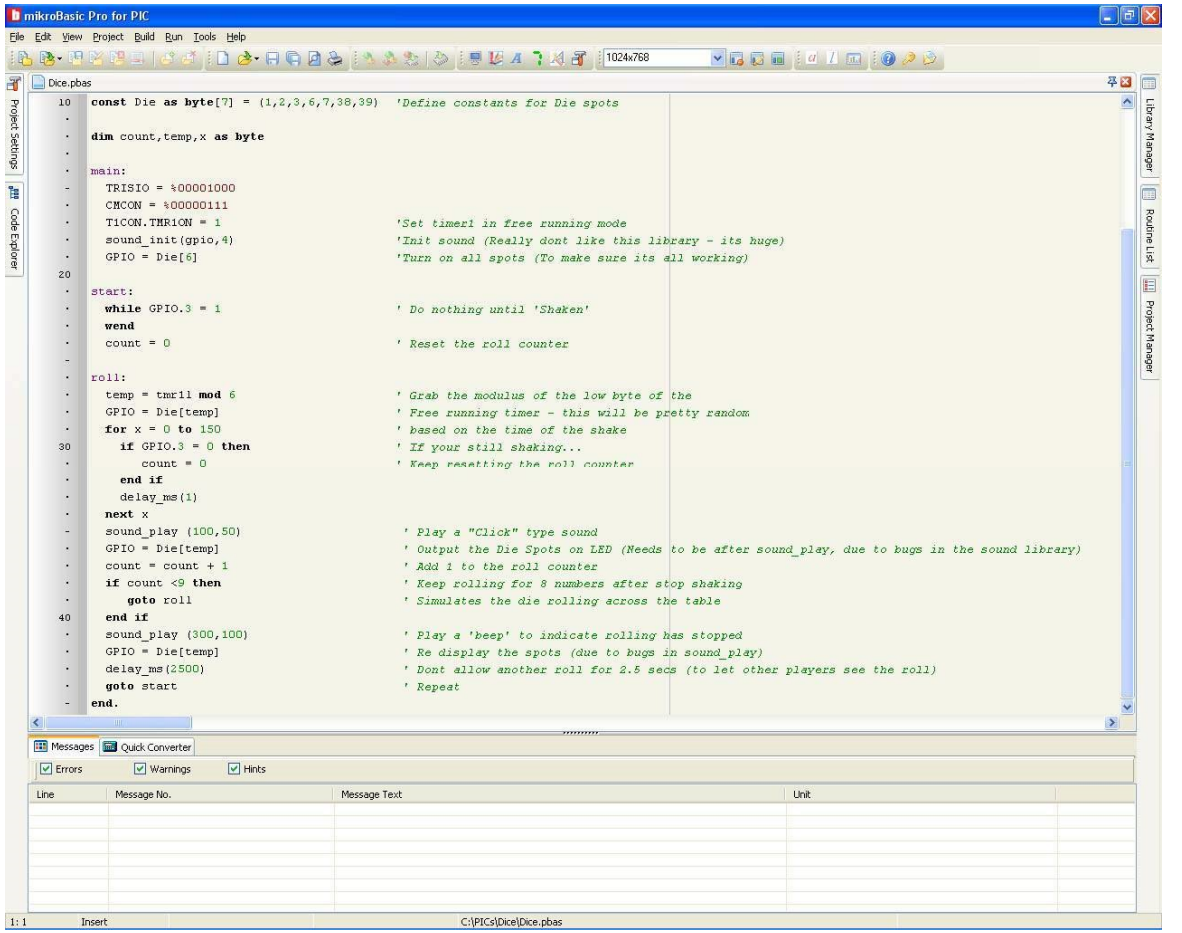


Программирование микропроцессоров

Существуют компиляторы для языков Форта и Бейсика. Используются также встроенные интерпретаторы Бейсика.

Компиляторы бейсика для МК:

- MikroBasic (архитектуры PIC, AVR, 8051 и ARM)
- Bascom (архитектуры AVR и 8051)
- FastAVR (для архитектуры AVR)
- PICBasic (для архитектуры PIC)
- Swordfish (для архитектуры PIC)



Для отладки программ используются:

- **программные симуляторы** (специальные программы для персональных компьютеров, имитирующие работу микроконтроллера),
- **внутрисхемные эмуляторы** (электронные устройства, имитирующие микроконтроллер, которые можно подключить вместо него к разрабатываемому встроенному устройству),
- **отладочный интерфейс**, например, JTAG.

Различия микроконтроллеров

Микроконтроллеры можно разделить по таким критериям:

- **разрядность** – это длина одного слова, обрабатываемого микроконтроллером;
- **система команд**;
- **архитектура памяти**.

По ***разрядности*** микроконтроллеры можно разделить:

- **8-битные**;
- **16-битные**;
- **32-битные**;
- **64-битные**.

По ***типу системы команд***.

- **RISC архитектура - Reduced Instruction Set Computer** или сокращенная система команд.
- **CISC архитектура - Complex Instruction Set Computer**, или полная система команд.

По ***архитектурт памяти*** можно разделить:

- **Архитектура фон Неймана** – используется общая область памяти команд и памяти данных
- **Гарвардская архитектура** – используется раздельная область памяти команд и памяти данных

Разрядность микроконтроллеров

Разрядность микроконтроллеров

По ***разрядности*** микроконтроллеры можно разделить:

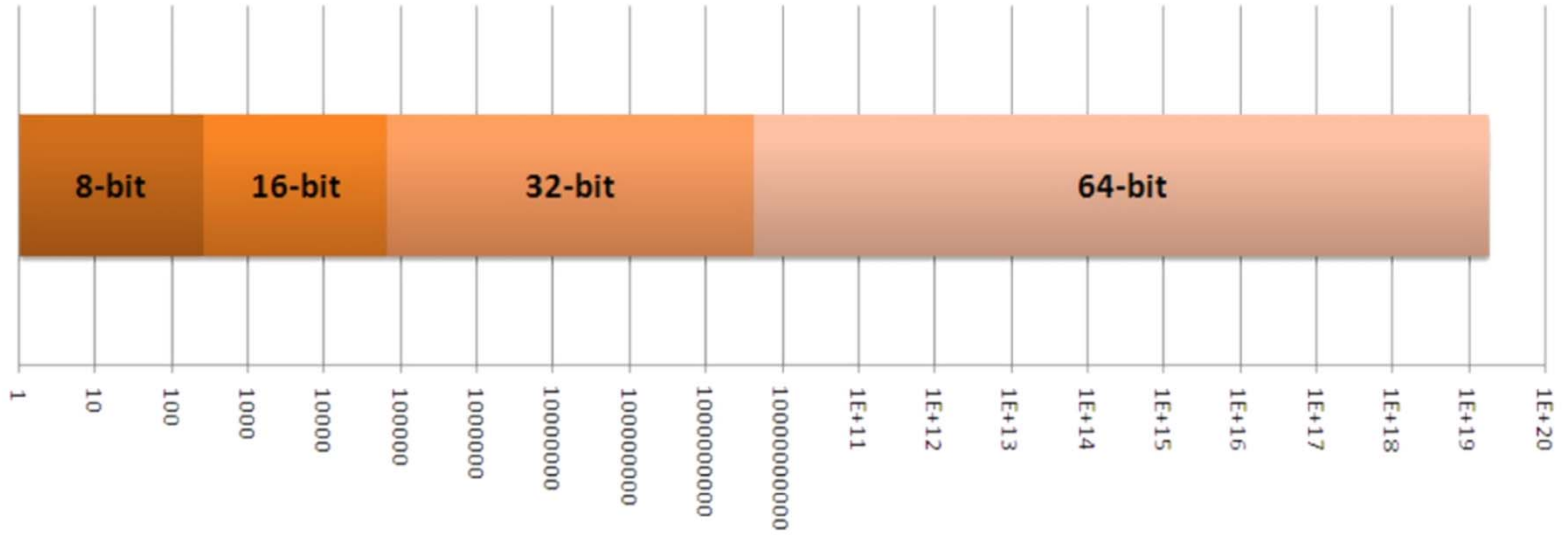
- **8-битные**;
- **16-битные**;
- **32-битные**;
- **64-битные**.

Первый микропроцессор i4004 был 4х разрядный и использовался в программируемом калькуляторе. С тех пор разрядность микропроцессора только увеличивалась.
Разрядность это количество бит, которые может одновременно передавать шина данных. В микроконтроллере, как и в микропроцессоре используется несколько шин:
Шина команд,
Шина данных,
Шина адреса для команд или данных.
Эти шины могут быть разной битности,

Разрядность микроконтроллеров

Разрядность это количество бит, которые может одновременно передавать шина.

- Существуют 8-битные и 32-битные микроконтроллеры PIC фирмы Microchip Technology,
- 8-битные микроконтроллеры AVR фирмы Atmel (с 2016 года производятся фирмой Microchip),
- 16-битные MSP430 фирмы TI, а также
- 32-битные микроконтроллеры архитектуры ARM, которую разрабатывает фирма ARM Limited и продаёт лицензии другим фирмам для их производства.



Количество битов	Минимальное значение	Максимальное значение
8	0	255 ($2^8 - 1$)
16	0	65 535 ($2^{16} - 1$)
32	0	4 294 967 295 ($2^{32} - 1$)
64	0	18 446 744 073 709 551 615 ($2^{64} - 1$)

Система команд микроконтроллеров

RISC
CISC

Системы команд микроконтроллеров

Классификация микроконтроллеров по ***типу системы команд***.
1. **RISC-архитектура**, или сокращенная система команд. Ориентирована на быстрое выполнение базовых команд за один, реже – два машинных цикла, а также имеет большое количество универсальных регистров и более длинный способ доступа к постоянной памяти.
2. **CISC-архитектура**, или полная система команд. Характерна прямая работа с памятью, имеет большее число команд, малое количество регистров (ориентирована на работу с памятью), длительность команд – от одного до четырех машинных циклов. Пример – процессоры Intel и AMD.

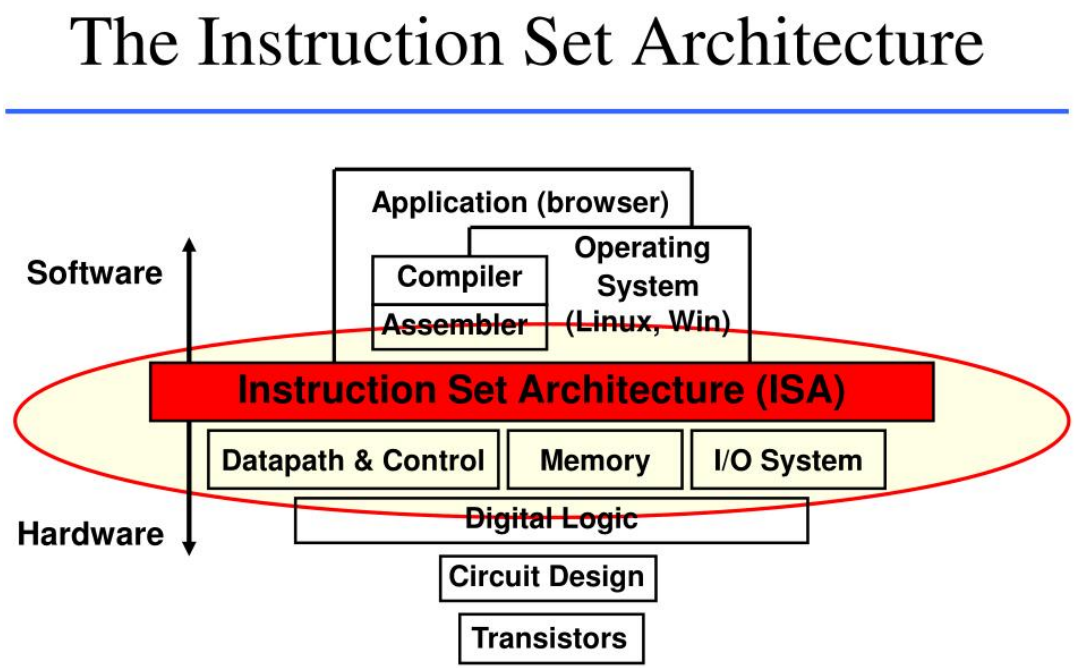
CISC-архитектура	**RISC-архитектура**
1. Многобайтовые команды	1. Однобайтовые команды
2. Малое количество регистров	2. Большое количество регистров
3. Сложные команды	3. Простые команды
4. Одна команда за один цикл	4. Несколько команд за один цикл
5. Одно исполнительное устройство	5. Несколько исполнительных устройств

Что такое архитектура набора команд (ISA)?

Количество инструкций, которые понимает процессор, ограничено. Существует фиксированное количество команд, которые понимает процессор. В мире представлено множество различных микропроцессоров, и они не используют одинаковый набор команд. Иными словами, они интерпретируют числа в инструкции по-разному. Одна архитектура микропроцессора трактует число 501012 как add r10, r12, а другая архитектура — как load r10, r12. Комбинация инструкций, которые понимает процессор, и регистров, которые ему доступны, называется архитектурой набора команды **(Instruction Set Architecture, ISA).**

Микропроцессоры, например, Intel и AMD, используют архитектуру набора команд x86. А микропроцессоры, например, A12, A13, A14 от Apple, понимают набор команд ARM. В этот список ARM-процессоров можно включить M1.

Набор команд x86 и ARM не является взаимозаменяемым. Программа компилируется под определенный набор команд, если, конечно, это не JavaScript, Java, C# или что-то подобное. В этом случае программа компилируется в байт-код, который похож на набор команд для несуществующего процессора. Для запуска такого кода требуется Just-In-Time компилятор или интерпретатор, который транслирует байт-код в инструкции, понятные для микропроцессора в вашем компьютере.



Почему произошел переход на совершенно другой набор команд?

Зачем использовать новый набор команд? Одна из конкурентов Intel компания Apple, при создании своего микропроцессора не могла использовать набор команд x86 в микропроцессорах Apple Silicon.

• Архитектура набора команд сильно влияет на архитектуру процессора.

Использование определенной архитектуры набора команд может усложнить или упростить задачу по созданию высокопроизводительного или энергоэффективного процессора.

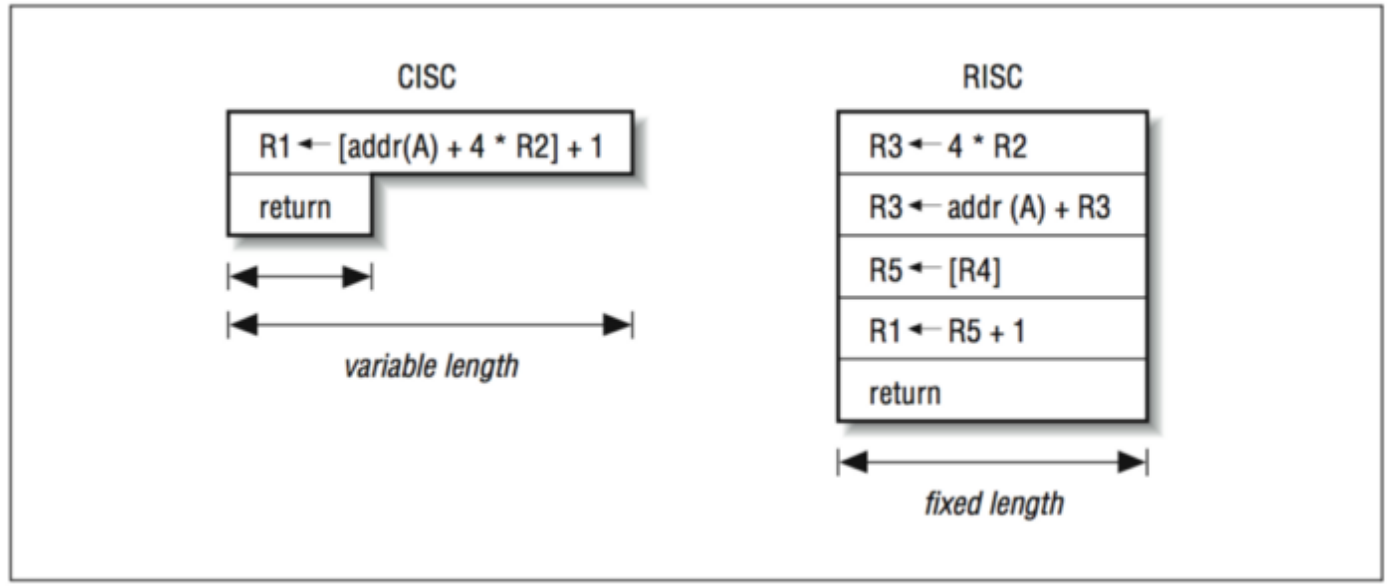
• Второй важный момент заключается в лицензировании. Apple не может свободно создавать свои процессоры с набором команд x86. Это часть интеллектуальной собственности Intel, которой не нужны конкуренты. Для сравнения, компания ARM не производит собственных микропроцессоров. Они занимаются проектированием архитектуры набора команд и предоставляют эталонные образцы микропроцессоров, которые ее реализуют.

Производители вмещают все на одной большой интегральной схеме, то есть на том, что называется системой на кристалле (System-On-a-Chip, SoC).

• Тесная интеграция оборудования дает повышенную производительность. Также негибкая система лицензирования x86 — еще один минус.

Отличие RISC и CISC

Архитектуры набора команд обычно следуют разным основополагающим философиям. Набор команд x86 — это то, что мы называем архитектурой CISC, в то время как архитектура ARM следует принципам RISC. В этом заключается основное различие.



Инструкции CISC могут быть любой длины. Максимальная теоретическая длина инструкции x86 может быть бесконечной, но на практике не превышает 15 байт. Инструкции RISC имеют ограниченную длину.

Отличие RISC и CISC

Аббревиатура **CISC** обозначает **Complex Instruction Set Computer**,
а **RISC — Reduced Instruction Set Computer.**
Сегодня объяснить разницу между этими наборами команд сложнее, чем во время их появления, так как в процессе развития они заимствовали идеи друг у друга. Более того, проводилась маркетинговая кампания по размытию границ между ними.

В 1987 году лучшим среди x86 был процессор **Intel 386DX**, а среди RISC — **MIPS R2000**.
Несмотря на то, что процессор Intel имеет вдвое больше транзисторов (**275 000** против **115 000** у MIPS) и вдвое больше кэш-памяти, процессор x86 проигрывает во всех тестах производительности.
Оба процессора работают на частоте 16 МГц, но RISC-процессор показывал результаты в 2-4 раза лучше.
Поэтому неудивительно, что в начале 90-х распространилась идея, что процессоры RISC значительно производительнее.

У Intel возникли проблемы с восприятием на рынке. Им было сложно убедить инвесторов и покупателей в том, что процессоры на устаревшей архитектуре CISC могут быть лучше процессоров RISC.
Так Intel стала позиционировать свои процессоры как RISC с простым декодером, который превращал команды CISC в команды RISC. Компания заявляла, что покупатель получает технологически совершенные процессоры RISC, которые понимают знакомый многим набор команд x86. Но внутри процессора x86 нет RISC-составляющей. Это просто маркетинговый ход. Боб Колвеллс (Bob Colwells), один из создателей Intel Pentium Pro с RISC-составляющей, сам говорил об этом.

Философия CISC

Эту философию сложно определить, так как микросхемы, которые мы определяем как **CISC**, очень разнообразны. Однако в них можно выделить общие закономерности.

В конце 1970, когда началась разработка **CISC-процессоров**, 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 и хранится в специальной памяти внутри микропроцессора. 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. **Микрокод** хранится в ROM-памяти, которая значительно дешевле оперативной памяти. Следовательно, уменьшение использования оперативной памяти через увеличение использования постоянной памяти.

Какое-то время все выглядело очень хорошо. Но со временем начались проблемы, связанные с подпрограммами в микрокоде. В них появились ошибки. Исправление ошибки в микрокоде в разы сложнее, чем в обычной программе. Нельзя получить доступ к этому коду и протестировать его как обычную программу.

Разработчики стали думать, что существует другой, более простой способ справиться с этими трудностями.

Философия RISC

Оперативная память стала дешеветь, компиляторы стали лучше, а большинство разработчиков перестало писать на ассемблере. Сперва программисты анализировали программы и заметили, что большинство сложных инструкций CISC не используются большинством программистов. Эти технологические изменения спровоцировали появление философии RISC. При использовании RISC часто нужно писать больше команд, чем в случае CISC. Архитектура RISC оптимизирована для компиляторов, но не для написания программ. Разработчики компиляторов затруднялись в выборе правильной сложной инструкции CISC. Вместо этого они предпочли использовать комбинацию нескольких простых инструкций для решения задачи.

Идея RISC заключается в замене сложных инструкций на комбинацию из нескольких простых. Таким образом не придется заниматься сложной отладкой микрокода. Вместо этого разработчики компилятора будут решать необходимые задачи алгоритмизации.

В аббревиатуре RISC слово (reduced) «сокращенный» можно понимать как сокращенное количество инструкций, но наиболее интересная интерпретация — это уменьшение сложности команд.

CISC

```
MUL 0A,1F
```

RISC

```
LOAD A, 0A
LOAD B, 1F
MUL A,B
STOR 1F, A
```

Конвейеризация: инновация RISC

Еще одна основная идея RISC — это конвейеризация.
Представьте процесс покупки в продуктовом магазине. Действия на кассе можно разделить на несколько шагов.

- Переместить покупки на конвейерную ленту и отсканировать штрих-коды на них.
- Использовать платежный терминал для оплаты.
- Положить оплаченное в сумку.

Если такое происходит без конвейеризации, то следующий покупатель сможет положить вещи на ленту только после того, как текущий покупатель заберет свои покупки. Аналогичное поведение изначально встречалось в CISC-процессорах, в которых по умолчанию нет конвейеризации.
Это неэффективно, так как следующий покупатель может начать использовать ленту, только когда предыдущий заберет товар. Получается, что ресурсы используются неэффективно. Представим, что каждое действие занимает фиксированный промежуток времени или один такт. Это значит, что обслуживание одного покупателя занимает три такта. Таким образом, за девять тактов будут обслужены три покупателя.
Подключим конвейеризацию к данному процессу. Как только покупатель начнет работать с платежным терминалом, следующий покупатель начнет выкладывать продукты на ленту.

Конвейеризация: инновация RISC

Как только покупатель начнет складывать продукты в сумку, следующий покупатель начнет работу с платежным терминалом. При этом третий покупатель начнет выкладывать покупки из корзины. В результате каждый такт кто-то будет собирать свои покупки и выходить из магазина. Таким образом, за девять тактов можно обслужить шесть покупателей. Можно сказать, что работа с кассой занимает три такта, но пропускная способность кассы — один покупатель в такт. В терминологии микропроцессоров это значит, что одна инструкция выполняется три такта, но средняя пропускная способность — одна инструкция в такт.

Это справедливо, если обработка каждого этапа занимает одинаковое количество времени. Если время каждого этапа сильно варьируется, то это работает не так хорошо. Например, если кто-то взял очень много продуктов и долго выкладывает их на ленту, то зона с платежным терминалом и упаковкой продуктов будет простаивать. Это негативно влияет на эффективность схемы.

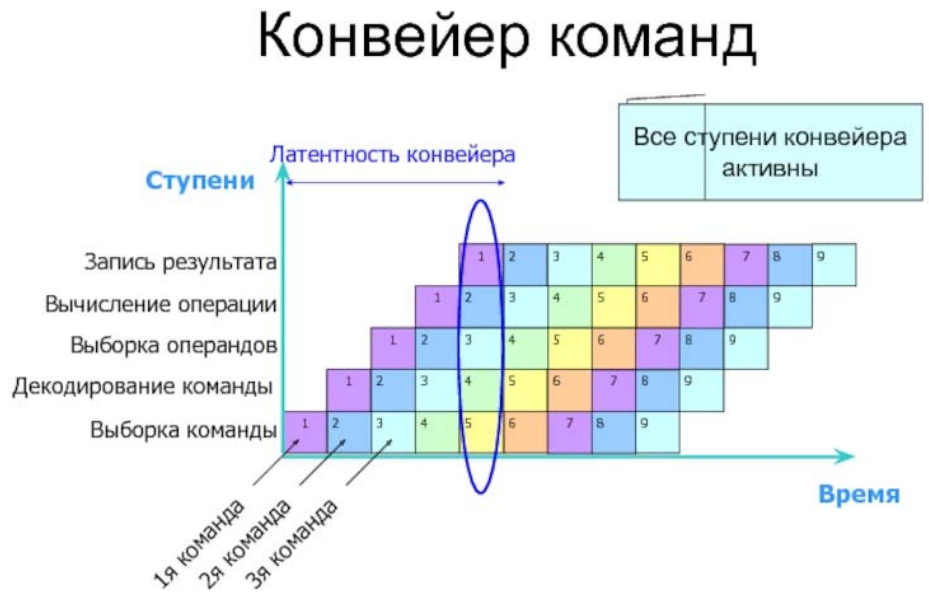
Проектировщики **RISC** прекрасно это понимали, поэтому попытались стандартизировать время выполнения каждой инструкции. Они разделены на этапы, каждый из которых выполняется примерно одинаковое количество времени. Таким образом, ресурсы внутри микропроцессора используются по максимуму.

Конвейеризация: инновация RISC

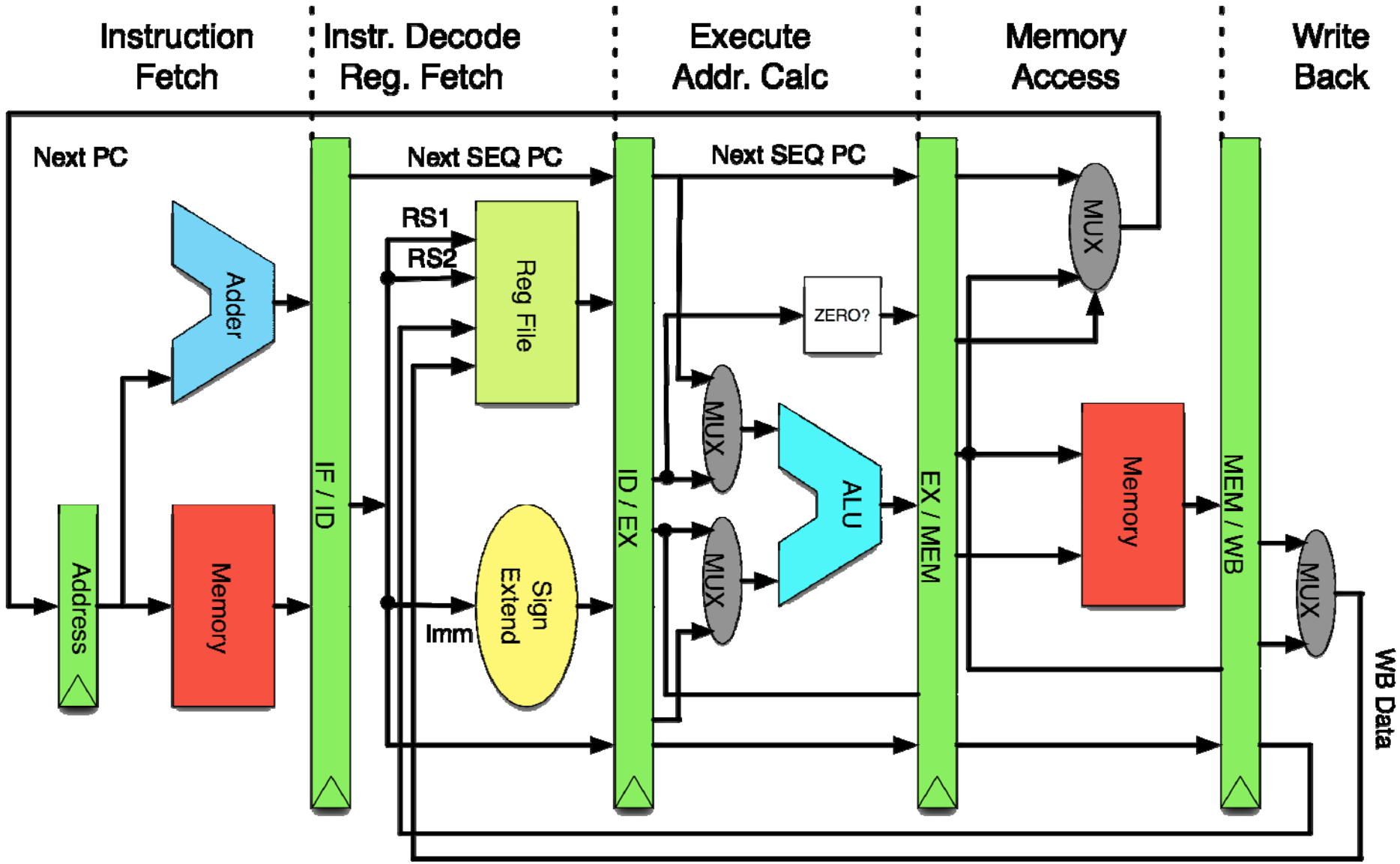
•Если рассмотреть ARM RISC-процессор, то мы обнаружим пятиступенчатый конвейер инструкций.

(Fetch) Извлечение инструкции из памяти и увеличение счетчика команд, чтобы извлечь следующую инструкцию в следующем такте.
•**(Decode) Декодирование** инструкции — определение, что эта инструкция делает. То есть активация необходимых для выполнения этой инструкции частей микропроцессора.
•**(Execute) Выполнение** включает использование арифметико-логического устройства (АЛУ) или совершение сдвиговых операций.
•**(Memory) Доступ к памяти**, если необходимо.
Это то, что делает инструкция *load*.
•**(Write Back) Запись результатов** в соответствующий регистр.

Инструкции ARM состоят из секций, каждая из которых работает с одним из этих этапов, а выполнение этапа обычно занимает один такт. То есть инструкции ARM очень удобно конвейеризировать.
Более того, каждая инструкция имеет одинаковый размер, то есть этап Fetch знает, где будет располагаться следующая инструкция, и ему не нужно проводить декодирование.



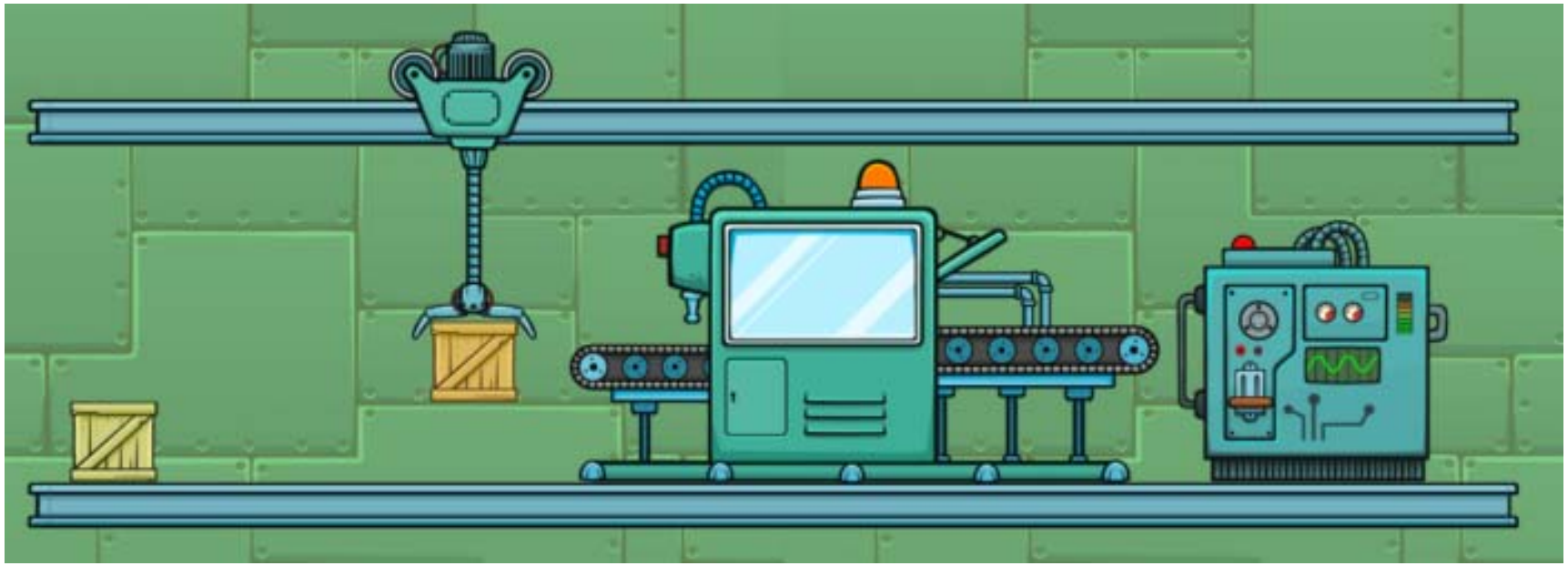
Конвейеризация: инновация RISC



Конвейеризация: инновация RISC

• С инструкциями CISC все не так просто. Они могут быть разной длины. То есть без декодирования фрагмента инструкции нельзя узнать ее размер и где располагается следующая инструкция.
• Вторая проблема CISC — сложность инструкций. Многократный доступ к памяти и выполнение множества вещей не позволяют легко разделить инструкцию CISC на отдельные части, которые можно выполнять поэтапно.

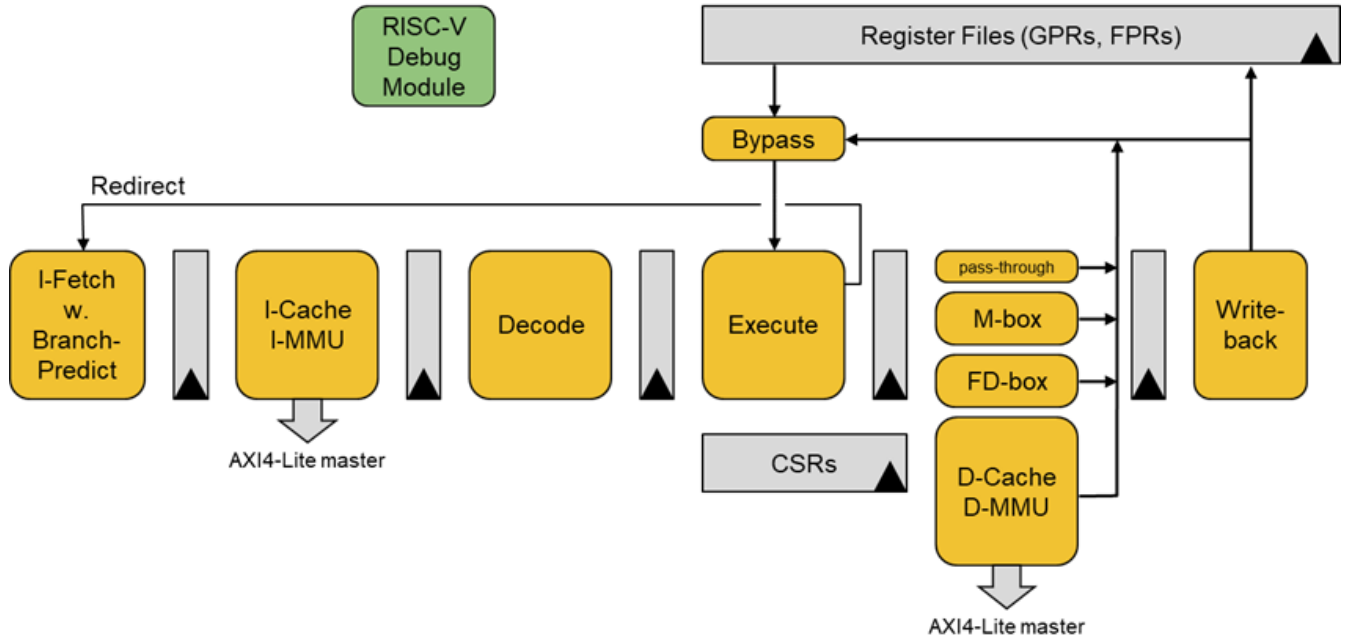
Конвейеризация — это особенность, которая позволила первым RISC-процессорам на голову обогнать своих конкурентов в тестах производительности.



Складской робот как аналогия для конвейеризации.

Архитектура Load / Store

Чтобы количество тактов, необходимых для каждой инструкции, было примерно одинаковым и удобным для конвейеризации, набор инструкций RISC четко отделяет загрузку из памяти и сохранение в память от остальных инструкций.



Например, в CISC может существовать инструкция, которая загружает данные из памяти, производит сложение, умножение, что-нибудь еще и записывает результат обратно в память.
В мире RISC такого быть не может. Операции типа сложения, сдвига и умножения выполняются только с регистрами. Они не имеют доступа к памяти.
Это было сделано что бы реализовать конвейеризацию. Иначе инструкции в конвейере могли зависеть друг от друга.

Большое количество регистров

Большая проблема для **RISC** — это упрощение инструкций, что ведет к увеличению их количества. Больше инструкций требуют больше памяти — недорогой, но медленной. Если программа RISC потребляет больше памяти, чем программа CISC, то она будет медленнее, так как процессор будет постоянно ждать медленного чтения из памяти.

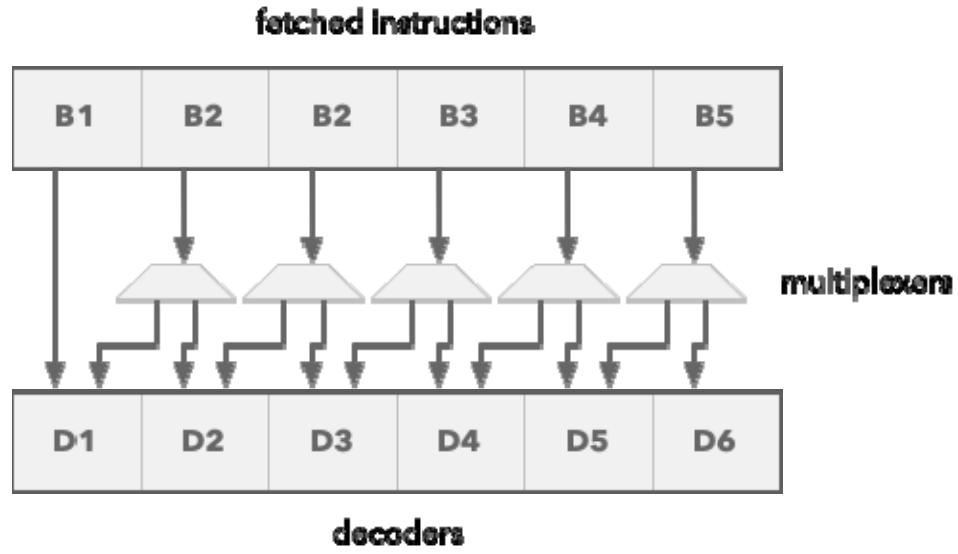
Проектировщики **RISC** сделали несколько наблюдений, которые позволили решить эту проблему. Они заметили, что множество инструкций перемещают данные между памятью и регистрами, чтобы подготовиться к выполнению команды. Имея большое количество регистров, они смогли сократить количество обращений к памяти.

Это потребовало улучшений в компиляторах. Компиляторы должны хорошо анализировать программы, чтобы понимать, когда переменные можно хранить в регистре, а когда их стоит записать в память. Работа с множеством регистров стала важной задачей для компиляторов, позволяющей ускорить работу на RISC-процессорах.

Инструкции в RISC проще. В них нет большого количества разных режимов адресации, поэтому, например, среди 32-битных команд есть больше бит, указывающих номер регистра.

Сжатый набор инструкций

Сжатый набор инструкций — это относительно новая идея для мира RISC, созданная для решения проблемы большого потребления памяти, с которой не сталкиваются CISC-процессоры.
Поскольку эта идея нова, то ARM пришлось модернизировать ее под существующую архитектуру. А вот современный RISC-V специально проектировался под сжатый набор инструкций и потому поддерживает их с самого начала.
Это несколько переработанная идея CISC, так как CISC инструкции могут быть как очень короткими, так и очень длинными.
Процессоры RISC не могут добавить короткие инструкции, так как это усложняет работу конвейеров. Вместо этого проектировщики решили использовать сжатые инструкции.
Это означает, что подмножество наиболее часто используемых 32-битных инструкций помещается в 16-битные инструкции. Таким образом, когда процессор выбирает инструкцию, он может выбрать две инструкции.
Так, например, в RISC-V есть специальный «флаг», который обозначает, сжатая это инструкция или нет. Если инструкция сжатая, то она будет разделена на две отдельные 32-битные инструкции.
Это процесс чтения сдвоенной команды, затем остальная часть микропроцессора работает как обычно. На вход подаются привычные однородные 32-битные инструкции, и все остальное работает предсказуемо.



Большие кэши

Кэши — это специальная форма очень быстрой памяти, которая располагается на микросхеме процессора. Они занимают дорогостоящую кремниевую площадь, необходимую процессору, поэтому есть ограничения по размеру кэшей.

Идея кэширования заключается в том, что большинство программ запускают небольшую часть себя гораздо чаще, чем остальную часть. Часто небольшие части программы повторяются бесчисленное количество раз, например, в случае циклов.

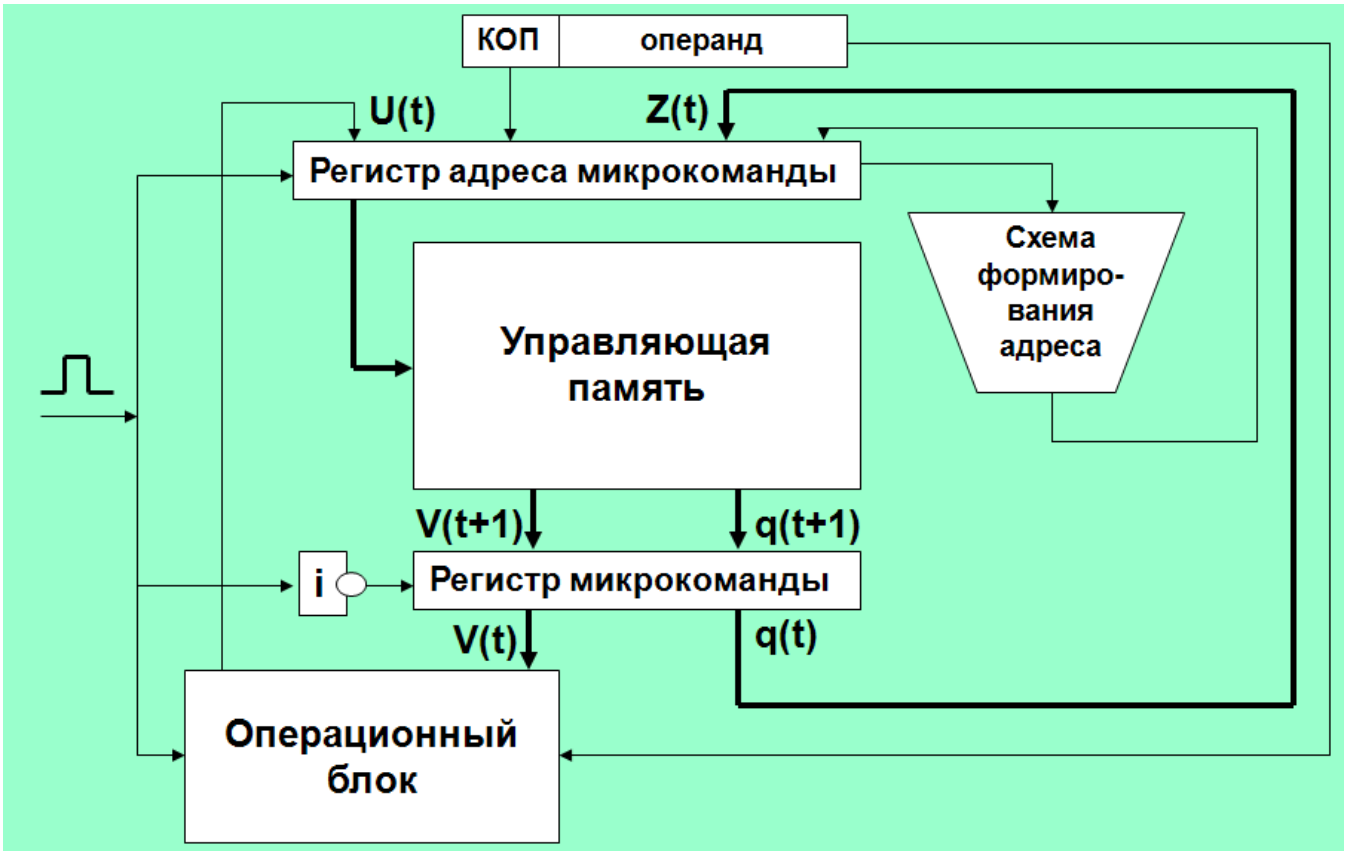
Следовательно, если поместить наиболее часто используемые части программы в кэш, то можно добиться значительного прироста скорости.

Это была ранняя стратегия RISC, когда программы для RISC занимали больше места, чем программы для CISC. Поскольку процессоры RISC были проще, для их реализации требовалось меньше транзисторов. Это оставляло больше кремниевой площади, которую можно было потратить на другие вещи, например, для кэшей.

Таким образом, имея большие кэши, процессоры RISC компенсировали то, что их программы больше, чем программы CISC.

микрооперации

Конечно, CISC не сидел сложа руки и не ждал, когда RISC его повергнет. Intel и AMD разработали собственные стратегии по эмуляции хороших решений RISC.
В частности, они искали способ конвейеризации инструкций, который никогда не работал с традиционными инструкциями CISC.
Было принято решение сделать внутренности CISC-процессора более RISC-похожими. Способ, которым это было достигнуто, — разбиение CISC-инструкции на более простые, названные микрооперациями.
Микрооперации, как инструкции RISC, легко конвейеризировать, потому что у них меньше зависимостей между друг другом и они выполняются за предсказуемое количество тактов.



В чем различие микроопераций и микрокода?

Микрокод — это маленькие программы в ROM-памяти, которые выполняются для имитации сложной инструкции. В отличие от микроопераций их нельзя конвейеризировать. Они не созданы для этого.

На самом деле микрокод и микрооперации существуют бок о бок. В процессоре, который использует микрооперации, программы микрокода будут использоваться для генерации серии микроопераций, которые помещаются в конвейер для последующего выполнения.

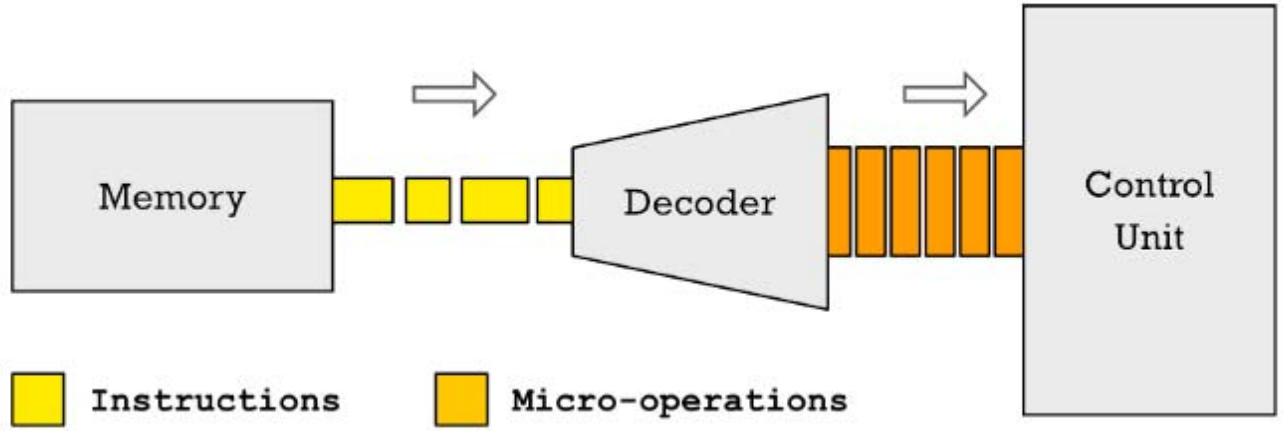
Имейте в виду, что микрокод в традиционном CISC-процессоре должен производить декодирование и выполнение. По мере выполнения микрокод берет под свой контроль различные ресурсы процессора, такие как АЛУ, регистры и так далее.

В современных CISC микрокод выполнит работу быстрее, потому что не использует ресурсов процессора. Он используется просто для генерации последовательности микроопераций.

Как микрооперации отличаются от RISC-инструкций

Инструкции RISC существуют на уровне набора команд. Это то, с чем работают компиляторы. Они думают о том, что вы хотите сделать, а мы пытаемся оптимизировать это.

Микрооперация — это нечто совершенно иное. Микрооперации, как правило, большие. Они могут быть больше 100 бит. Неважно, насколько они большие, потому что они существуют временно. Это различает их от инструкций RISC, которые составляют программы и могут занимать гигабайты памяти. Инструкции не могут быть сколь угодно большими.



Инструкции прибывают из памяти, обычно из высокоскоростного кэша. Далее они входят в декодер, который разбивает каждую инструкцию на одну или несколько микроопераций. Хотя они выполняют меньше одной инструкции, они значительно больше.

Микрооперации специфичны для каждой модели процессора. Каждый бит указывает часть процессора, которую необходимо включить или выключить при исполнении.
В общем случае нет необходимости в декодировании, если можно сделать большую инструкцию.
Каждый бит отвечает за определенный ресурс в процессоре.
Таким образом, разные процессоры с одинаковым набором команд будут иметь разные микрокоды.

Как микрооперации отличаются от RISC-инструкций

Фактически некоторые RISC-процессоры используют микрокод для некоторых инструкций, как CISC-процессоры. Одним из таких примеров является сохранение и восстановление состояния регистров при вызове подпрограмм. Когда одна программа переходит к другой подпрограмме для выполнения задачи, эта подпрограмма будет использовать некоторые регистры для локальных вычислений. Код, вызывающий подпрограмму, не хочет, чтобы его данные в регистрах изменялись случайным образом, поэтому он должен их сохранить в памяти.

Это настолько частое явление, что добавление конкретной инструкции для сохранения нескольких регистров в память было слишком заманчивым. В противном случае эти инструкции могут съесть много памяти. Поскольку это предполагает многократный доступ к памяти, имеет смысл добавить это как программу микрокода.

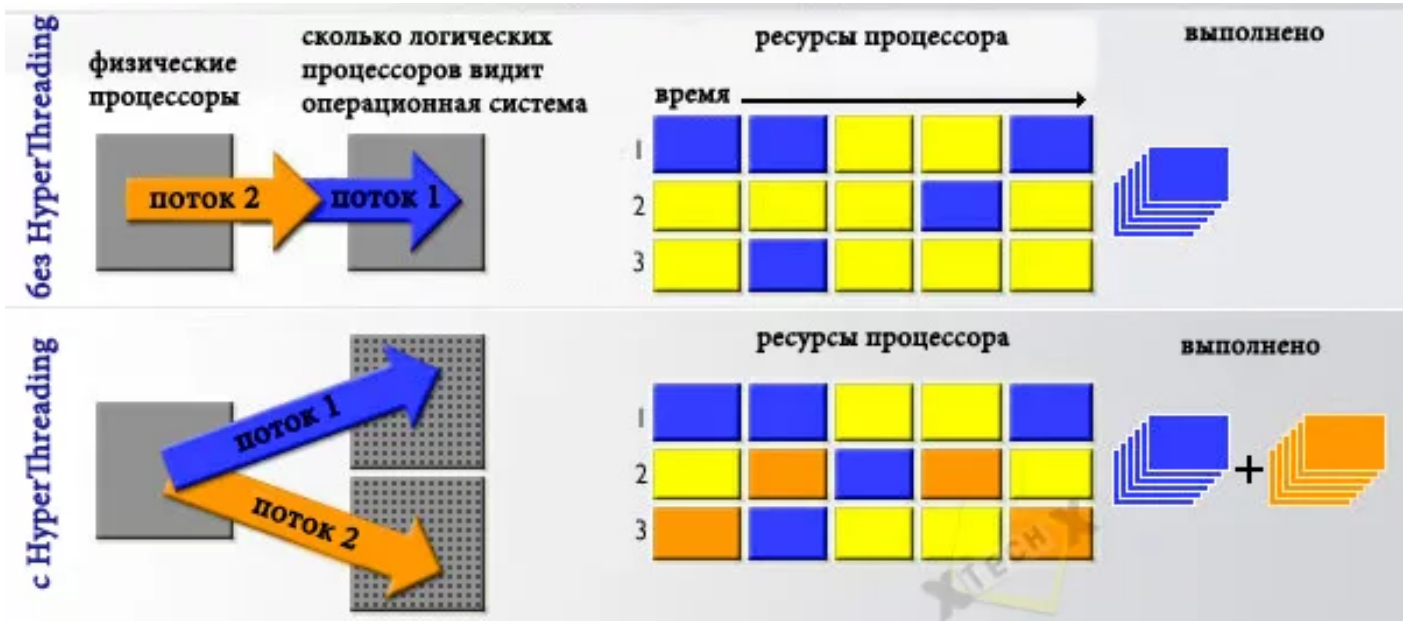
Однако не все RISC процессоры делают это. Например, RISC-V пытается быть более «чистым» и не имеет специальной инструкции для этого. Команды RISC-V оптимизированы для конвейеризации. Более строгое следование философии RISC делает конвейеризацию более эффективной.

Гипертрединг (аппаратные потоки)

Еще один трюк, который используется CISC, — это гипертрединг.
Напомню, что микрооперации непростые. Конвейер команд не будет заполнен полностью на постоянной основе, как у RISC.
Поэтому используется трюк под названием гипертрединг. Процессор CISC берет несколько потоков инструкций. Каждый поток инструкций разбивается на части и конвертируется в микрооперации. Поскольку этот процесс несовершенен, вы получите ряд пробелов в конвейере. Но, имея дополнительный поток инструкций, вы можете поместить в эти промежутки другие микрооперации и таким образом заполнить конвейер.
Эта стратегия актуальна и для RISC-процессоров, потому что не каждая инструкция может исполняться каждый такт. Доступ к памяти, например, занимает больше времени. Аналогично для сохранения и восстановления регистров с помощью сложной инструкции, которую предоставляют некоторые RISC-процессоры. В коде также есть переходы, которые вызывают пробелы в конвейере.
Следовательно, более продвинутые и производительные процессоры RISC, такие как IBM POWER, тоже будут использовать аппаратные потоки.
В моем понимании трюк с гипертредингом более выгоден для процессоров CISC. Создание микроопераций — менее идеальный процесс, и он создает больше пробелов в конвейере, следовательно, гипертрединг дает больший прирост производительности.

Гипертрединг (аппаратные потоки)

Если конвейер всегда заполнен, то от гипертредининга / аппаратных потоков нет никакой пользы.
Аппаратные потоки могут представлять угрозу безопасности. Intel столкнулась с проблемами безопасности, потому что один поток инструкций может влиять на другой.
Как правило, аппаратные потоки дают примерно 20% прирост производительности. То есть процессор с 5 ядрами и гипертредингом будет приблизительно похож на процессор с 6 ядрами без него. Но данное значение зависит во многом от архитектуры процессора.
В любом случае, это одна из причин, почему ряд производителей высокопроизводительных чипов ARM, таких как Ampere, выпускают 80-ядерный процессор без гипертредининга.
Более того процессоры ARM не использует аппаратные потоки.



Организация памяти микроконтроллеров

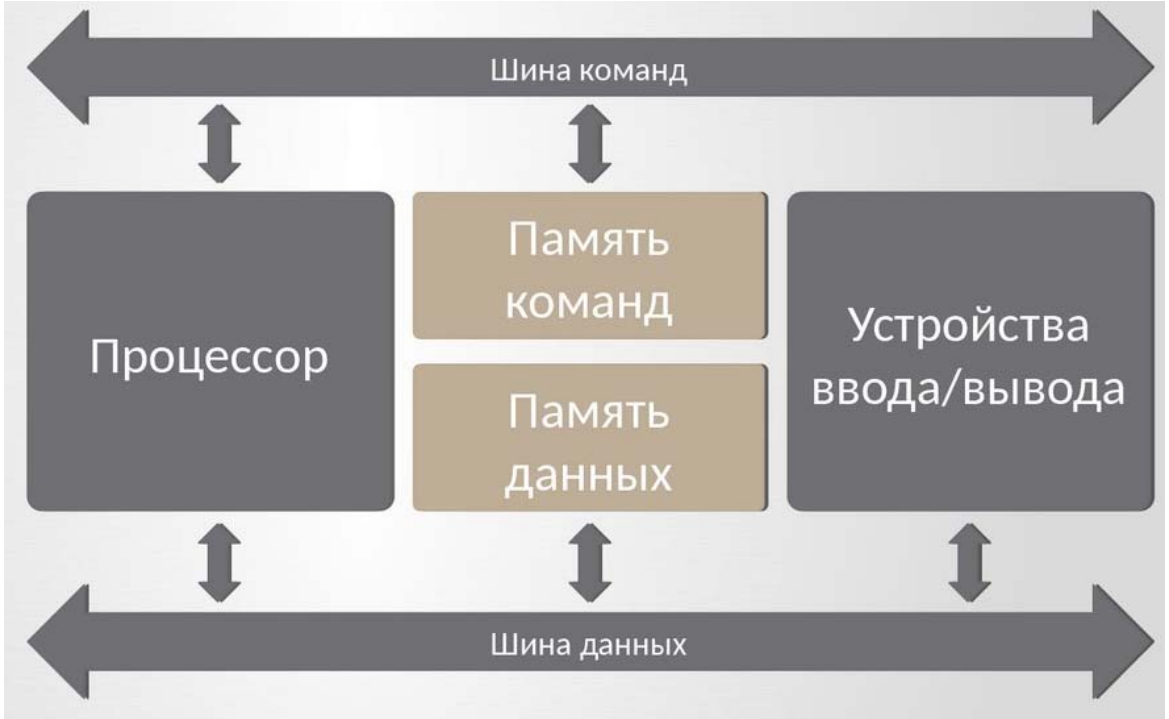
Архитектура фон Неймана
Гарвардская архитектура

Организация памяти микроконтроллеров

В отличие от обычных компьютерных микропроцессоров, в микроконтроллерах часто используется гарвардская архитектура памяти, то есть раздельное хранение данных в ОЗУ, а команд — в ПЗУ.

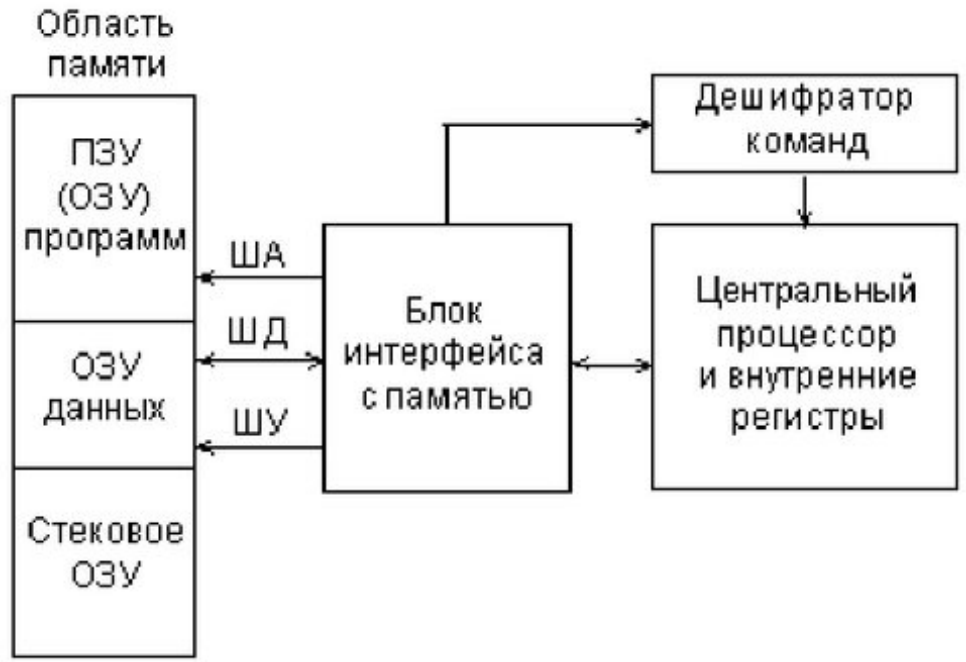
Кроме ОЗУ, микроконтроллер может иметь встроенную энергонезависимую память для хранения программы и данных. Многие модели контроллеров вообще не имеют шин для подключения внешней памяти.

Наиболее дешёвые типы памяти допускают лишь однократную запись, либо хранимая программа записывается в кристалл на этапе изготовления (конфигурацией набора технологических масок). Такие устройства подходят для массового производства в тех случаях, когда программа контроллера не будет обновляться. Другие модификации контроллеров обладают возможностью многократной перезаписи программы в энергонезависимой памяти.



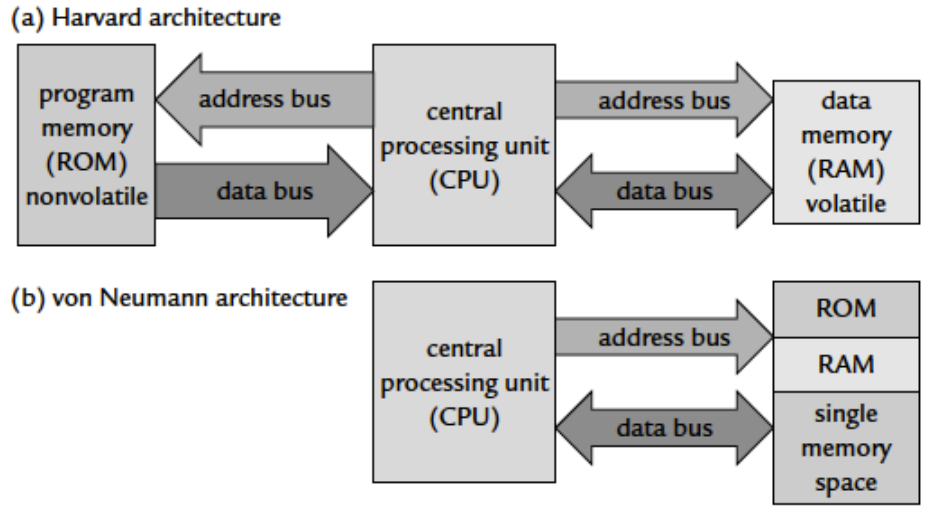
Классификация микроконтроллеров по типу памяти:

• **Архитектура фон Неймана** – основная особенность в том, что присутствует общая область памяти для команд и данных.
Это разработка Принстонского университета, более известная как архитектура фон Неймана, названная так по имени авторитетного учёного-консультанта и разработчика, первым предоставившего отчёт об архитектуре, к которой пришли в ходе плодотворных дискуссий в команде создателей; авторами же идей, заложенных в этой архитектуре, являлись Джон Преспер Экерт и Джон Уильям Мокли. Разработана в процессе работы над ЭНИАКом, а затем над EDVAC.
Принципы логической архитектуры были оформлены в документе, так как на первой странице документа стояло только имя фон Неймана, у читавших документ сложилось ложное впечатление, что автором всех идей, изложенных в работе, является именно он.



Классификация микроконтроллеров по типу памяти:

• **Архитектура фон Неймана** – (модель фон Неймана, Принстонская архитектура) —принцип совместного хранения команд и данных в памяти компьютера. Вычислительные машины такого рода часто обозначают термином «машина фон Неймана»



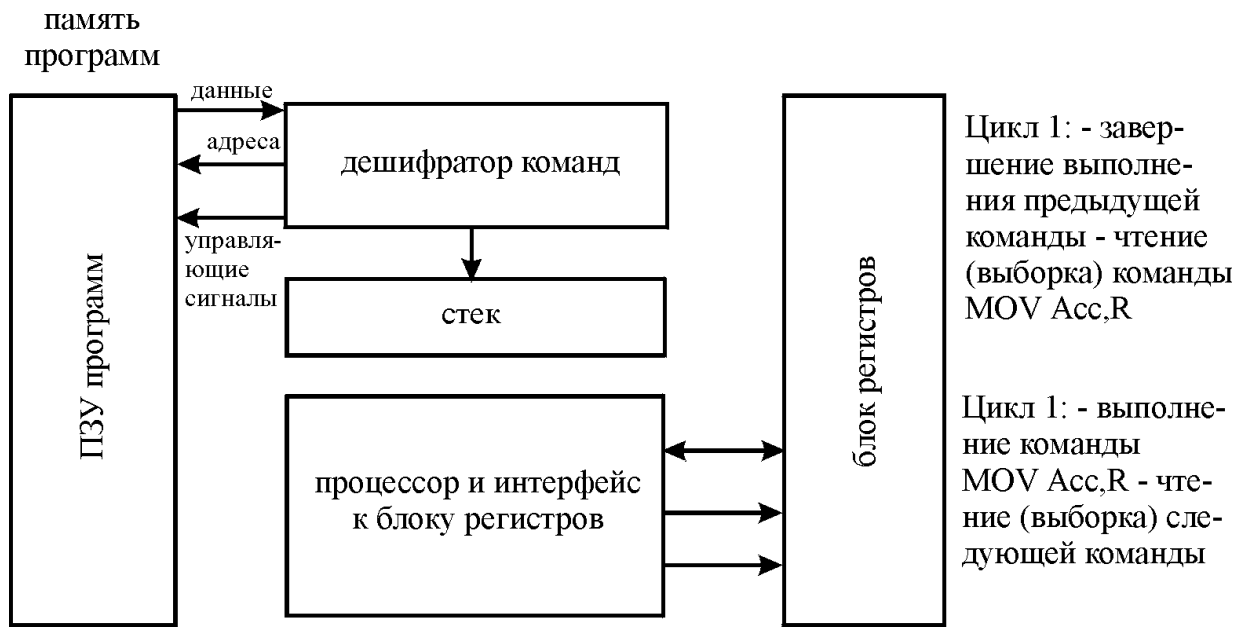
Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования; то есть одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему.

Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой приём носит название модификации команд и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной вычислительной машины.

Классификация микроконтроллеров по типу памяти:

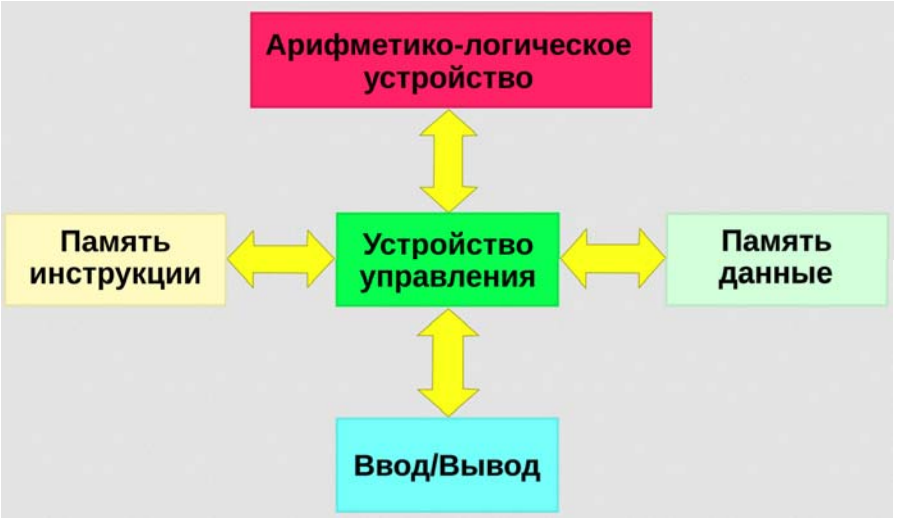
Гарвардская архитектура – основное отличие от архитектуры фон Неймана заключается в раздельной памяти данных и программ.

• В конце 1930-х годов в Гарвардском университете Говардом Эйкеном была разработана архитектура компьютера Марк I. Оригинальная идея была продемонстрирована Эйкеном компании IBM в октябре 1937 года. Архитектура была разработана Говардом Эйкеном в конце 1930-х годов в Гарвардском университете. Разделение каналов позволяло одновременно пересылать и обрабатывать команды и данные, благодаря чему значительно повышалось общее быстродействие компьютера. Впервые использовалась на компьютерах семейства Mark. В первом компьютере Эйкена «Марк I» для хранения инструкций использовалась перфорированная лента, а для работы с данными — электромеханические регистры.



Классификация микроконтроллеров по типу памяти:

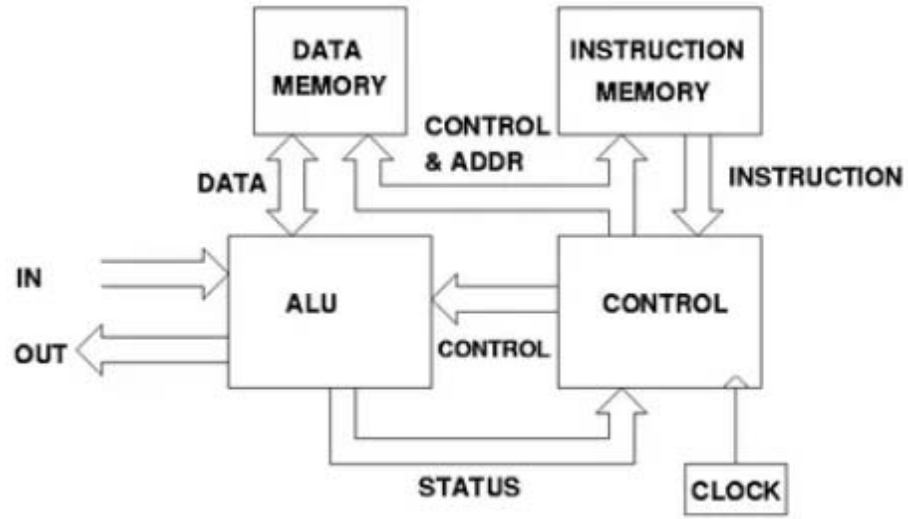
• Память инструкций и память данных представляют собой разные физические устройства;
• канал инструкций и канал данных также физически разделены.
Типичные операции (*сложение и умножение*) требуют от любого вычислительного устройства нескольких действий:
• выборку двух операндов;
• выбор инструкции и её выполнение;
• сохранение результата.
В гарвардской архитектуре характеристики устройств памяти для инструкций и памяти для данных не обязательно должны быть одинаковыми. В частности, ширина слова, тактирование, технология реализации и структура адресов памяти могут различаться. В некоторых системах инструкции могут храниться в памяти только для чтения, в то время как для сохранения данных обычно требуется память с возможностью чтения и записи. В некоторых системах требуется значительно больше памяти для инструкций, чем памяти для данных, поскольку данные обычно могут подгружаться с внешней или более медленной памяти. Такая потребность увеличивает битность (ширину) шины адреса памяти инструкций по сравнению с шиной адреса памяти данных.



Классификация микроконтроллеров по типу памяти:

Первым компьютером, в котором была использована идея *гарвардской архитектуры*, был **Марк I.**

Гарвардская архитектура используется в **ПЛК** и **микроконтроллерах**, таких, как **Microchip PIC, Atmel AVR, Intel 4004, Intel 8051, процессорах серии TMC320 фирмы Texas Instruments, DSP фирм Motorola и Analog Devices**, а также в кэш-памяти первого уровня **x86-микропроцессоров**, делящейся на два равных либо различных по объёму блока для данных и команд.

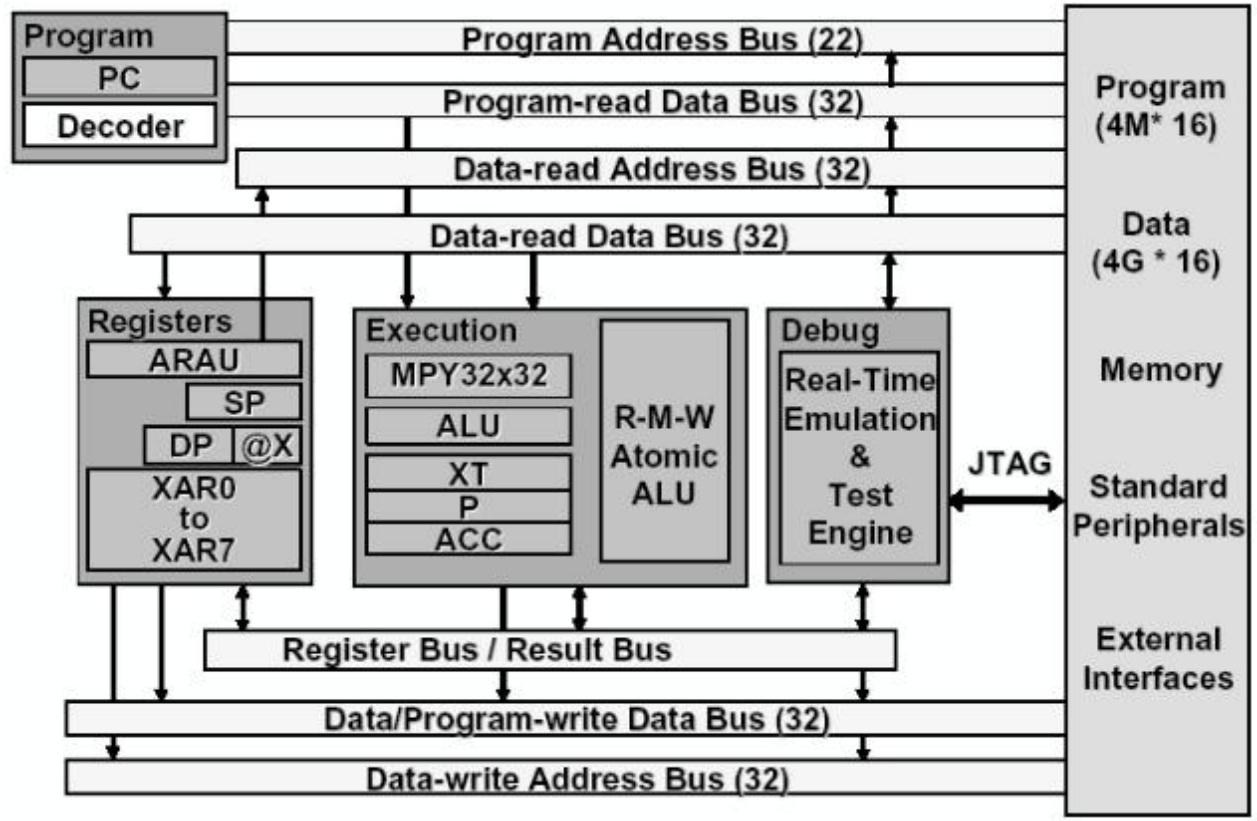


Классификация микроконтроллеров по типу памяти:

Модифицированная гарвардская архитектура

При разделении каналов передачи команд и данных на кристалле процессора последний должен иметь почти вдвое больше интерфейсных выводов, так как шина адреса и шина данных составляют основную часть выводов микропроцессора.

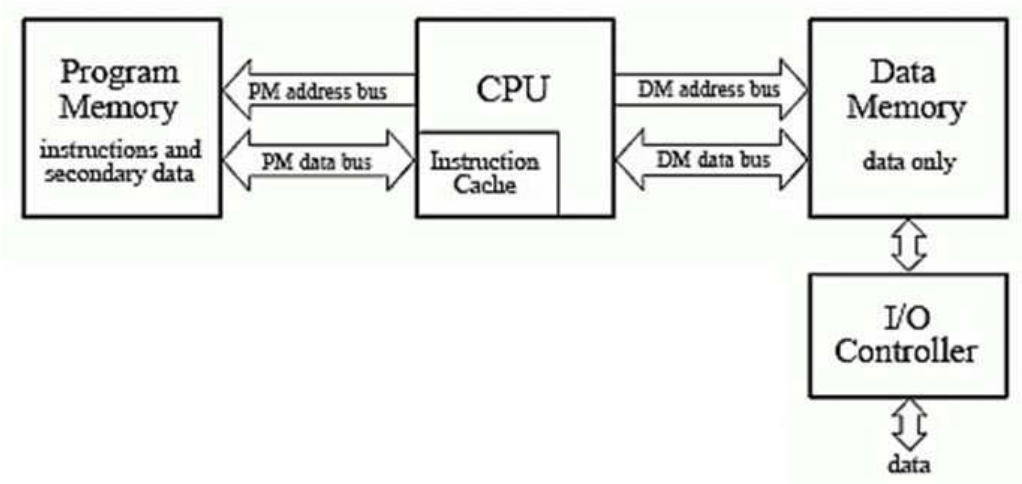
Способом разрешения этой проблемы стала идея использовать общие шину данных и шину адреса для всех внешних данных, а внутри процессора использовать шину данных, шину команд и две шины адреса. Такую концепцию стали называть модифицированной гарвардской архитектурой. Такая схемотехника применяется в современных сигнальных процессорах. Ещё дальше по пути уменьшения стоимости пошли при создании микроконтроллеров. В них одна шина команд и данных применяется и внутри кристалла. Разделение шин в модифицированной гарвардской структуре осуществляется при помощи раздельных управляющих сигналов: чтения, записи или выбора области памяти.



Классификация микроконтроллеров по типу памяти:

Расширенная гарвардская архитектура

Часто требуется выбрать три составляющие: два операнда и инструкцию (в алгоритмах цифровой обработки сигналов это наиболее распространенная задача в БПФ-, КИХ- и БИХ-фильтрах). Для этого существует кэш-память. В ней может храниться инструкция — следовательно, обе шины остаются свободными и появляется возможность передать два операнда одновременно. Использование кэш-памяти вместе с разделёнными шинами получило название «Super Harvard Architecture» («SHARC») — расширенная гарвардская архитектура.

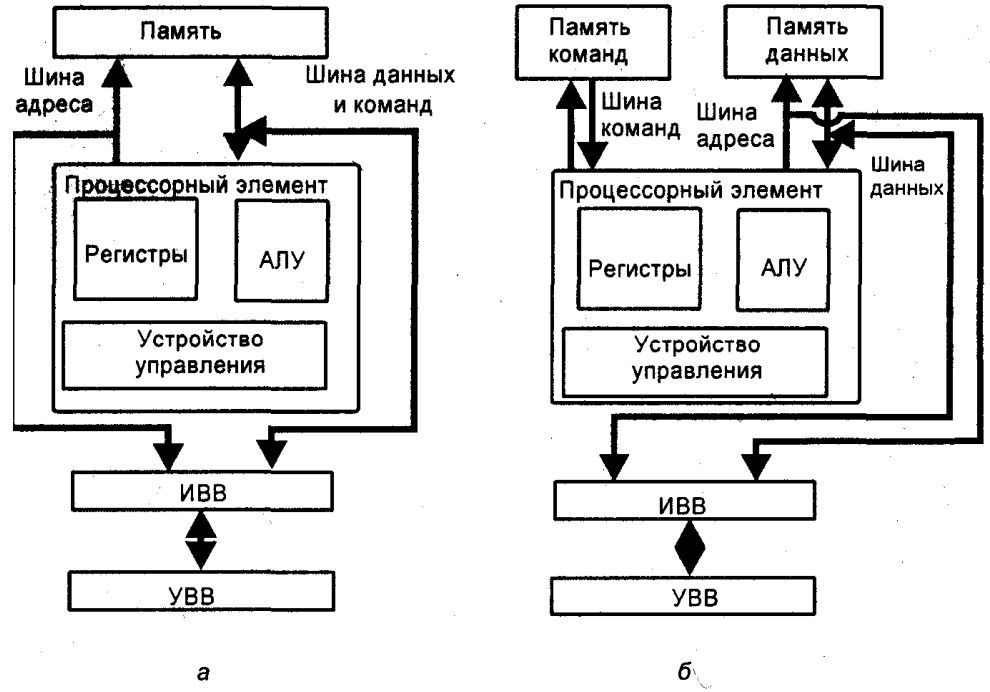


Примером могут служить процессоры «**Analog Devices**»:
ADSP-21xx — модифицированная гарвардская архитектура,
ADSP-21xxx (SHARC) — расширенная гарвардская архитектура.

Классификация микроконтроллеров по типу памяти:

Гибридные модификации с архитектурой фон Неймана

Существуют гибридные архитектуры, сочетающие достоинства как гарвардской, так и фон-неймановской архитектур. Современные CISC-процессоры обладают раздельной кэш-памятью 1-го уровня для команд и данных, что позволяет им за один рабочий такт получать одновременно и команду, и данные для её выполнения. То есть процессорное ядро аппаратно гарвардское, но программно оно фон-неймановское, что упрощает написание программ. Обычно в данных процессорах одна шина используется и для передачи команд, и для передачи данных, что схемотехнически упрощает систему. Современные варианты таких процессоров могут иногда содержать встроенные контроллеры сразу нескольких разнотипных шин для работы с различными типами памяти — например, DDR RAM и Flash. Тем не менее, и в этом случае шины, как правило, используются и для передачи команд, и для передачи данных без разделения, что делает данные процессоры ещё более близкими к фон-неймановской архитектуре при сохранении достоинств гарвардской архитектуры.



Что такое микропроцессор

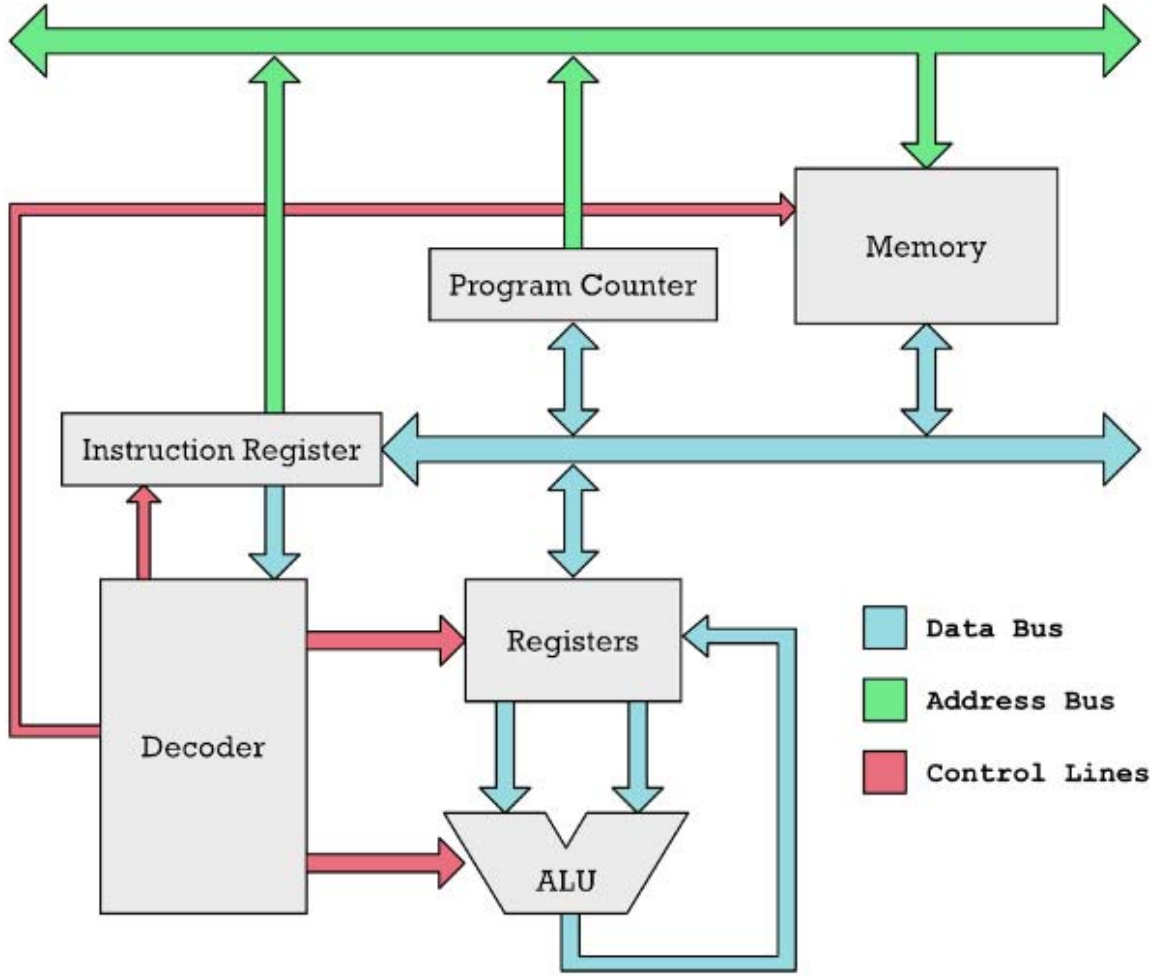
В общем случае процессор — это мозг компьютера. Он читает инструкции из памяти, которые указывают, что делать компьютеру. Инструкции — это просто числа, которые интерпретируются специальным образом.
В памяти нет ничего, что позволяло бы отличить обычное число от инструкции. Поэтому разработчики операционных систем должны быть уверены, что инструкции и данные лежат там, где процессор ожидает их найти. Микропроцессоры (CPU) выполняют очень простые операции. Вот пример нескольких инструкций, которые выполняет процессор:

load r1, 150
load r2, 200
add r1, r2
store r1, 310

Это человекочитаемая форма того, что должно быть просто списком чисел для компьютера. Например, *load r1, 150* в обычном RISC микропроцессоре представляется в виде 32-битного числа. Это значит, что число представлено 32 символами, каждый из которых 0 или 1.
load в первой строчке перемещает содержимое ячейки памяти 150 в регистр r1. Оперативная память компьютера (RAM) — это хранилище миллиардов чисел. Каждое число хранится по своему адресу, и так микропроцессор получает доступ к правильному числу.
В нашем примере add r1, r2 складывает содержимое r1 и r2 и полученный результат записывает в r1. В конце мы сохраняем полученный результат в оперативной памяти в ячейке с адресом 310 с помощью команды store r1, 310.

Что такое микропроцессор

Упрощенная диаграмма операций в микропроцессоре. Инструкции помещаются в регистр инструкций, где происходит декодирование. Декодер активирует нужные части процессора и операция выполняется.



Что такое микропроцессор

Что такое регистр? Эта концепция достаточно старая. Старые механические кассовые аппараты были основаны на этой концепции. В те времена регистр был чем-то вроде механического приспособления, в котором хранилось число, с которым вы хотели работать. Часто в таких аппаратах был аккумуляторный регистр, в который вы могли добавлять числа, а регистр сохранял сумму.

Ваши электронные калькуляторы работают по такому же принципу. Чаще всего на дисплее отображается содержимое аккумуляторного регистра, а вы выполняете действия, которые влияют на его содержимое.
Аналогичное справедливо для микропроцессора. В нем есть множество регистров, которым даны имена — например, A, B, C или r1, r2, r3, r4 и так далее. Инструкции микропроцессора обычно производят операции над этими регистрами.

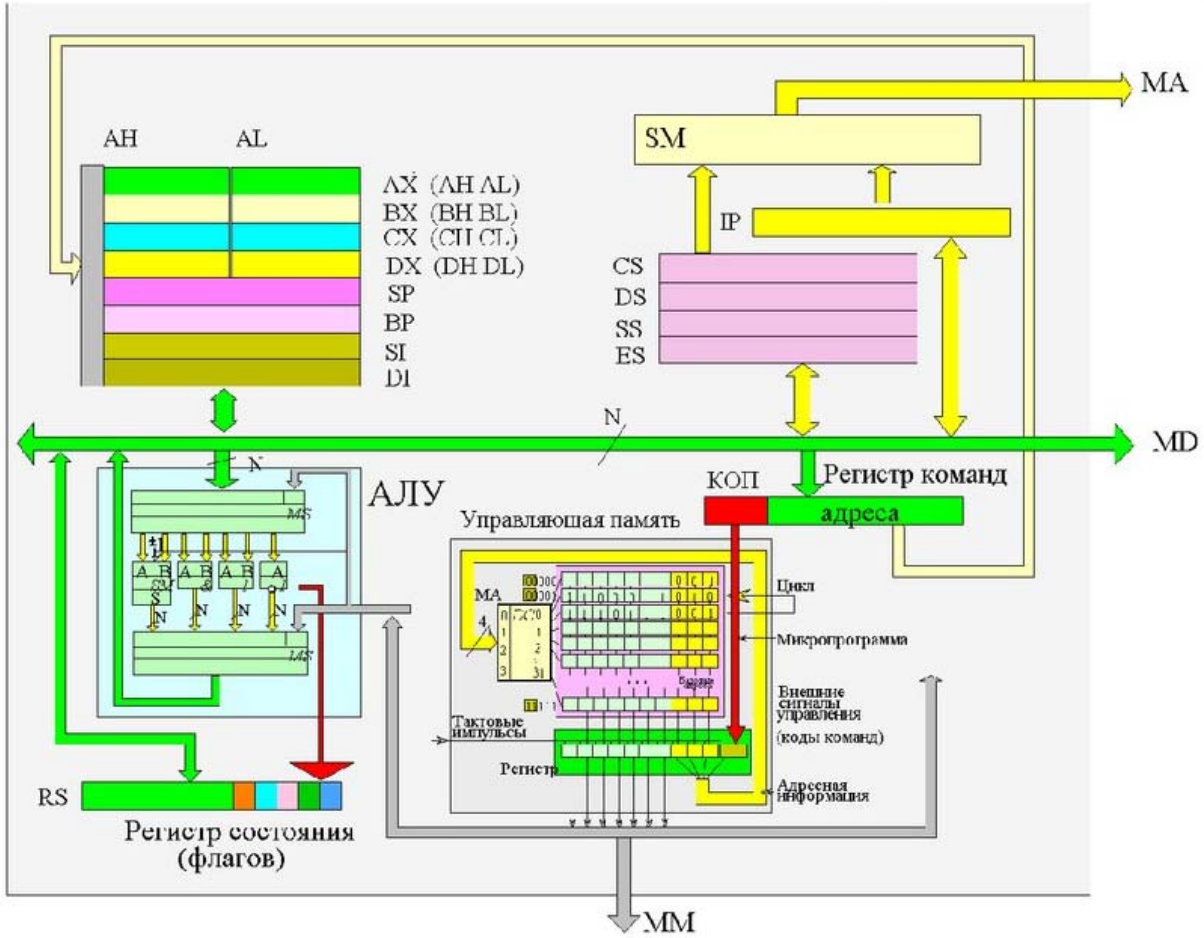
Арифметический калькулятор **Феликс**.
Русский механический калькулятор.
Внизу виден аккумуляторный регистр, сохранявший до тринадцати десятичных знаков.
Наверху — входной регистр, вмещающий пять знаков.
Слева внизу — счетный регистр.

Регистры

Реги́стр проце́ссора — поле заданной длины во внутрипроцессорной сверхбыстрой оперативной памяти (СОЗУ). Используется самим процессором, может быть как доступным, так и не доступным программно.

• ***Программно недоступные регистры*** — любые процессорные регистры, к которым невозможно так или иначе обратиться из выполняемой программы. Например, при выборке из памяти очередной команды она помещается в регистр команд, обращение к которому программист прописать не может.

• ***Программно доступные регистры*** есть те, к которым возможно так или иначе обратиться из выполняемой программы.



Практически каждый такой регистр обозначается своим именем-идентификатором на уровне языка ассемблера и соответствующим числовым кодом-идентификатором на уровне машинного языка.

Регистры

По уровню доступности программно доступные регистры неодинаковы и делятся на две большие подгруппы:

• ***Системные регистры*** — любые регистры, программно доступные только системным программам (например, ядру операционной системы), имеющим достаточный для этого уровень системных привилегий/прав. В терминах многих машинных систем такой уровень привилегий часто называется «уровнем/режимом ядра» или «режимом супервизора». Для всех прочих программ — работающих в «режиме пользователя» — эти регистры недоступны. Примеры таких регистров: управляющие регистры и теневые регистры дескрипторов сегментов.

• ***Регистры общего назначения (РОН)*** — регистры, доступные любым программам. В частности, регистры, используемые без ограничения в арифметических и логических операциях, но имеющие определённые аппаратные ограничения (например, в строковых РОН). Эти регистры не характерны для эпохи мейнфреймов типа IBM/370 и стали популярными в микропроцессорах архитектуры X86 — Intel 8085, Intel 8086 и последующих.

• ***Специальные регистры*** содержат данные, необходимые для работы процессора — смещения базовых таблиц, уровни доступа и т. д.

Часть специальных регистров принадлежит устройству управления, которое управляет процессором путём генерации последовательности микрокоманд.

Доступ к значениям, хранящимся в регистрах, осуществляется непосредственно на тактовой частоте процессора и, как правило, в несколько раз быстрее, чем доступ к полям в оперативной памяти (даже если кеш-память содержит нужные данные), но объём оперативной памяти намного превосходит суммарный объём процессорных регистров, суммарная «ёмкость» регистров общего назначения/данных для x86-процессоров (например, Intel 80386 и более новых) 8 регистров по 4 байта = 32 байта (В x86-64-процессорах — 16 по 8 байт = 128 байт и некоторое количество векторных регистров).

Регистры

В таблице показано количество регистров общего назначения в нескольких распространённых архитектурах микропроцессоров. Так, в SPARC и MIPS регистр номер 0 не сохраняет информацию и всегда считывается как 0, а в процессорах x86 с регистром ESP (указатель на стек) могут работать лишь некоторые команды.

Архитектура	Целочисленных регистров	FP- регистров	Примечания
x86-32	8	8	
x86-64	16	16	
IBM System/360	16	4	
z/Architecture	16	16	
Itanium	128	128	
SPARC	31	32	Регистр 0 (глобальный) всегда запущен
IBM Cell	4~16	1~4	
IBM POWER	32	32	
Power Architecture	32	32	
Alpha	32	32	
6502	3	0	
W65C816S	5	0	
PIC	1	0	
AVR	32	0	
ARM 32-bit[4]	16	различное	
ARM 64-bit[5]	31	32	
MIPS	31	32	Регистр 0 всегда занулён
RISC-V	31	32	Дополнительно есть регистр 0, который всегда возвращает ноль
Эльбрус 2000	256	совмещены с целочисленными	32 двухразрядных регистра, 256 = 32 глобальных регистра + 224 регистра стека

Архитектура ARM

ARM (Advanced RISC Machines) — одна из крупнейших в мире бесфабричных компаний, проектирующих процессоры, разработчик и лицензиар архитектуры 32-разрядных и 64-разрядных RISC-процессоров (с архитектурой ARM), ориентированных на использование в портативных и мобильных устройствах, а также в серверах и суперкомпьютерах. На 2020 год семейство ARM-процессоров является самым популярным среди прочих центральных процессоров. В 2020 году японский суперкомпьютер ***Фугаку*** на процессорах с архитектурой ARM занял 1-е место по производительности в мировом рейтинге Top500.

Изначально основана как совместное предприятие компаний ***Acorn, Apple и VLSI***; с 2016 года принадлежит японской телекоммуникационной и инвестиционной корпорации SoftBank. Центральный офис находится в Кембридже, Великобритания. Офисы и центры проектирования ARM расположены по всему миру, включая Францию, Индию, Швецию и США.

Самой успешной версией ядра, продажи которой достигали сотен миллионов штук, ранее был ***ARM7TDMI***.

В 2005 году было произведено 1,6 миллиарда ядер, около миллиарда ядер ARM пошло на мобильные телефоны. В январе 2008 года было произведено более 10 миллиардов ядер, а за 2011 год количество ядер оценивалось в 7,9 млрд.

В июле 2016 года совет директоров ARM Holdings продал за 24,3 млрд £ свою компанию японской телекоммуникационной корпорации SoftBank. В сентябре 2020 года Nvidia заключила сделку с SoftBank по приобретению компании ARM за 40 млрд долларов. Сделкой заинтересовались соответствующие органы США, Евросоюза, Великобритании и Китая. В феврале 2022 года стороны отказались от сделки. В 2023 году SoftBank провела размещение 10 % акций ARM на бирже NASDAQ.

Архитектура ARM

Архитектура ARM (***Advanced RISC Machine*** — усовершенствованная RISC-машина; иногда — Acorn RISC Machine) — система команд и семейство описаний и готовых топологий 32-битных и 64-битных микропроцессорных / микроконтроллерных ядер, разрабатываемых компанией ARM Limited.

Среди лицензиатов готовых топологий ядер ARM — компании AMD, Apple, Analog Devices, Atmel, Xilinx, Cirrus Logic, Intel, Marvell, NXP, STMicroelectronics, Samsung, LG, MediaTek, Qualcomm, Sony, Texas Instruments, Nvidia, Freescale, Миландр, ЭЛВИС, HiSilicon, Байкал электроникс.

Многие лицензиаты проектируют собственные топологии ядер на базе системы команд ARM: DEC StrongARM, Freescale i.MX, Intel XScale, NVIDIA Tegra, ST-Ericsson Nomadik, Krait и Kryo в Qualcomm Snapdragon, Texas Instruments OMAP, Samsung Hummingbird, LG H13, Apple A6 и HiSilicon K3.

Британская компания Acorn Computers задумалась над переходом от относительно слабых процессоров MOS Technology 6502 к более производительным решениям. Acorn тестировала множество процессоров и решилась на разработку собственного процессора, и их инженеры начали изучать документацию проекта ***RISC***, разработанного в Университете Калифорнии в Беркли. Они подумали, что раз уж *группе студентов* удалось создать вполне конкурентоспособный процессор, то их инженерам это будет несложно.

Сначала Уилсон приступила к разработке системы команд, создавая симулятор нового процессора. После успешного моделирования, собралась небольшая команда для реализации модели на аппаратном уровне.

Архитектура ARM

Официальный проект ***Acorn RISC Machine*** был начат в октябре 1983 года. ***VLSI Technology*** была выбрана в качестве поставщика кремниевых компонентов. Разработку возглавили Уилсон и Фербер. Их основной целью было достижение низкой латентности обработки прерывания, как у MOS Technology 6502. Архитектура доступа к памяти, взятая от 6502, позволила разработчикам достичь хорошей производительности без использования дорогостоящего в реализации модуля DMA. Первый процессор был произведен VLSI 26 апреля 1985 года — именно тогда он впервые заработал и был назван ***ARM1***. Первые серийные процессоры под названием ***ARM2*** стали доступны в следующем году.

В ARM2 была 32-разрядная шина данных, 26-битное адресное пространство и 16 32-разрядных регистров. Программный код должен был лежать в первых 64 мегабайтах памяти, а программный счётчик был ограничен 26 битами, так как верхние 4 и нижние 2 бита 32-битного регистра служили флагами. ARM2 стал, возможно, самым простым из популярных 32-битных процессоров в мире, имея всего лишь 30 тысяч транзисторов (для сравнения, в сделанном на 6 лет раньше процессоре Motorola 68000 было 68 тысяч транзисторов). Многое из этой простоты обусловлено отсутствием микрокода (который в процессоре 68000 занимает от одной четверти до одной трети площади кристалла) и отсутствием кэша, как и во многих процессорах того времени. Эта простота привела к низким затратам энергии, в то время как ARM был гораздо более производителен, чем Intel 80286. У его преемника — процессора ARM3 — уже был кэш 4 кб, что ещё больше увеличило производительность.

Архитектура ARM

В конце 1980-х годов ***Apple Computer и VLSI Technology*** начали работать с Acorn Computers над новыми версиями ядра ARM. Работа была настолько важна, что Acorn преобразовала команду разработчиков в 1990 году в новую компанию под названием ***Advanced RISC Machines***. По этой причине ARM иногда расшифровывают как ***Advanced RISC Machines*** вместо ***Acorn RISC Machine***. ***Advanced RISC Machines*** стала ARM, когда её родительская компания ARM Holdings вышла на Лондонскую фондовую биржу и NASDAQ в 1998 году.

Новая работа ***Apple-ARM*** в конечном итоге превратилась в ***ARM6***, впервые выпущенный в 1992 году. Apple использовала основанный на базе ARM6 процессор **ARM610** в качестве основы для своего продукта ***Apple Newton PDA***. В 1994 году Acorn стала использовать **ARM610** как главный процессор в своих компьютерах RISC PC. Компания DEC также купила лицензию на архитектуру ARM6 (чем вызвала небольшую путаницу, поскольку они также производили процессоры **Alpha**) и начала производить StrongARM. На 233 МГц этот процессор требовал всего 1 Вт мощности (более поздние версии требовали гораздо меньше). Позднее **Intel** получил права на эту работу в результате судебного процесса. Intel воспользовалась возможностью дополнить свою устаревшую линейку I960 процессором StrongARM и позднее разработала свою версию ядра под торговой маркой XScale, которую они впоследствии продали компании ***Marvell***.

Ядро ARM сохранило все тот же размер после всех этих изменений. У ARM2 было 30 тысяч транзисторов.

В основном процессоры семейства завоевали сегмент массовых мобильных продуктов (сотовые телефоны, карманные компьютеры) и встраиваемых систем средней и высокой производительности (от сетевых маршрутизаторов и точек доступа до телевизоров).

Процессоры ARM

В настоящее время значимыми являются несколько семейств процессоров ARM:

- ***ARM7 (с тактовой частотой до 60-72 МГц),*** предназначенные, например, для недорогих мобильных телефонов и встраиваемых решений средней производительности. Активно вытесняется семейством Cortex.
- ***ARM9, ARM11 (с частотами до 1 ГГц)*** для более мощных телефонов, карманных компьютеров и встраиваемых решений высокой производительности.
- ***Cortex A*** — новое семейство процессоров на смену ARM9 и ARM11.
- ***Cortex M*** — новое семейство процессоров на смену ARM7, также призванное занять новую для ARM нишу встраиваемых решений низкой производительности. В семействе присутствуют четыре значимых ядра:
- ***Cortex-M0, Cortex-M0+*** (более энергоэффективное) и ***Cortex-M1*** (оптимизировано для применения в ПЛИС) с архитектурой ***ARMv6-M***;
- ***Cortex-M3*** с архитектурой ***ARMv7-M***;
- ***Cortex-M4*** (добавлены SIMD-инструкции, опционально FPU) и ***Cortex-M7*** (FPU с поддержкой чисел одинарной и двойной точности) с архитектурой ***ARMv7E-M***;
- ***Cortex-M23 и Cortex-M33*** с архитектурой ***ARMv8-M ARMv8-M***.

В 2010 году производитель анонсировал процессоры ***Cortex-A15*** под кодовым названием Eagle, ARM утверждает, что ядро ***Cortex A15*** на 40 процентов производительнее на той же частоте, чем ядро ***Cortex-A9*** при одинаковом числе ядер на чипе. Изделие, изготовленное по 28-нанометровому техпроцессу, имеет 4 ядра, может функционировать на частоте до 2,5 ГГц и будет поддерживаться многими современными операционными системами. Популярное семейство микропроцессоров xScale фирмы Marvell (до 27 июня 2007 года — Intel[15]) в действительности является расширением архитектуры ARM9, дополненной набором инструкций Wireless MMX, специально разработанных фирмой Intel для поддержки мультимедийных приложений.

Архитектура

Уже давно существует справочное руководство по архитектуре ARM, которое разграничивает все типы интерфейсов, которые поддерживает ARM, так как детали реализации каждого типа процессора могут различаться. Архитектура развивалась с течением времени и, начиная с ARMv7, были определены 3 профиля:

- ***A (application)*** — для устройств, требующих высокой производительности (смартфоны, планшеты);
- ***R (real time)*** — для приложений, работающих в реальном времени;
- ***M (microcontroller)*** — для микроконтроллеров и недорогих встраиваемых устройств.

Профили могут поддерживать меньшее количество команд.

Режимы

Процессор может находиться в одном из следующих операционных режимов:

- ***User mode*** — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
- ***Fast Interrupt (FIQ)*** — режим быстрого прерывания (меньшее время срабатывания).
- ***Interrupt (IRQ)*** — основной режим прерывания.
- ***System mode*** — защищённый режим для использования операционной системой.
- ***Abort mode*** — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
- ***Supervisor mode*** — привилегированный пользовательский режим.
- ***Undefined mode*** — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию.

Переключение режима процессора происходит при возникновении соответствующего исключения или же модификацией регистра статуса.

Набор команд ARM Thumb

Режим, в котором исполняется 32-битный набор команд***. ARM Base Instruction Set***:
ADC, ADD, AND, B/BL, BIC, CMN, CMP, EOR, LDM, LDR/LDRB, MLA, MOV, MUL, MVN, ORR, RSB, RSC, SBC, STM, STR/STRB, SUB, SWI, SWP, TEQ, TST
Набор команд ***Thumb***
Для улучшения плотности кода процессоры, начиная с ARM7TDMI, снабжены режимом «***thumb***». В этом режиме процессор выполняет альтернативный набор 16-битных команд. Большинство из этих 16-разрядных команд переводится в нормальные команды ARM. Уменьшение длины команды достигается за счёт сокрытия некоторых операндов и ограничения возможностей адресации по сравнению с режимом полного набора команд ARM.
В режиме ***Thumb*** меньшие коды операций обладают меньшей функциональностью. Например, только ветвления могут быть условными, и многие коды операций имеют ограничение в виде доступа только к половине главных регистров процессора. Более короткие коды операций в целом дают большую плотность кода, хотя некоторые операции требуют дополнительных команд. В ситуациях, когда порт памяти или ширина шины ограничены 16 битами, более короткие коды операций режима Thumb становятся гораздо производительнее по сравнению с обычным 32-битным ARM-кодом, так как меньший программный код придется загружать в процессор при ограниченной пропускной способности памяти.
Первым процессором с декодером ***Thumb***-команд был ***ARM7TDMI***. Все процессоры семейства ARM9, а также XScale, имели встроенный декодер ***Thumb***-команд.

Набор команд Thumb-2

Thumb-2 — технология, появившаяся в ARM1156 core, анонсированном в 2003 году. Он расширяет ограниченный 16-битный набор команд ***Thumb*** дополнительными 32-битными командами, чтобы задать набору команд дополнительную ширину. Цель ***Thumb-2*** — достичь плотности кода, как у Thumb, и производительности, как у набора команд ARM на 32 битах.

Thumb-2 расширяет как команды ARM, так и команды ***Thumb*** ещё большим количеством команд, включая управление битовым полем, табличное ветвление, условное исполнение. Новый язык ***«Unified Assembly Language» (UAL)*** поддерживает создание команд, как для ARM, так и для Thumb из одного и того же исходного кода. Версии Thumb на ARMv7 выглядят, как код ARM. Это требует осторожности и использования новой команды if-then, которая поддерживает исполнение до 4 последовательных команд испытываемого состояния. Во время компиляции в ARM-код она игнорируется, но во время компиляции в код Thumb-2 генерирует команды.

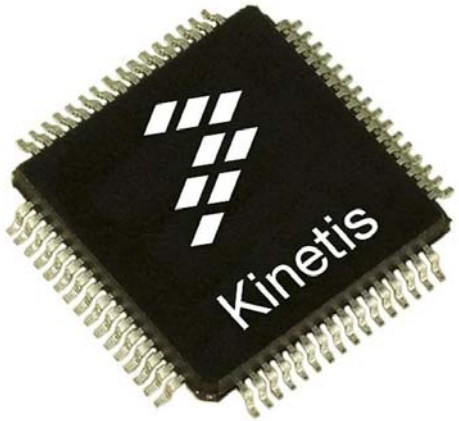
Например: ; if (r0 == r1)

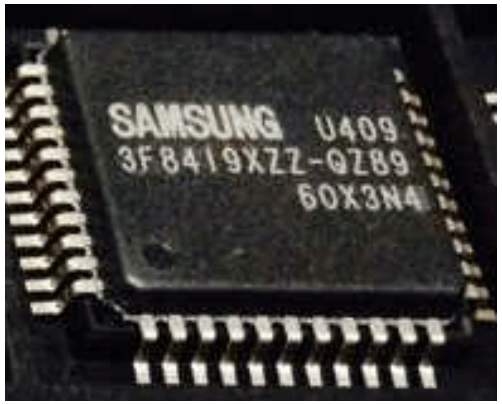
```
CMP r0, r1
ITE EQ ; ARM: no code ... Thumb: IT instruction
; then r0 = r2;
MOVEQ r0, r2 ; ARM: conditional; Thumb: condition via ITE 'T' (then)
; else r0 = r3;
MOVNE r0, r3 ; ARM: conditional; Thumb: condition via ITE 'E' (else)
; recall that the Thumb MOV instruction has no bits to encode "EQ" or "NE"
```

Все кристаллы ARMv7 поддерживают набор команд Thumb-2, а некоторые кристаллы, вроде ***Cortex-m3***, поддерживают только ***Thumb-2***. Остальные кристаллы Cortex и ***ARM11*** поддерживают наборы команд как ***Thumb-2, так и ARM***.

Производители микропроцессоров

Существуют 8-битные, 16-битные и 32-битные микроконтроллеры **PIC фирмы Microchip Technology**, **микроконтроллеры AVR фирмы Atmel** (с 2016 года производятся фирмой Microchip), 16-битные **MSP430 фирмы TI**, а также 32-битные микроконтроллеры архитектуры ARM, которую разрабатывает фирма **ARM Limited** и продаёт лицензии другим фирмам для их производства. Несмотря на популярность в России микроконтроллеров Atmel, PIC, MSP430, на 2009 год мировой рейтинг по объёму продаж выглядел иначе: первое место с большим отрывом занимала **Renesas Electronics**, на втором — **Freescale**, на третьем — **Samsung**, затем шли **Microchip** и **TI**, далее — все остальные.





Производители микропроцессоров

1. Корпорация **Nuvoton Technology (NTC)** основана и зарегистрирована в Тайване. Начав работу в качестве дочерней компании **Winbond Electronics**, **Nuvoton Technology** в 2008 году открыла свой офис и с этого момента присутствует на рынке микроэлектроники с собственным брендом. На фондовой бирже Тайваня (TSE) компания присутствует с осени 2010 года.
Продукция Nuvoton выпускается на фабриках **Winbond и TSMC**. Компании не коснулись санкционные ограничения США и Европы на поставки своей продукции в Россию. Штаб-квартира, дизайн-центр и обе фабрики Nuvoton расположены в непосредственной близости друг от друга в научно-промышленном кластере г. Синьчу (Тайвань). В составе Nuvoton три подразделения: микроконтроллеры и микросхемы для аудиорешения для облачных и вычислительных систем (включая персональные компьютеры), а также собственное производство пластин. В 2020 году Nuvoton приобрела полупроводниковый бизнес Panasonic (контроллеры питания и датчики для автоэлектроники, смартфонов и систем безопасности). Nuvoton основала дочерние компании в США, Китае, Израиле и Индии.

Производители микроконтроллеров RISC-V

Микроконтроллеры, выпущенные в 2017—2019 годах:

Western Digital: SweRV Core (32 бита, 2 ядра, 1,8 ГГц, 28 нм)[31][32]

SiFive: FE310 (32 бита, 1 ядро, 870 МГц — 28 нм, 370 МГц — 55 нм)[23][24]

Kendryte: K210 (64 бита, 2 ядра + нейроускоритель, 600 МГц, 28 нм, 500 мВт)[33][34][35]

GreenWaves: GAP8 (32 бита, 8+1 ядро + нейроускоритель, 250 МГц, 55 нм, 100 мВт)[36]

NXP: RV32M1 (32 бита, 2 гибридных ядра ARM-M4F/RISC-V + ARM-M0+/RISC-V, 48-72 МГц)[37]

WCH: CH572 (60 МГц, корпус QFN28)[38] контроллер BLE + Zigbee + USB + Ethernet + Touchkey

HUAMI: MHS001 Huangshan № 1 (4 ядра, нейроускоритель, 55 нм, 240 МГц)[39] процессор для IoT

GigaDevice: GD32VF103 (1 ядро, 32 бита, 108 МГц, ОЗУ до 32 кБ, ПЗУ до 128 кБ)[40][41] микроконтроллер (не путать с семейством GD32F103).

FADU: Annapurna FC3081/FC3082 (64 бита, многоядерный, 7 нм, 1,7 Вт)[42][43][44] контроллер для NVMe SSD

BitMain: Sophon Edge TPU BM1880 (64 бита, 1 ядро RV64GC 1 ГГц + 2 ядра ARM A53 1,5 ГГц, 2,5 Вт) нейроускоритель 1 TOPS на INT8 для IoT и краевых вычислений

Текон: Дружба (32 бита, 1 ядро, 250 МГц, 28 нм, 0,5 Вт

Производители микроконтроллеров RISC-V

Микроконтроллеры, выпущенные в 2020 году:

ONiO: ONiO.zero (16/32 бита, 1 кБ ПЗУ, 2 кБ ОЗУ, 8/16/32 кБ ППЗУ, 1-24 МГц, 0,36-1,44 Вт, встроенный радиоэлектро генератор на 800/900/1800/1900/2400 МГц) BLE, 802.15.4 UWB[49][50]

WCH: CH32V103 (32 бита, 10/20КБ ОЗУ, 32/64 КБ ППЗУ, до 80 МГц, корпуса LQFP48, QFN48 или LQFP64)[51] универсальный контроллер с USB 2.0, SPI, I2C, GPIO, USART, TouchKey, RTC, TIM, ADC

Миландр: К1986ВК025 (32-битное ядро ВМ-310S CloudBEAR, ОЗУ 112 Кбайт, ППЗУ 256+8 Кбайт, ПЗУ 16 Кбайт, 60 МГц, 90 нм фабрика TSMC, 7 каналов 24-битных метрологических АЦП, сопроцессоров для шифров «Кузнечик», «Магма» и AES, корпус QFN88 10 x 10 мм)

Espressif: ESP32-C3 (32-битное ядро RV32IMC, 400 Кбайт SRAM, 384 Кбайт ПЗУ, 160 МГц, Wi-Fi, Bluetooth LE 5.0, по контактам совместим с ESP8266)[52]

Bouffalo Lab: BL602 и BL604 (32-битный, динамическая частота от 1 МГц до 192 МГц, 276 КБ SRAM, 128 КБ ПЗУ, Wi-Fi, Bluetooth LE)[53]

Cmsemicon: ANT32RV56xx (ядро RV32EC, 48 МГц, 32+8 Кбайт SRAM, 64 Кбайт)[54]

Микроконтроллеры, выпущенные в 2021 году:

Микрон (Россия): МІК32 (32-битное RV32IMC ядро SCR1 Syntacore, 1-32 МГц, фабрика Микрон, ОЗУ 16 КБ, ППЗУ 8 КБ, 64 входа/выхода, АЦП 12 бит 8 каналов до 1 МГц;, ЦАП 12 бит 4 канала до 1 МГц, криптография ГОСТ Р 34.12-2015 «Магма», «Кузнечик» и AES 128). Технологические нормы 180 нм.

в 2022 году компания Espressif Systems сообщила о переходе ESP32 на чипы с RISC-V ядрами (ESP32-C и ESP32-H).

Ссылки

https://habr.com/ru/companies/selectel/articles/542074/
https://ru.wikipedia.org/wiki/ARM

Спасибо за внимание

ЧУ ПО «Социально-технологический колледж»

Преподаватель: Борисов Алексей Альбертович