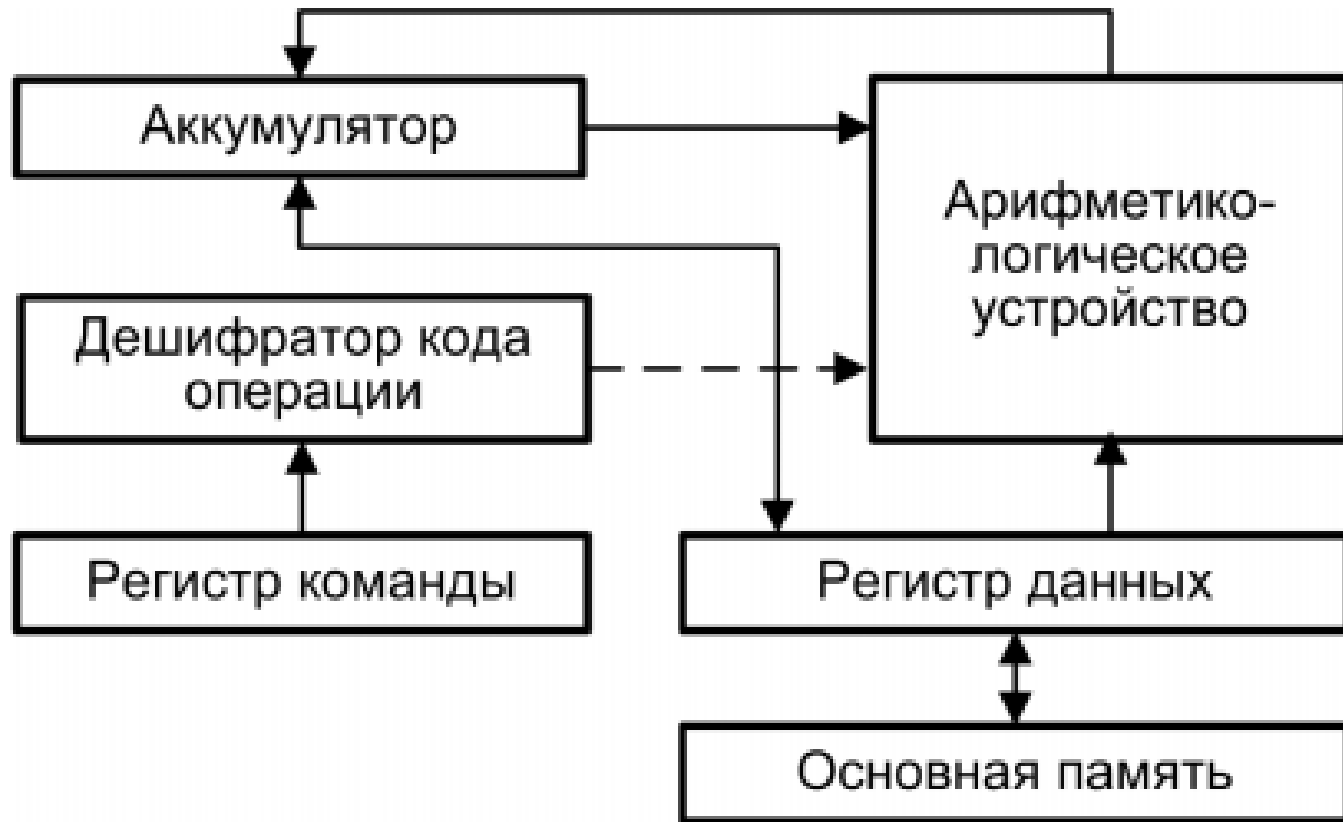


# Классификация вычислительных архитектур

- По месту хранения операндов
  - Аккумуляторная
  - Стековая
  - Регистровая
  - Регистровая с выделенным доступом к памяти
  
- По используемым наборам команд
  - CISC – архитектуры
  - RISC – архитектуры

# Классификация по месту расположения операндов

# Аккумуляторная архитектура



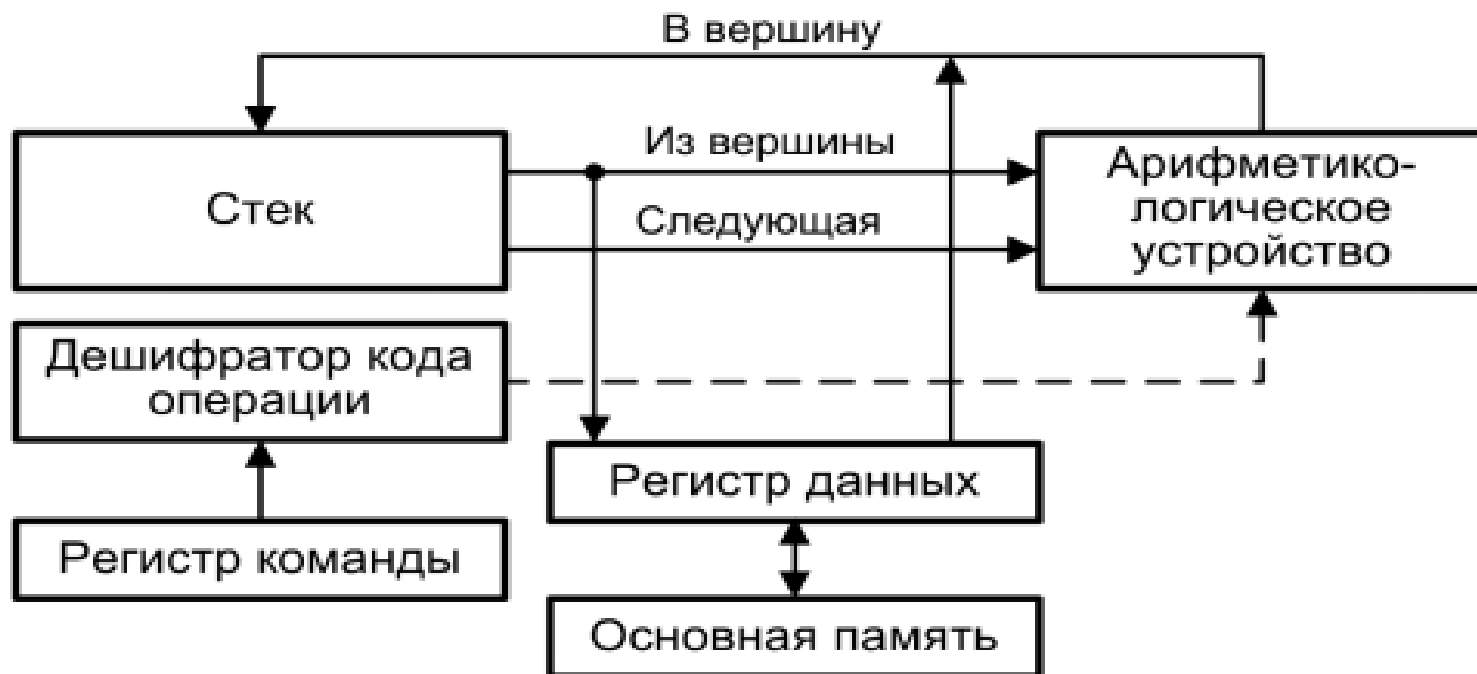
```
load Pa ; загрузка в аккумулятор из ОП
add Pb ; сложение
store Pc ; запись результата в ОП
```

*(IBM 7090, DEC PDP-8 и другие ранние архитектуры)*

# Аккумуляторная архитектура

- Исторически первая архитектура (*IBM 7090, DEC PDP-8*)
- Сначала оба операнда находятся в ОП
- Перед выполнением команды один операнд помещается в регистр аккумулятора (**команда Load**), второй остается в ОП (точнее в промежуточном регистре данных)
- Результат выполнения помещается в аккумулятор, потом в память (**команда Store**)
- Особенности:
  - + короткие команды (адрес одного операнда предопределен - аккумулятор)
  - - частое обращение к памяти (доп. регистров нет)

# Стековая архитектура



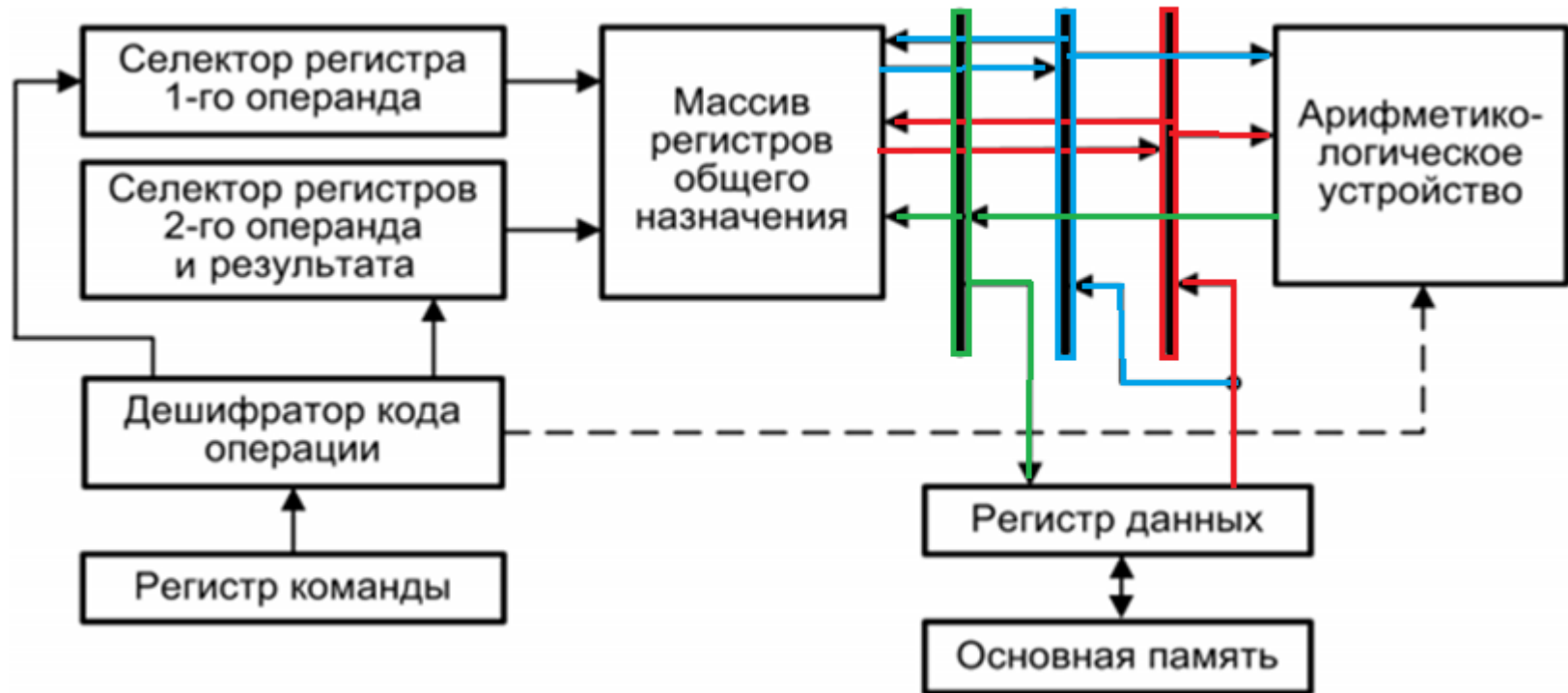
```
push Pa ;запись первого операнда в вершину стека из ОП
push Pb ;запись второго операнда в вершину стека из ОП
add      ;сложение
pop Pc   ;выборка результата из вершины и запись в ОП
```

- Сопроцессоры с плавающей точкой Intel, AMD
- **PicoJava** - микропроцессор для аппаратного выполнения байт кода без JVM (SUN для встраиваемых систем Fujitsu, NEC и Siemens)

# Стековая архитектура

- Регистры процессора образуют стек (верхние ячейки стека могут быть в процессоре, а остальной стек в памяти). Операнды и результат помещаются в стек
- Особенности
  - + компактный машинный код ( в командах отсутствуют адреса операндов )
  - + упрощение способа обращения к подпрограммам и обработки прерываний.
  - + хороша для языков использующих польскую обратную запись Лисп, Пролог
  - + виртуальная машина Java имеет стековую организацию
  - - нет произвольного доступа к памяти, из-за чего компилятору трудно создать эффективный программный код
  - - сложно создать большой регистровый стек в процессоре, а стек в памяти уменьшает быстродействие
- Сложно организовать конвейер

# Регистровая архитектура



- *x8086 и другие архитектуры*

# Регистровая архитектура

- Процессор включает в себя массив регистров, к каждому можно обратиться по номеру (*x8086 и другие архитектуры*)
- Выделяют три подвида команд обработки:
  - регистр-регистр;
  - регистр-память;
  - память-память
  - обращение к памяти из команды

В ALU операнды могут поступать как с регистров, так и с памяти

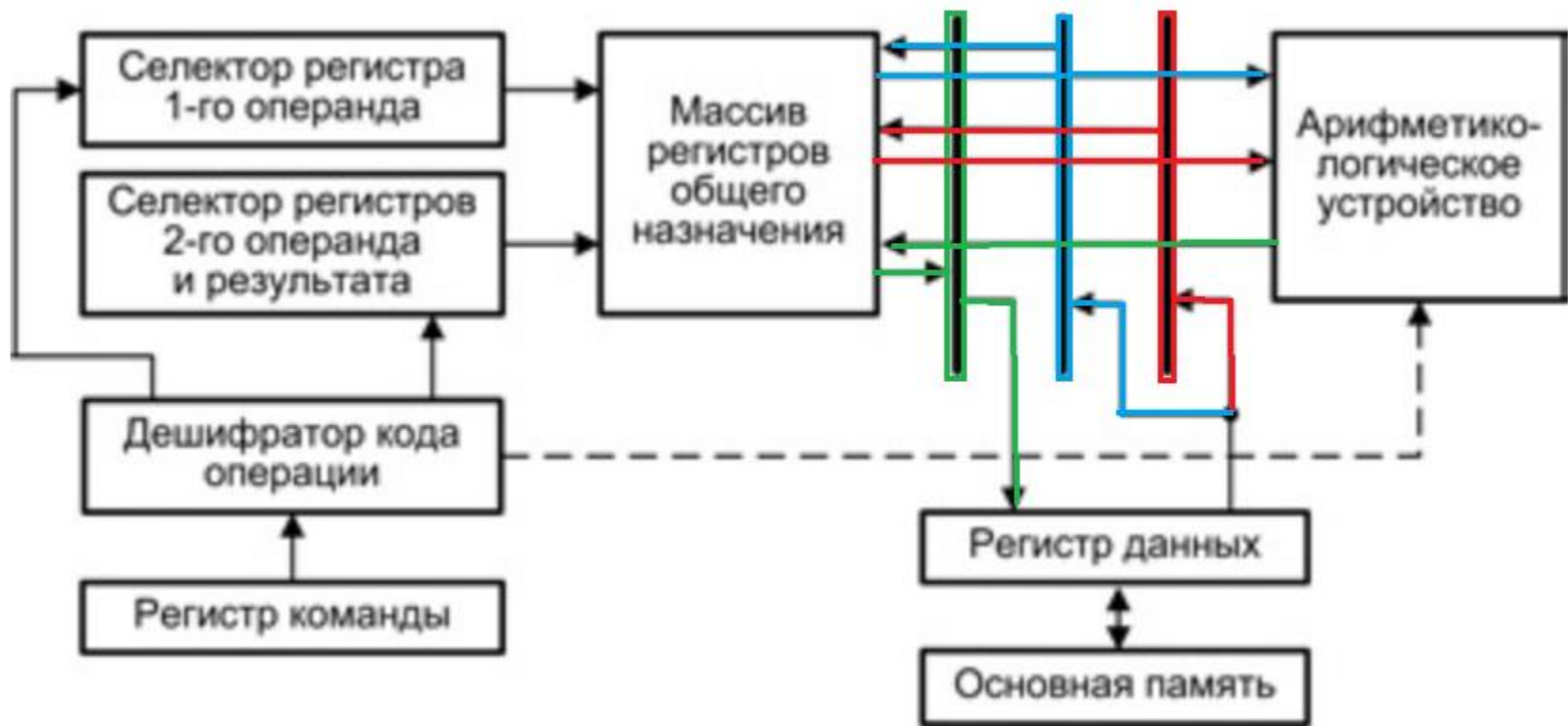
Особенности:

- + выше быстродействие, чем у аккумуляторной (меньше обращений к ОП)
- - команды длиннее чем у аккумуляторной (длиннее код)

- *mov si,offset str1*
- *mov di,offset str2*
- *cld*
- *movsb*



# Архитектура с выделенным доступом к памяти



- Процессоры MIPS, RISC V, ARM и др.

# Архитектура с выделенным доступом к памяти

- Развитие регистровой архитектуры (MIPS, ARM)
- Доступны только команды **регистр-регистр**
- Обращение к памяти только для загрузки и сохранения (*Load/Store architecture*)
- В ALU операнды могут поступать только с регистров.
- Простые, короткие команды фиксированной длины
- Особенности:
  - + Высокая производительность
  - - Менее компактный код (большее количество простых команд)

# Классификация вычислительных архитектур в зависимости от архитектуры набора команд (ISA , instruction set architecture)

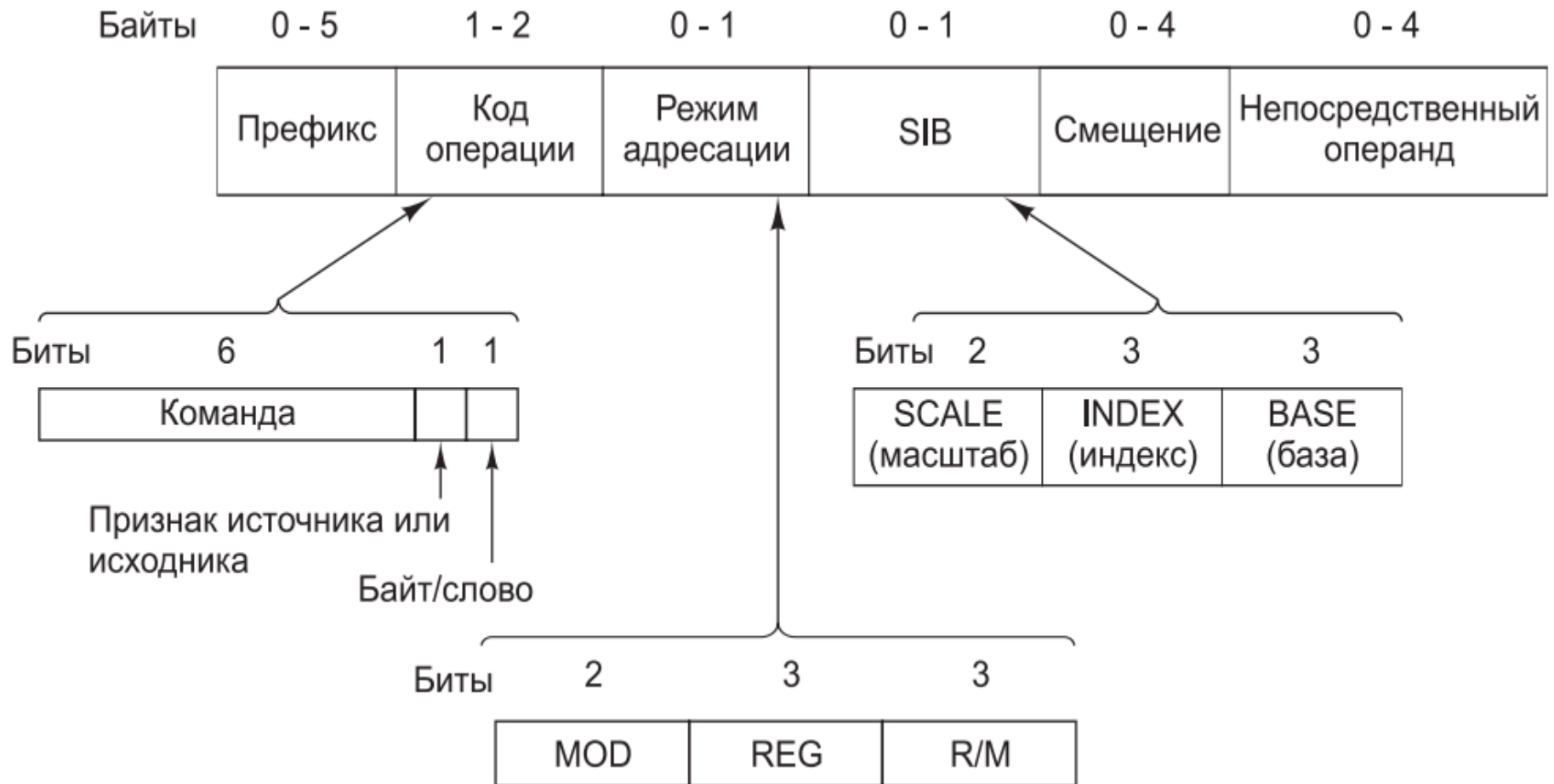
ISA – это то, как видит компьютер программист.  
Она определена набором команд (языком) и местом  
нахождения операндов (регистры и память).

# Архитектура системы команд



# CISC (Complex Instruction Set Computing)

## Формат команд CISC-процессоров Intel



В байтах указана возможная длина полей команды

# CISC архитектура

- Малое количество пользовательских регистров.
- Команды имеют разную длину (1-15 байт) и время выполнения.
- Большое количество режимов адресации (доступ к памяти из команды)
- Двухадресные команды (команды с разрушением)
- В систему команд добавлены «удобные» для программиста сложные команды.
  
- Особенности:
  - +Ускорение разработки программ
  - +Код программы занимает в памяти меньше места
  
  - -Для сложных команд «Медленное» микропрограммное устройство управления.
  - -Сложнее сделать конвейер
  - Сложнее процессор
  - Более низкая скорость выполнения команд
  - Более высокое потребление мощности

# RISC архитектура

## (Reduced instruction set computer)



- Дэвид Паттерсон автор термина RISC
- В проекте Berkeley RISC (1980 - 1984) при исследовании работы CISC процессора Motorola 68000 установил, что:
  - 80% времени работы процессор выполняет 20% реализованных в нём команд (правило Парето)
  - Многие сложные команды оказались мало востребованными и не генерировались компиляторами того времени

# Особенности RISC-процессоров

- Архитектура с выделенным доступом к памяти
- Команды
  - регистр-регистр
  - загрузки/сохранения (*Load/Store*)
- Фиксированная длина команд и меньшее их количество
- Малое количество форматов команд
- Малое количество способов адресации
- Трехадресные команды (без разрушения)
- Расширенный объём регистровой памяти процессора : от 32 и более пользовательских регистров;
- Устройство управления на основе «жесткой логики»

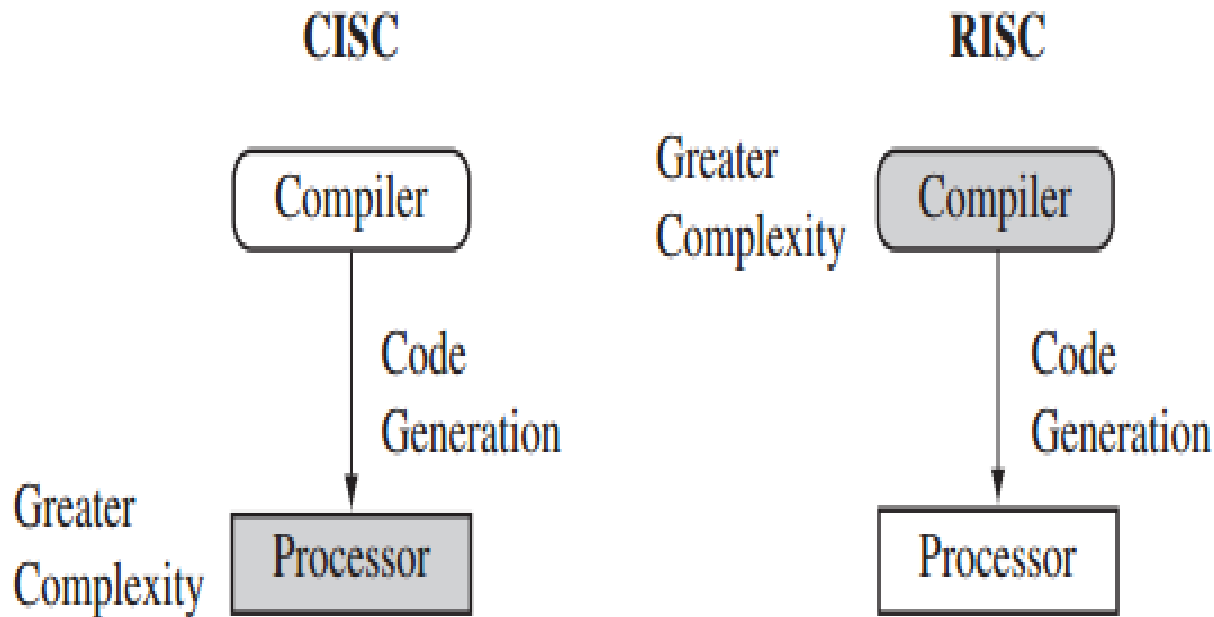


# Форматы команд RISC-процессора MIPS

## (MIPS 32)

Тип	-31- формат (бит) -0-					
R	код операции (6)	rs (5)	rt (5)	rd (5)	shamt (5)	функция (6)
I	код операции (6)	rs (5)	rt (5)	константа (immediate) (16)		
J	код операции (6)	адрес (26)				

# Особенности RISC



- Более сложный компилятор
- Программа занимает больше памяти (на 30%)
- Проще реализовать конвейер
- Проще процессор – меньшее потребление мощности, больше свободной площади на кристалле (батарейный тип)

# Причины перехода от CISC к RISC

- Конвейеризация процессора
- Увеличение объема памяти
- Совершенствование компилятора.

# Гибридные CISC и RISC

- Все x86-процессоры являются CISC-процессорами
- Начиная с Intel486DX, CISC-процессоры имеют RISC-ядро.
- Перед выполнением сложные CISC команды преобразуются в набор RISC микроопераций с помощью встроенного аппаратного декодера/транслятора

# MIPS (Microprocessor without Interlocked Pipeline Stages - микропроцессор без задержки конвейера)

- Архитектура характеризуется:
  - Регистровая архитектура с выделенным доступом к памяти (load, store)
  - RISC – архитектура системы команд
  - Всего 3 формата инструкций.
  - Низкое потребление мощности;
  - Высокая производительность.
- В 2021 разработка архитектуры MIPS прекращена в пользу архитектуры RISC V.



# MIPS

## ■ Область применения

- терминалы оплаты,
- бытовая техника
- коммутаторы и маршрутизаторы ( Cisco, Linksys, ZyXEL и MikroTik)
- кабельные и ADSL-модемы,
- лазерные принтеры, цифровые приставки, роботы.

## ■ На рынке мобильных платформ представлена слабо по двум причинам:

- Большая конкуренция со стороны архитектуры ARM;
- Отсутствие полной поддержки со стороны мобильных ОС

# Набор регистров процессора MIPS

## Набор регистров MIPS

Название	Номер	Назначение
\$0	0	Константный ноль
\$at	1	Временный регистр для нужд ассемблера
\$v0-\$v1	2–3	Возвращаемые функциями значения
\$a0-\$a3	4–7	Аргументы функций
\$t0-\$t7	8–15	Временные переменные
\$s0-\$s7	16–23	Сохраняемые переменные
\$t8-\$t9	24–25	Временные переменные
\$k0-\$k1	26–27	Временные переменные операционной системы (ОС)
\$gp	28	Глобальный указатель (англ.: global pointer)
\$sp	29	Указатель стека (англ.: stack pointer)
\$fp	30	Указатель кадра стека (англ.: frame pointer)
\$ra	31	Регистр адреса возврата из функции

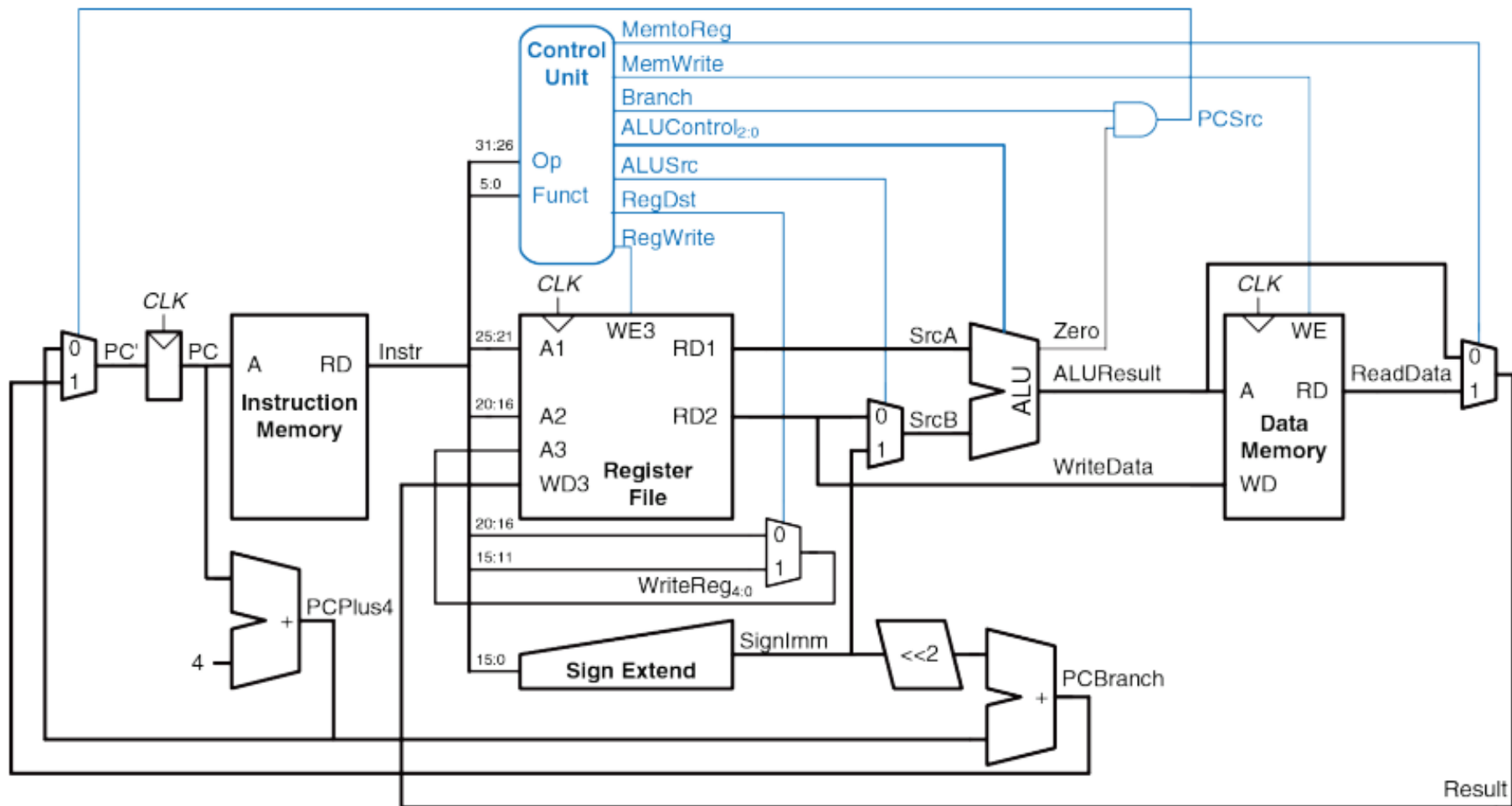
# Форматы команд RISC-процессора MIPS - MIPS32,.

## Три формата команд

Тип	-31- формат (бит) -0-					
R	код операции (6)	rs (5)	rt (5)	rd (5)	shamt (5)	функция (6)
I	код операции (6)	rs (5)	rt (5)	константа (16) <i>(immediate)</i>		
J	код операции (6)	адрес (26)				



# Однотактный процессор MIPS



Выборка команды

Декодирование

Выполнение

Доступ к памяти

Запись результата

Fetch

Decode

Execute

Memory

Writeback

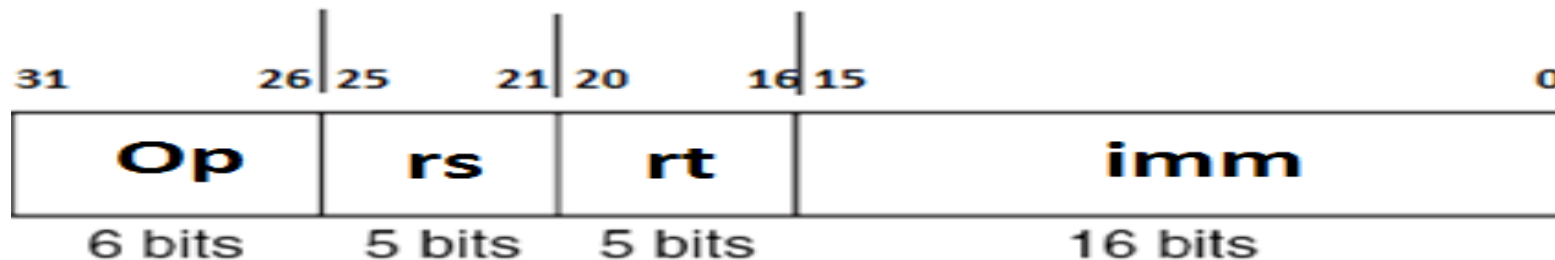
# Команда загрузка слова LW

***lw*** ***\$rt***, *imm*(***\$rs***)

$[rt] = [rs] + \text{Signimm}$

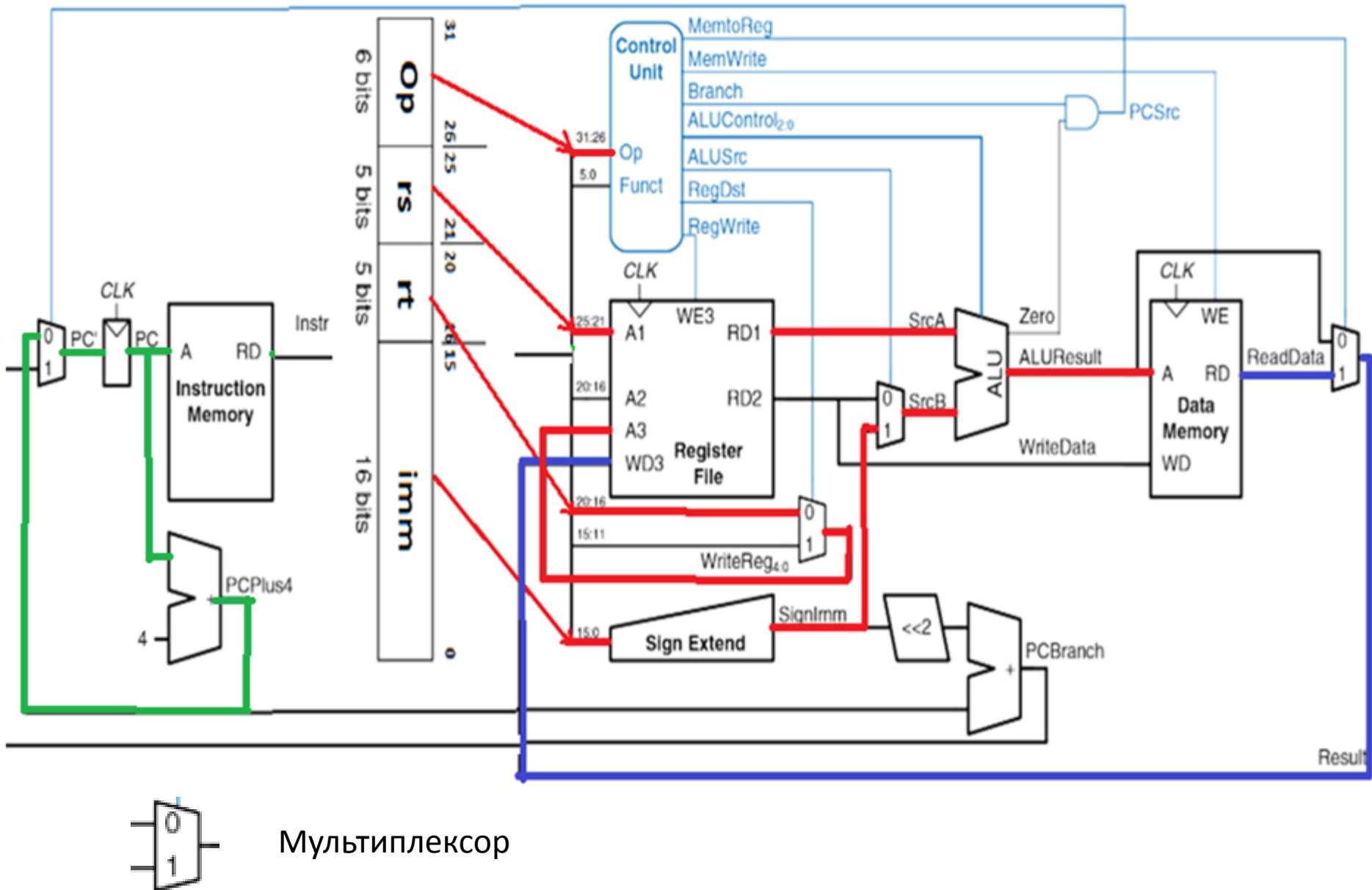
Содержимое ячейки памяти по Адресу  
вычисляемому как сумма содержимого регистра **rs**  
и знакового расширения непосредственного  
операнда ***Signimm*** загрузить в регистр ***rt***

*lw \$s3, 1(\$s4)*



- *imm* (*immediate*) – непосредственный операнд
- *rs* – номер регистра источника
- *rt* – номер временного регистра результата

# Тракт команды LW



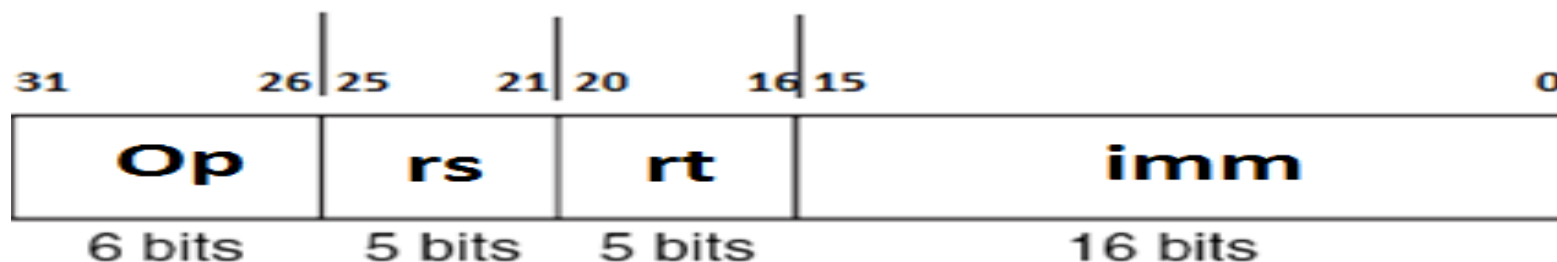
# Команда сохранения слова в память SW

**sw** \$*rt*, *imm*(\$*rs*)

[ *rs* ] + Signimm = [ *rt* ]

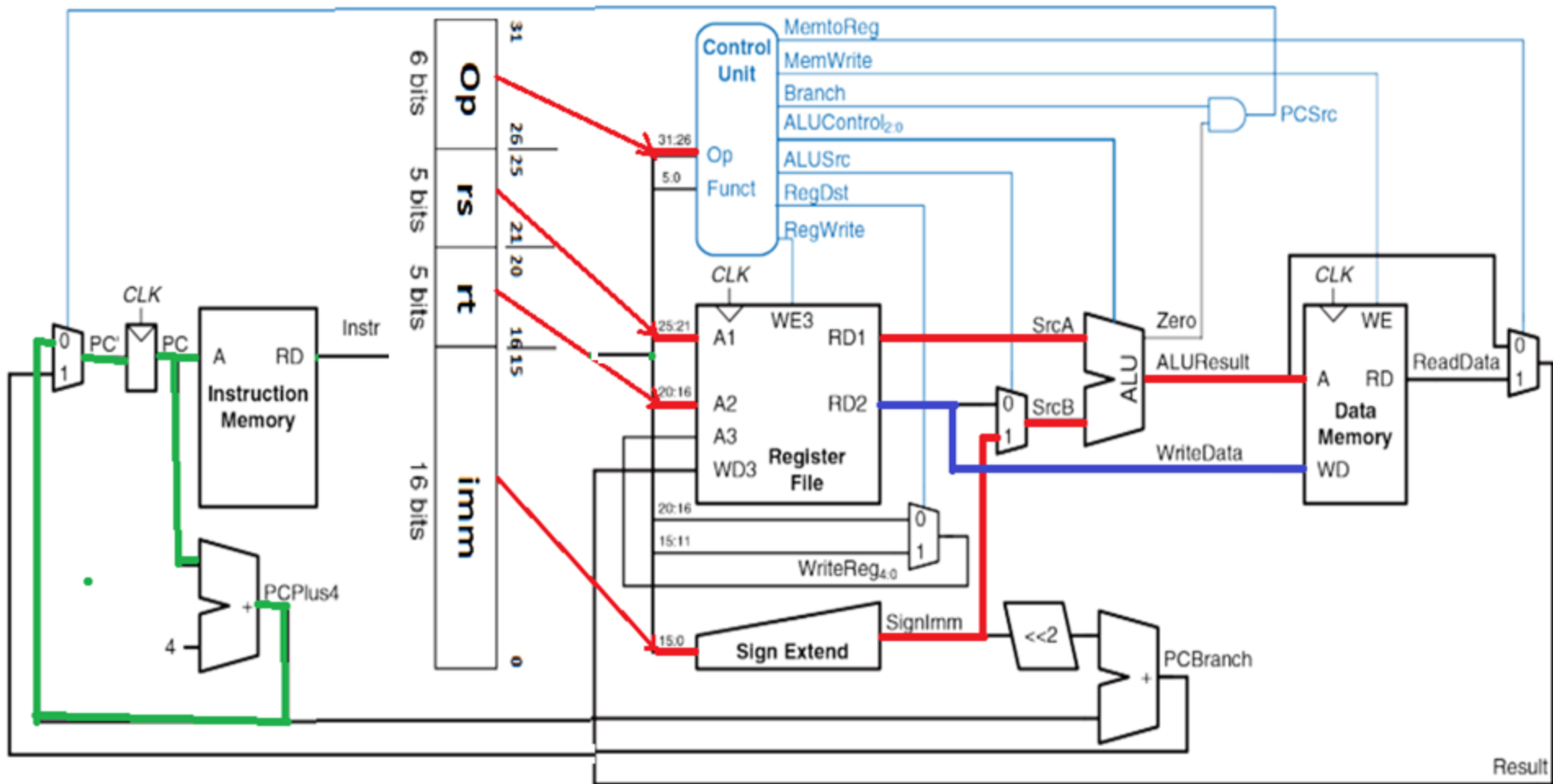
*sw* \$s3, 1(\$s4)

В ячейку памяти по Адресу вычисляемому как сумма содержимого регистра **rs** и знакового расширения непосредственного операнда **Signimm**, сохранить содержимое регистра **rt**



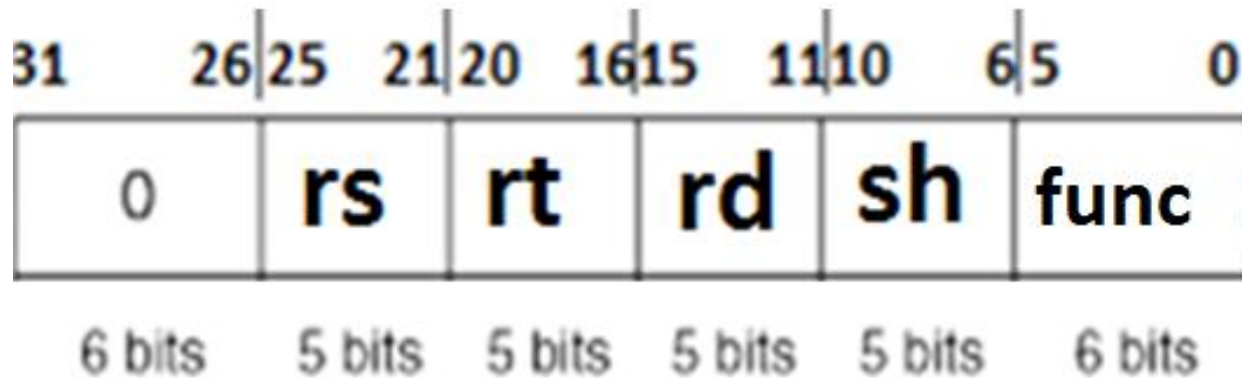
- *imm* (*immediate*) – непосредственный операнд
- *rs* – регистр источник
- *rt* – временный регистр

# Тракт команды Sw

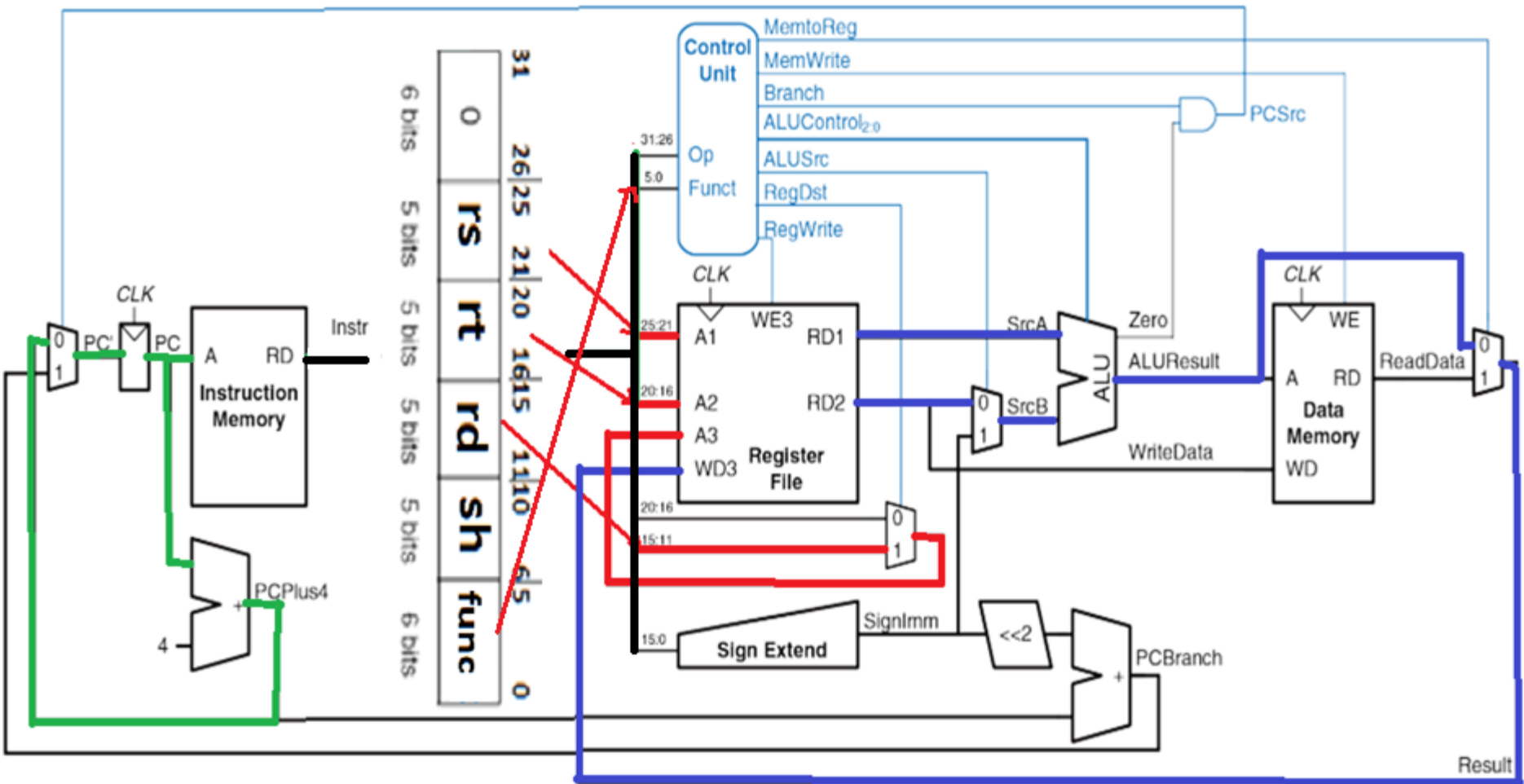


## Сложение add

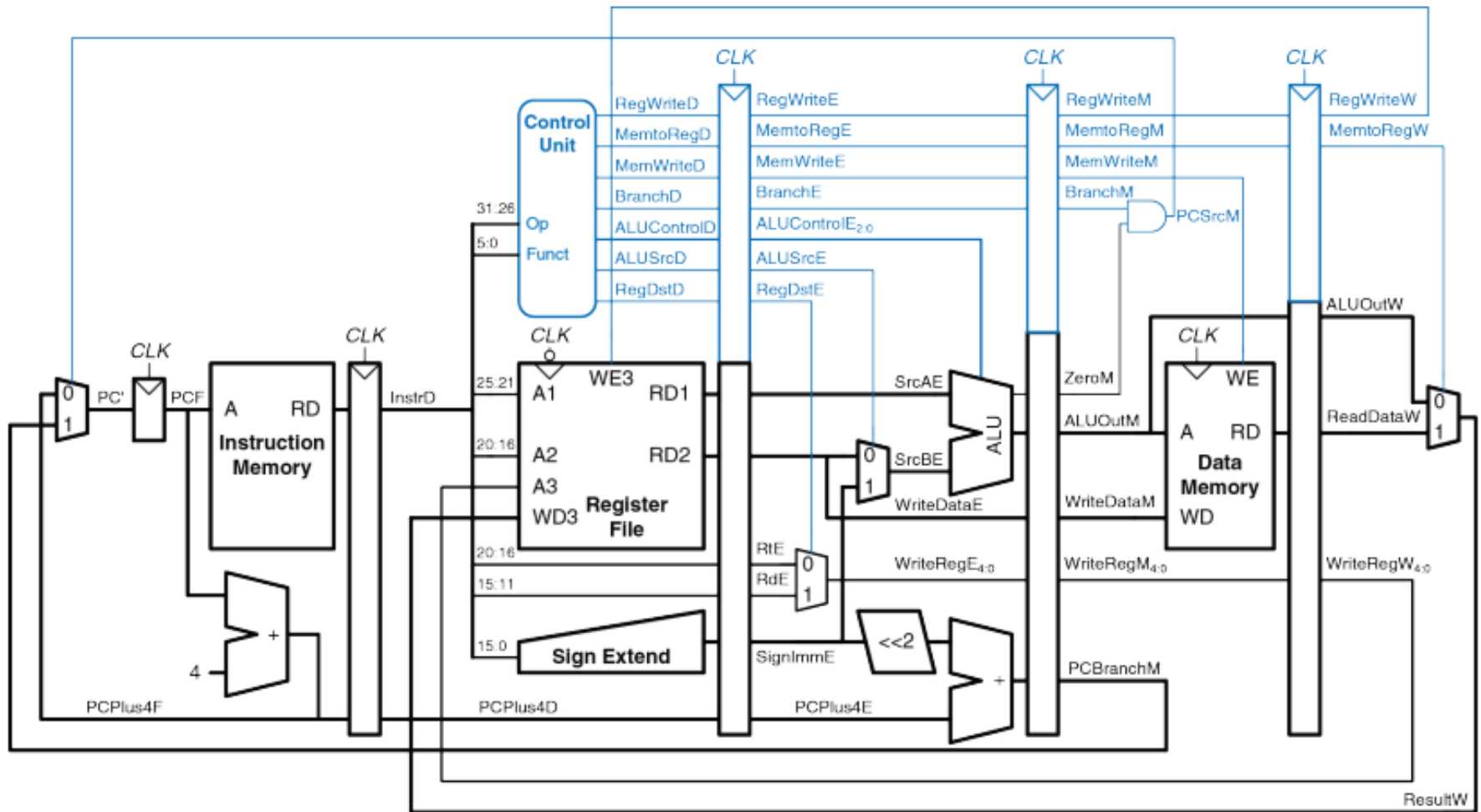
- add \$rd, \$rs, \$rt      add \$s0, \$s1, \$s2
- $[rd] = [rs] + [rt]$



# Тракт команды add



# Конвейерный процессор MIPS





# Архитектура RISC-V

- 2010г - представлена в Калифорнийском университете в Беркли разработчиком Кресте Асановичем
- 2015 - « RISC-V International» некоммерческая организация по продвижению архитектуры RISC-V.
- <https://riscv.org/>



**Крсте Асанович** начинал создавать RISC-V как летний проект. Он работает профессором информатики в Калифорнийском университете в Беркли и занимает пост председателя правления некоммерческой организации RISC-V International, ранее известной как RISC-V Foundation. Он также является соучредителем SiFive, компании, которая разрабатывает и продает чипы, платы и дополнительные средства разработки для RISC-V.

# Набор регистров

## Набор регистров RISC-V

Название	Номер	Назначение
zero	x0	Константа нуля
ra	x1	Адрес возврата (от англ. <i>return address</i> )
sp	x2	Указатель стека (от англ. <i>stack pointer</i> )
gp	x3	Глобальный указатель (от англ. <i>global pointer</i> )
Tp	x4	Указатель потока (от англ. <i>thread pointer</i> )
t0–t2	x5–x7	Временные переменные
s0/fp	x8	Сохраняемая переменная / Указатель фрейма стека
s1	x9	Сохраняемая переменная
a0–a1	x10–x11	Аргументы функций / Возвращаемые значения
a2–a7	x12–x17	Аргументы функций
s2–s11	x18–x27	Сохраняемые переменные
t3–t6	x28–x31	Временные переменные

# Регистры с плавающей точкой

32 дополнительных регистра с плавающей точкой			
f0–7	ft0–7	Floating-point temporaries	Вызывающий
f8–9	fs0–1	Floating-point saved registers	Вызываемый
f10–11	fa0–1	Floating-point arguments/return values	Вызывающий
f12–17	fa2–7	Floating-point arguments	Вызывающий
f18–27	fs2–11	Floating-point saved registers	Вызываемый
f28–31	ft8–11	Floating-point temporaries	Вызывающий

# Формат команд

Тип	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Регистр/ регистр	funct7							rs2				rs1				funct3			rd				код операции				1	1				
С операндом	±	imm[10:0]											rs1				funct3			rd				код операции				1	1			
С длинным операндом	±	imm[30:12]																			rd				код операции				1	1		
Сохранение	±	imm[10:5]					rs2				rs1				funct3			imm[4:0]				код операции				1	1					
Ветвление	±	imm[10:5]					rs2				rs1				funct3			imm[4:1]			[11]	код операции				1	1					
Переход	±	imm[10:1]										[11]	imm[19:12]							rd				код операции				1	1			

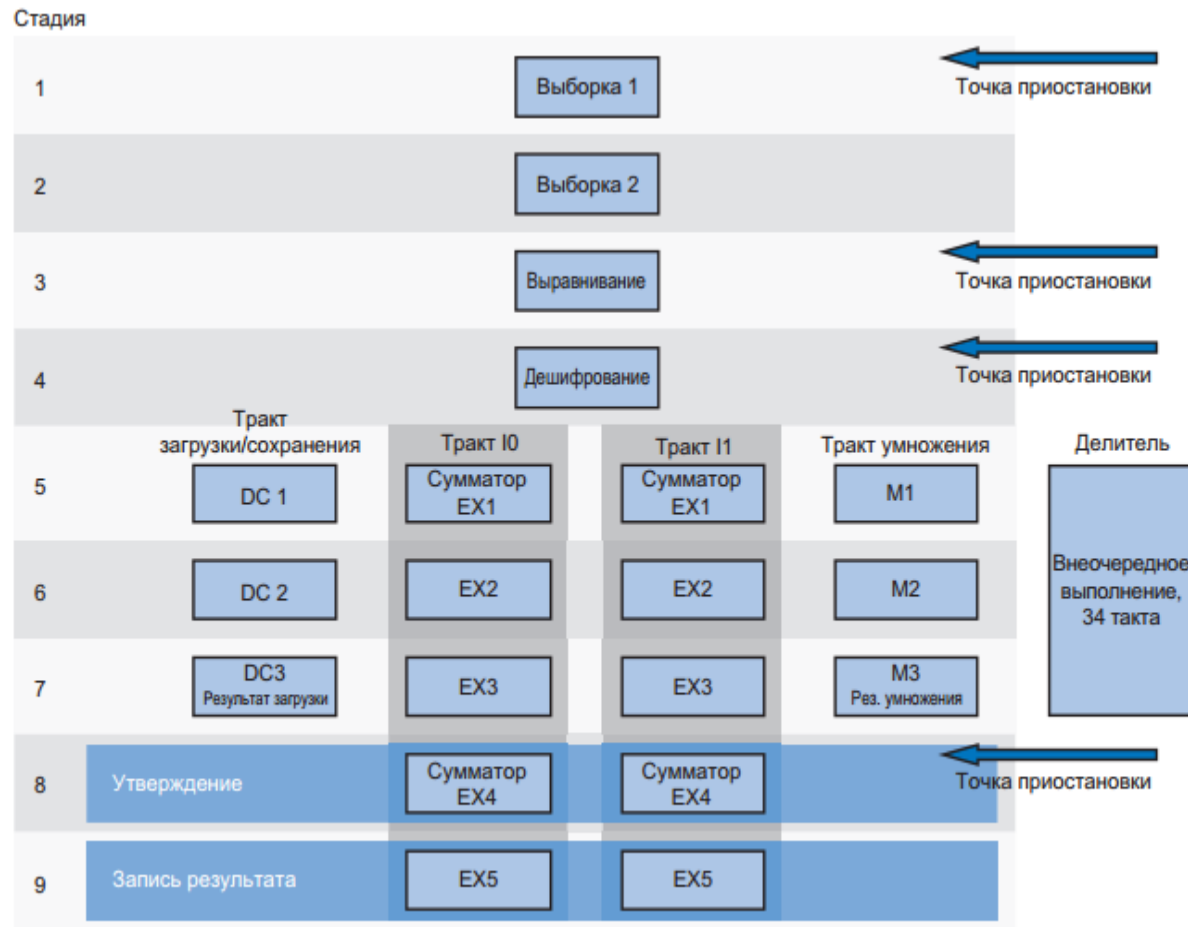
- Шесть форматов команд
- Расположение битов констант в командах сделано для минимизации аппаратных затрат(мультиплексоров, линий связи и.т.д) и упрощает процесс извлечения констант.

# Базовые наборы команд и их расширения

Сокращение	Наименование	Версия	Статус
Базовые наборы			
RV32I	Базовый набор с целочисленными операциями, 32-битный	2.1	Ratified
RV64I	Базовый набор с целочисленными операциями, 64-битный	2.1	Ratified
RV32E	Базовый набор с целочисленными операциями для <a href="#">встраиваемых систем</a> , 32-битный, 16 регистров	1.9	Draft
RV128I	Базовый набор с целочисленными операциями, 128-битный	1.7	Draft
Часть 1 Стандартные непривилигерованные наборы команд			
M	Целочисленное умножение и деление (Integer Multiplication and Division)	2.0	Ratified
A	<a href="#">Атомарные операции</a> (Atomic Instructions)	2.1	Ratified
F	Арифметические операции с плавающей запятой над числами одинарной точности (Single-Precision Floating-Point)	2.2	Ratified
D	Арифметические операции с плавающей запятой над числами двойной точности (Double-Precision Floating-Point)	2.2	Ratified
Q	Арифметические операции с плавающей запятой над числами четверной точности	2.2	Ratified
C	Сокращённые имена для команд (Compressed Instructions)	2.2	Ratified
Counters	Инструкции для счетчиков производительности и таймеров – наборы <b>Zicntr</b> и <b>Zihpm</b>	2.0	Draft
L	Арифметические операции над десятичными числами с плавающей запятой (Decimal Floating-Point)	0.0	Open
B	<a href="#">Битовые операции</a> (Bit Manipulation)	0.36	Open
J	<a href="#">Двоичная трансляция</a> и поддержка <a href="#">динамической компиляции</a> (Dynamically Translated Languages)	0.0	Open
T	<a href="#">Транзакционная память</a> (Transactional Memory)	0.0	Open

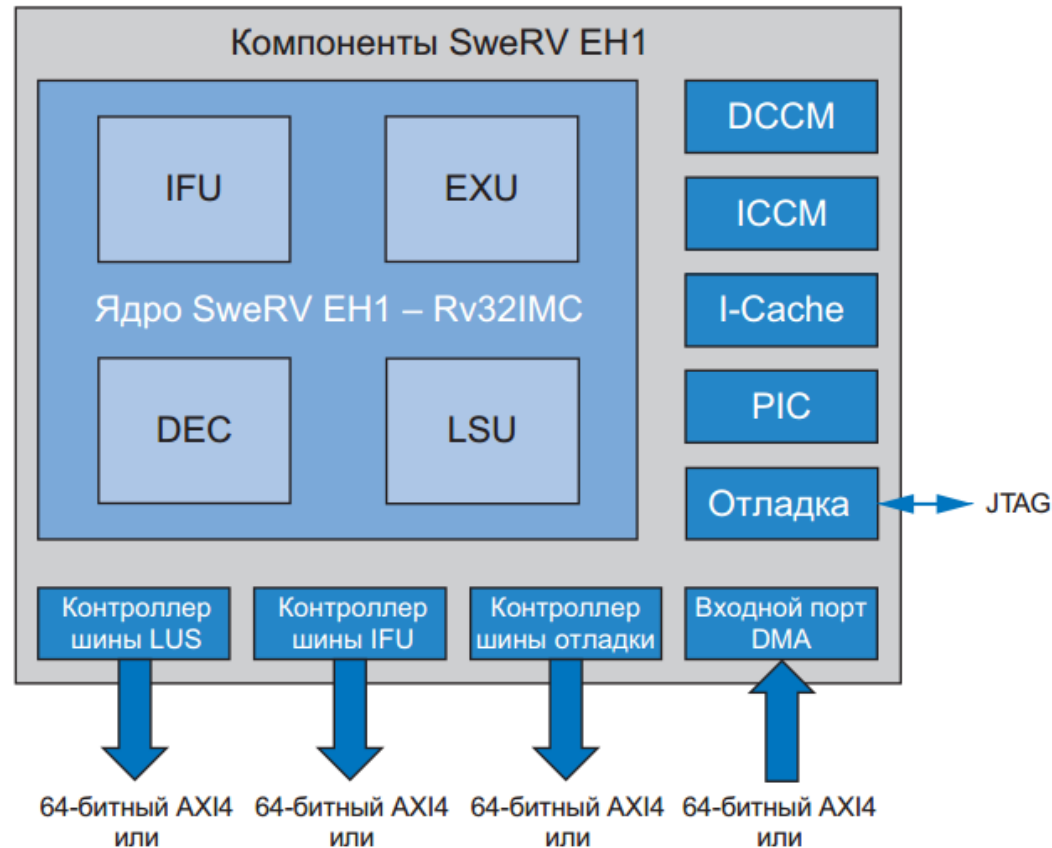
В зависимости от назначения может использоваться базовый набор команд + нужное расширение и к ней требуемая архитектура процессора

# RISC V ядро SweRVEH1 от Western Digital Corporation



**9-стадийный конвейер ядра SweRV EH1**

# RISC V ядро SweRVEH1 от Western Digital Corporation



- IFU – блок извлечения инструкций
- EXU – блок исполнения
- DEC – блок декодирования
- LSU – блок загрузки сохранения
- DCCM, ICCM – кэш данных

# IP- ядра Risc V в России

- Российские компании предлагающие готовые IP-ядра по микроархитектуре Risc V:
- **CloudBEAR** - BM-310, BI-350, BI-651, BI-671
  - [Миландр](#): K1986BK025 (32-битное ядро BM-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)
- **Syntacore** - семейство SCR
  - [Микрон](#) (Россия): MIK32 (32-битное RV32IMC ядро **SRC1** 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](#))



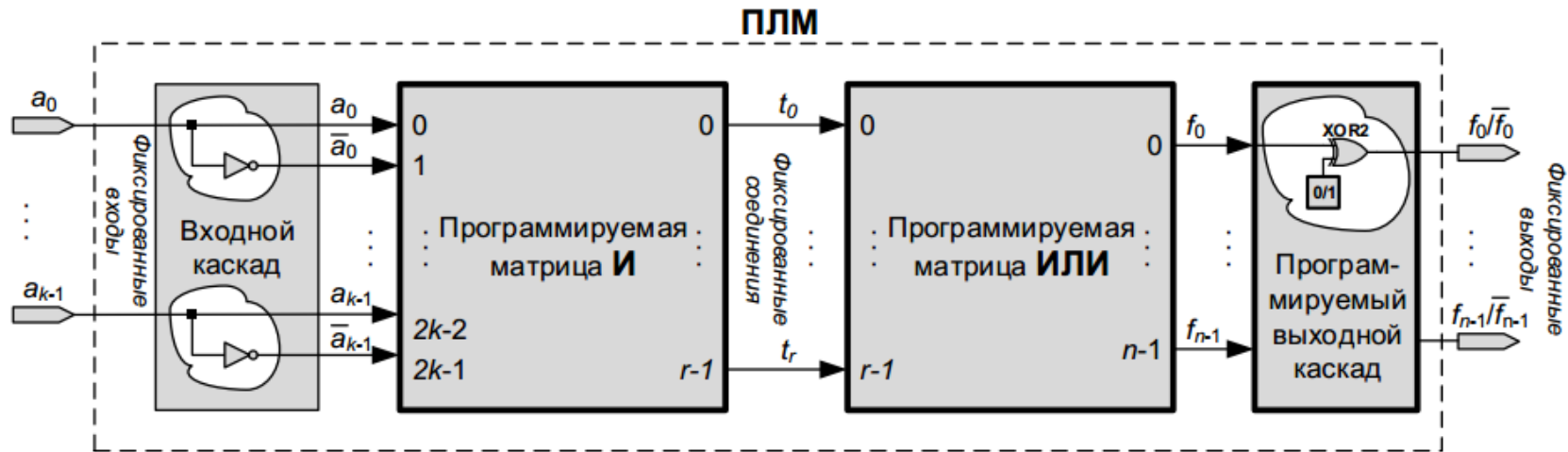
# IP (Intellectual Property) -ядра

- **IP– ядра** сложные готовые блоки для проектирования микропроцессоров (микросхем)
- Различают три основных класса блоков:
  - **программные IP-ядра** ( *soft blocks*) — блоки, реализованные на языке описания аппаратуры;
  - **схемотехнические IP-ядра** ( *firm blocks*) — блоки, реализованные в виде принципиальной схемы;
  - **физические IP-ядро** ( *hard blocks*) — блоки, реализованные на аппаратном уровне в виде законченной топологии.

■

**Как программно перенести проект Risc V**

# Программируемые логические матрицы



ПЛМ представляют собой набор базовых логических элементов :

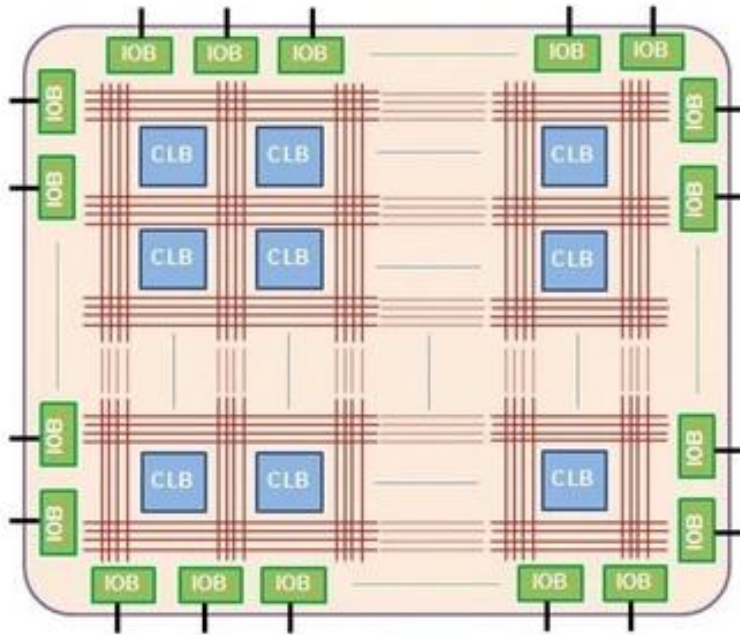
**Матрицу элементов И, Матрицу элементов ИЛИ, Матрицу элементов НЕ**

Элементы и связи между ними программируются с помощью языка Verilog и VHDL

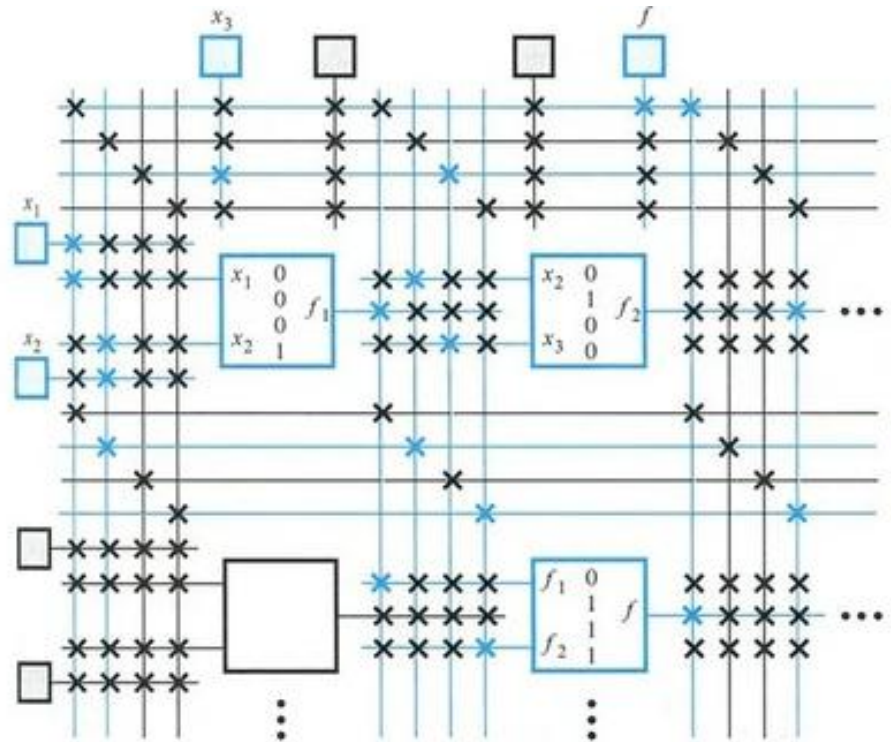
Пример регистра на Verilog

```
module flop(input  logic      clk,
            input  logic [3:0] d,
            output logic [3:0] q);
    always_ff @(posedge clk)
        q <= d;
endmodule
```

# FPGA — Field-Programmable Gate Array



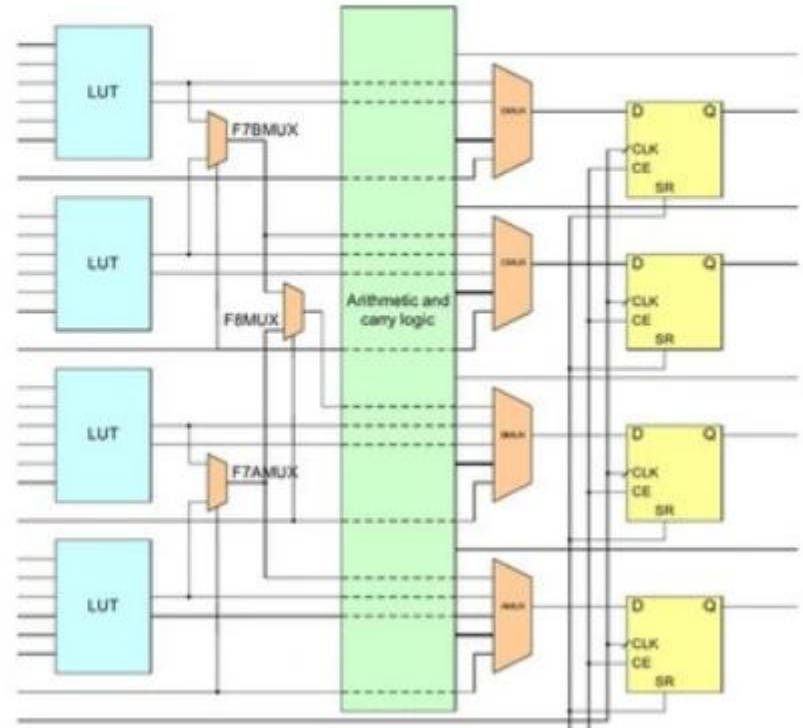
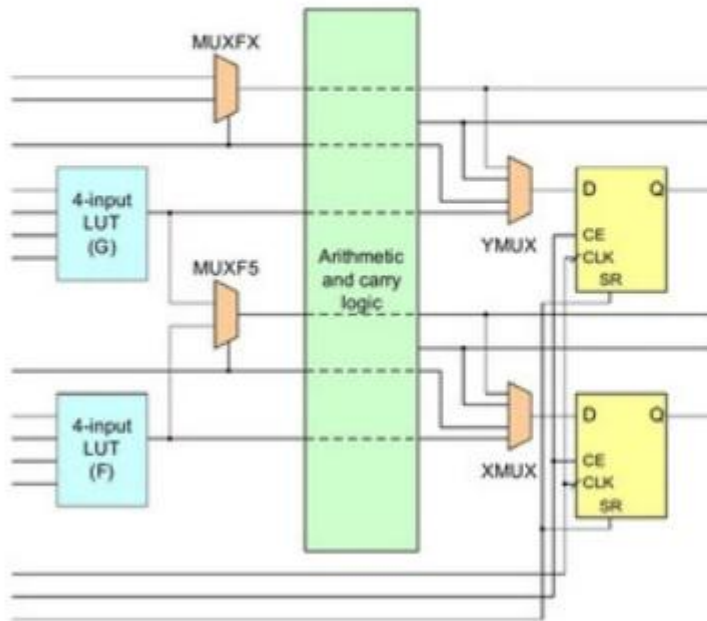
Общая схема FPGA



Секция программируемой FPGA

- CLB – конфигурируемые логические блоки
- IOB – блоки ввода вывода

# Начинка конфигурируемого логического блока (CLB)



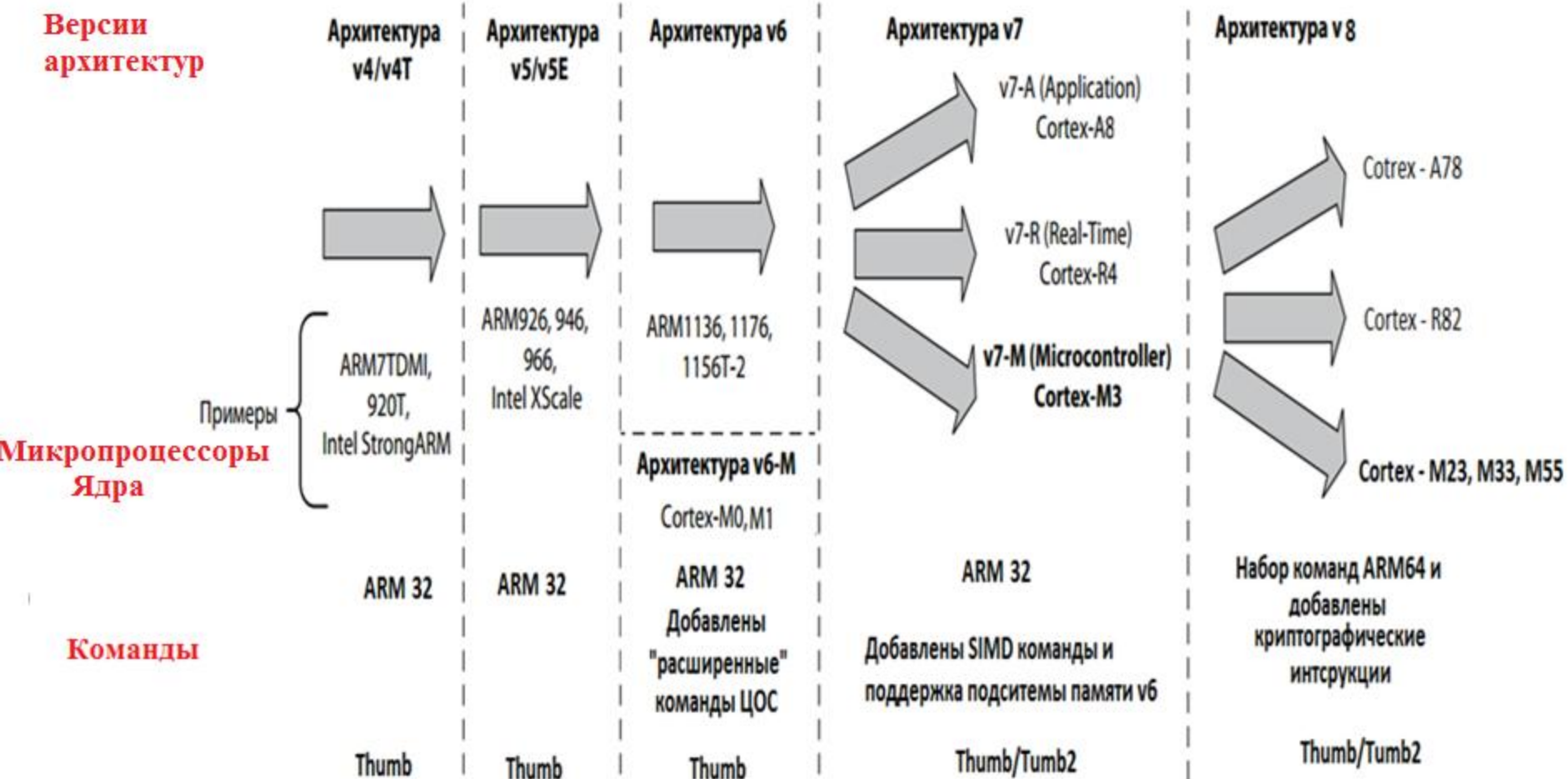
- LUT — Look-Up Tables. реализует любую логическую функцию используя в качестве данных операнды с входов.
- Специализированные управляемые пользователем мультиплексоры (MUX) для комбинационной логики.
- Блок арифметической логики, который позволяет делать суммирование и умножение операндов.
- Несколько триггеров для хранения информации (желтые блоки)

# **ARM (Advanced RISC Machine)**

# ARM

- ARM (Advanced RISC Machine) 1985
- 90% рынка мобильных телефонов и встраиваемых систем
- Низкое потребление
- Разрабатываемы ядра продаются в виде интеллектуальной собственности (IP intellectual property) другим фирмам для изготовления.
- Профиль А – для высокопроизводительных платформ.
- Профиль R – для многофункциональных встраиваемых систем, работающих в режиме реального времени.
- Профиль М – для встраиваемых систем на базе микроконтроллеров (только набор Thumb).
-

# Эволюция архитектур ARM





# Регистры процессора + cpsr

## Набор регистров ARM

Название	Назначение
R0	Аргумент, возвращаемое значение, временная переменная
R1–R3	Аргумент, временная переменная
R4–R11	Сохраненная переменная
R12	Временная переменная
R13 (SP)	Указатель стека
R14 (LR)	Регистр связи
R15 (PC)	Счетчик команд

r0...r12 – регистры общего назначения;

r13 –SP (Stack Pointer) –указатель стека.

r14 – LR (Link Register) – регистр связи: при вызове подпрограммы адрес возврата автоматически запоминается в r14,.откуда затем считывается при возврате;

r15 – PC (Programm Counter) – счетчик команд;

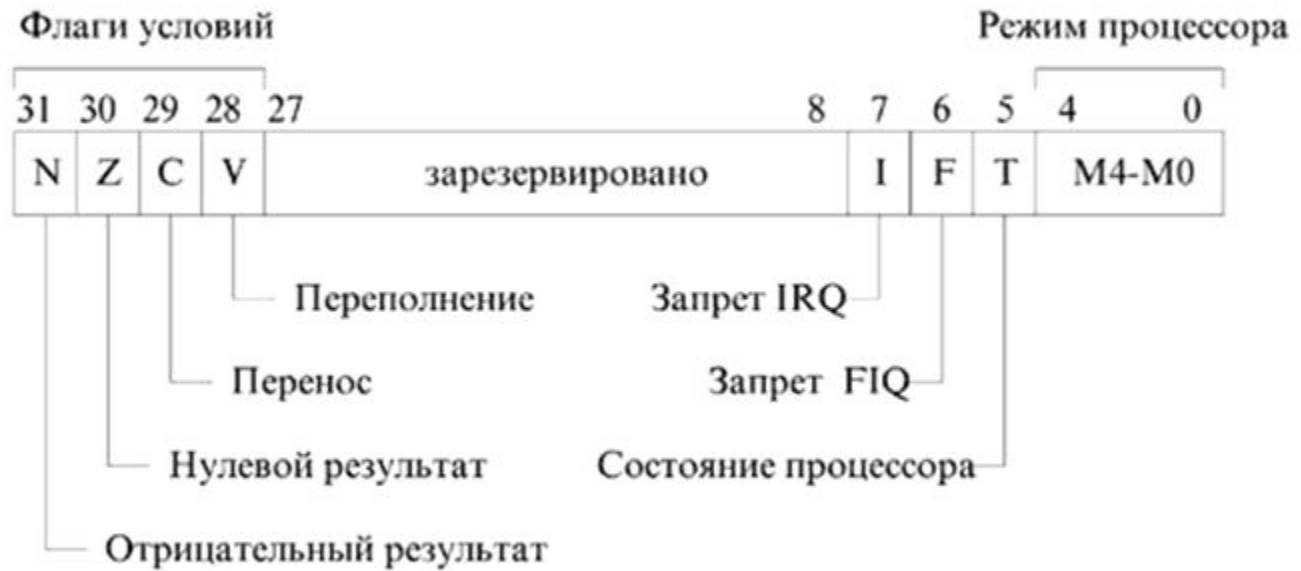
Всего 16 Регистров + Регистр Текущего Статуса Программы

**Current Program Status Register (cpsr)**

**Для ARM 64 – 32 регистра**

# Регистр текущего статуса программы

## Регистр состояния CPSR



T – задаёт состояние процессора: ARM (T = 0) или THUMB (T = 1);

# Режимы работы процессора

## Режимы выполнения процессора ARM

Режим	CPSR <sub>4:0</sub>
Пользователя	10000
Супервизора	10011
Аварийный (Ошибка доступа к памяти)	10111
Неопределенный (Не допустимая инструкция)	11011
Прерывания (IRQ)	10010
Быстрого прерывания (FIQ)	10001

# Режимы работы процессора


- Режимы задаются в регистре CPSR.
- При запуске процессор начинает работу в режиме Supervisor.
- Переход в режим User реализуется путём записи в регистр CPSR значения 10000. Обратное переключение из режима User в Supervisor производится только при поступлении команды программного прерывания SWI.
- Таким образом, обеспечивается доступ пользователя к ресурсам операционной системы.
- Переход в режимы IRQ, FIQ, Аварийный, Ошибка Неопределенный - осуществляется автоматически при поступлении соответствующих запросов или событий.

# Регистры режимов при выполнении команд набора ARM

Режим пользователя	Режим быстрого прерывания	Супервизорный режим	Аварийный режим	Режим прерывания	Неопределенный режим
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

Регистры статуса программы в состоянии ARM

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 - банкированный регистр

В каждом режиме отдельные регистры дублируются/банкируются

# Банки регистров

- В зависимости от режима процессор работает с отдельным набором (банком) регистров. При этом некоторые регистры используются во всех режимах, а содержимое некоторых регистров необходимо сохранять при переходе из режима в режим.
- Для этой цели используются дополнительные банки регистров (регистры банкируются). Например, указатель стека, регистр адреса возврата сохраняются в текущем банке и при переходе в другой режим вместо них будут использоваться аналогичные регистры, но с другого банка.
- Счетчик команд не банкируется.
- При изменении режима работы содержимое CPSR копируется в SPSR (Saved Programm Status Register) , откуда затем считывается при возврате.
- Запись в CPSR в режиме пользователя запрещена.

# Режим быстрого прерывания

- При возникновении прерывания обработчик должен позаботиться о сохранении нужных регистров процессора.
- На это необходимо дополнительное время.
- При переходе в режим быстрого прерывания текущее состояние регистров процессора R8-R13 остается в текущем банке, а обработчик прерывания начинает использовать новые регистры R8\_fig – R13\_fig из банка режима быстрого прерывания.
- Тем самым время на сохранение регистров не требуется.

# Основные форматы команд ARM

31	2827	1615				87				0				Тип команды								
Cond	0 0	I	Код операции		S	Rn	Rd	Операнд2								Data processing / PSR Transfer						
Cond	0 0 0 0 0 0				A	S	Rd	Rn	RS	1 0 0 1		Rm	Multiply									
Cond	0 0 0 0 1			U	A	S	RdHi	RdLo	RS	1 0 0 1		Rm	Long Multiply									
Cond	0 0 0 1 0			B	0 0	Rn	Rd	0 0 0 0	1 0 0 1		Rm	Swap										
Cond	0 1	I	P	U	B	W	L	Rn	Rd	Смещение								Load/Store Byte/Word				
Cond	1 0 0		P	U	S	W	L	Rn	Список регистров										Load/Store Multiple			
Cond	0 0 0		P	U	1	W	L	Rn	Rd	Смещение <sub>1</sub>	1	S	H	1	Смещение <sub>2</sub>	Halfword transfer: Immediate offset						
Cond	0 0 0		P	U	0	W	L	Rn	Rd	0 0 0 0	1	S	H	1	Rm	Halfword transfer: Register offset						
Cond	1 0 1		L	Смещение												Branch						
Cond	0 0 0 1			0 0 1 0			1 1 1 1		1 1 1 1		1 1 1 1		0 0 0 1		Rn	Branch Exchange						
Cond	1 1 0		P	U	N	W	L	Rn	CRd	CPNum	Смещение								Coprocessor data transfer			
Cond	1 1 1 0			Op1			CRn	CRd	CPNum	Op2	0	CRm	Coprocessor data operation									
Cond	1 1 1 0			Op1		L	CRn	Rd	CPNum	Op2	1	CRm	Coprocessor register transfer									
Cond	1 1 1 1			Номер прерывания												Software interrupt						



# ARM использует предикатное выполнение ветвлений

Мнемонические обозначения условий

cond	Мнемоника	Название	CondEx
0000	EQ	Равно	$Z$
0001	NE	Не равно	$\bar{Z}$
0010	CS/HS	Флаг переноса поднят / беззнаковое больше или равно	$C$
0011	CC/LO	Флаг переноса сброшен / беззнаковое меньше	$\bar{C}$
0100	MI	Минус / отрицательное	$N$
0101	PL	Плюс / положительное или ноль	$\bar{N}$
0111	VS	Переполнение / флаг переполнения поднят	$V$
1000	VC	Переполнения нет / флаг переполнения сброшен	$\bar{V}$
1001	HI	Беззнаковое больше	$\bar{Z}C$
1010	LS	Беззнаковое меньше или равно	$Z \text{ OR } \bar{C}$
1011	GE	Знаковое больше или равно	$\bar{N} \oplus \bar{V}$
1100	LT	Знаковое меньше	$N \oplus V$
1101	GT	Знаковое больше	$\bar{Z}(\bar{N} \oplus \bar{V})$
1110	LE	Знаковое меньше или равно	$Z \text{ OR } (N \oplus V)$
	AL (или пусто)	Всегда / безусловно	Игнорируется

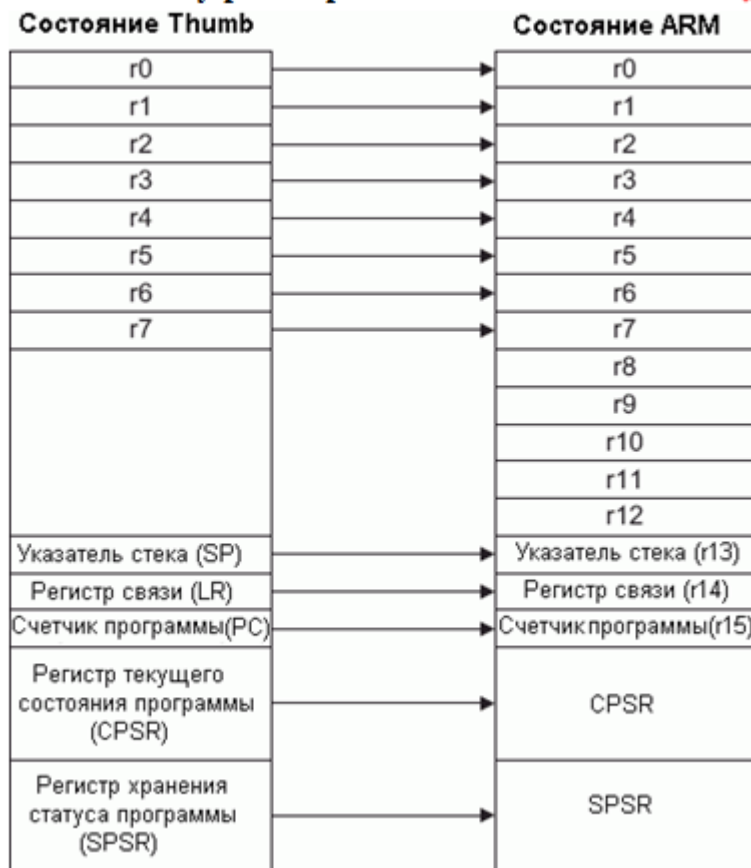
- Если пусто, то поле cond игнорируется

# Сжатые команды Thumb и Thumb2

- Thumb
  - 16-битовые команды, повторяют обычные команды ARM, но с рядом ограничений:
    - могут обращаться только к восьми младшим регистрам;
    - один и тот же регистр играет роль источника и приемника (два адреса);
    - поддерживаются более короткие непосредственные операнды;
    - отсутствует условное выполнение (отсутствует поле cond).
- Thumb2
  - Смесь 16- и 32-битных инструкций.
  - Инструкции thumb-2 дают улучшение :
    - плотности кода на 26% по сравнению с 32-битными инструкциями ARM
    - производительности на 25% по сравнению с 16-битными инструкциями Thumb.
    - некоторые команды набора Thumb2 содержат поле cond.

# Соотношения между наборами Thumb и ARM

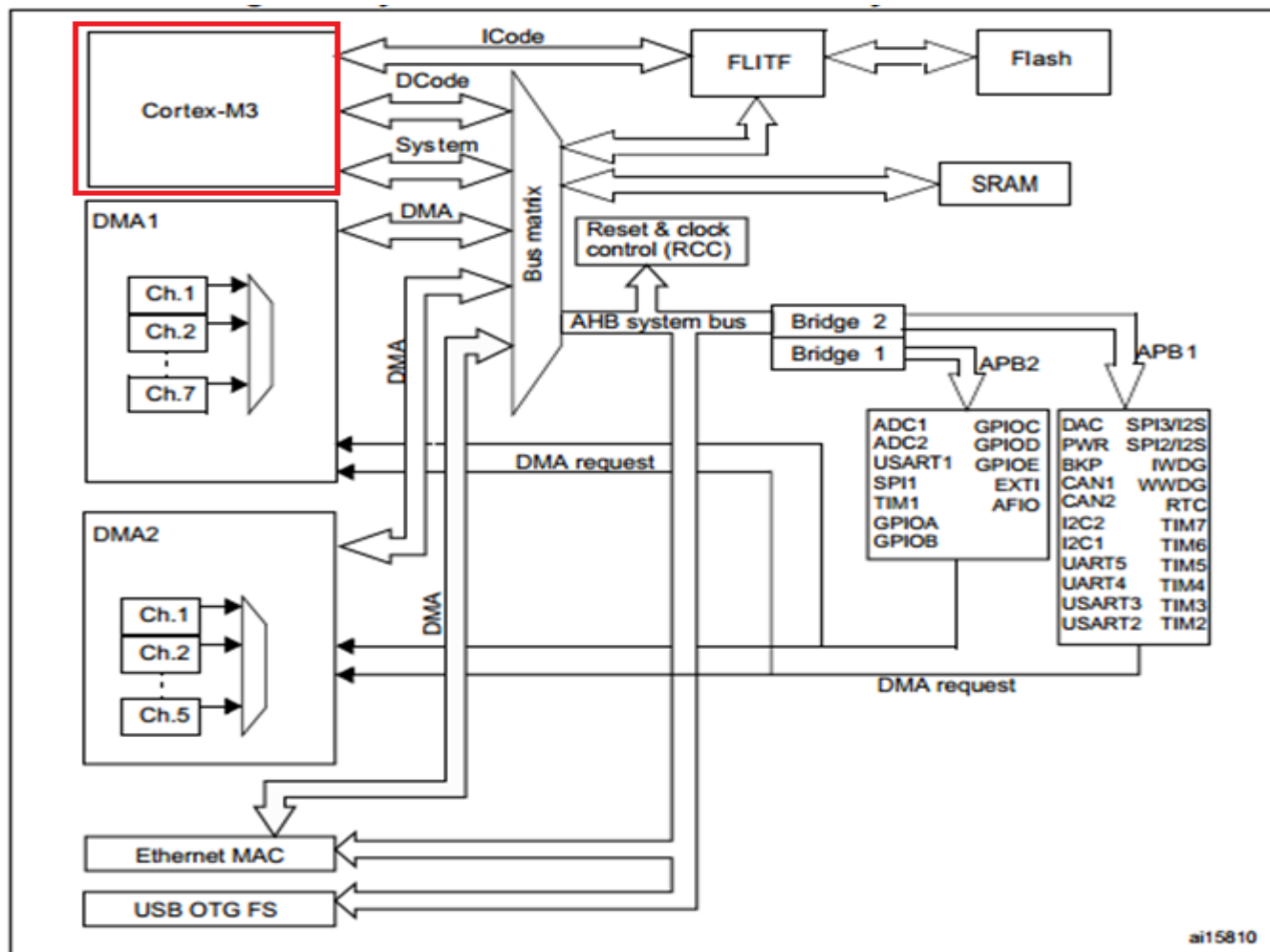
## Соотношение между регистрами в состояниях ARM и Thumb



# Форматы набора Thumb

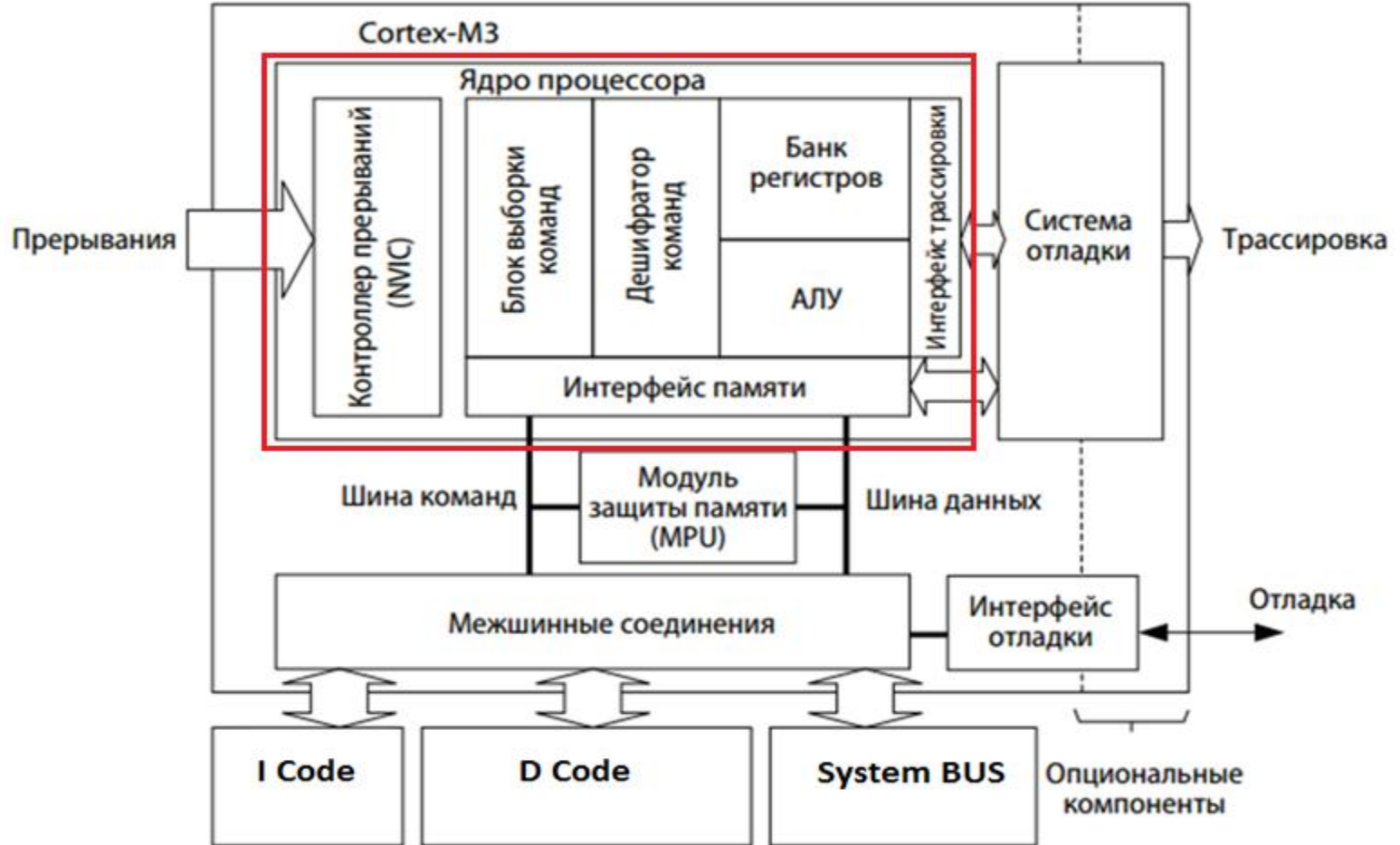
15															0																								
0 1 0 0 0 0							funct				Rm				Rdn				<funct>S Rdn, Rdn, Rm (data-processing)																				
0 0 0 ASR LSR							imm5							Rm				Rd				LSLS/LSRS/ASRS Rd, Rm, #imm5																	
0 0 0 1 1 1 SUB							imm3							Rm				Rd				ADDS/SUBS Rd, Rm, #imm3																	
0 0 1 1 SUB							Rdn				imm8													ADDS/SUBS Rdn, Rdn, #imm8															
0 1 0 0 0 1 0 0							Rdn [3]		Rm						Rdn[2:0]						ADD Rdn, Rdn, Rm																		
1 0 1 1 0 0 0 0 SUB							imm7													ADD/SUB SP, SP, #imm7																			
0 0 1 0 1							Rn				imm8													CMP Rn, #imm8															
0 0 1 0 0							Rd				imm8													MOV Rd, #imm8															
0 1 0 0 0 1 1 0							Rdn [3]		Rm						Rdn[2:0]						MOV Rdn, Rm																		
0 1 0 0 0 1 1 1 L							Rm						0 0 0						BX/BLX Rm																				
1 1 0 1							cond				imm8													B<cond> imm8															
1 1 1 0 0							imm11													B imm11																			
0 1 0 1							L B H		Rm				Rn				Rd				STR(B/H)/LDR(B/H) Rd, [Rn, Rm]																		
0 1 1 0 L							imm5							Rn				Rd				STR/LDR Rd, [Rn, #imm5]																	
1 0 0 1 L							Rd				imm8													STR/LDR Rd, [SP, #imm8]															
0 1 0 0 1							Rd				imm8													LDR Rd, [PC, #imm8]															
1 1 1 1 0							imm22[21:11]													1 1 1 1 1					imm22[10:0]										BL imm22				

# SoC - Микроконтроллер STM32



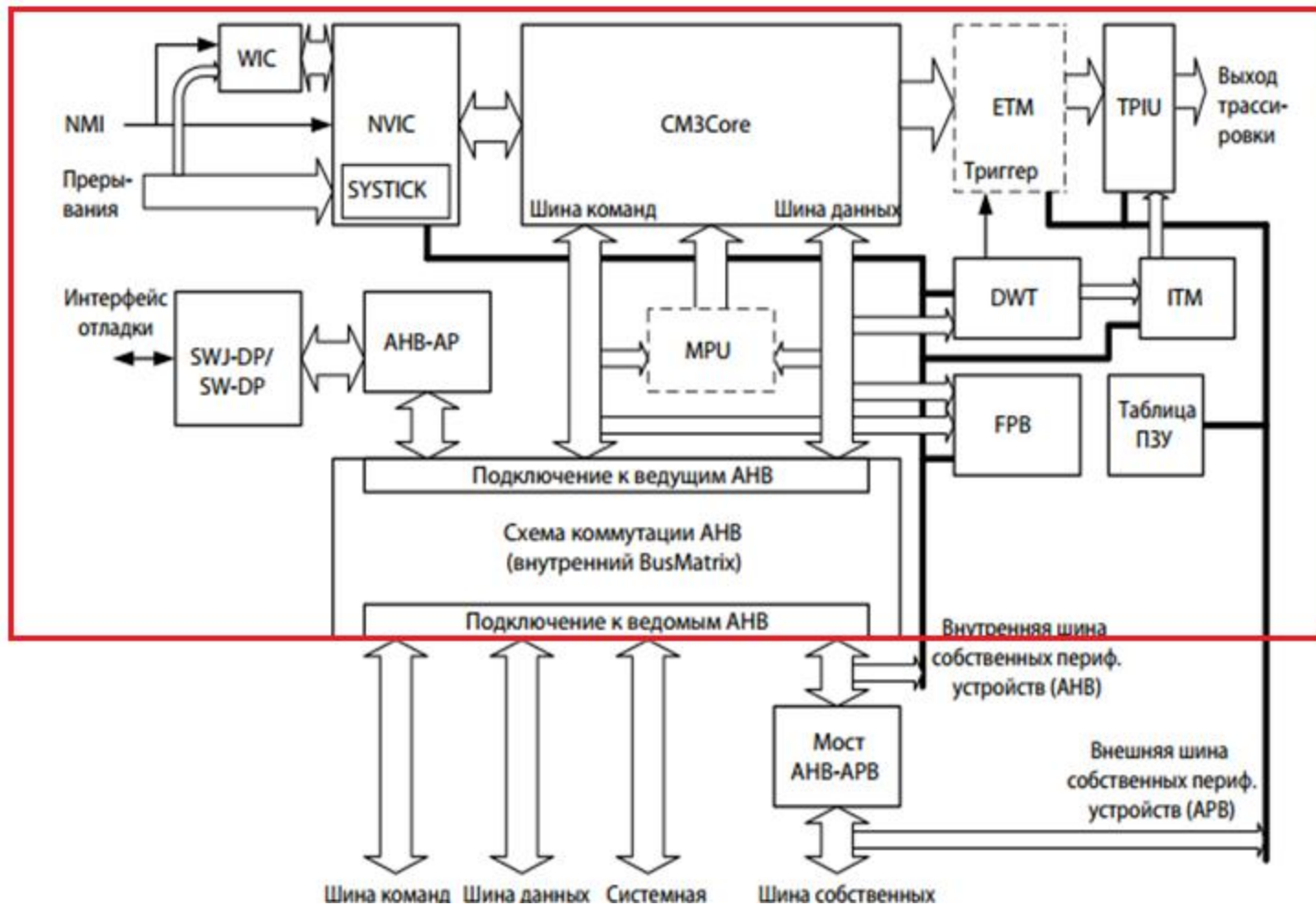
Soc (System-on-a-Chip)

# Ядро процессора Cortex - M3



- Трех-ступенчатый конвейер (выборка, дешифрация, выполнение)
- Набор команд Thumb 2

## ARM процессор Cortex - M3 (еще раз)



# Микроконтроллер STM 32 f103 на базе микропроцессора Cortex M3

