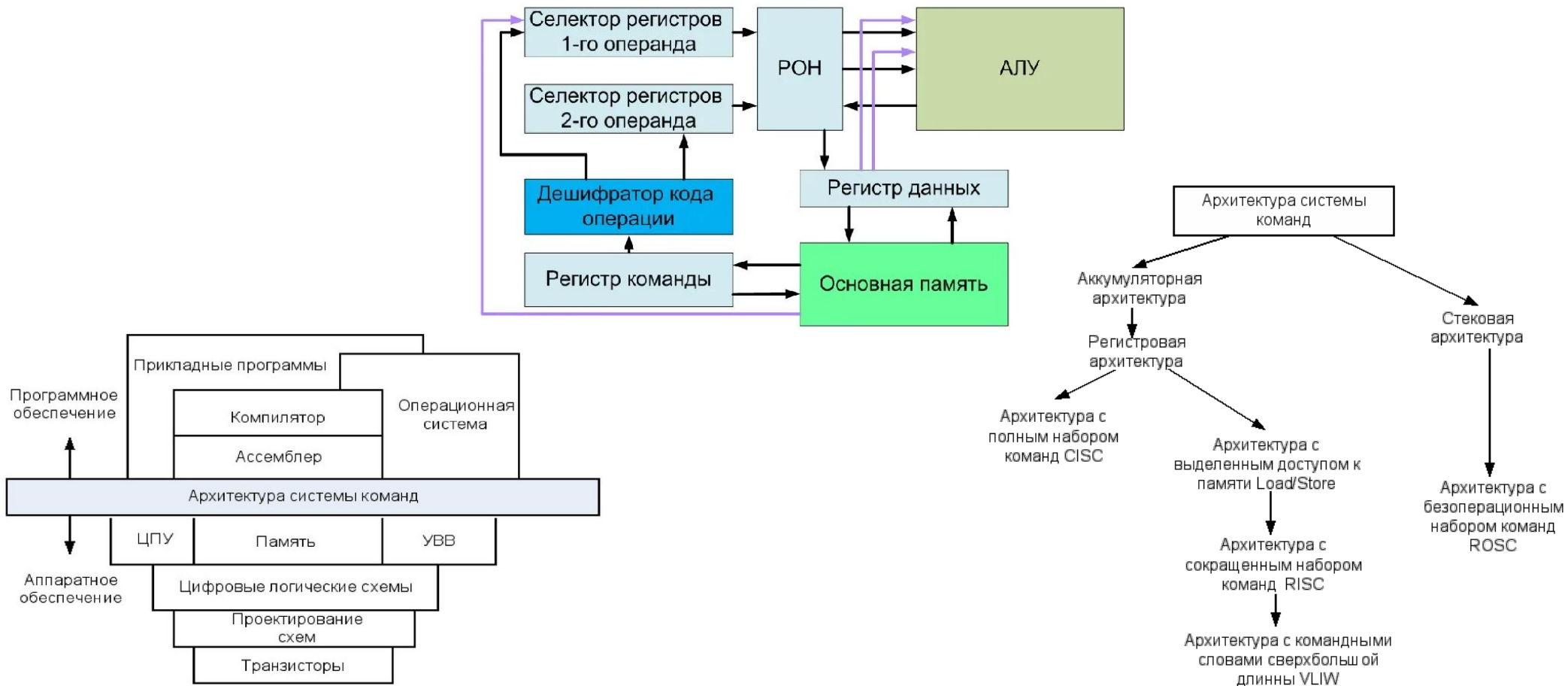


# Архитектура уровня набора (системы) команд



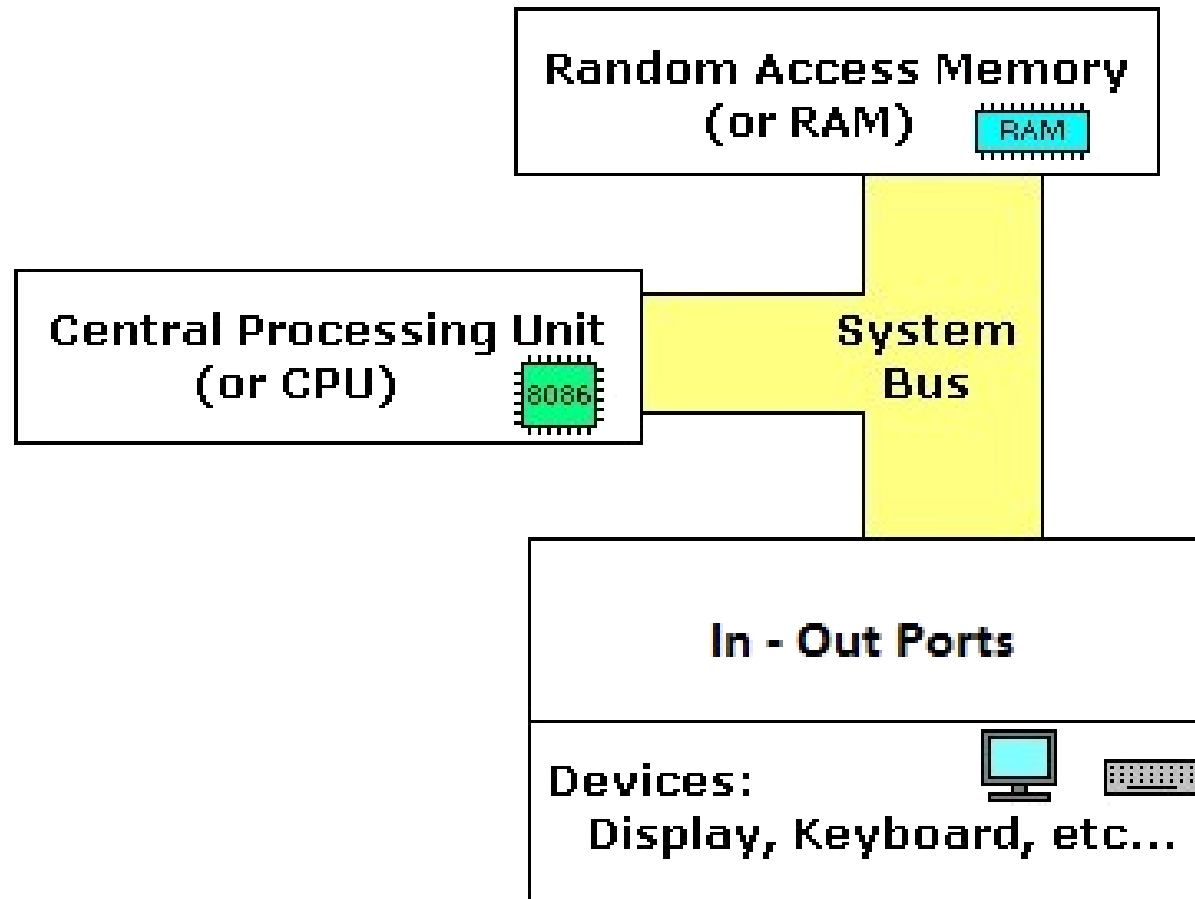
# Определения

Архитектура набора команд (instruction set architecture, ISA) —

часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора. На этом уровне определяются реализованные в микропроцессоре конкретного типа:

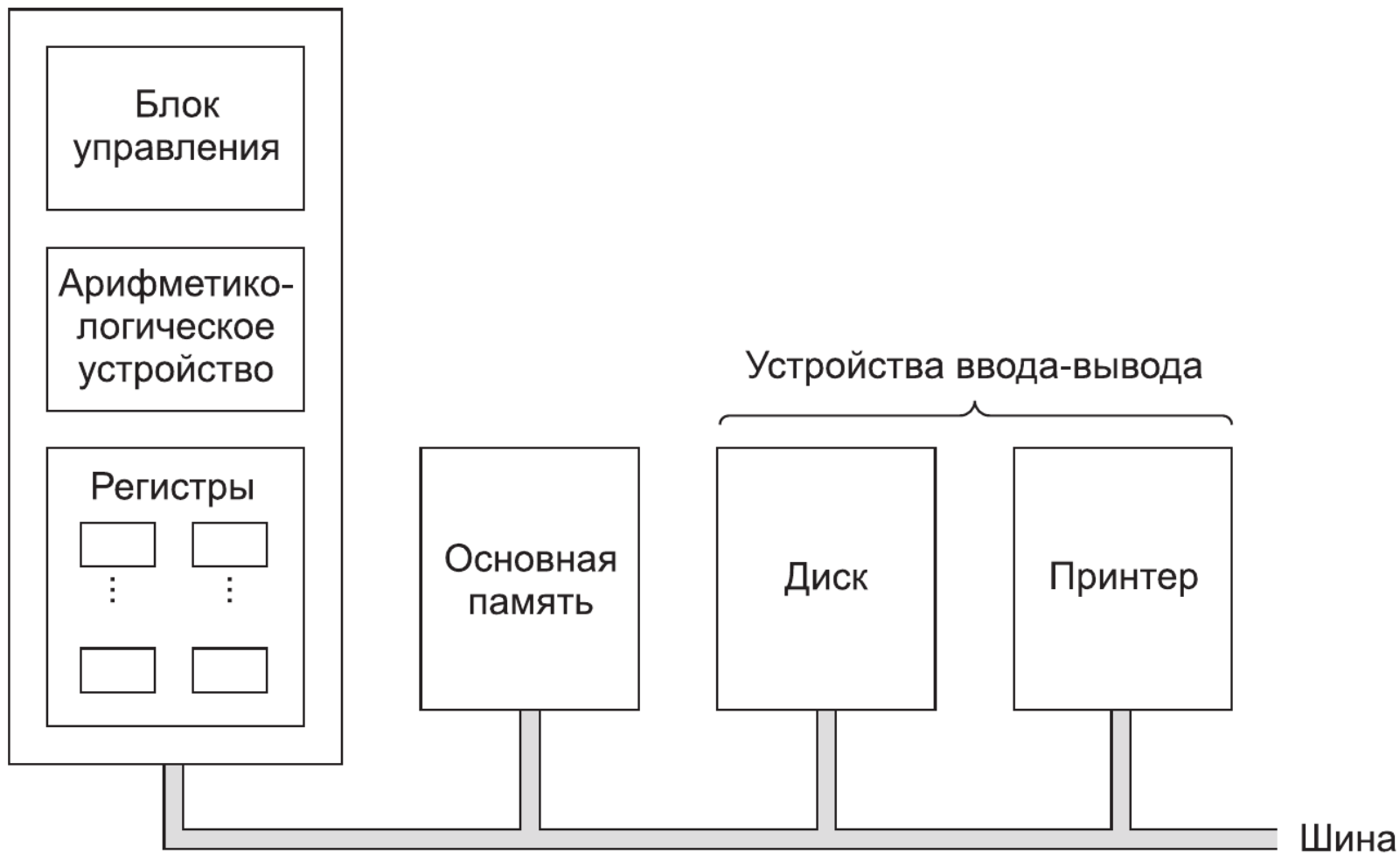
- машинные команды (система команд),
- типы внутренних данных (например, с плавающей запятой, целочисленные и т. д.),
- организация памяти,
- регистры,
- режимы адресации,
- взаимодействие с внешними устройствами ввода/ вывода,
- обработчики прерываний и исключительных состояний.

# Обобщенная структура компьютера

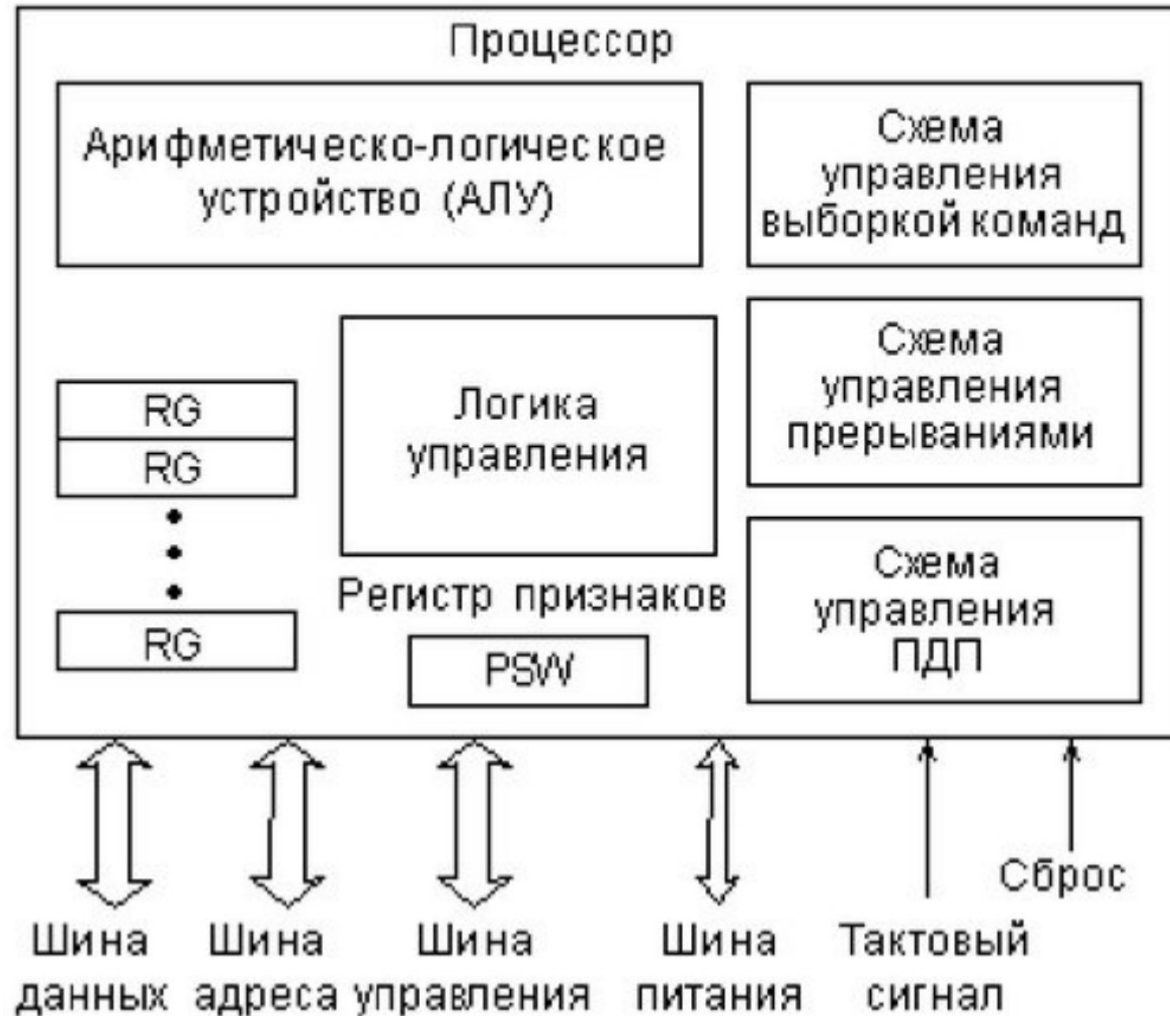


# Больше деталей...

Центральный процессор



## Еще больше деталей...



# Определения

Система команд (набор команд) — соглашение о предоставляемых архитектурой средствах программирования, а именно:

- определённых типах данных,
- инструкций (операций),
- системы регистров,
- методов адресации,
- моделей памяти,
- способов обработки прерываний и исключений,
- методов ввода и вывода.

# Общая структура формата команды



**Формат команды** - структура команды с разметкой номеров разрядов, определяющих - границы отдельных полей команды.

**Операционная часть:** содержит код операции и задает вид операции (+, -, \*, /...)

**Адресная часть:** содержит информацию об адресах операндов и результата, иногда адрес следующей команды.

Структура команды очень сильно влияет на производительность: чем больше команда, тем больше времени уходит на ее считывание и дешифрирование.

Часто система команд достаточно велика, и разрядность шины адреса составляет до 128 разрядов.

Это приводит к появлению очень длинных команд и снижению эффективности.

Длина команд должна быть согласована с длиной обрабатываемых данных (упрощается аппаратура).

Это, как правило, приводит к длине команд, кратным разрядности шины данных.

# Классификация системы команд

По функциональному назначению

Способ передачи управления

По количеству адресов

По длине команды

По способу кодирования

По способам (методам) адресации

По механизму однозначности команды

По типу архитектуры



# Классификация команд по функциональному назначению

## Классификация системы команд



# Классификация системы команд

Способ передачи управления

С явной передачей управления  
(указание адреса перехода в команде)

С неявной передачей управления  
(переход по счетчику команд)

По количеству адресов

Безадресные

Одноадресные

Двухадресные

Трехадресные

По длине команды

Однобайтовые

Двухбайтовые

...

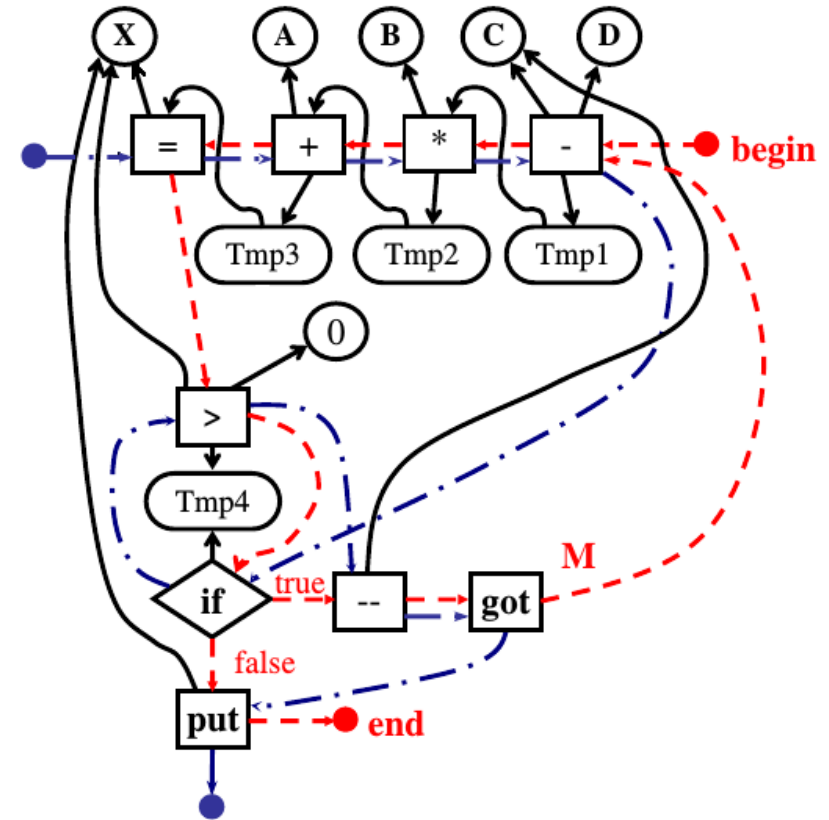
По способу кодирования

С фиксированным полем кода операций

С расширяющимся полем кода операций

## Классификация команд по числу адресов (и способу передачи управления)

M:  $X = A + B * (C - D);$   
 $\text{if}(X > 0) \{C--; \text{goto } M;\}$   
 $\text{else put}(X);$



- - отношения между операндами
- - -> - отношения управления
- . - -> - отношения расположения

# Классификация команд по числу адресов (и способу передачи управления)

КОП
-----

**Push(A); Push(B); +; C = Pop();**  
переходы по счетчику команд

КОП	Адрес операнда
-----	-------------------

**RegAcc = A; RegAcc += B; C = RegAcc;**  
переходы по счетчику команд

КОП	Адрес 1 операнда	Адрес 2 операнда
-----	---------------------	---------------------

**C = A; C += B;**  
переходы по счетчику команд

КОП	Адрес 1 операнда	Адрес 2 операнда	Адрес результата
-----	---------------------	---------------------	---------------------

**C = A+B;**  
переходы по счетчику команд

КОП	Адрес 1 операнда	Адрес 2 операнда	Адрес результата	Адрес следующей команды
-----	---------------------	---------------------	---------------------	----------------------------

**C = A+B & goto M;**

--	--	--	--

Поля адресной части

# Классификация команд по способу кодирования операции

Пример: Intel 8080

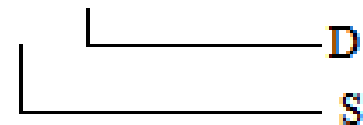
КОП	Адрес 1 операнда	Адрес 2 операнда
-----	---------------------	---------------------

КОП	Адрес 1 операнда
-----	---------------------

КОП
-----

01	DDD	SSS
----	-----	-----

MOV r1,r2



1 0 0 0 0	SSS
-----------	-----

ADD r



1 1 1 1 1 0 1 1
-----------------

КОП

EI

2-адресная: 01 - признак команды MOV

другие: 00,11,10 - признак расширенного КОП. Один или несколько КОП должны указывать на наличие полностью расширенного КОП.

Дешифратор команды МП должен, фактически, рассматривать команду последовательно слева направо.

# Классификация системы команд



# Классификация команд по способу адресации

## Подразумеваемый операнд

*В команде не содержится явных указаний об адресе операнда, операнд подразумевается и фактически задается кодом операции:*

**INC r    (r) <- (r) + 1**

## Подразумеваемый адрес

*В команде отсутствует адрес операнда или результата, но этот адрес подразумевается:*

**ADD B    (A) <- (A) + (B)**

## Непосредственная адресация

*В команде содержится не адрес, а сам операнд. Используется обычно для констант. Выборка операнда и формирование его адреса не нужны.*

## Прямая адресация

*Исполнительный адрес соответствует адресному коду. В команде находится сам адрес.*

**Недостаток: длинный адрес, необходимо считывать.**

# Классификация команд по способу адресации

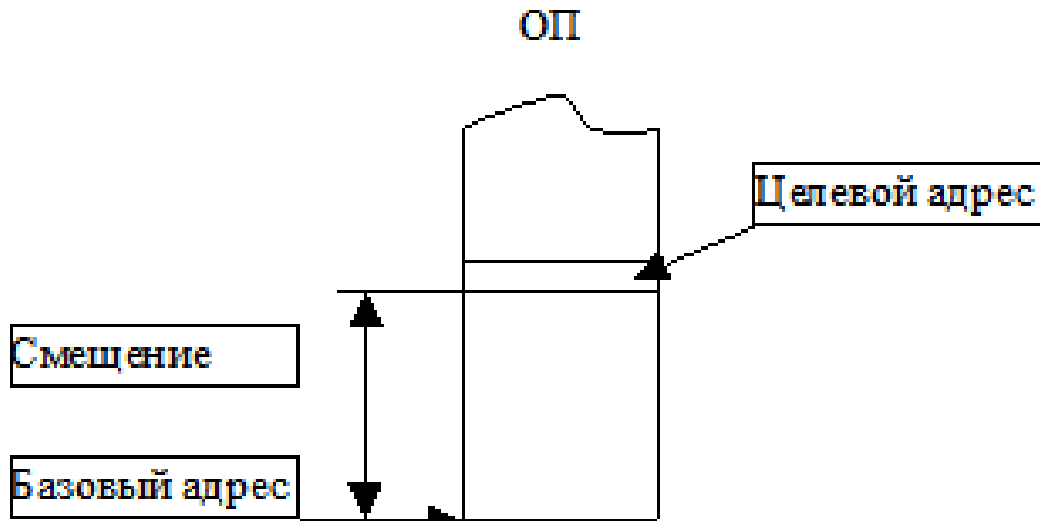
## Относительная адресация (базирование)

Исполнительный адрес определяется суммой адресного кода команды и некоторого числа (базового адреса)

$$A_{\text{эффисп}} = A_{\text{баз}} + A_{\text{ком}}$$

Базовый адрес часто хранится в специальном регистре (регистр базы). В команде выделяется поле для указания номера базового регистра.

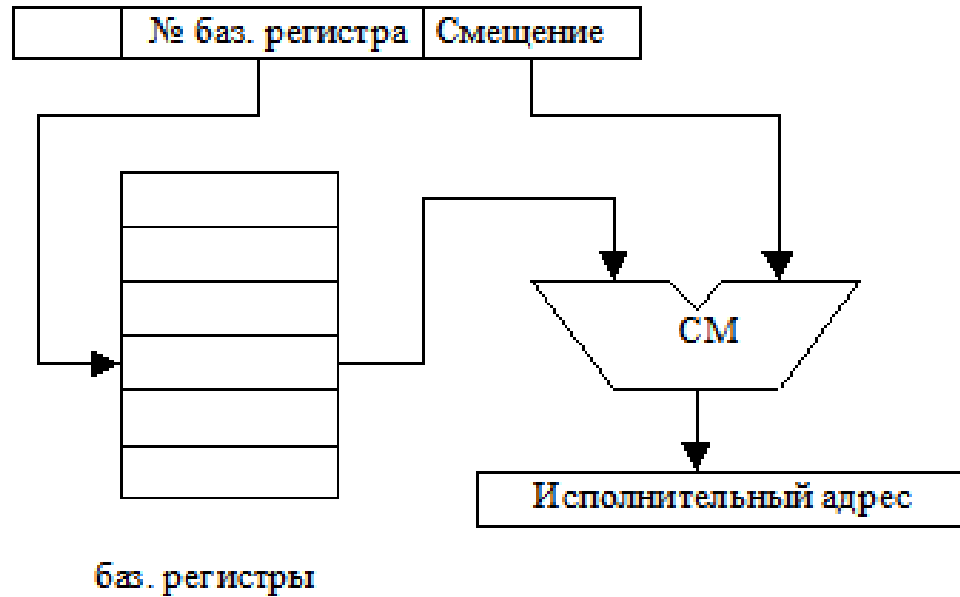
**Достоинство:** меньшая длина адресного кода, при обращении к любой ячейки памяти. В команде указывается только смещение.



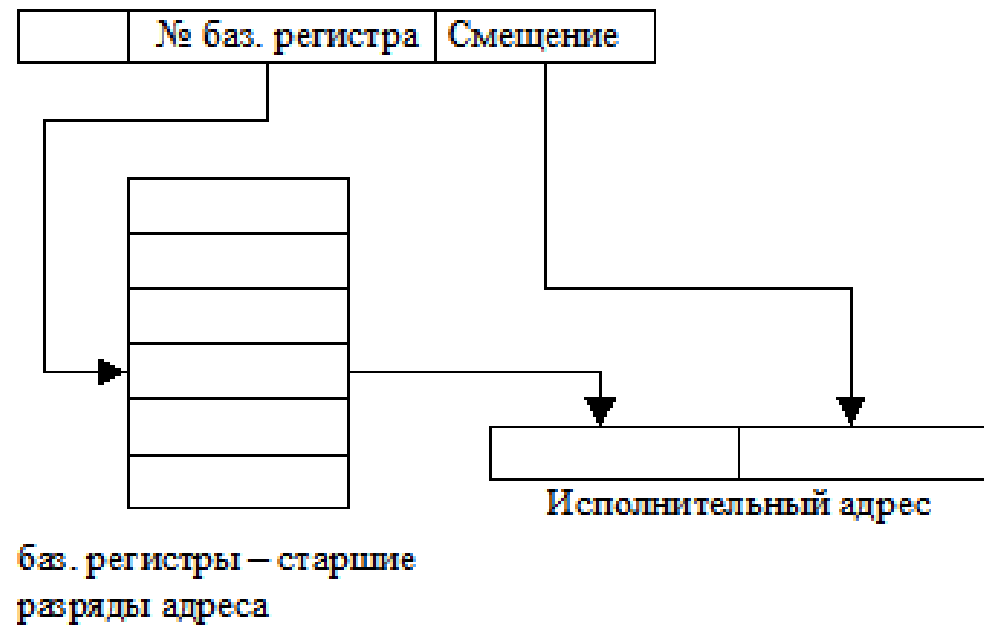


# Варианты относительной адресации

а)



б)



а) суммирование (используется чаще, но сложение - долго)

б) совмещение (базовый адрес содержит старшие разряды, а следующий младшие разряды)

Недостаток: невозможна адресация всей ОП.

# Классификация команд по способу адресации

## Укороченная адресация

Адресное поле содержит только часть адреса (младшие или старшие разряды).

Остальные подразумеваются.

Используется только с другими способами адресации.

Применяется для уменьшения длины команды.

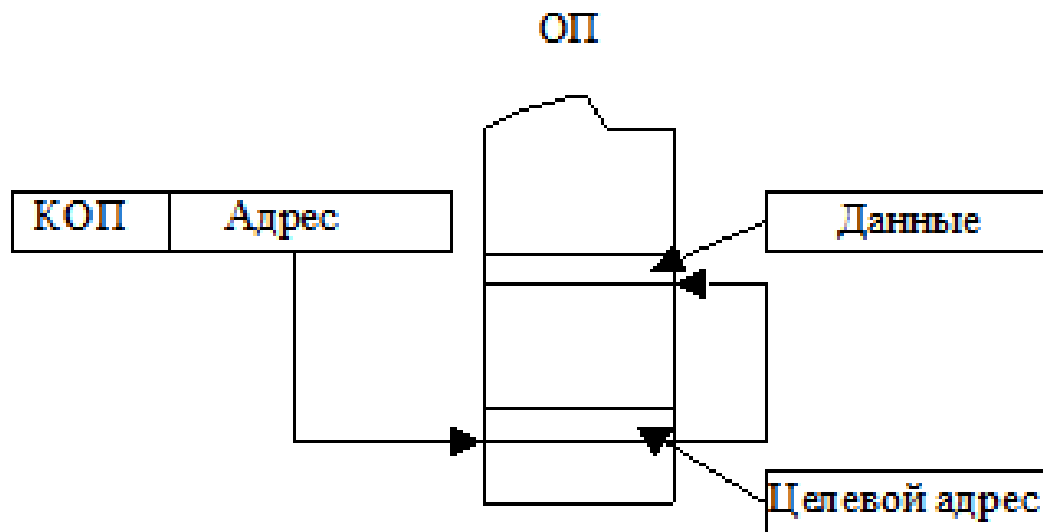
## Регистровая адресация

Частный случай укороченной адресации

**16 RОН - 4 разряда адреса**

## Косвенная адресация

Адрес в команде указывает адрес ячейки памяти, в которой находится адрес операнда.



# Классификация команд по способу адресации

## Автоинкрементная и автодекрементная адресация

Эффективна при работе с массивами. Используется часто совместно с косвенной адресацией, чтобы для обработки каждого элемента массива не загружать новое значение адреса, а использовать автоматическое увеличение или уменьшение на 1 содержимого регистра с адресом.

## Адресация слов переменной длины

Используется для выполнения операций над данными переменной длины. Адресация таких данных реализуется путем указания в команде местоположения в памяти начала слова и его длины.

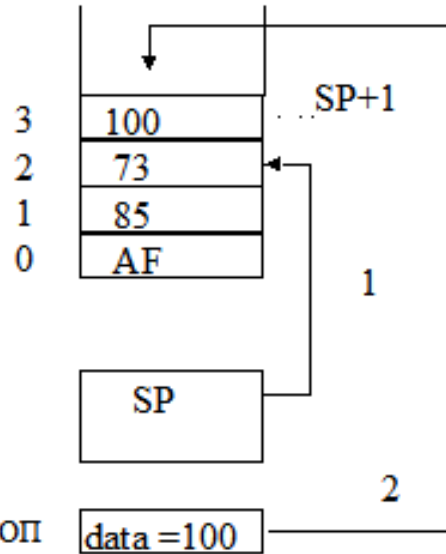
## Стековая адресация

Используется для выполнения операций над стеком. Часто используется в безадресных командах. Обычно через специально выделенный регистр стека.

# Использование польской постфиксной записи

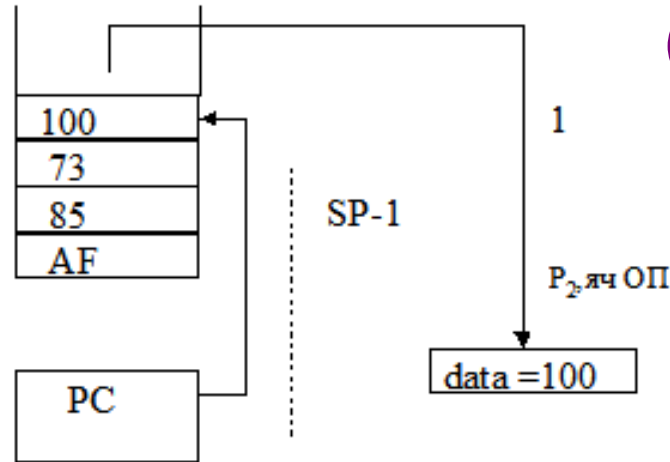
$$(k+l-m)*(p-s) \Rightarrow kl+m-ps-*$$

Запись в стек



1) SP+1    2) запись

Чтение из стека



1) чтение    2) SP-1

Адрес K
Адрес L
+
Адрес M
-
Адрес P
Адрес S
-
*

# Классификация по способу задания однозначности



В большинстве случаев формат данных, с которыми выполняется команда, указывается в самом коде операции.

Это приводит к избыточности кодирования команд, т.к. при наличии нескольких команд работающих с одними и теми же данными фактически каждый раз указывается их тип.

# Тегирование

При тегировании каждое хранимое в памяти слово снабжается указателем - тегом, определяющим тип данных (целое, двоичное число, число с ПТ, адрес, строка, и т.д.)

Тег	Данные
-----	--------

В поле тега кроме типа указывают длину и формат данных, другие параметры. Теги формируются компилятором и прикрепляются к данным.

Наличие тегов придает данным свойство самоопределяемости.

Достоинства:

- теговая организация памяти способствует реализации принципа независимости программ от данных.
- приводит к экономии памяти, т.к. отсутствует информационная избыточность на задание типов и размеров операндов при их использование несколькими командами.

Недостатки:

- уменьшается скорость работы МП из-за установки соответствия типа команд типам данных происходящей не на этапе компиляции, а при выполнении программы.

# Тегирование

Пример: в обычной ЭВМ тип данных определяется контекстно задаче, то есть, непосредственно командой, которая использует эти данные:

В этом случае код команды задает вид (тип) данных. Таким образом, команд сложения может быть очень много: с ФТ, с ПТ, с разрядностью 8, 16, 32 бита.

Теговая организация позволяет достигнуть инвариантности команд относительно данных, что приводит к значительному сокращению набора команд машины (в нашем случае остается всего одна команда сложения). Это упрощает структуру процессора, облегчает работу программиста, облегчает обнаружение ошибок при операциях с некорректными данными.

ADD B	ADD BX	ADD EBX
8 разр.	16 разр.	32 разр.

# Дескрипторы

**Дескрипторы** - служебные слова, содержащие информацию (описание) массивов данных и команд.

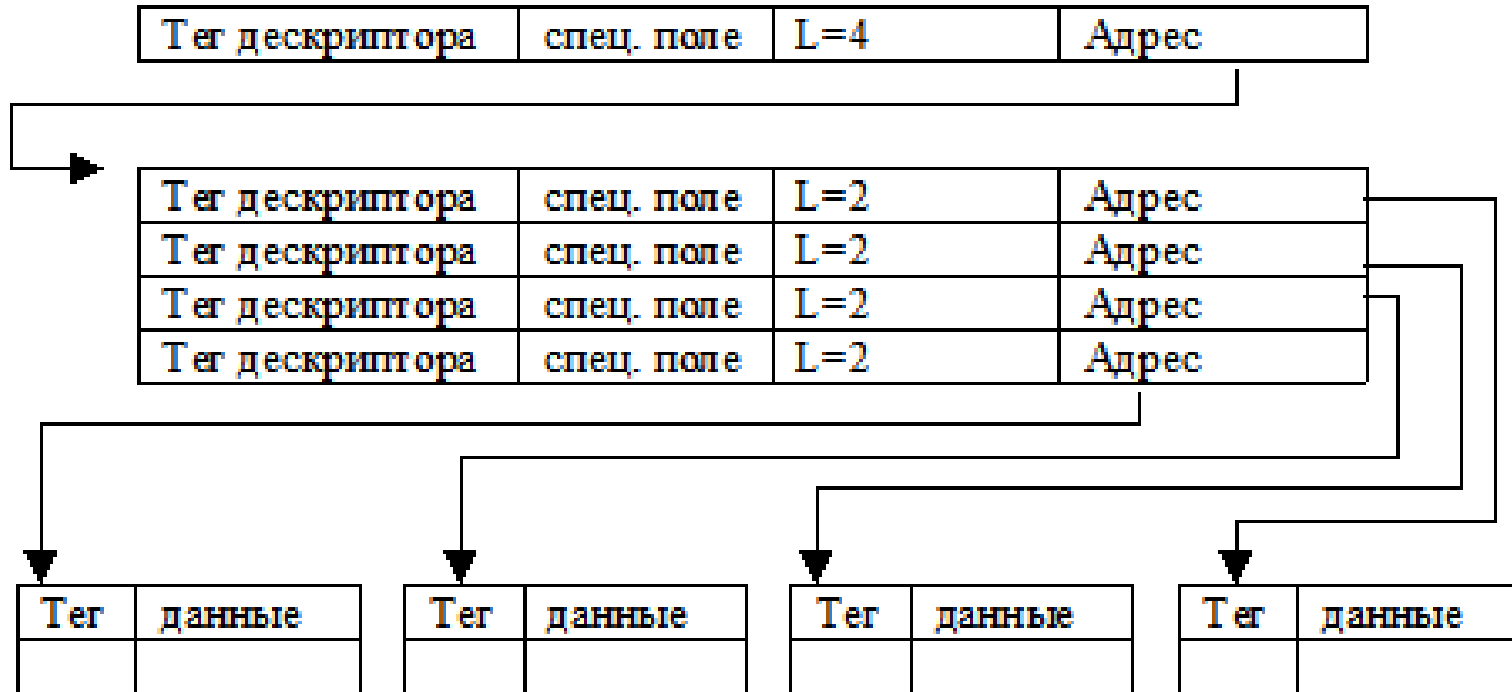
Могут применяться как с тегами так и без них.

Дескриптор содержит сведения о:

- размере массива данных;
- местоположении массива в ОП или во внешней памяти;
- адресе начала массива;
- режиме защиты данных.



## Пример: описание двумерного массива дескрипторами древовидной структуры.



*Наличие в архитектуре ЭВМ дескрипторов подразумевает, что обращение к информации в памяти производится через них, что можно рассматривать как дальнейшее развитие аппарата косвенной адресации.*

# Тип архитектуры процессора

**CISC** (complex instruction set computing или complex instruction set computer) — тип процессорной архитектуры, которая характеризуется следующим набором свойств:

- нефиксированное значение длины команды;
- арифметические действия кодируются в одной команде;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию.

**RISC** (reduced instruction set computer — компьютер с набором коротких [простых, быстрых] команд) — архитектура процессора, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения меньшим.

*Это также облегчает повышение тактовой частоты и делает более эффективной суперскалярность (распараллеливание инструкций между несколькими исполнительными блоками).*

**VLW** (very long instruction word — «очень длинная машинная команда») — архитектура с несколькими вычислительными устройствами. Одна инструкция процессора содержит несколько операций, выполняемых параллельно. Фактически это «видимое программисту» микропрограммное управление. Машинный код представляет собой немного свёрнутый микрокод для непосредственного управления аппаратурой.

# CISC

Представитель CISC-архитектуры: процессоры на основе команд **x86**.

Процессоры x86 и x86-64 являются одними из самыми распространённых. Они доминируют в сегментах рабочих станций, персональных компьютеров, серверов начального и среднего уровня, а также мейнфреймов

Поздние x86-процессоры (Intel Pentium 4, Pentium D, Core, AMD Athlon, Phenom) **имеют RISC-ядро**, хотя и CISC-совместимы. Формально они считаются гибридными. В гибридных CISC-процессорах CISC-инструкции преобразовываются в набор внутренних RISC-команд, при этом одна команда x86 может порождать несколько RISC-команд (в случае процессоров типа P6 — до четырёх RISC-команд в большинстве случаев), исполнение команд происходит на **суперскалярном конвейере** одновременно по несколько штук.

Основной недостаток CISC-архитектуры в сравнении с RISC — более сложный подход к распараллеливанию вычислений на уровне **микроархитектуры**.

# RISC

## Мотивы

Многие компиляторы не задействовали все возможности сложных машинных инструкций.

На сложные методы адресации уходит много времени из-за дополнительных обращений к медленной памяти.

Лучше исполнять более простые команды. При этом процессор упрощается. В нём остаётся место для большего числа регистров, за счёт которых можно сократить количество обращений к памяти.

Перенос большей части программы на регистры внутри процессора позволяет использовать более высокую его тактовую частоту вместо обращения к общей памяти с меньшей тактовой частотой.

# RISC

## Особенности

- Фиксированная длина машинных команд и простой формат команды.
- Специализированные команды для операций с памятью — чтения или записи. Операции вида Read-Modify-Write отсутствуют. Любые операции типа «изменить» выполняются только над содержимым регистров.
- Большое количество регистров общего назначения (32 и более).
- Отсутствие поддержки операций вида «изменить» над укороченными типами данных — байт, 16-разрядное слово.
- Отсутствие микропрограмм внутри самого процессора. То, что в CISC-процессоре выполняется микропрограммами, в RISC-процессоре выполняется как обыкновенный машинный код (хотя и помещённый в специальное хранилище) , не отличающийся принципиально от кода ядра ОС и приложений.

## Особенности

VLIW можно считать логическим продолжением идеологии RISC, расширяющей её на архитектуры с несколькими вычислительными узлами. В команде явно указывается, что именно должен делать каждый узел процессора. Поэтому длина команды может достигать 128 или даже 256 разрядов.

В суперскалярных процессорах также есть несколько вычислительных узлов, но задача распределения работы между ними решается аппаратно. Это сильно усложняет устройство процессора. В процессорах VLIW задача распределения решается во время компиляции. В командах явно указано, какое вычислительное устройство какую операцию должно выполнять.

## Реализации

Многопроцессорный вычислительный комплекс «Эльбрус-3».  
микропроцессоры серии «Эльбрус» («Эльбрус 2000», «Эльбрус S»).

## Достоинства

Упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор. Поскольку отсутствуют большие и сложные схемы управления, сильно снижается энергопотребление.

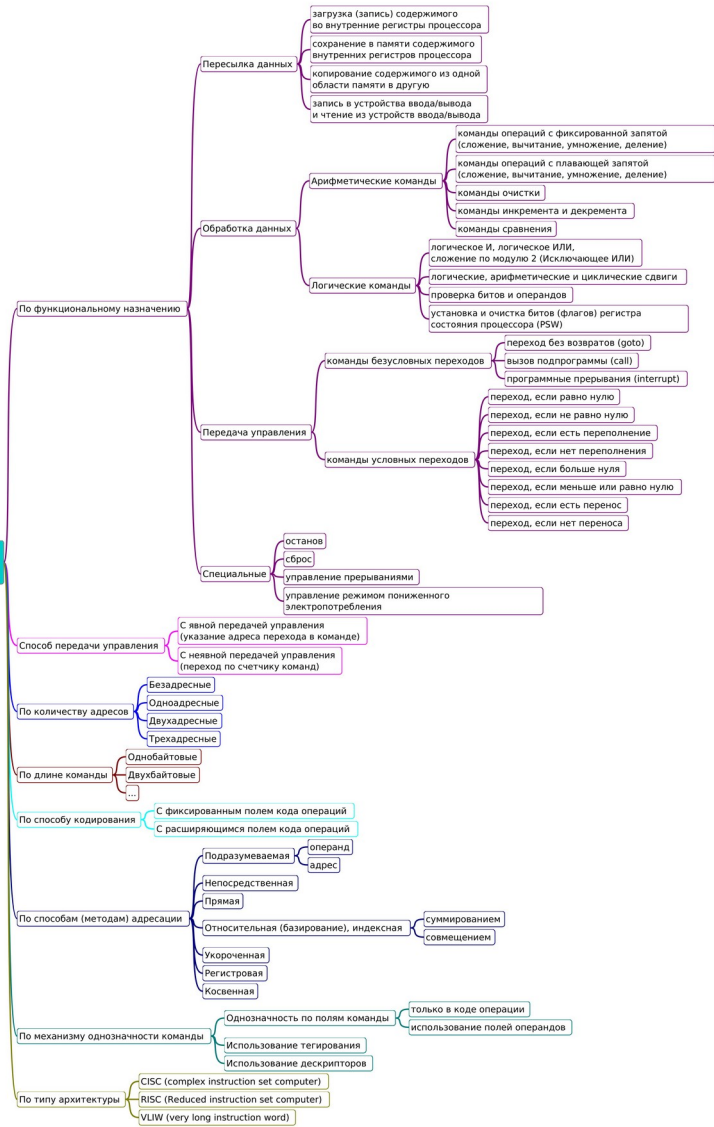
## Недостатки

Код обладает невысокой плотностью. Из-за большого количества пустых инструкций для простаивающих устройств программы могут быть гораздо длиннее, чем аналогичные программы для традиционных архитектур.

Архитектура выглядит довольно экзотической и непривычной для программиста. Из-за сложных внутренних зависимостей кода программирование вручную на уровне машинных кодов является сложным.

Приходится полагаться на оптимизацию компилятора.

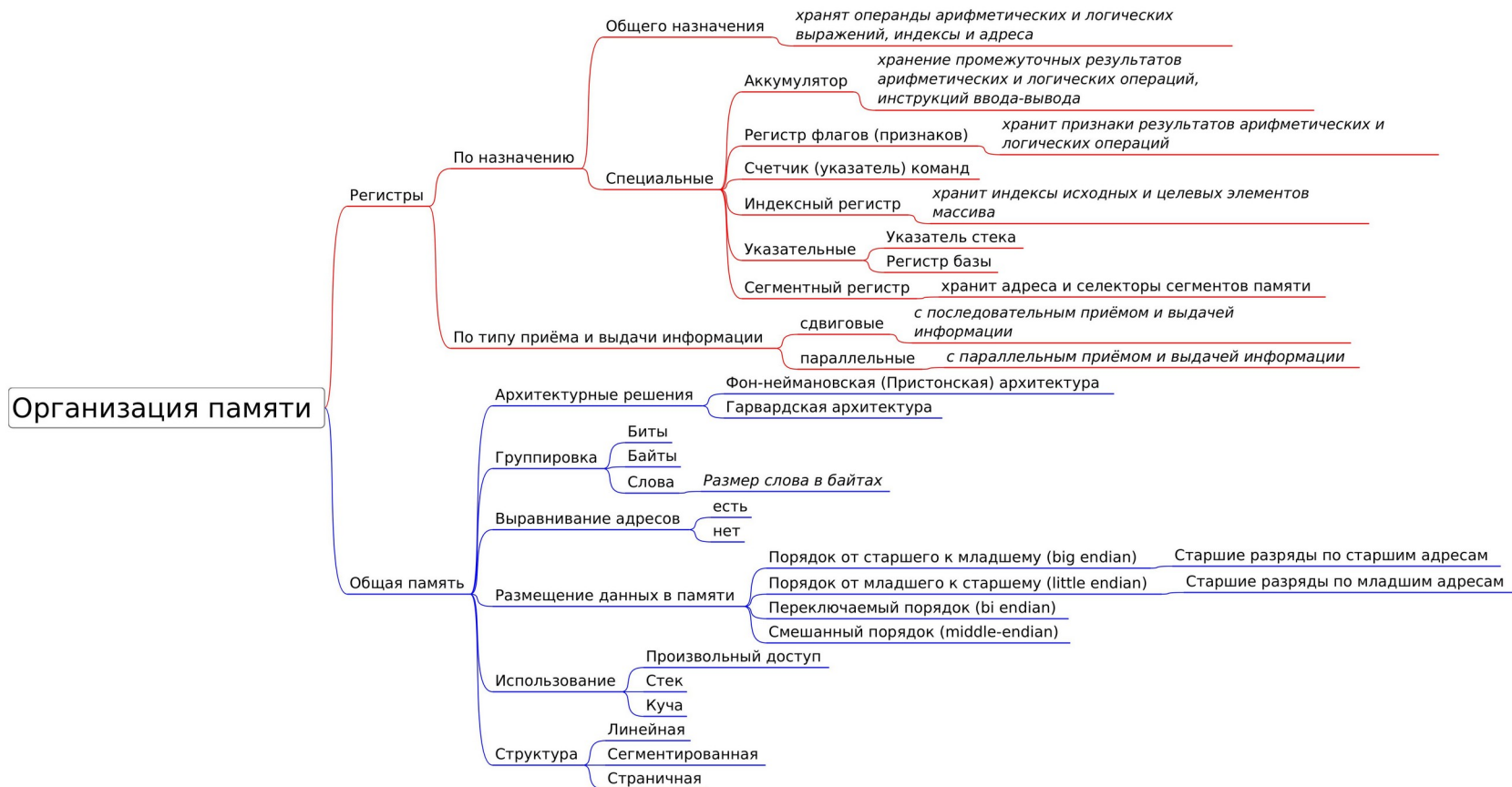
## Классификация системы команд





# Организация памяти

На уровне системы команд речь идет об организации памяти с точки зрения программиста



# Регистровая память

**Регистр процессора** — блок ячеек памяти, образующий сверхбыструю оперативную память (СОЗУ) внутри процессора; используется самим процессором и большей частью недоступен программисту: например, при выборке из памяти очередной команды она помещается в регистр команд, к которому программист обратиться не может.

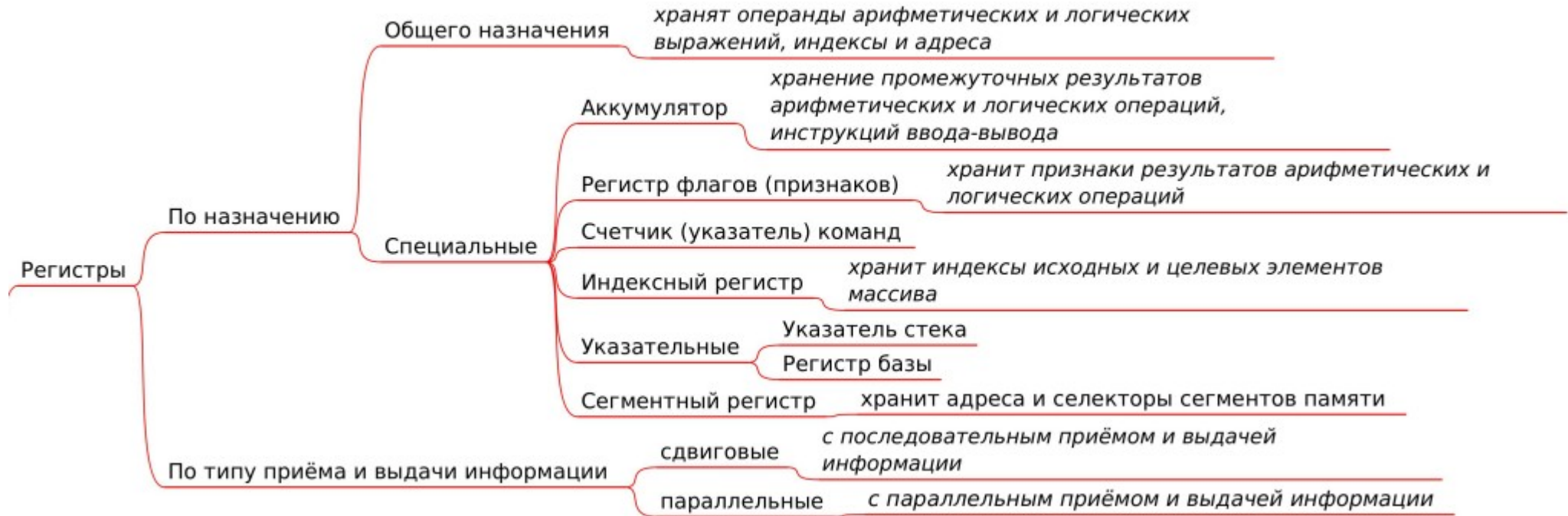
*(Википедия)*

**Регистры общего назначения (РОН)** — подмножество регистров процессора, используемое без ограничения в арифметических и логических операциях, но имеющее определённые ограничения в специальном применении. ора; используется самим процессором и большей частью недоступен программисту: например, при выборке из памяти очередной команды она помещается в регистр команд, к которому программист обратиться не может.

**Специальные регистры** — набор регистров, обеспечивающих работу процессора.

*Могут быть доступными и недоступными из системы команд.*

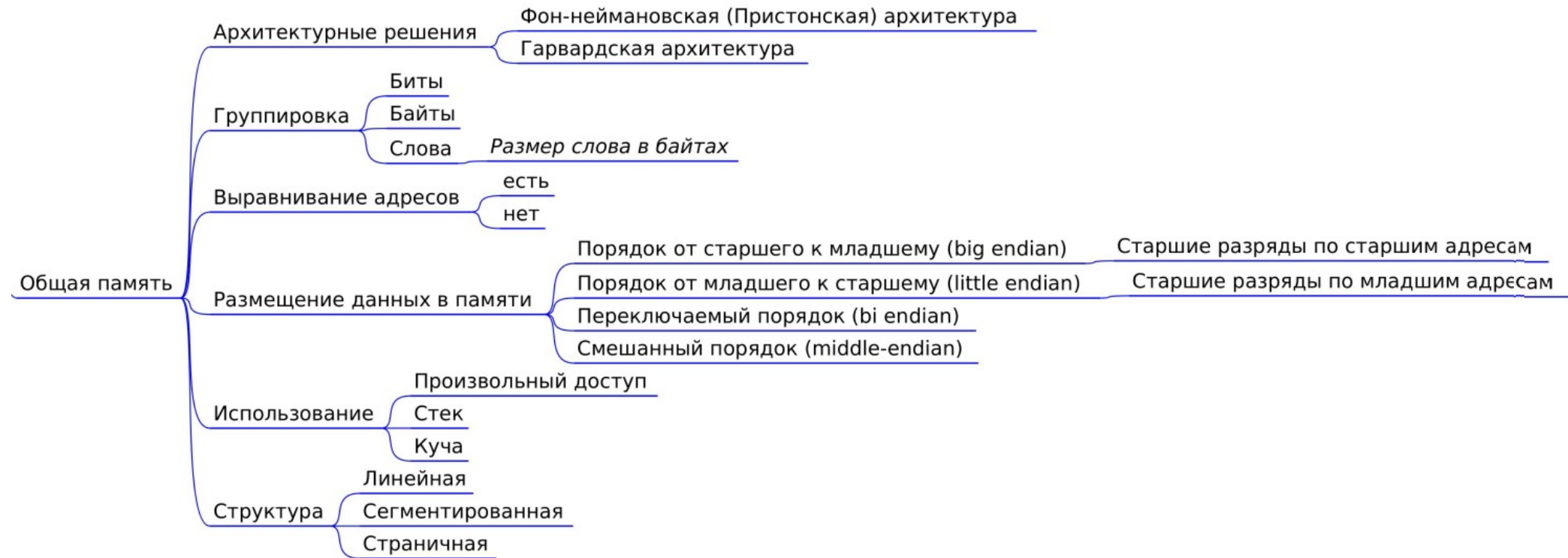
# Регистровая память



# Регистровая память в различных архитектурах

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 занулен
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 <sup>[4]</sup>	16	различное	
ARM 64-bit <sup>[5]</sup>	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, 32 глобальных регистров, 224 регистра стека процедур <sup>[6]</sup>

# Общая память



# Модель фон Неймана

## Принципы фон Неймана

### ***1. Принцип однородности памяти***

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

### ***2. Принцип адресности***

Основная память состоит из пронумерованных ячеек. Процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.

### ***3. Принцип программного управления***

Все вычисления должны быть представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти и выполняются в естественной последовательности, то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

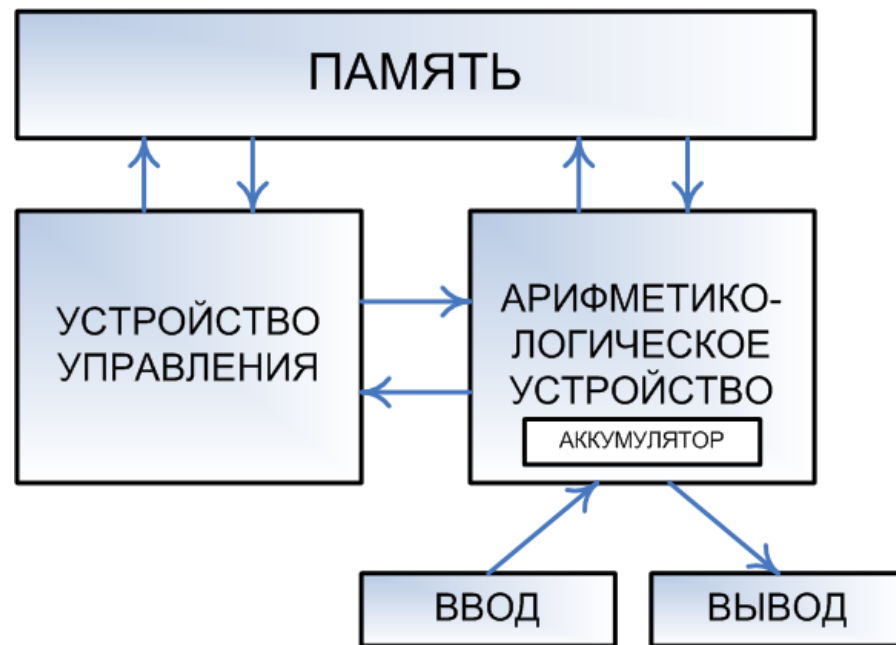
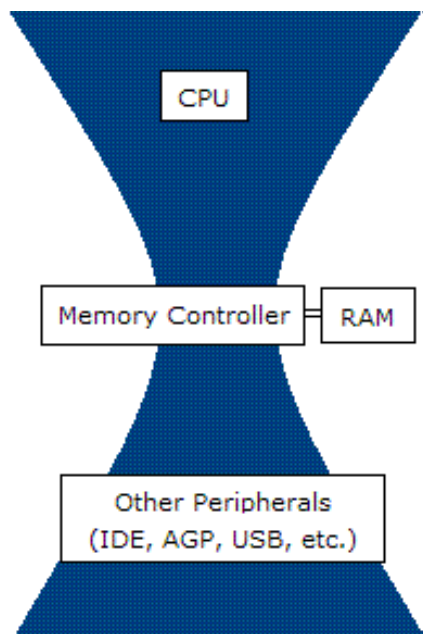
# Модель фон Неймана

## Недостатки фон Неймановской архитектуры

### 1. Ограничение пропускной способности памяти

Память программ и память данных не могут быть доступны в одно и то же время, что существенно ограничивает скорость работы процессора (сильнее, чем если бы программы и данные хранились в разных местах).

*Проблема решается на уровне микроархитектуры и распределения памяти*



# Гарвардская архитектура

## Осoвные особенности

- 1. Хранилище инструкций и хранилище данных представляют собой разные физические устройства.*
- 2. Канал инструкций и канал данных также физически разделены.*

## Достоинство

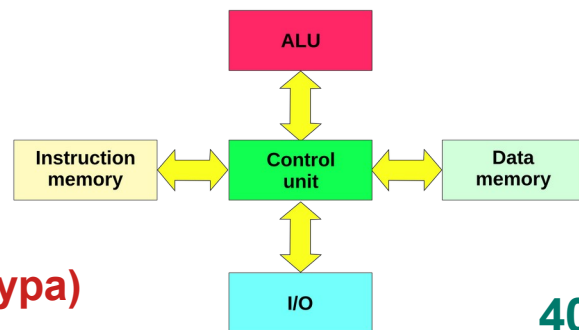
Процессор может считывать очередную команду и оперировать памятью данных одновременно и без использования кэш-памяти. При определенной сложности схемы он быстрее, чем компьютер с архитектурой фон Неймана, поскольку потоки команд и данных расположены на отдельных физически не связанных между собой аппаратных каналах.

Исходя из физического разделения шин команд и данных, разрядности этих шин могут различаться и физически не могут пересекаться.

## Недостаток

Высокая стоимость

**В настоящее время существуют гибридные решения (микроархитектура)**





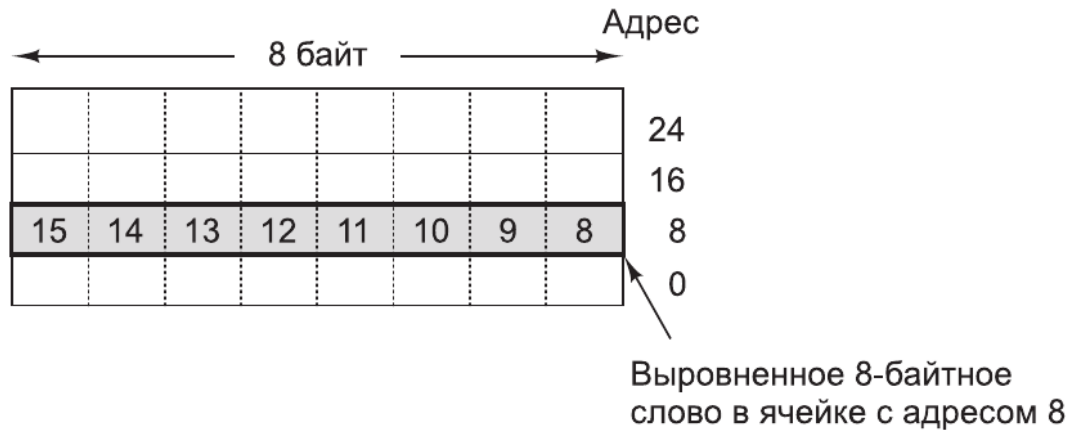
# Выравнивание данных

**Выравнивание данных в оперативной памяти компьютеров** — способ размещения данных в памяти особым образом для ускорения доступа.

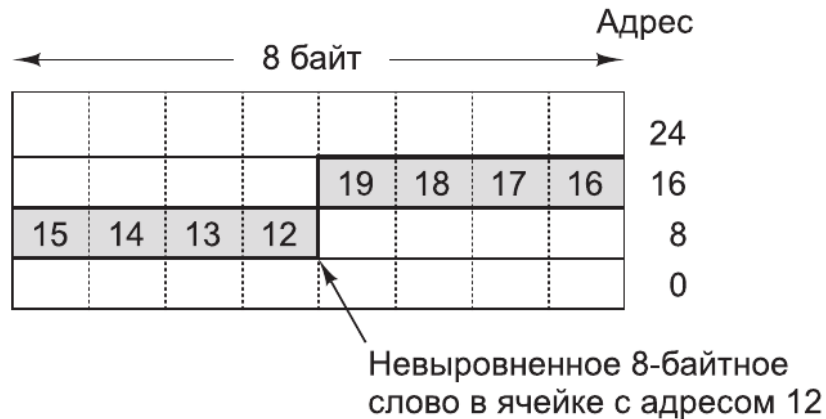
Машинное слово в разных процессорах имеет свой размер. Доступ к общей памяти осуществляется по словам. Если адрес обращения к памяти не кратен размеру слова, чтение будет происходить за два цикла обращения. Выравнивание данных при размещении в памяти по адресам, кратным размеру слова, позволяет ускорить выполнение программы.

адрес	данные
9	
8	это слово не выровнено
7	
6	
5	
4	
3	это слово выровнено
2	
1	
0	

# Выравнивание данных



а

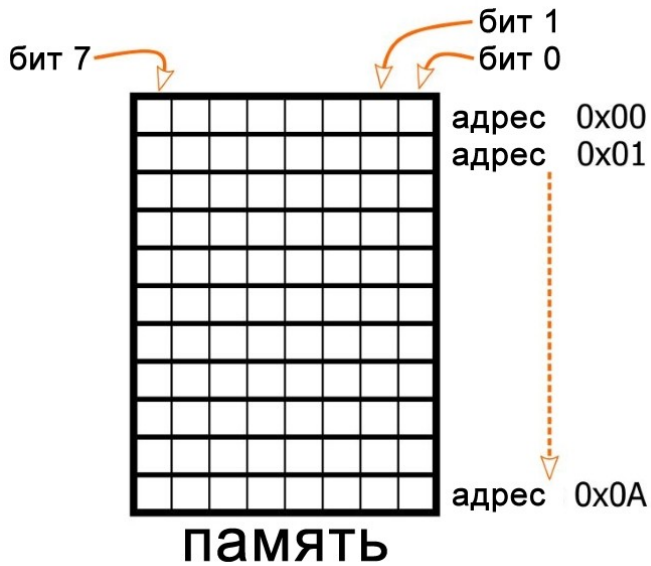


б

# Размещение данных в памяти

Последовательность байтов в памяти может быть заполнена длинными данными в разном порядке (или быть передана по каналам связи). Выбор порядка записи байтов произволен и часто определяется только соглашениями. **Варианты:**

- 1) **Порядок от старшего к младшему (big-endian)**. Соответствует привычному порядку записи арабских цифр ( $A_n, \dots, A_0$ ).
- 2) **Порядок от младшего к старшему (little-endian)**.  $A_0, \dots, A_n$ .
- 3) **Переключаемый порядок (bi-endian)**. Многие процессоры могут работать и в порядке от младшего к старшему, и в обратном. Обычно порядок байтов выбирается программно во время инициализации операционной системы, но может быть выбран и аппаратно переключками на материнской плате.
- 4) **Смешанный порядок (middle-endian)**. Иногда используется при работе с числами, длина которых превышает машинное слово. Число представляется последовательностью машинных слов, которые записываются в формате, естественном для данной архитектуры, но сами слова следуют в обратном порядке. В процессорах VAX и ARM используется смешанное представление для длинных вещественных чисел.



**прямой порядок  
big endian**

D <sub>31</sub>	D <sub>30</sub>	D <sub>29</sub>	D <sub>28</sub>	D <sub>27</sub>	D <sub>26</sub>	D <sub>25</sub>	D <sub>24</sub>	0x00
D <sub>23</sub>	D <sub>22</sub>	D <sub>21</sub>	D <sub>20</sub>	D <sub>19</sub>	D <sub>18</sub>	D <sub>17</sub>	D <sub>16</sub>	0x01
D <sub>15</sub>	D <sub>14</sub>	D <sub>13</sub>	D <sub>12</sub>	D <sub>11</sub>	D <sub>10</sub>	D <sub>9</sub>	D <sub>8</sub>	0x02
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	0x03

**обратный порядок  
little endian**

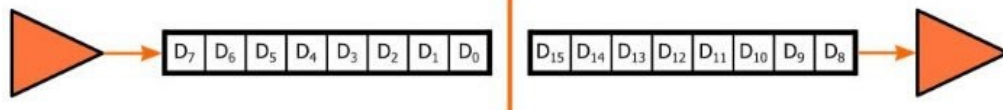
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	0x00
D <sub>15</sub>	D <sub>14</sub>	D <sub>13</sub>	D <sub>12</sub>	D <sub>11</sub>	D <sub>10</sub>	D <sub>9</sub>	D <sub>8</sub>	0x01
D <sub>23</sub>	D <sub>22</sub>	D <sub>21</sub>	D <sub>20</sub>	D <sub>19</sub>	D <sub>18</sub>	D <sub>17</sub>	D <sub>16</sub>	0x02
D <sub>31</sub>	D <sub>30</sub>	D <sub>29</sub>	D <sub>28</sub>	D <sub>27</sub>	D <sub>26</sub>	D <sub>25</sub>	D <sub>24</sub>	0x03

*Хранение данных с прямым порядком и с обратным порядком. "D" относится к 32-разрядному слову данных, а номера индексов указывают на отдельные биты от MSb (D<sub>31</sub>) до LSb (D<sub>0</sub>)*

**обратный порядок little endian**  
(первым передается младший байт)



**прямой порядок big endian**  
(первым передается старший байт)



```

1  #include <stdio.h>
2
3  union {
4      unsigned int i;
5      char c[sizeof(unsigned int)];
6  } foo;
7
8
9  int main() {
10     foo.i = 1;
11     if (foo.c[0] == 1) {
12         printf("Little endian\n");
13     } else {
14         printf("Big endian\n");
15     }
16
17
18     foo.i = 0x01020304;
19     for(int i = 0; i < sizeof(unsigned int); i++) {
20         printf("%p: %X\n", &foo.c[i], (unsigned int)foo.c[i]);
21     }
22     return 0;
23 }

```

```

Little endian
0x55c3bc49403c: 4
0x55c3bc49403d: 3
0x55c3bc49403e: 2
0x55c3bc49403f: 1

```

# BIG-ENDIAN



# LITTLE-ENDIAN



Big Endian	Little Endian	Оба варианта
Архитектуры процессоров		
AVR32 FR-V H8300 PA-RISC S390 Motorola 680x0 PowerPC SPARC	Alpha CRIS Blackfin Intel 64 IA-32 (x86) MN10300 AT91SAM7	ARM SuperH (sh) M32R MIPS Xtensa
Протоколы		
TCP/IP	USB	

# *Используемые источники*

1. Таненбаум Э. Архитектура компьютера. 6-е изд. – СПб.: Изд. Питер, 2017. – 816 с.
2. Майерс Г.Дж. Архитектура современных ЭВМ. В 2-х книгах. – М.: Мир, 1985. – 366+311 с.

## **Википедия**

Система команд: [https://ru.wikipedia.org/wiki/Система\\_команд](https://ru.wikipedia.org/wiki/Система_команд)

Регистр процессора: [https://ru.wikipedia.org/wiki/Регистр\\_процессора](https://ru.wikipedia.org/wiki/Регистр_процессора)

Архитектура фон Неймана: [https://ru.wikipedia.org/wiki/Архитектура\\_фон\\_Неймана](https://ru.wikipedia.org/wiki/Архитектура_фон_Неймана)

Гарвардская архитектура: [https://ru.wikipedia.org/wiki/Гарвардская\\_архитектура](https://ru.wikipedia.org/wiki/Гарвардская_архитектура)

Порядок байтов: [https://ru.wikipedia.org/wiki/Порядок\\_байтов](https://ru.wikipedia.org/wiki/Порядок_байтов)