



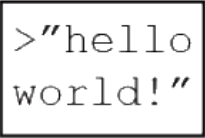


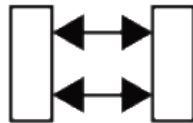
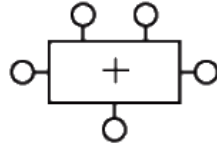



# Архитектуры процессорных систем

## Лекция 10. Виды и классификация архитектур

Цикл из 16 лекций о цифровой схемотехнике, способах построения и архитектуре компьютеров

# План лекции

- Архитектура
- Классификация архитектур
  - по способу хранения операндов
  - по составу и сложности команд
  - по способу реализации условных переходов
- Классификация команд процессора
  - по функциональному назначению
  - по способам адресации
- Примеры реальных архитектур

Application Software	
Operating Systems	
<b>Architecture</b>	
Micro-architecture	
Logic	
Digital Circuits	
Devices	
Physics	

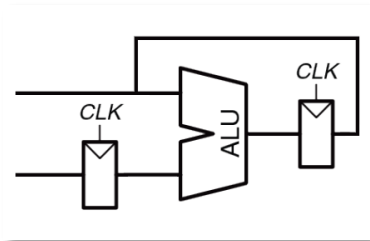
– абстрактная модель функциональных возможностей процессора  
(средства, которыми может пользоваться программист / функциональная организация)

# Instruction Set Architecture (ISA)

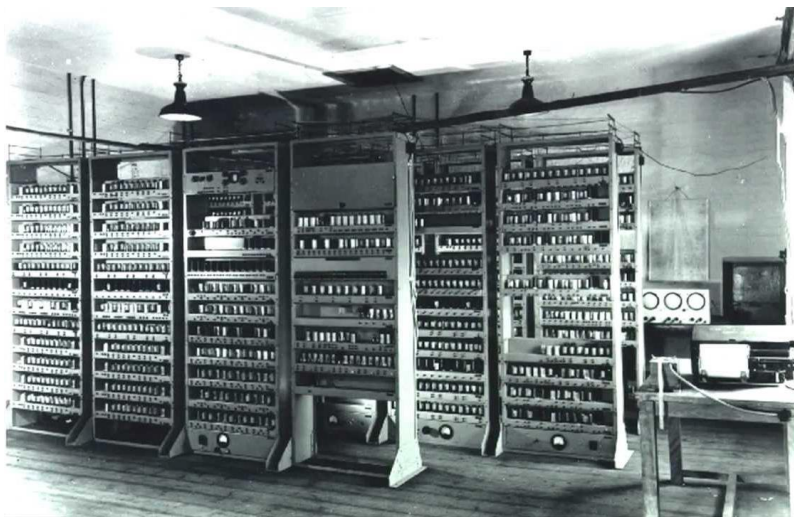
- Система команд
- Средства для выполнения команд
  - Форматы данных
  - Системы регистров
  - Способы адресации
  - Модели памяти

## Instruction Set Architectures

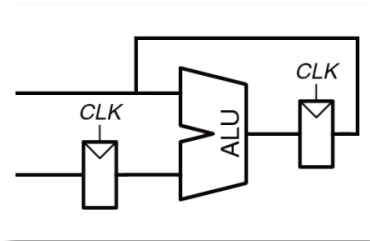
## Instruction Set Architectures



**Аккумуляторная архитектура**  
(EDSAC, 1950)



## Instruction Set Architectures

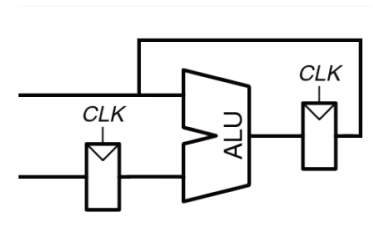


**Аккумуляторная архитектура**  
(EDSAC, 1950)

Регистровая архитектура  
(IBM 360, 1964)



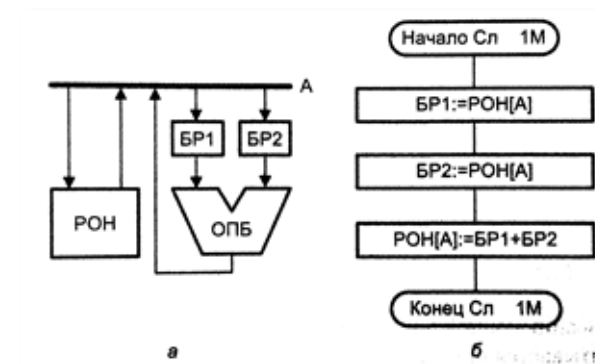
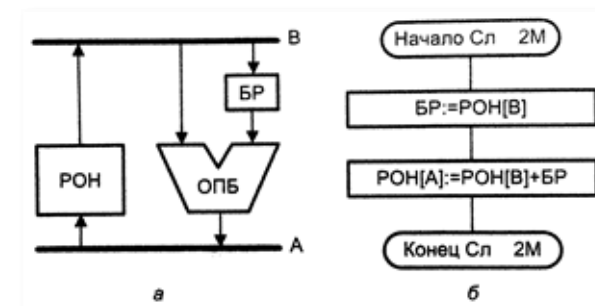
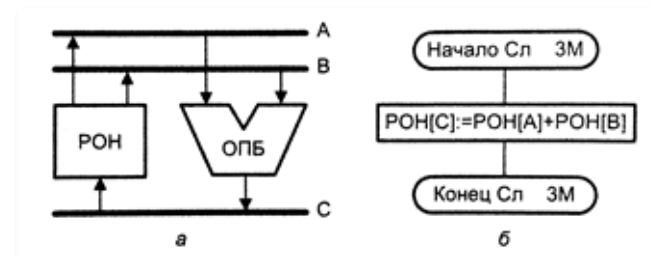
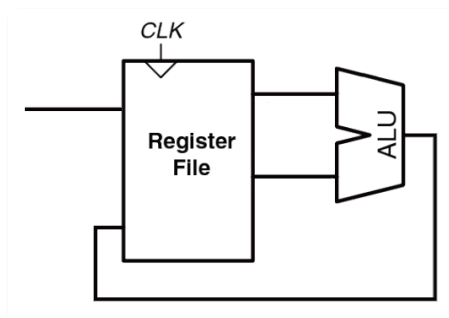
# Instruction Set Architectures



**Аккумуляторная архитектура**  
(EDSAC, 1950)

**Регистровая архитектура**  
(IBM 360, 1964)

**Load/Store архитектура**  
(CDC 6600, Cray1, 1963–76)





## Instruction Set Architectures

### Аккумуляторная архитектура

(EDSAC, 1950)

### Регистровая архитектура

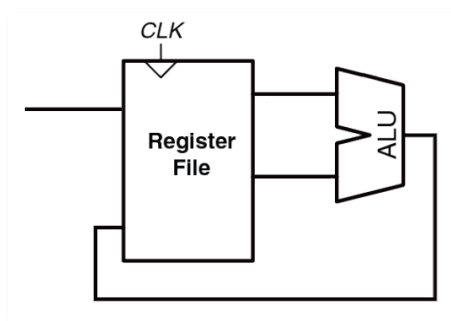
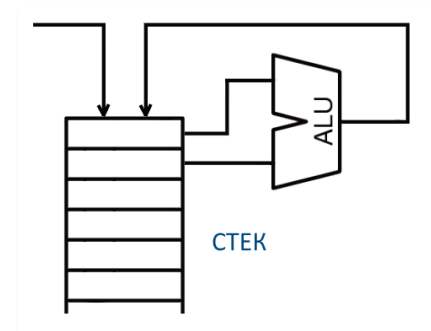
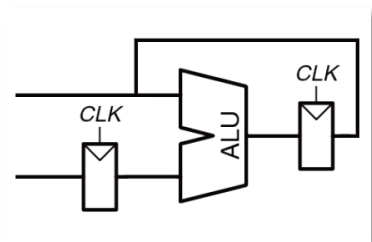
(IBM 360, 1964)

### Load/Store архитектура

(CDC 6600, Cray1, 1963–76)

### Стековая архитектура

(B5500, B6500, 1963–66)



# Запись математических выражений

- Инфиксная запись ( $a + b$ )
- Префиксная запись ( $+ a b$ )
- Постфиксная запись ( $a b +$ ) – обратная польская нотация
  - Любое выражение можно представить в постфиксном виде
  - Обработка постфиксных выражений наиболее оптимальным образом использует обращение к памяти

Например:

$(a + b) * (c + d) - e$

$ab+cd+*e-$

в инфиксной записи соответствует

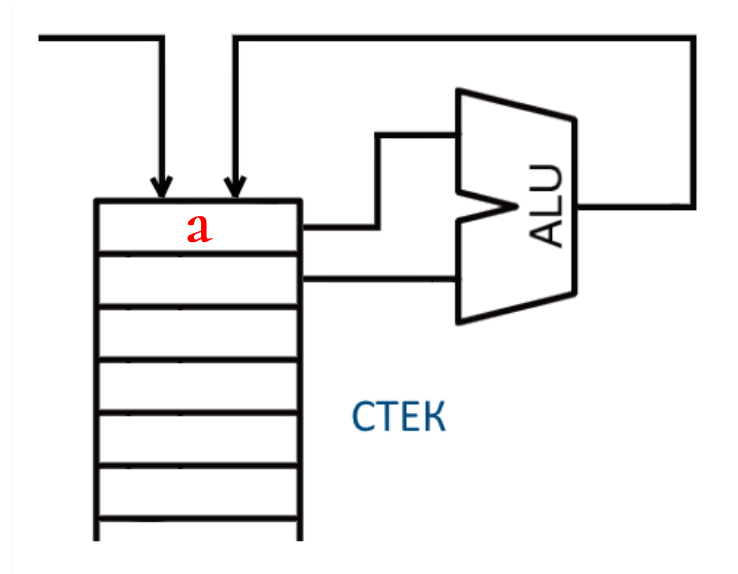
в постфиксной записи

# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

a

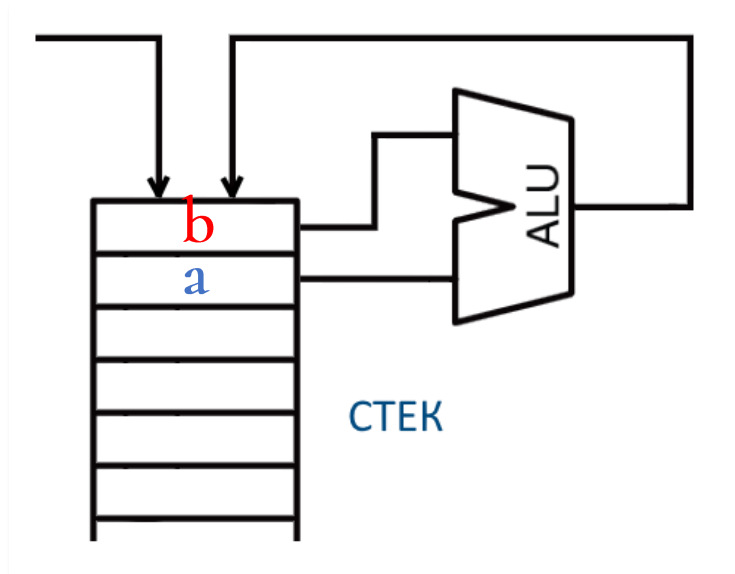


# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

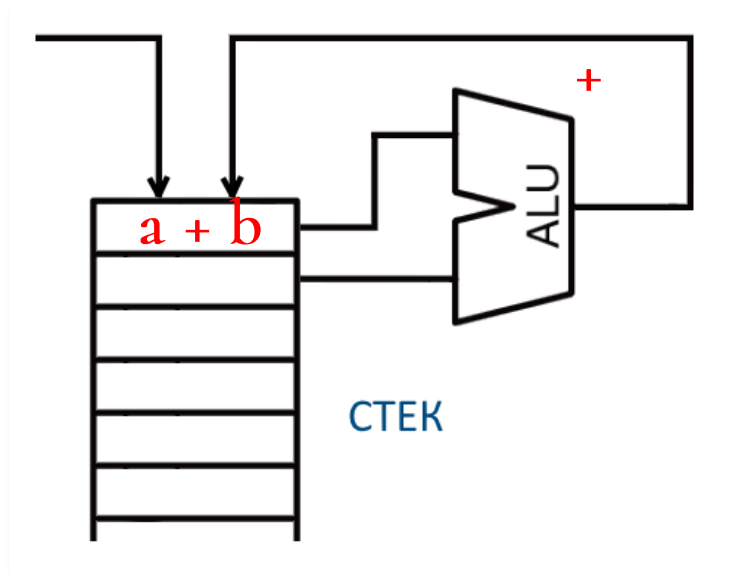
b



# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

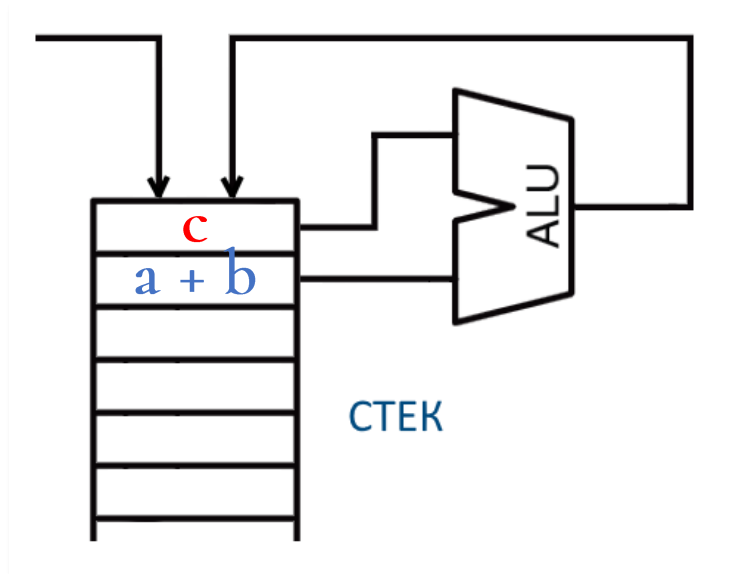


# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

c

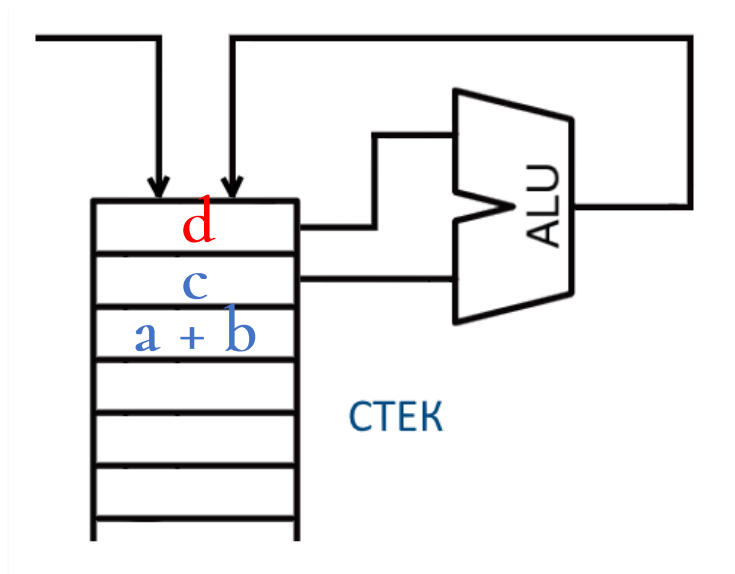


# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

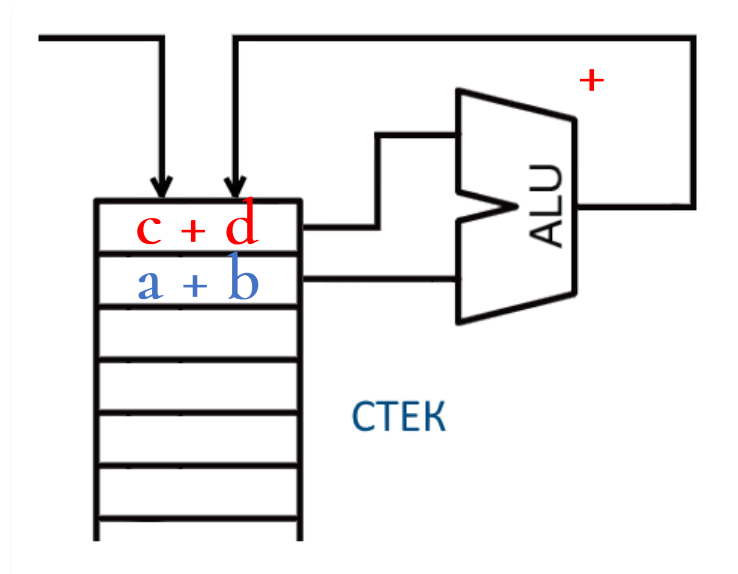
d



# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-





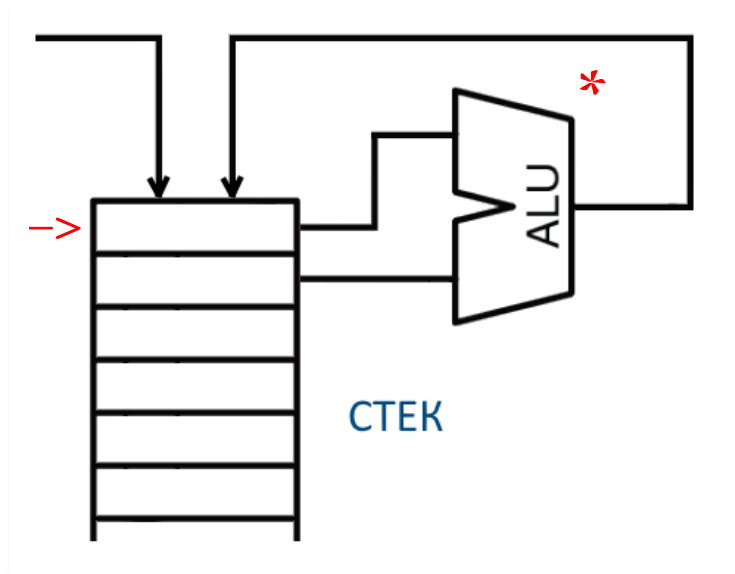
# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

$(a + b) * (c + d)$

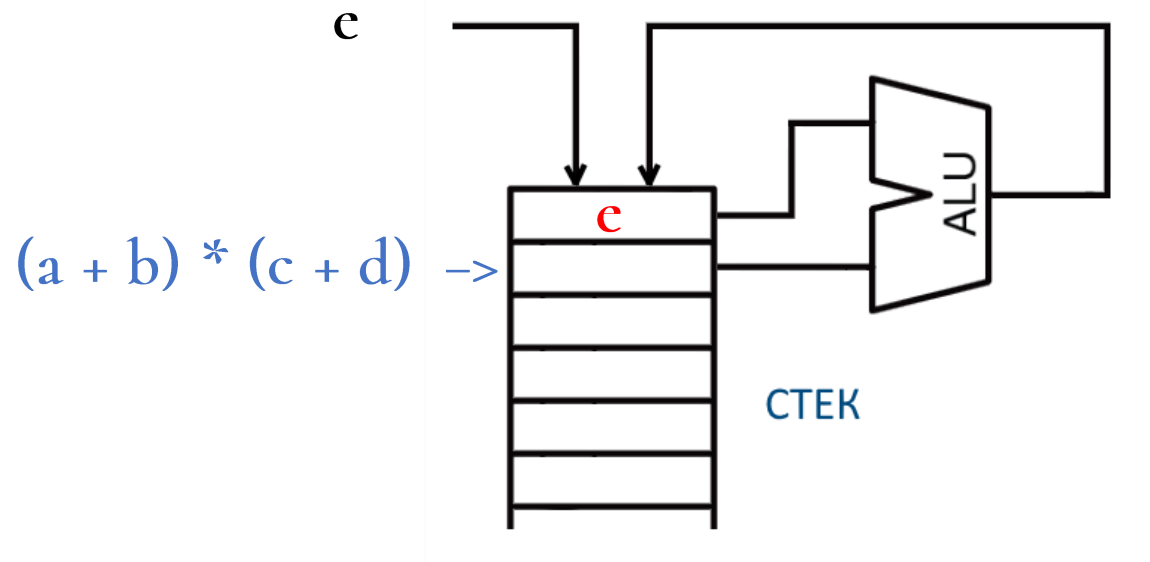
->



# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

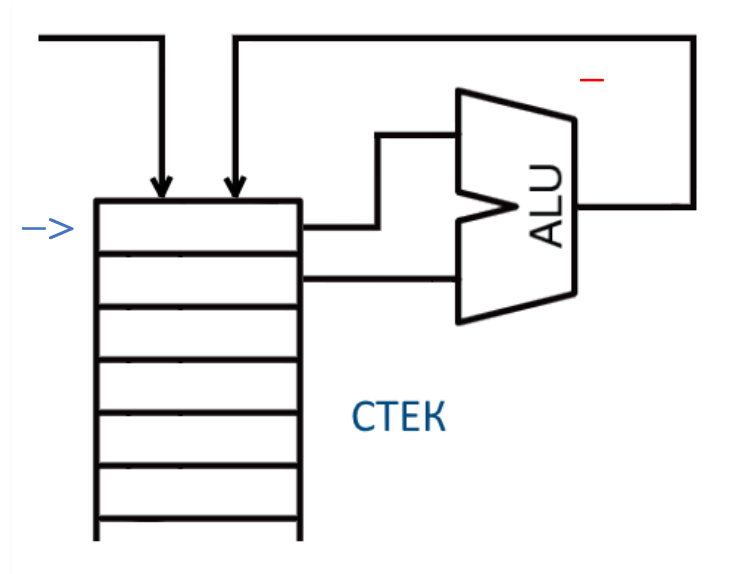


# Обратная польская нотация

$$(a + b) * (c + d) - e$$

ab+cd+\*e-

$(a + b) * (c + d) - e \rightarrow$



# Алгоритм преобразования инфиксной записи в постфиксную

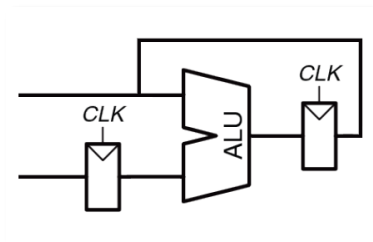
СПО – стек последовательности операций (воображаемая память сохранения операций по принципу первым пришел – последним вышел)

1. Входная строка с выражением просматривается слева направо
2. Если очередной символ входной строки – **операнд**, он **сразу перезаписывается** в постфиксную строку
3. Левая скобка, а также символ математической **операции** в случае, **если СПО пуст**, всегда заносится в СПО
4. Когда при просмотре **встречается правая скобка, символ**, находящийся на вершине СПО, выталкивается из СПО и **заносится в постфиксную строку**. Процедура повторяется до появления левой скобки. При встрече они аннигилируют
5. Если **СПО пуст или символ операции** во входной строке имеет более высокий приоритет, чем символ на вершине СПО, символ операции из входной строки **заносится в СПО**
6. Если приоритет символа во входной строке **равен или ниже приоритета** символа на вершине СПО, символ из вершины СПО выталкивается в постфиксную строку. Процедура повторяется до тех пор, пока на вершине СПО не появится символ с меньшим приоритетом, либо левая скобка, либо СПО станет пустой. После этого символ из входной строки заносится в СПО
7. После достижения **последнего символа** входной строки содержание СПО последовательно выталкивается в постфиксную строку

# Стековые архитектуры

- Burrough's B5000 (1960)
- Burrough's B6700
- HP 3000
- ICL 2900
- Symbolics 3600
- **Современные**
  - Inmos Transputer
  - Java Virtual Machine
  - Intel x87 Floating Point Unit

# Instruction Set Architectures



## Аккумуляторная архитектура

(EDSAC, 1950)

## Регистровая архитектура

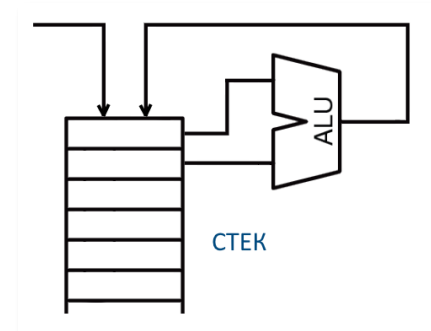
(IBM 360, 1964)

## Load/Store архитектура

(CDC 6600, Cray1, 1963–76)

## Стековая архитектура

(B5500, B6500, 1963–66)



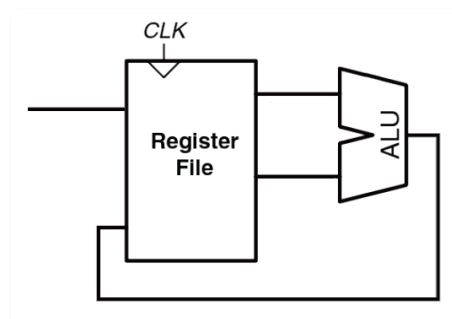
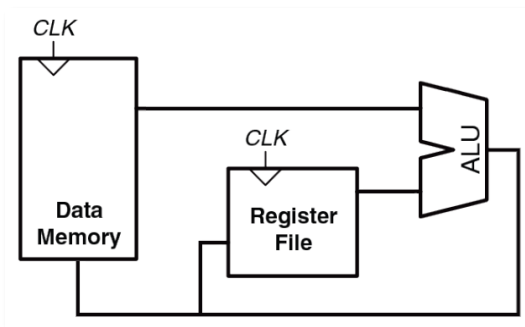
Архитектура с полным набором команд – CISC

register-memory

Выделенный доступ к памяти

(Мостовая архитектура)

(VAX, Intel 432, 1977–80)



# Особенности CISC

CISC (Complex Instruction Set Computer – архитектура с полным набором команд)

- Большое количество различных машинных команд (сотни), каждая из которых выполняется за несколько тактов центрального процессора;
- Устройство управления с программируемой логикой;
- Небольшое количество регистров общего назначения (РОН);
- Различные форматы команд с разной длиной;
- Преобладание двухадресной адресации;
- Развитый механизм адресации операндов, включающий различные методы косвенной адресации.

# Instruction Set Architectures

## Аккумуляторная архитектура

(EDSAC, 1950)

## Регистровая архитектура

(IBM 360, 1964)

## Load/Store архитектура

(CDC 6600, Cray1, 1963–76)

## Стековая архитектура

(B5500, B6500, 1963–66)

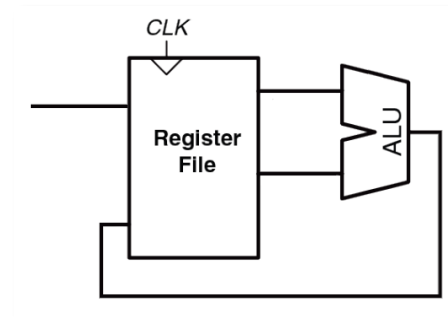
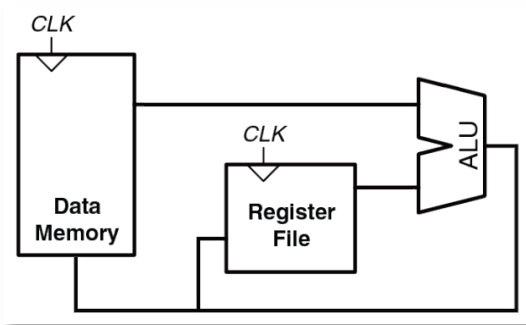
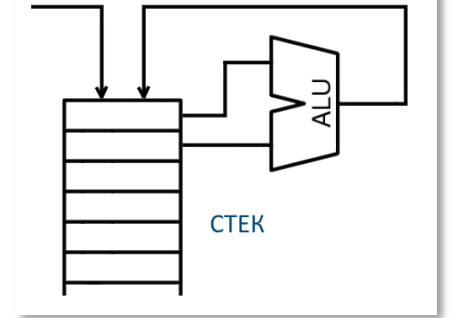
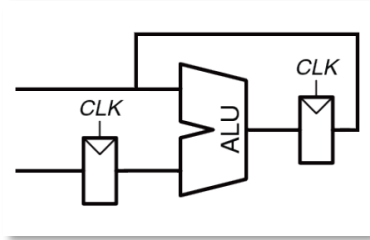
Архитектура с полным набором команд – CISC  
register-memory

Выделенный доступ к памяти  
(**Мостовая архитектура**)

(VAX, Intel 432, 1977–80)

Архитектура с сокращенным набором команд – RISC

(MIPS, SPARC, IBM RS6000, ... 1987)





# Особенности RISC

RISC (Reduced Instruction Set Computer – архитектура с сокращенным набором команд)

- Выполнение всех (или, по крайней мере, 75% команд) за один цикл;
- Стандартная однословная длина всех команд, равная естественной длине слова и ширине шины данных и допускающая унифицированную конвейерную обработку всех команд;
- Малое число стандартных команд (не более 128);
- Малое количество форматов команд (около 4);
- Малое число способов адресации (не более 4);
- Доступ к памяти только посредством команд чтения и записи (load/store);
- Все команды, за исключением чтения и записи, используют внутрипроцессорные межрегистровые пересылки;
- Устройство управления с аппаратной (жесткой) логикой;
- Относительно большой (порядка 32 регистров) процессорный файл регистров общего назначения (число РОН в современных RISC-микропроцессорах может превышать 500).

# Преимущества RISC

- Для технологии RISC характерна сравнительно простая структура устройства управления (остается больше места для других узлов ЦП и для дополнительных устройств: кэш-памяти, блока арифметики с плавающей запятой, части основной памяти, блока управления памятью, портов ввода/вывода).
- Унификация набора команд, ориентация на конвейерную обработку, унификация размера команд и длительности их выполнения, устранение периодов ожидания в конвейере.
- RISC обладает рядом средств для непосредственной поддержки ЯВУ и упрощения разработки компиляторов ЯВУ, благодаря чему эта архитектура в плане поддержки ЯВУ почти равна CISC.

# Недостатки RISC

- Сокращенное число команд: на выполнение ряда функций приходится тратить несколько команд вместо одной в CISC. Это удлиняет код программы, увеличивает загрузку памяти и трафик команд между памятью и ЦП. Исследования показали, что RISC-программа в среднем на 30% длиннее CISC-программы, реализующей те же функции.
- Хотя большое число регистров дает существенные преимущества, само по себе оно усложняет схему декодирования номера регистра, тем самым увеличивается время доступа к регистрам.
- УУ с аппаратной логикой, реализованное в большинстве RISC-систем, менее гибко, более склонно к ошибкам, затрудняет поиск и исправление ошибок, уступает при выполнении сложных команд.
- Однословная команда исключает прямую адресацию для полноразрядного адреса, поэтому ряд производителей допускают небольшую часть команд двойной длины

# Instruction Set Architectures

## Аккумуляторная архитектура

(EDSAC, 1950)

## Регистровая архитектура

(IBM 360, 1964)

## Load/Store архитектура

(CDC 6600, Cray1, 1963–76)

## Стековая архитектура

(B5500, B6500, 1963–66)

Архитектура с полным набором команд – CISC  
register-memory

Выделенный доступ к памяти  
(**Мостовая архитектура**)

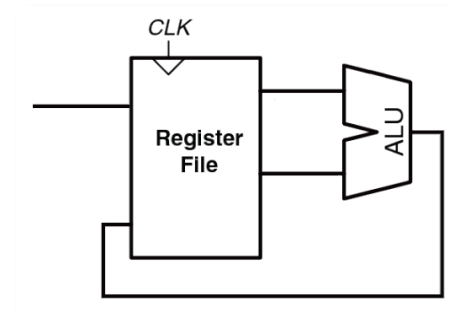
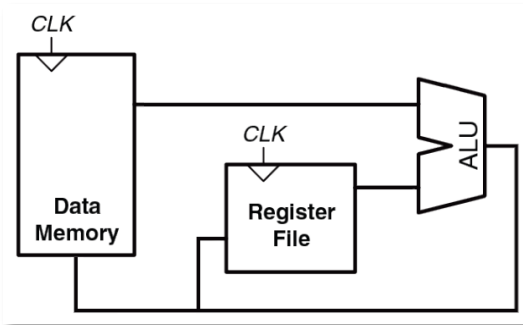
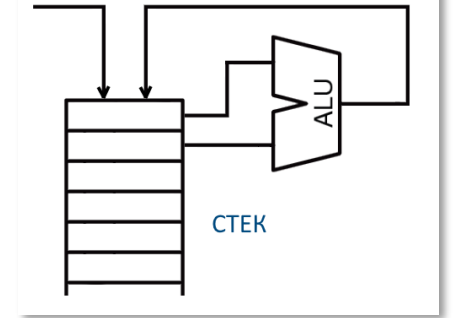
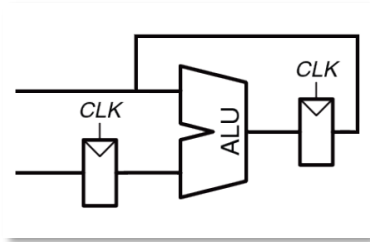
(VAX, Intel 432, 1977–80)

Архитектура с сокращенным набором команд – RISC

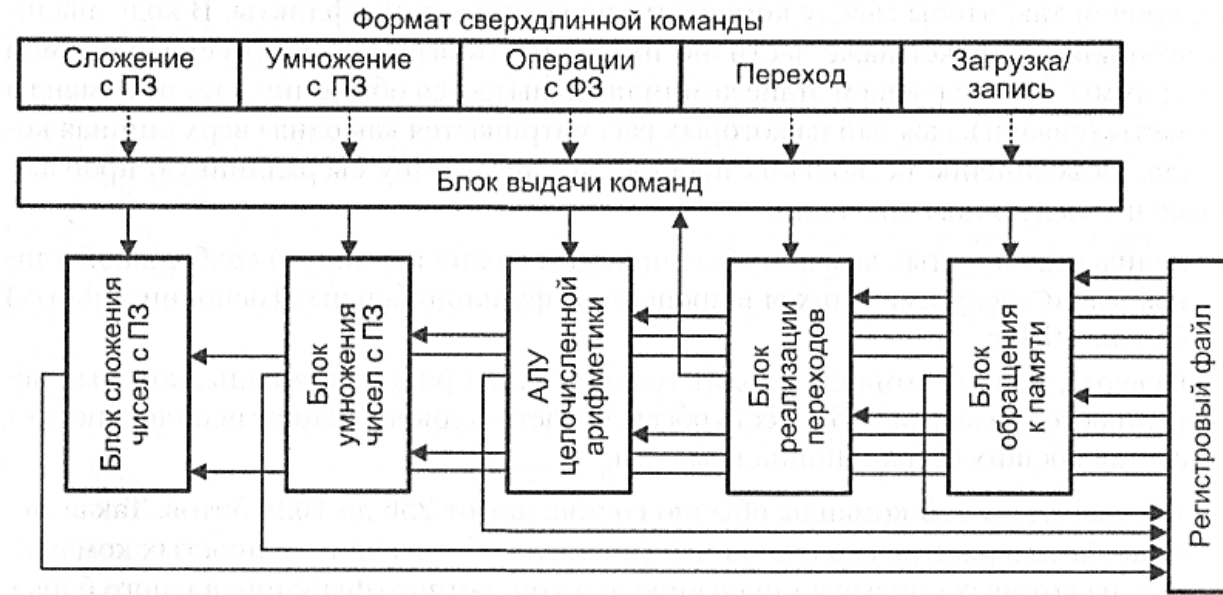
(MIPS, SPARC, IBM RS6000, ... 1987)

Архитектура с командными словами сверхбольшой длины – VLIW

(Itanium, конец 1990-х)



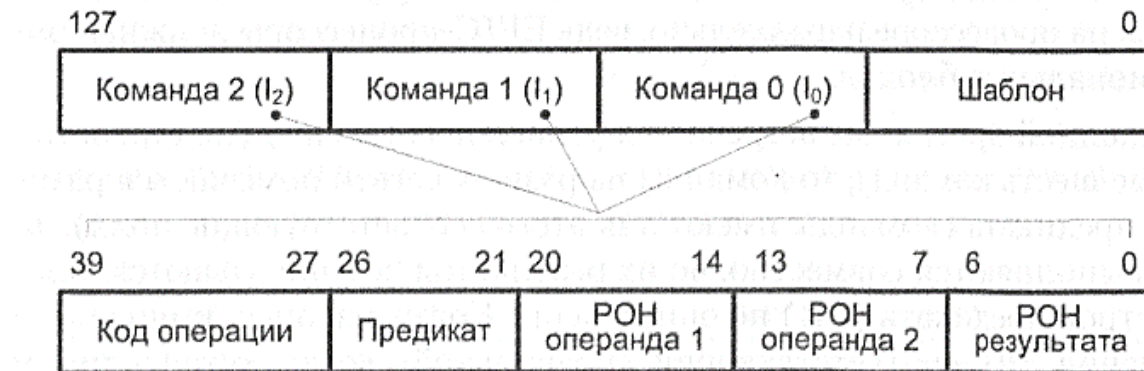
# VLIW (Very Long Instruction Word)

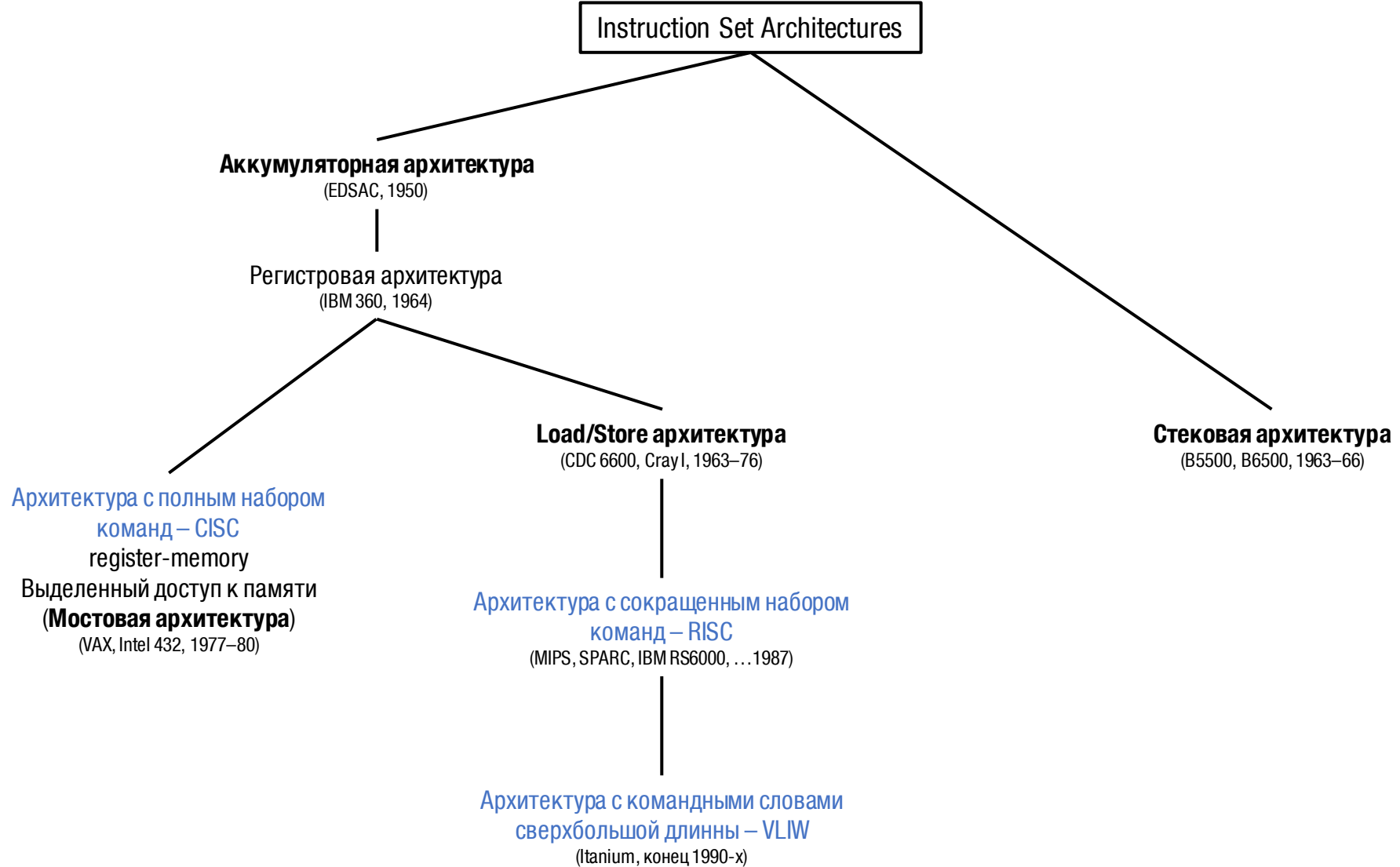


# Недостатки VLIW

- Усложнение регистрового файла и, прежде всего, связей этого файла с вычислительными устройствами;
- Трудности создания компиляторов, способных найти в программе независимые команды, связать такие команды в длинные строки и обеспечить их параллельное выполнение;
- Большой объем программного кода;
- Невозможность миграции программ написанных под другие архитектуры;
- Сложность отладки.

# EPIC (Explicitly Parallel Instruction Computing)







# Регистры в различных архитектурах

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register-memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-memory, memory-memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register-memory	1980
Intel 80386	8	Register-memory	1985
ARM	16	Load-store	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992
HP/Intel IA-64	128	Load-store	2001
AMD64 (EMT64)	16	Register-memory	2003
RISC-V	32	Load-store	2010

# Типы данных и размеры

- **Типы**

- Целочисленные (Integer)
- Двоично-десятичный код (Binary Coded Decimal – BCD)
- Числа в формате с плавающей точкой (Floating Point)
  - IEEE 754 (float)
  - Cray Floating Point
  - Intel Extended Precision (80-bit)
  - TensorFloat
  - BrainFloat
  - MSFP
  - Posit
- Пакованная арифметика (Packed Vector Data)
- Адреса

- **Размеры**

- Целочисленные (8, 16, 32, 64 бита)
- Числа в формате с плавающей точкой (8, 16, 32, 40, 64, 80 бит)

# Классификация архитектур

По способу реализации условных переходов

- Переход происходит или не происходит исходя из значений флагов, формируемых операционными устройствами
- Специальные операции меняют содержимое регистра состояний (State Register). Команда условного перехода происходит, если содержимое регистра состояний соответствует выполняемой инструкции
  - Плюс использования – возможность предикация

# Классификация инструкций

- По функциональному назначению
- По количеству операндов и способам адресации

# Классификация инструкций

По функциональному назначению:

- Инструкции работы с данными
  - Перемещение данных (`mov`, `lw`, `sw`, `push`, `pop`...)
  - Модификация данных
    - Арифметические (`add`, `sub`, `mul`, `div`...)
    - Логические (`and`, `or`, `not`...)
    - Сдвига (`sll`, `srl`, `rol`, `ror`...)
    - Сравнения (`slt`, `cmp`...)
- Инструкции управления
  - Управление программой
    - Условный переход (`beq`, `bne`...)
    - Безусловный переход (`jal`, `goto`...)
  - Управление процессором (`halt`, `nop`, `rst`...)

# Способы адресации операндов

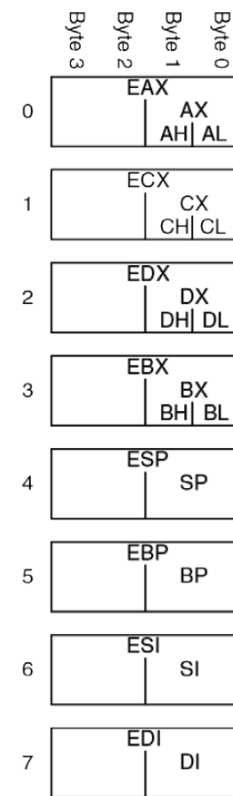
- **Непосредственная** (операнд содержится внутри инструкции)
- **Прямая** (указывается адрес операнда в основной памяти)
- **Регистровая** (указывается адрес регистра в регистровом файле)
- **Косвенно-регистровая** (указывается адрес регистра, в котором лежит адрес операнда в основной памяти)
- **Косвенно-регистровая со смещением** (указывается адрес регистра, в котором лежит базовый адрес, к которому прибавляется константа из инструкции, полученное число является адресом операнда в основной памяти)
- **Относительная** (является суммой регистра специального назначения и константы)

# Количество операндов в инструкции

- 0 – нет операндов
- 0 – подразумеваемый операнд
- 1 – один операнд
- 2 – два операнда
- 3 – три операнда
- ...

# Сравнение RISC и CISC

Характеристики	MIPS	x86
Количество регистров	32, общего назначения	8, некоторые ограничения по использованию
Количество операндов	3 (2 источника, 1 назначение)	2 (1 источник, 1 источник/назначение)
Расположение операндов	Регистры или непосредственные операнды	Регистры, непосредственные операнды или память
Размер операнда	32 бита	8, 16 или 32 бита
Коды условий	Нет	Да
Типы команд	Простые	Простые и сложные
Размер команд	Фиксированный, 4 байта	Переменный, 1–15 байтов





# Адресация операндов в x86

Источник/ Назначение	Источник	Пример	Выполняемая функция
Регистр	Регистр	add EAX, EBX	EAX $\leftarrow$ EAX + EBX
Регистр	Непосредственный операнд	add EAX, 42	EAX $\leftarrow$ EAX + 42
Регистр	Память	add EAX, [20]	EAX $\leftarrow$ EAX + Mem[20]
Память	Регистр	add [20], EAX	Mem[20] $\leftarrow$ Mem[20] + EAX
Память	Непосредственный операнд	add [20], 42	Mem[20] $\leftarrow$ Mem[20] + 42

Пример	Назначение
add EAX, [20]	EAX $\leftarrow$ EAX + Mem[20]
add EAX, [ESP]	EAX $\leftarrow$ EAX + Mem[ESP]
add EAX, [EDX+40]	EAX $\leftarrow$ EAX + Mem[EDX+40]
add EAX, [60+EDI*4]	EAX $\leftarrow$ EAX + Mem[60+EDI*4]
add EAX, [EDX+80+EDI*2]	EAX $\leftarrow$ EAX + Mem[EDX+80+EDI*2]

# Регистр состояния x86

Название	Назначение
CF (Carry Flag, флаг переноса)	Показывает, что при выполнении последней арифметической операции результат вышел за пределы разрядной сетки. Указывает на то, что произошло переполнение при беззнаковых вычислениях. Также используется как флаг переноса при работе с числами, разрядность которых превышает разрядность архитектуры
ZF (Zero Flag, флаг нуля)	Показывает, что результат последней операции равен нулю
SF (Sign Flag, флаг знака)	Показывает, что результат последней операции был отрицательным (старший бит результата равен 1).
OF (Overflow Flag, флаг переполнения)	Показывает, что произошло переполнение при вычислениях со знаковыми числами в дополнительном коде

Инструкция	Назначение	Действие после CMP D, S
JZ/JE	Ветвление, если $ZF = 1$	Ветвление, если $D = S$
JNZ/JNE	Ветвление, если $ZF = 0$	Ветвление, если $D \neq S$
JGE	Ветвление, если $SF = OF$	Ветвление, если $D \geq S$
JG	Ветвление, если $SF \neq OF$ и $ZF = 0$	Ветвление, если $D > S$
JLE	Ветвление, если $SF \neq OF$ и $ZF = 1$	Ветвление, если $D \leq S$
JL	Ветвление, если $SF \neq OF$	Ветвление, если $D < S$
JC/JB	Ветвление, если $CF = 1$	
JNC	Ветвление, если $CF = 0$	
JO	Ветвление, если $OF = 1$	
JNO	Ветвление, если $OF = 0$	
JS	Ветвление, если $SF = 1$	
JNS	Ветвление, если $SF = 0$	

# Управление программой в x86

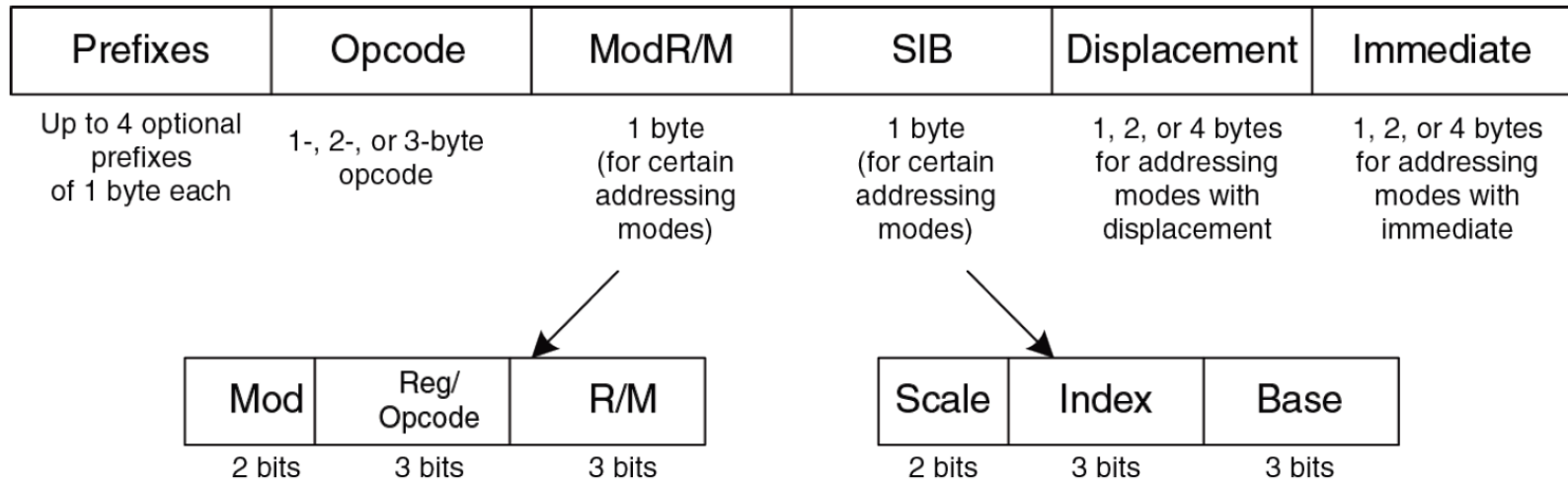
Инструкция	Назначение	Функция
JMP	Безусловный переход	Переход по относительному адресу: $EIP = EIP + S$ Переход по абсолютному адресу: $EIP = S$
JCC	Ветвление (условный переход)	Если установлен флаг, то $EIP = EIP + S$
LOOP	Проверка условия цикла	$ECX = ECX - 1$ Если $ECX \neq 0$ , то $EIP = EIP + imm$
CALL	Вызов функции	$ESP = ESP - 4$ ; $MEM[ESP] = EIP$ ; $EIP = S$
RET	Возврат из функции	$EIP = MEM[ESP]$ ; $ESP = ESP + 4$

# Некоторые инструкции ISA x86

Инструкция	Назначение	Функция
IDIV/DIV	Знаковое/беззнаковое деление	EDX:EAX/D EAX = частное; EDX = остаток
SAR/SHR	Арифметический/логический сдвиг вправо	$D = D \ggg S$ / $D = D \gg S$
SAL/SHL	Сдвиг влево	$D = D \ll S$
ROR/ROL	Циклический сдвиг вправо/влево	Циклически сдвинуть D на S разрядов
RCR/RCL	Циклический сдвиг вправо/влево через бит переноса	Циклически сдвинуть CF и D на S разрядов
BT	Проверка бита	CF = D[S] (бит номер S из D)
BTR/BTS	Проверить бит и сбросить/установить его	CF = D[S]; D[S] = 0 / 1
TEST	Установить флаги по результатам проверки бит	Установить флаги по результатам D И S
MOV	Скопировать операнд	D = S
PUSH	Поместить на стек	ESP = ESP - 4; Mem[ESP] = S
POP	Прочитать из стека	D = MEM[ESP]; ESP = ESP + 4
CLC, STC	Сбросить/установить флаг переноса	CF = 0 / 1

Инструкция	Назначение	Функция
ADD/SUB	Сложение/вычитание	$D = D + S$ / $D = D - S$
ADDC	Сложение с переносом	$D = D + S + CF$
INC/DEC	Увеличение/уменьшение	$D = D + 1$ / $D = D - 1$
CMP	Сравнение	Установить флаги по результатам D - S
NEG	Инверсия	$D = -D$
AND/OR/XOR	Логическое «И/ИЛИ/ИСКЛЮЧАЮЩЕЕ ИЛИ»	$D = D \text{ операция } S$
NOT	Логическое НЕ	$D = \bar{D}$
IMUL/MUL	Знаковое/беззнаковое умножение	EDX:EAX = EAX × D

# Кодирование инструкций x86

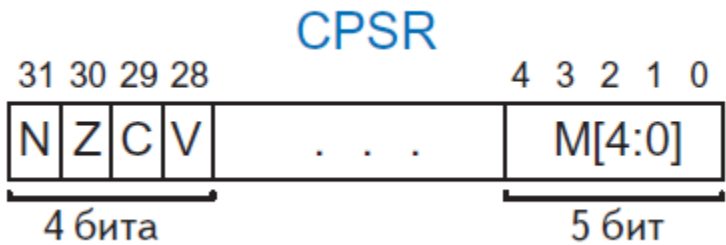


# Архитектура ARM

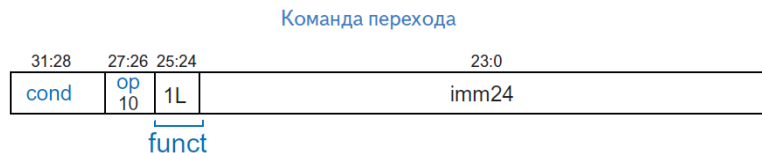
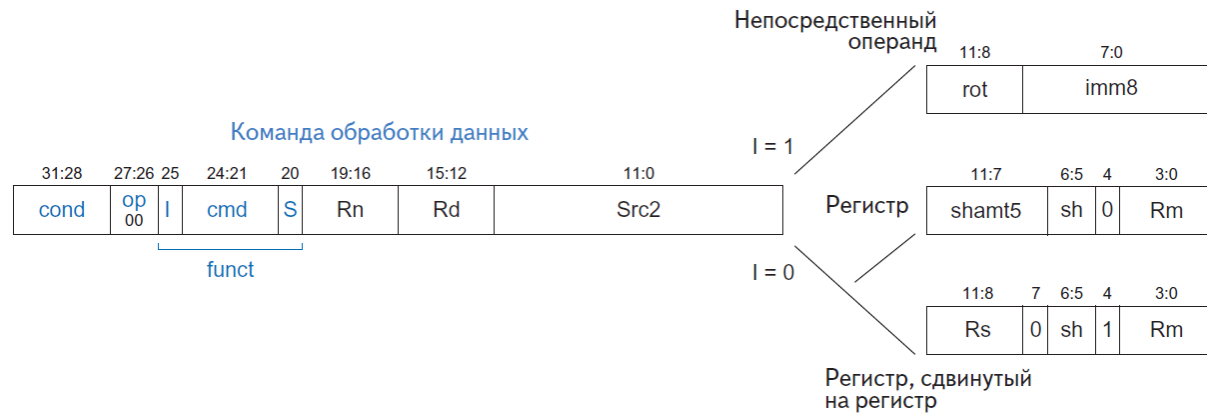
Название	Назначение
R0	Аргумент, возвращаемое значение, временная переменная
R1–R3	Аргумент, временная переменная
R4–R11	Сохраненная переменная
R12	Временная переменная
R13 (SP)	Указатель стека
R14 (LR)	Регистр связи
R15 (PC)	Счетчик команд

Флаг	Название	Описание
N	Negative	Результат выполнения команды отрицателен, т. е. бит 31 равен 1
Z	Zero	Результат выполнения команды равен нулю
C	Carry	Команда привела к переносу
V	oVerflow	Команда привела к переполнению

Режим	CPSR <sub>4:0</sub>
Пользователя	10000
Супервизора	10011
Аварийный	10111
Неопределенный	11011
Прерывания (IRQ)	10010
Быстрого прерывания (FIQ)	10001



# Кодирование инструкций ARM



# Примеры команд ARM

cmd	Название	Описание	Операция
0000	AND Rd, Rn, Src2	Поразрядное И	$Rd \leftarrow Rn \& Src2$
0001	EOR Rd, Rn, Src2	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ	$Rd \leftarrow Rn \wedge Src2$
0010	SUB Rd, Rn, Src2	Вычитание	$Rd \leftarrow Rn - Src2$
0011	RSB Rd, Rn, Src2	Инвертированное вычитание	$Rd \leftarrow Src2 - Rn$
0100	ADD Rd, Rn, Src2	Сложение	$Rd \leftarrow Rn + Src2$
0101	ADC Rd, Rn, Src2	Сложение с переносом	$Rd \leftarrow Rn + Src2 + C$
0110	SBC Rd, Rn, Src2	Вычитание с переносом	$Rd \leftarrow Rn - Src2 - \bar{C}$
0111	RSC Rd, Rn, Src2	Инвертированное вычитание с переносом	$Rd \leftarrow Src2 - Rn - \bar{C}$
1000 (S=1)	TST Rn, Src2	Тест	Установить флаги в соответствии с $Rn \& Src2$
1001 (S=1)	TEQ Rn, Src2	Тест на эквивалентность	Установить флаги в соответствии с $Rn \wedge Src2$
1010 (S=1)	CMP Rn, Src2	Сравнение	Установить флаги в соответствии с $Rn - Src2$
1011 (S=1)	CMN Rn, Src2	Сравнение с противоположным	Установить флаги в соответствии с $Rn + Src2$
1100	ORR Rd, Rn, Src2	Поразрядное ИЛИ	$Rd \leftarrow Rn   Src2$

L	Название	Описание	Операция
0	B label	Перейти	$PC \leftarrow (PC+8) + imm24 \ll 2$
1	BL label	Перейти и связать	$LR \leftarrow (PC+8) - 4; PC \leftarrow (PC+8) + imm24 \ll 2$

op	B	op2	L	Название	Описание	Операция
01	0	—	0	STR Rd, [Rn, ±Src2]	Сохранить регистр	$Mem[Adr] \leftarrow Rd$
01	0	—	1	LDR Rd, [Rn, ±Src2]	Загрузить регистр	$Rd \leftarrow Mem[Adr]$
01	1	—	0	STRB Rd, [Rn, ±Src2]	Сохранить байт	$Mem[Adr] \leftarrow Rd_{7:0}$
01	1	—	1	LDRB Rd, [Rn, ±Src2]	Загрузить байт	$Rd \leftarrow Mem[Adr]_{7:0}$
00	—	01	0	STRH Rd, [Rn, ±Src2]	Сохранить полуслово	$Mem[Adr] \leftarrow Rd_{15:0}$
00	—	01	1	LDRH Rd, [Rn, ±Src2]	Загрузить полуслово	$Rd \leftarrow Mem[Adr]_{15:0}$
00	—	10	1	LDRSB Rd, [Rn, ±Src2]	Загрузить байт со знаком	$Rd \leftarrow Mem[Adr]_{7:0}$
00	—	11	1	LDRSH Rd, [Rn, ±Src2]	Загрузить полуслово со знаком	$Rd \leftarrow Mem[Adr]_{15:0}$

Команды	Описание	Назначение
LDM, STM	Загрузить/сохранить несколько	Сохранить или восстановить регистры при вызове функции
SWP / SWPB	Обменять (байт)	Атомарная загрузка и сохранения для синхронизации процессов
LDRT, LDRBT, STRT, STRBT	Загрузить/сохранить слово/байт с трансляцией	Разрешить операционной системе доступ к памяти в виртуальном адресном пространстве
SWI <sup>1</sup>	Программное прерывание	Возбудить исключение, часто используется для вызова операционной системы
CDP, LDC, MCR, MRC, STC	Доступ к сопроцессору	Взаимодействие с факультативным сопроцессором
MRS, MSR	Скопировать в регистр состояния или из него	Сохранить регистр состояния на время обработки исключения