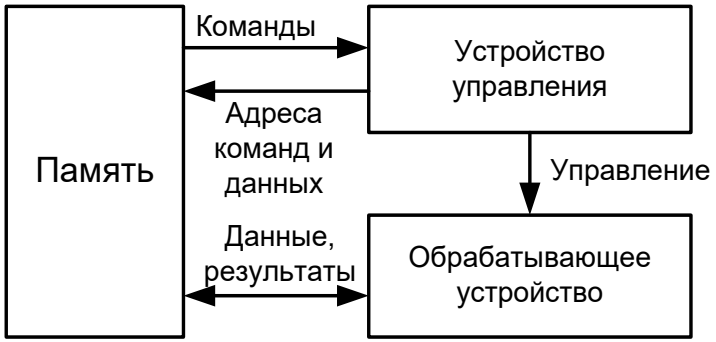
IX. Принципы построения и архитектура ЭВМ

Общие принципы построения современных ЭВМ

Принципы Фон-Неймана

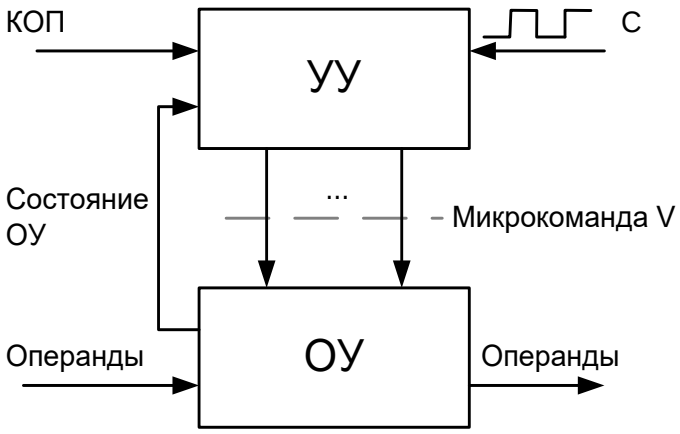
-Двоичное кодирование информации
-Программное управление
-Адресность памяти
-*Однородность памяти*

ОКОД, SISD



-Гарвардская архитектура
(ОП для хранения команд и ОП для хранения данных)
Принстонская архитектура
(ОП для хранения команд и данных)

Принципы микропрограммного управления



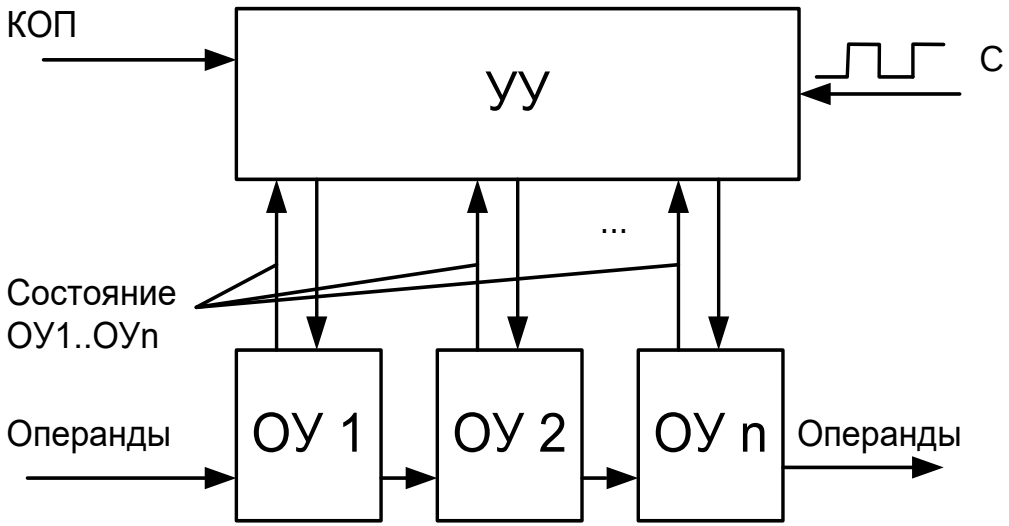
Любое цифровое устройство можно рассматривать, как совокупность операционного и управляющего блока.

Любая команда или последовательность команд реализуется в операционном блоке за несколько тактов

Последовательность сигналов управления должна выдаваться устройством управления в соответствии с поступающей на вход командой и текущим состоянием операционного блока

Состояние линий управления в каждом такте задает микрокоманду. Совокупность микрокоманд, необходимых для реализации команды называется микропрограммой.

Принцип конвейерной обработки

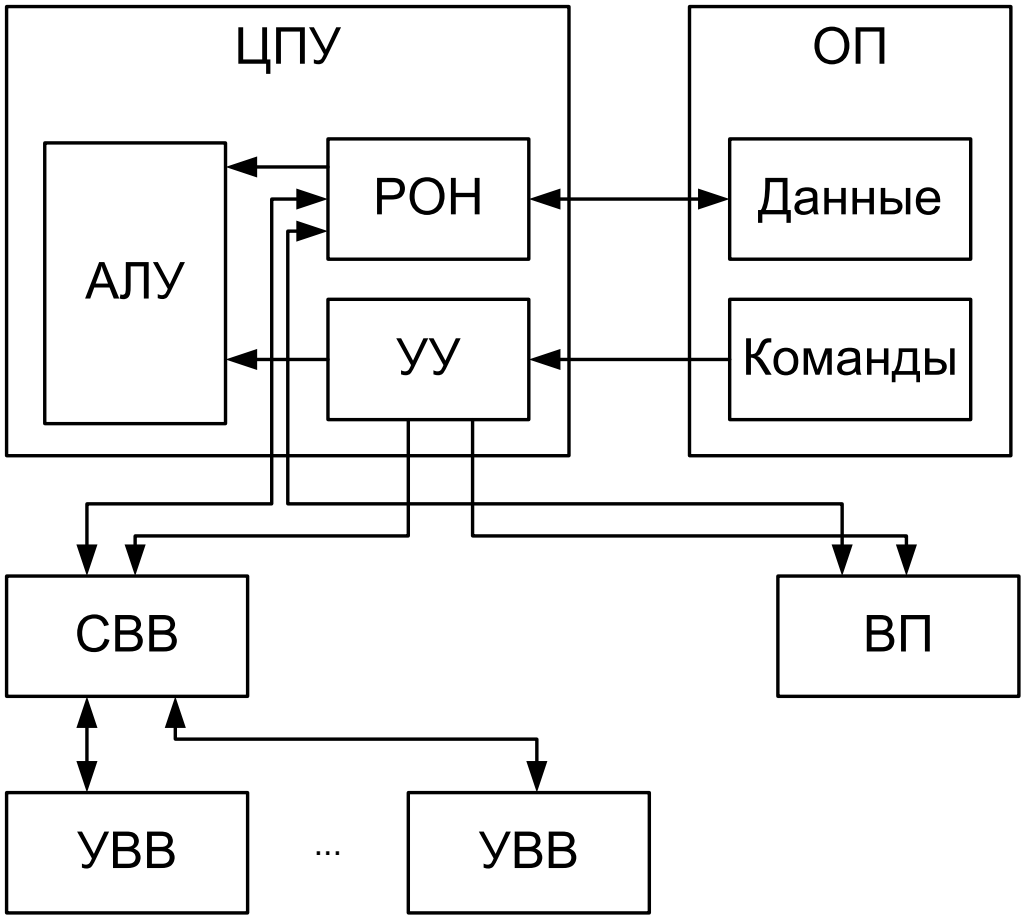


Конвейерная обработка представляет собой процесс, при котором сложные действия разделяются на более короткие стадии. Их параллельное выполнение для последовательности действий позволяет более полно использовать обрабатывающие ресурсы конвейера.

Структура современных ЭВМ

-Центральное процессорное устройство (ЦПУ).
  - Арифметико-логическое устройство (АЛУ)
  -Устройство управления (УУ)
  -Регистры общего назначения (РОН)
-Основная память
-Система ввода-вывода
-Управление внешними устройствами
-Внешняя память
-Система передачи информации
-Система синхронизации
-Система прерываний
-Система прямого доступа к памяти
-Система подвода питания/земли и система энергосбережения
Система контроля и повышения отказоустойчивости

ЭВМ с непосредственными связями

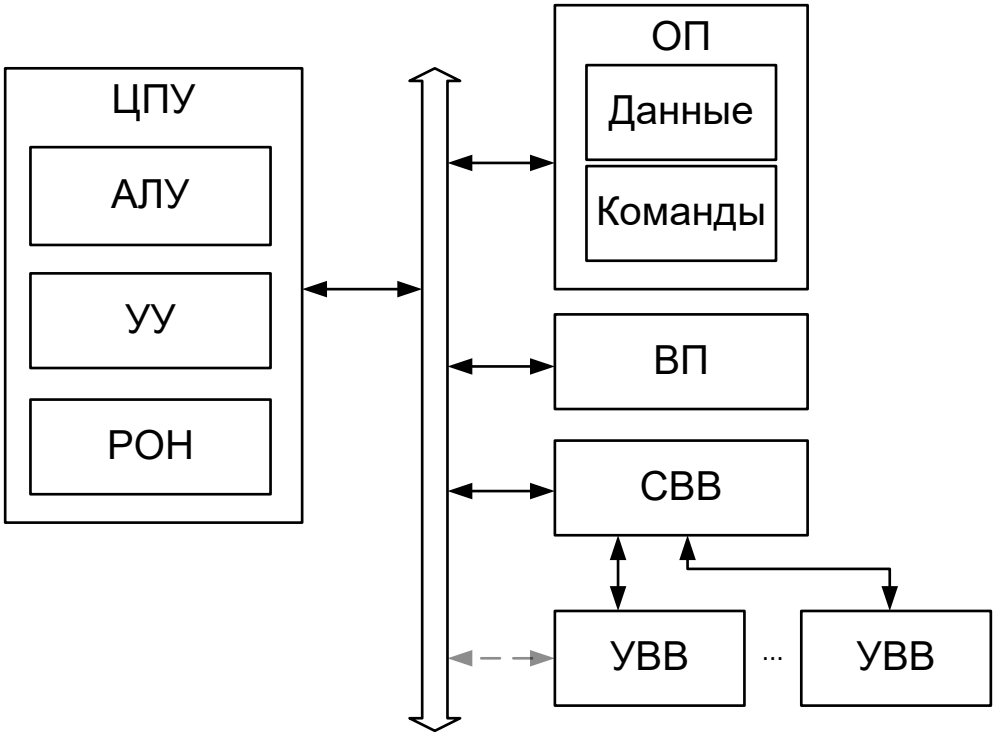


-(+) При построении оптимальных линий связи вычислительная машина обладает максимальным быстродействием.

(-) Ограничение на количество выводов микросхем не позволяет организовать широкие шины.

(-) Реконфигурация системы требует изменения характеристик линий связи.

ЭВМ с магистральной структурой

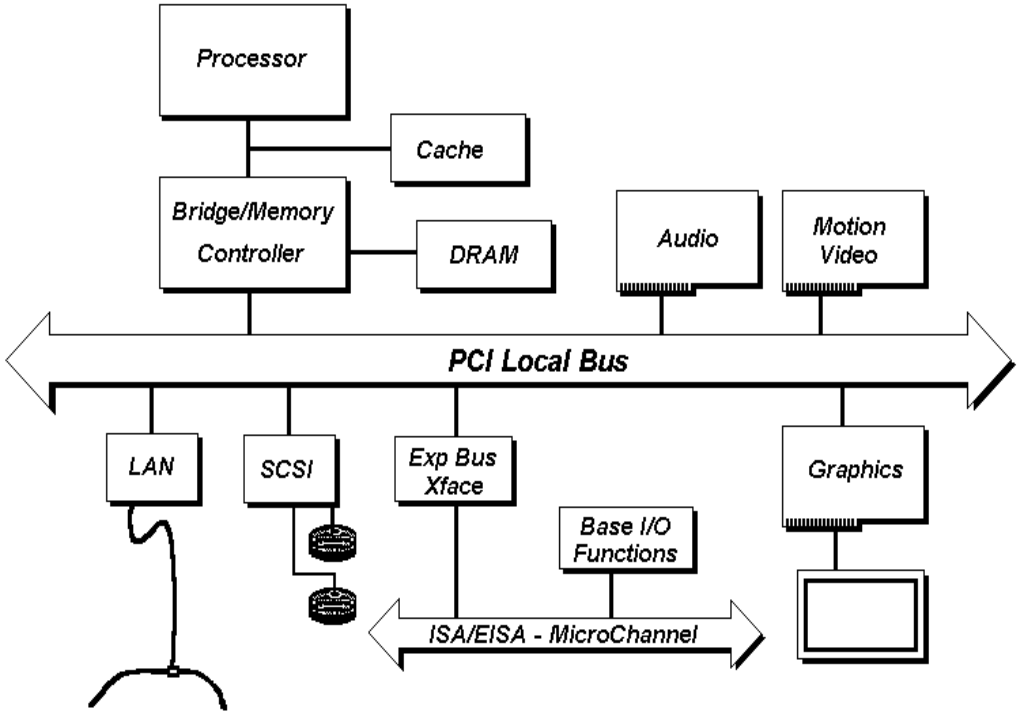


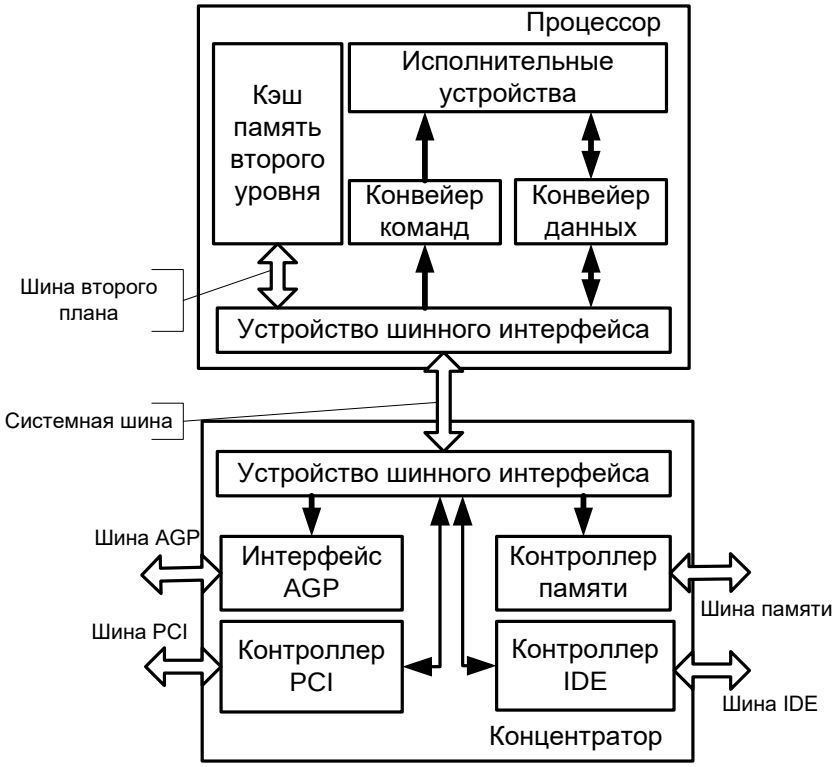
(+) Общая шина позволяет легко реконфигурировать систему.

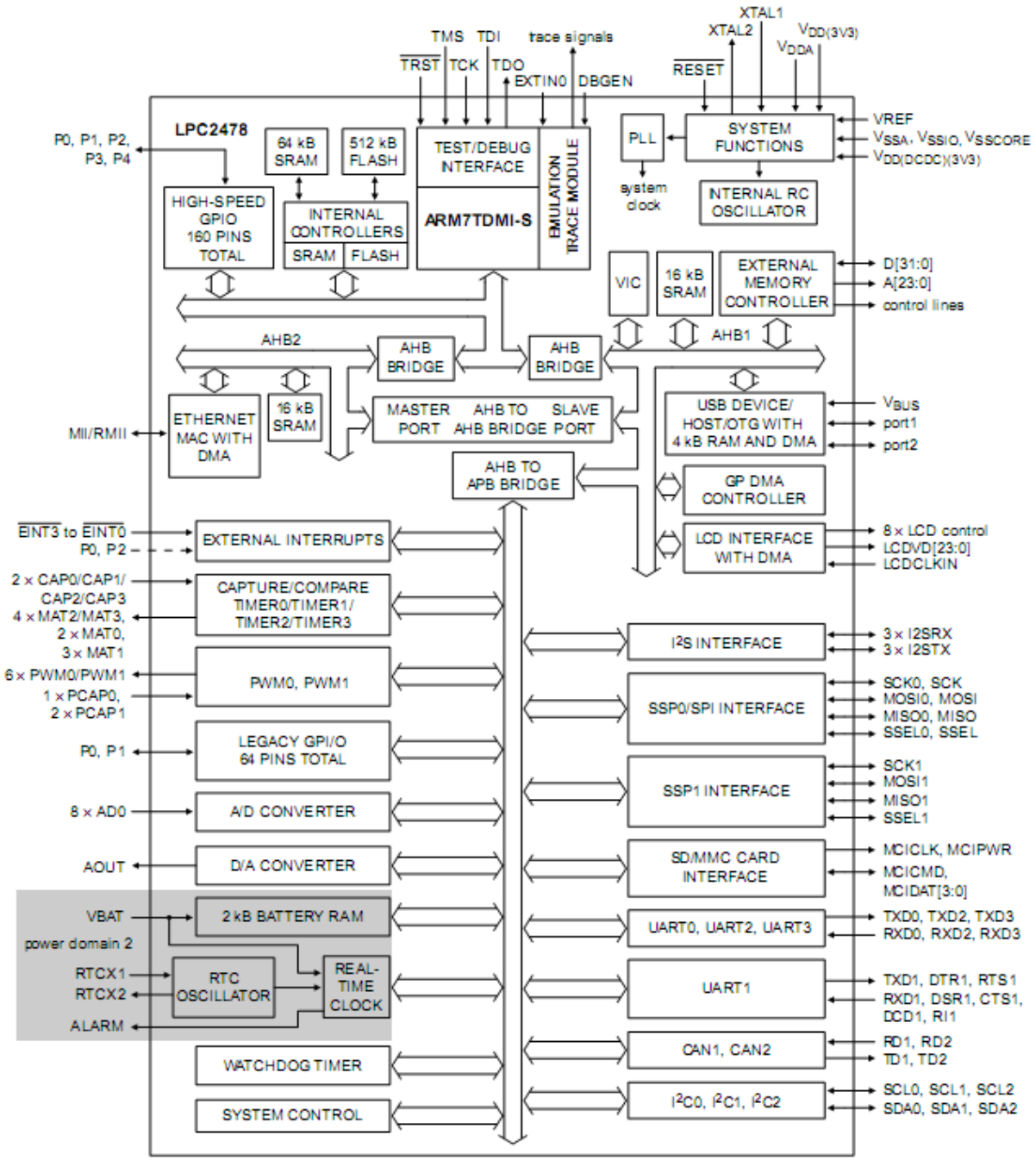
(-) Шина является узким местом.

- Шина, используемая всеми устройствами системы для передачи данных называется системной.
- Для разгрузки системной шины используют иерархию шин.
- По назначению, разделяют шины адреса, шины данных и шины управления.

Примеры построения ЭВМ с иерархией шин

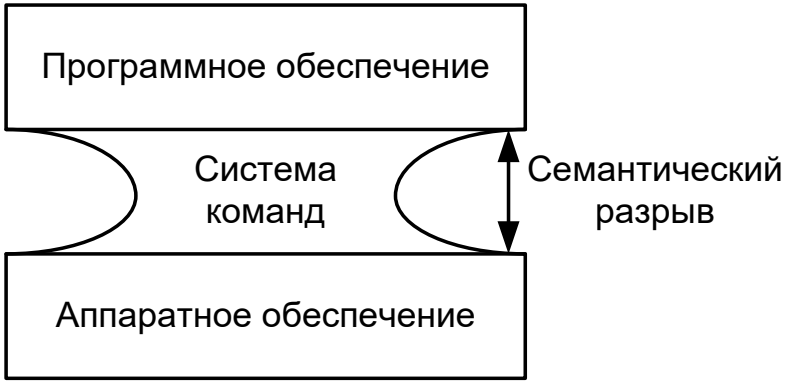





LPC2478
P0, P1, P2, P3, P4
64 kB SRAM
512 kB FLASH
HIGH-SPEED GPIO 160 PINS TOTAL
INTERNAL CONTROLLERS
SRAM
FLASH
TEST/DEBUG INTERFACE
ARM7TDMI-S
EMULATION TRACE MODULE
TMS
TDI
trace signals
TRST
TCK
TDO
EXTIN0
DBGEN
RESET
XTAL1
XTAL2
VDD(3V3)
VDDA
PLL
system clock
SYSTEM FUNCTIONS
VREF
VSSA, VSSIO, VSSCORE
VDD(DCDC)(3V3)
INTERNAL RC OSCILLATOR
VIC
16 kB SRAM
EXTERNAL MEMORY CONTROLLER
D[31:0]
A[23:0]
control lines
AHB2
AHB BRIDGE
AHB BRIDGE
AHB1
MII/RMII
ETHERNET MAC WITH DMA
16 kB SRAM
MASTER AHB TO AHB BRIDGE
SLAVE PORT PORT
USB DEVICE/ HOST/OTG WITH 4 kB RAM AND DMA
VBUS
port1
port2
AHB TO APB BRIDGE
GP DMA CONTROLLER
LCD INTERFACE WITH DMA
8 x LCD control
LCDVD[23:0]
LCDCLKIN
EINT3 to EINT0
P0, P2
EXTERNAL INTERRUPTS
2 x CAP0/CAP1/ CAP2/CAP3
4 x MAT2/MAT3, 2 x MAT0, 3 x MAT1
CAPTURE/COMPARE TIMER0/TIMER1/ TIMER2/TIMER3
6 x PWM0/PWM1
1 x PCAP0, 2 x PCAP1
PWM0, PWM1
P0, P1
LEGACY GPI/O 64 PINS TOTAL
8 x AD0
A/D CONVERTER
AOUT
D/A CONVERTER
VBAT
power domain 2
2 kB BATTERY RAM
RTCX1
RTCX2
RTC OSCILLATOR
REAL-TIME CLOCK
ALARM
WATCHDOG TIMER
SYSTEM CONTROL
I2S INTERFACE
3 x I2SRX
3 x I2STX
SSP0/SPI INTERFACE
SCK0, SCK
MOSI0, MOSI
MISO0, MISO
SSEL0, SSEL
SSP1 INTERFACE
SCK1
MOSI1
MISO1
SSEL1
SD/MMC CARD INTERFACE
MCICLK, MCIPWR
MCICMD, MCIDAT[3:0]
UART0, UART2, UART3
TXD0, TXD2, TXD3
RXD0, RXD2, RXD3
UART1
TXD1, DTR1, RTS1
RXD1, DSR1, CTS1, DCD1, RI1
CAN1, CAN2
RD1, RD2
TD1, TD2
I2C0, I2C1, I2C2
SCL0, SCL1, SCL2
SDA0, SDA1, SDA2

Основные тенденции развития ЭВМ

- Повышение степени интеграции элементной базы
  - -Увеличение набора команд
    - -Увеличение степени аппаратной поддержки.
- Обратная совместимость
- Наличие семантического разрыва



Проблема семантического разрыва

Технология программирования непрерывно развивается, что позволяет увеличивать функциональность программ и сокращать время их разработки. Создание проблемно-ориентированных языков высокого уровня усугубляет принципиальное отличие языка машинных команд, реализуемого компьютером, от языков, используемых при написании программ. Данная проблема носит название "семантического разрыва" и выражается в неоправданном падении производительности вычислительной системы.

Архитектура системы команд

В команде указывается, какую операцию выполнять (КОП), над какими операндами выполнять операцию, а также куда поместить операнд.

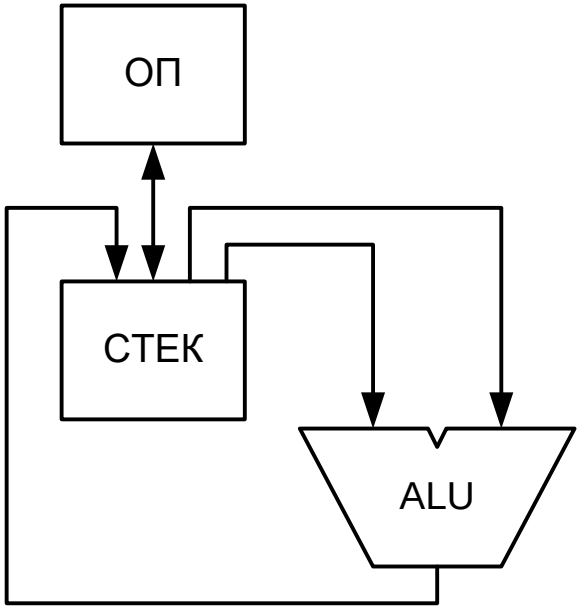
RISC – Reduced Instruction Set Computer; CISC – Complex Instruction Set Computer; VLIW – Very Long Instruction Word; ROSC - Removed Operand Set Computer

Сравнение CISC, RISC и VLIW архитектур СК

Характеристика	CISC	RISC	VLIW
Длина команды	Различная	Одинаковая	Одинаковая
Расположение полей в командах	Различное	Одинаковое	Одинаковое
Количество регистров	Малое. Регистры специализи-рованные	Большое. Регистры универсальные	Большое. Регистры универсальные
Доступ к памяти	Кодируется в команде. Выполняется по микрокоманде	Выполняется по специальной команде	Выполняется по специальной команде
Длительность выполнения команд	Различная	Одинаковая (для большинства команд)	Различная

Стековая архитектура СК

(+) При размещении операндов в стековой памяти (LIFO) архитектура команд упрощается (большое количество действий выполняется аппаратно)



Операции:
- занесение в стек (PUSH);
- извлечение из стека (POP);
- выполнение действий на стеком (извлечение операндов из вершины стека, выполнение действий, помещение результата в вершину стека)

Для выполнение арифметических операций их преобразуют к постфиксной форме (Польской записи).

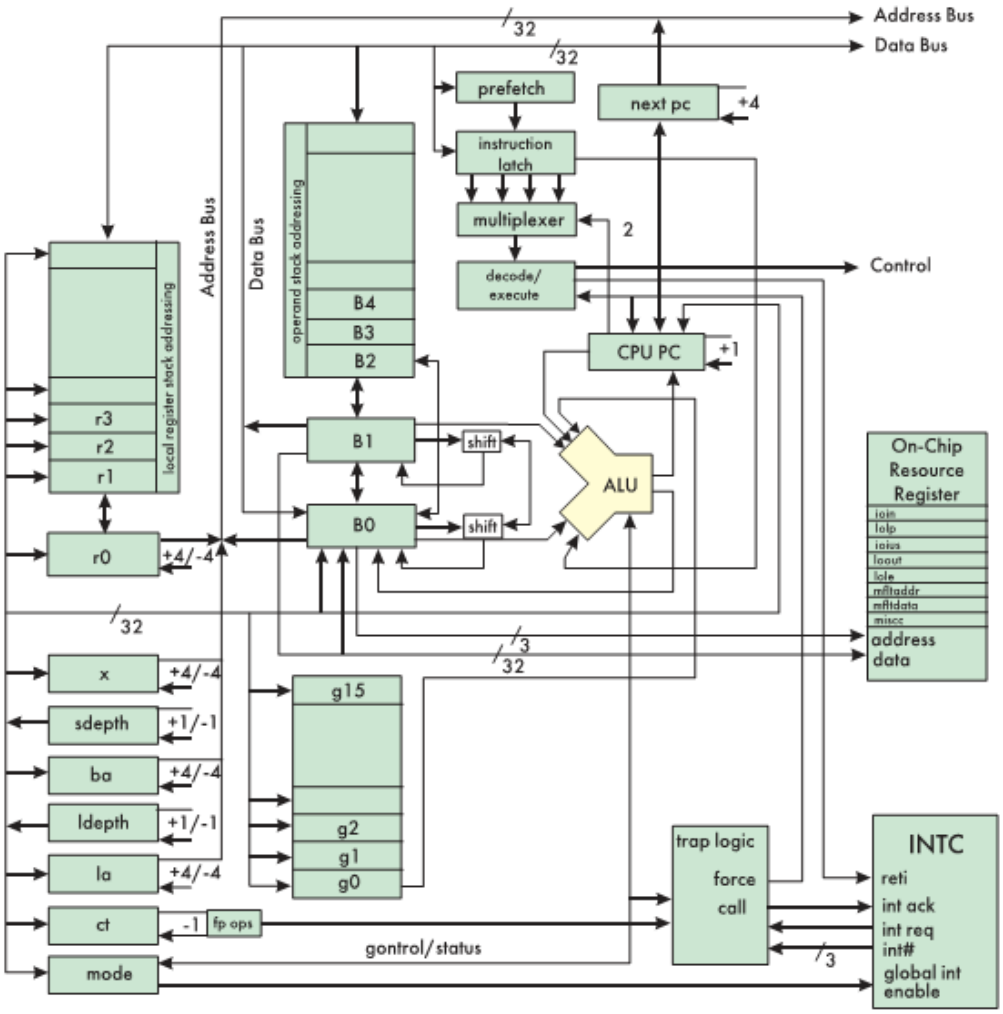
Пример: a = a + b * (c -d); Постфиксная форма: abcd-*+;
Действия: PUSH a; PUSH b; PUSH c; PUSH d; SUB; MUL; ADD; POP a.

(-) Отсутствие прямого доступа к памяти ограничивает область применения.
(-) Сложность организации параллельной обработки.

Стековые процессоры (Форт-процессоры)

ПРИМЕР

Блок-схема микропроцессора IGNITE



Сравнение выполнения программы на RISC-процессоре и на стековом микропроцессоре

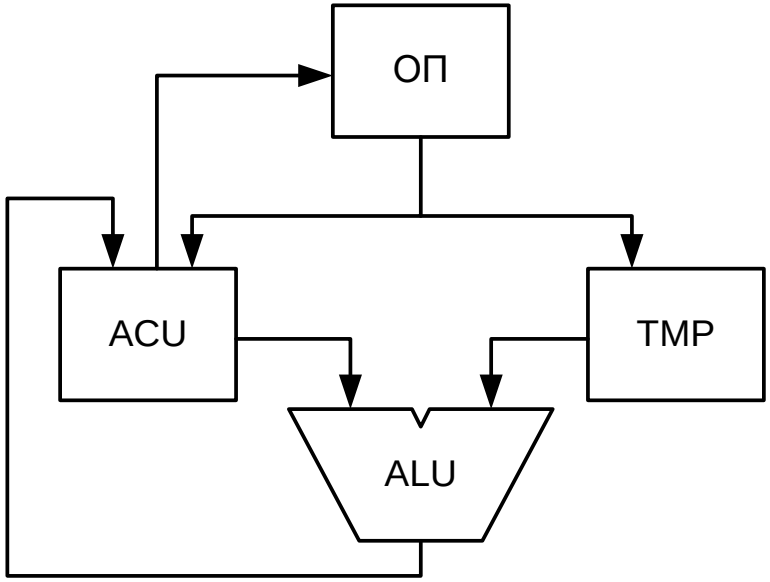
Номер команды	RISC MPU	IGNITE
1	add #1, g2, g5	push g1
		push g2
		inc #1
2	sub g1, g5, g5	sub
3	add g5, g3, g5	push g3
		add
4	shl g4, #1, temp	push g4
		shl #1
5	sub g5, temp, g5	sub
		pop g5
	Всего 20 байт	**Всего 10 байт**

Набор микросхем **TDS9092 FORTH CHIPS**

Использованы материалы статьи: Иосиф Каршенбойм, Стековые микропроцессоры, или новое - это хорошо забытое новое, Компоненты и технологии, №9, 2008

Аккумуляторная архитектура СК

Один из операндов должен обязательно находиться в специальном регистре-аккумуляторе. Результат также сохраняется в аккумулятору.



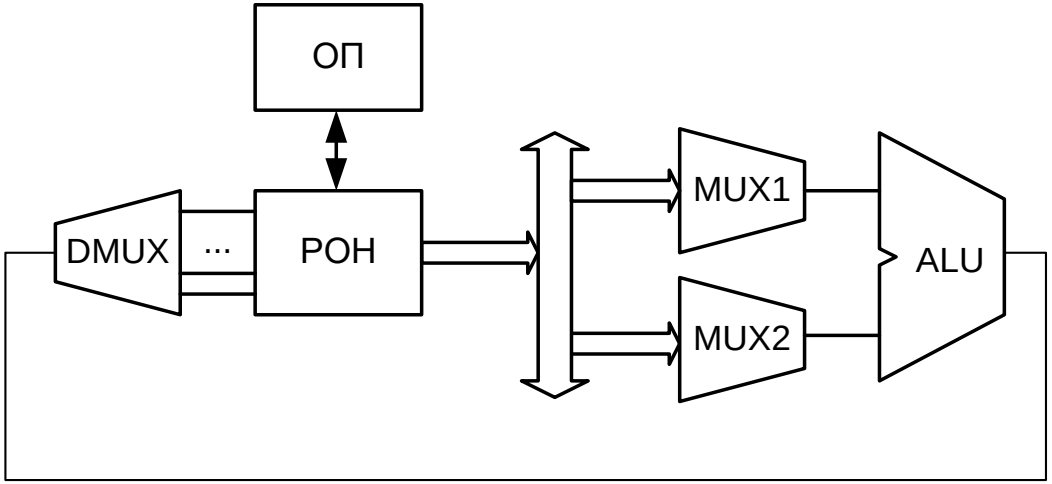
Операции:
- занесение в аккумулятор (LOAD);
- извлечение из аккумулятора (STORE);
- выполнение действий над операндами (извлечение первого операнда из аккумулятора, извлечение второго операнда из ОП и помещение во временный теневой регистр TMP, выполнение действий, помещение результата в аккумулятор).

Пример: a = a + b * (c -d); Определение троек: T1=c-d; T2=b*T1; T3=a+T2;
Действия:LOAD c; SUB D; MUL b; ADD a; STORE a.

(+) В команде необходимо указывать только адрес второго операнда.
(+) Ускоряются длинные вычисления (a*b/c+d-e).
(-) Наличие одного аккумулятора является узким местом, т.к. временно ненужный результат необходимо перезаписывать в другой регистр или ОП.

Регистровая архитектура СК

В состав процессора входит большое количество однотипных регистров. В команде необходимо указать номера регистров, хранящих операнды, а также номер регистра операнда.



Для данной архитектуры возможны варианты размещения операндов: оба операнда в памяти; один операнд в памяти и один в РОН; оба операнда в РОН.

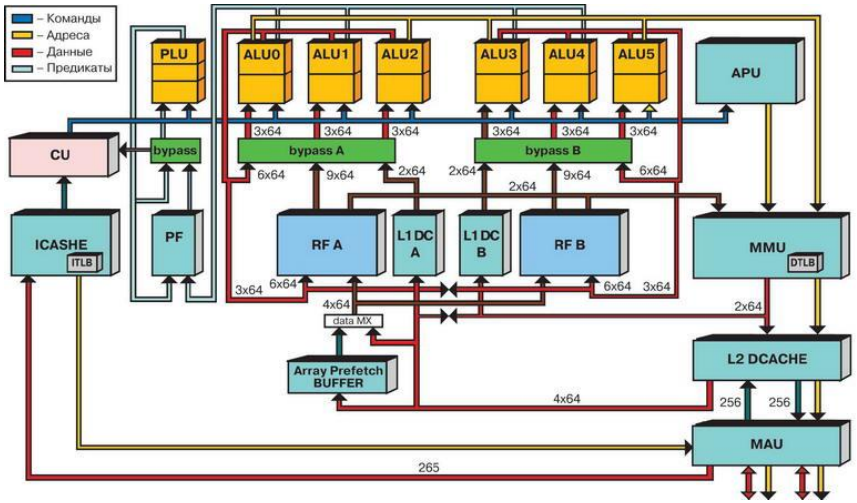
Для уменьшения размерности команд и для упрощения декодирования накладывают ограничения на размещение операндов.

Архитектура VLIW – Very Long Instruction Word

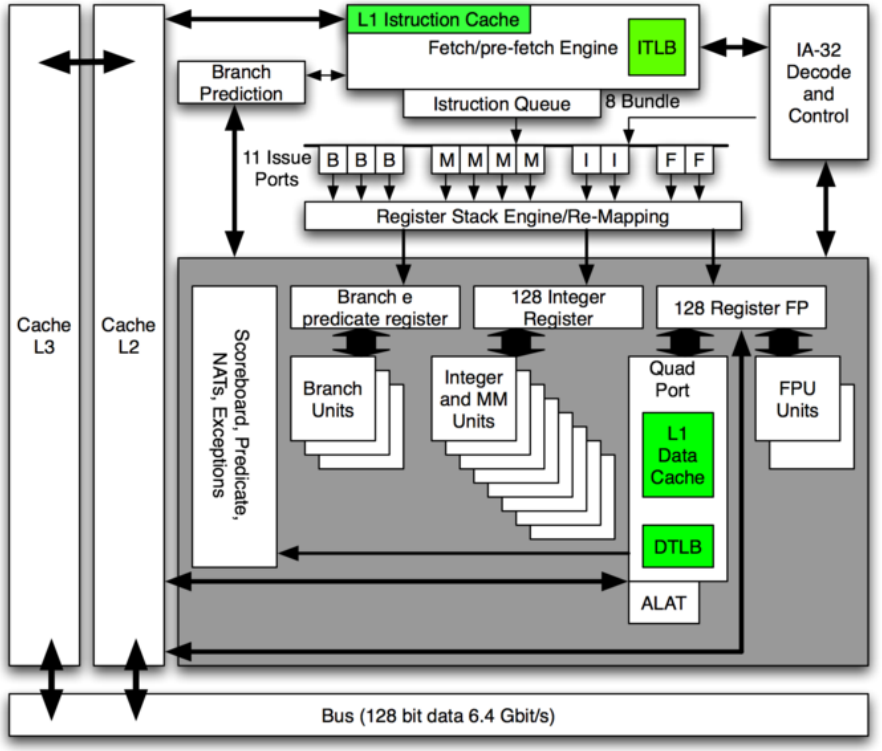
В процессорах VLIW задача распределения решается во время компиляции и в инструкциях явно указано, какое вычислительное устройство должно выполнять какую команду.

ПРИМЕР

Эльбрус-3 и его микропроцессорное исполнение Эльбрус 2000 (E2K) также являются VLIW процессорами.

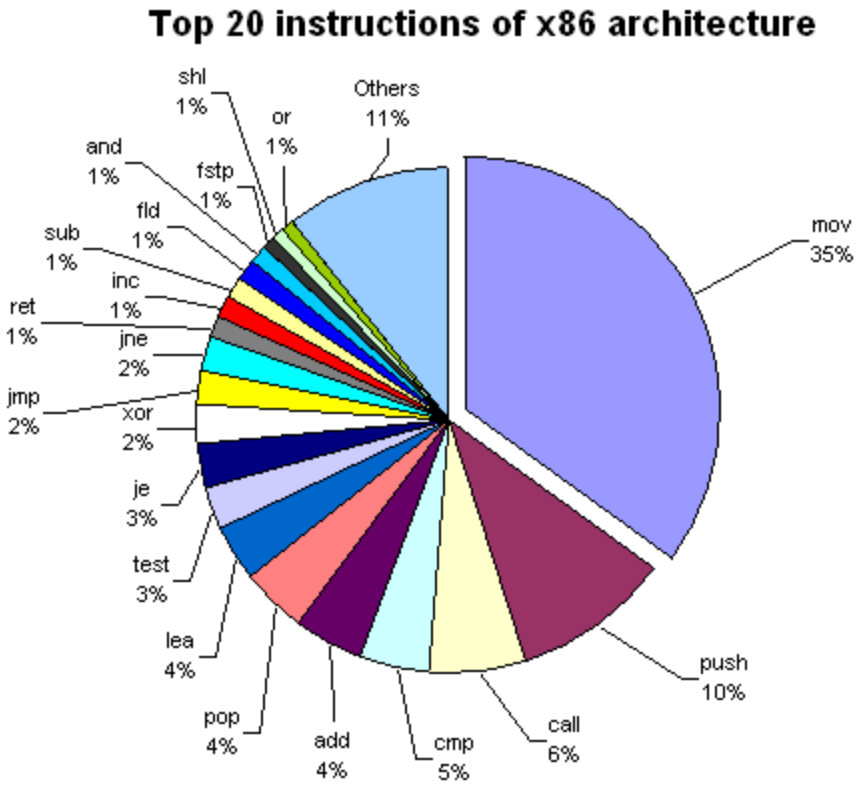


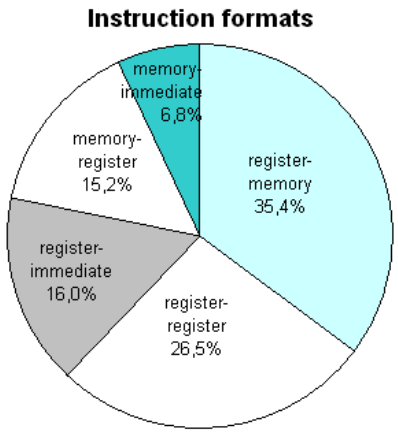
Микропроцессор Intel Itanium имеет как традиционную систему команд IA-32, так и систему команд «с явным параллелизмом» (англ. Explicitly Parallel Instruction Computing, EPIC), исполняемую VLIW-ядром

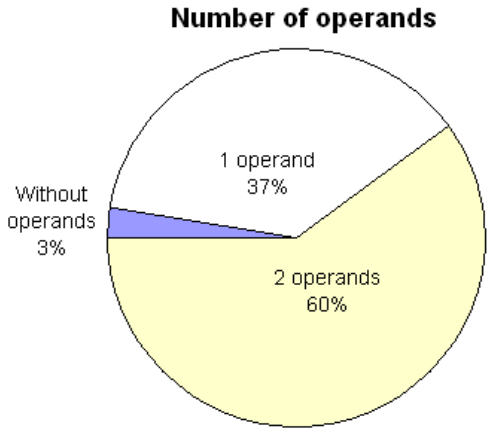


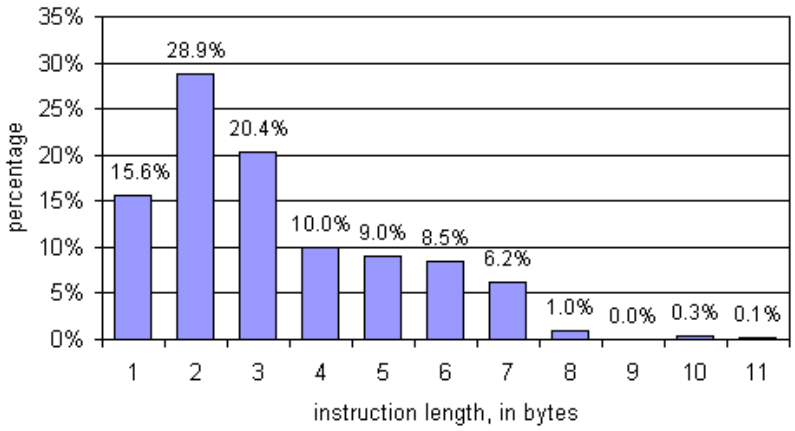
Вариант	(+)	(-)
Оба операнда находятся в регистрах	Простота аппаратной реализации. Простота параллельной обработки.	Избыточность в команде из-за сложности кодирования с кратностью 8 бит
Один операнд находится в регистре, а один в памяти	Код компактен. Данные поступают в ALU без промежуточного хранения в РОН	Наличие адреса в команде усложняет дешифрацию и сокращает возможное кол-во РОН, адресуемых в команде.
Оба операнда находятся в памяти	Код наиболее компактен. Возможность выполнения простых действий наиболее быстро без занесения в РОН	Команды имеют максимальную длину. Из-за наличия коротких и длинных команд трудно оптимизировать тракты передачи данных и декодеры инструкций

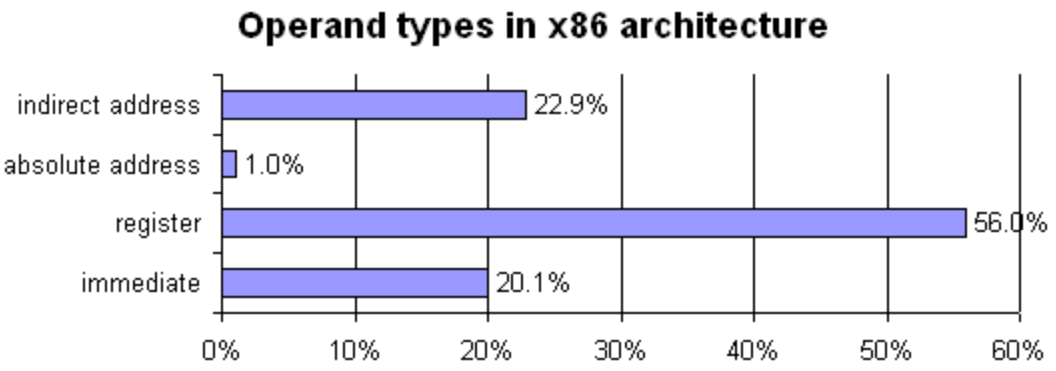
Статистические данные для x86 команд











Типы команд.

- Команды пересылки данных.
  - регистр-регистр
  - регистр-память
  - память-память
- Команды арифметической и логической обработки (сложение, вычитание, умножение, деление, инкремент, декремент, сравнение, операции над ЧПЗ, логические операции, операции сдвига).
  Сдвиг: логический, арифметический, циклический, циклический через дополнительным разряд.
- Команды работы со строками (могут быть реализованы набором других команд, однако удобны при работе с символьной информацией).
- Команды векторной обработки (позволяет выполнять однотипные действия над большим количеством однородных данных). Пример арифметики с насыщением:

```
  1011 0111 1010
+ 0001 1001 1000
  1100 1111 1111
```

- Команды преобразования: служат для табличного преобразования данных из одной системы кодов в другую (2-10 <-> 2)

•Команды ввода/вывода. Служат для управления, проверки состояния и обмена данными с периферийными устройствами.

- Команды вывода в порт
- Команды ввода из порта.

•Команды управления потоком команд. Данные команды служат для указания очередности выполняемых команд.

Вычисление адреса очередной команды может выполняться несколькими способами:

- увеличением адреса на длину исполненной (естественный порядок).
- изменением адреса на длину следующей (перешагивание)
- изменением адреса на значение, указанное в текущей команде (короткий переход).
- непосредственное указание следующей команды (длинный переход).

Перечисленные команды могут выполняться лишь по некоторому условию (уловные переходы).

Команды условного перехода составляют 80% команд управления.

Команды безусловного перехода: вызовы и возвраты из процедур, и.т.д.

Форматы команд.

Операционная часть	Адресная часть

1. Четырехадресная команда.

КОП	1 операнд	2 операнд	результат	Адр след ком.

2. Трехадресная команда

КОП	1 операнд	2 операнд	результат

3. Двухадресная команда.

КОП	1 операнд	2 оп-д/результат

Характерна для CISC-архитектуры

4. Аккумуляторная архитектура

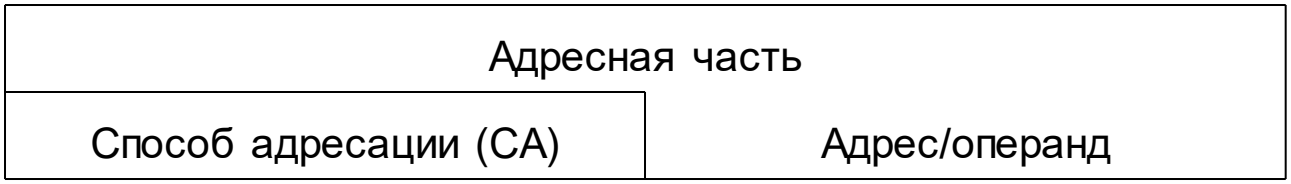
КОП	1 операнд

Второй операнд хранится в аккумуляторе. Данный формат команд характерен для RISC-архитектур.

5. Нульоперандная команда.

КОП

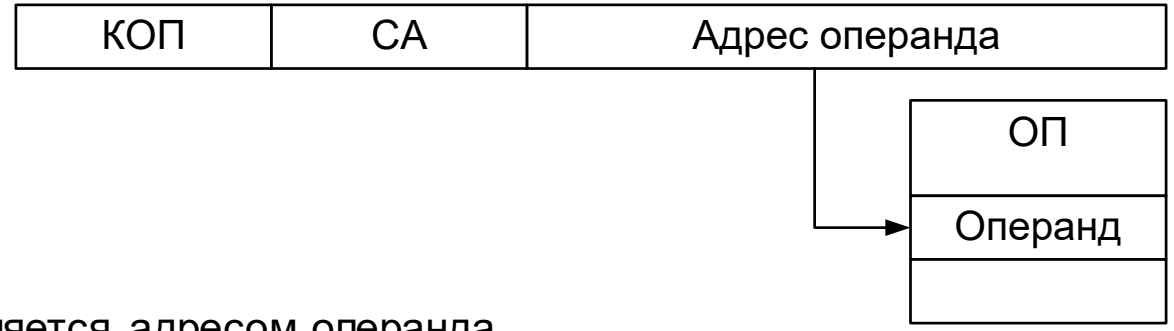
Способы адресации



Непосредственная адресация

Вместо адреса команда содержит непосредственно операнд.
(+) команда выполняется быстро
(-) непосредственный операнд может не войти в команду

Прямая адресация



Адрес в команде является адресом операнда
(+) если операнд находится в памяти, то это самый быстрый способ указать на него
(-) заранее определенный адрес влияет на переносимость программы.
(-) Адрес занимает много места

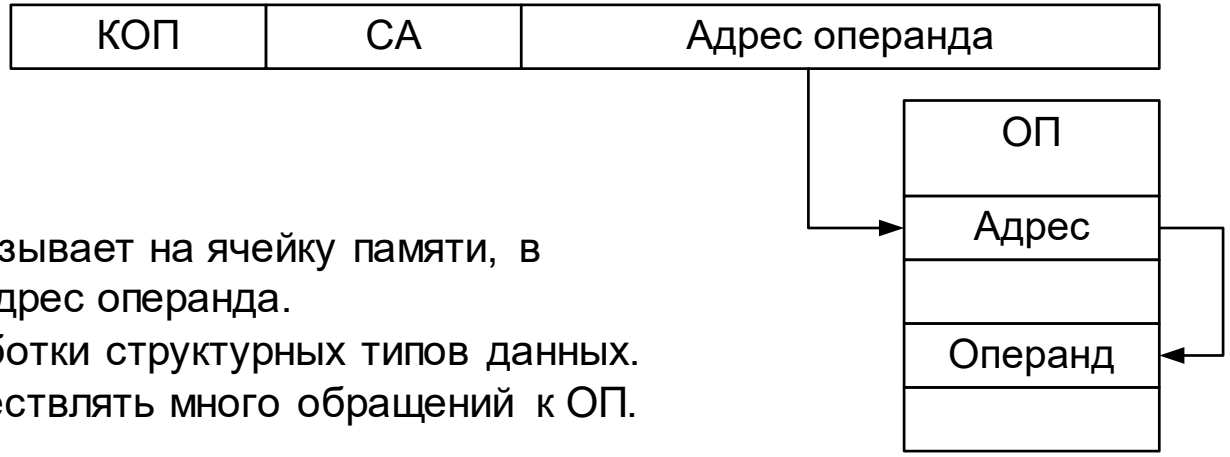
Неявная адресация

Операнд подразумевается (следует из КОП).
(+) Команда занимает мало места
(-) только такие командах нельзя использовать для построение всей системы команд.

Регистровая адресация

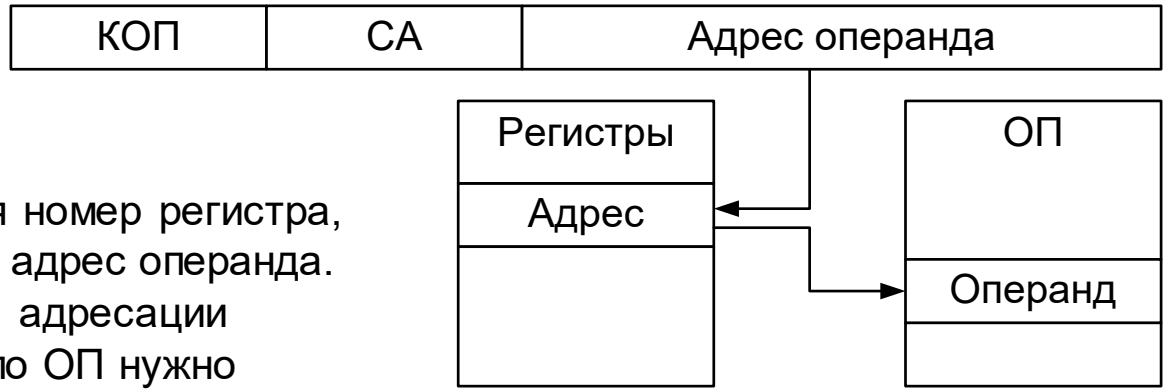
Адрес в команде указывает не на ячейку ОП, а на регистр.
(+) Быстрее прямой адресации
(-) Количество регистров ограничено

Косвенная адресация



Адрес в команде указывает на ячейку памяти, в которой находится адрес операнда.
(+) удобна для обработки структурных типов данных.
(-) приходится осуществлять много обращений к ОП.

Косвенная регистровая адресация

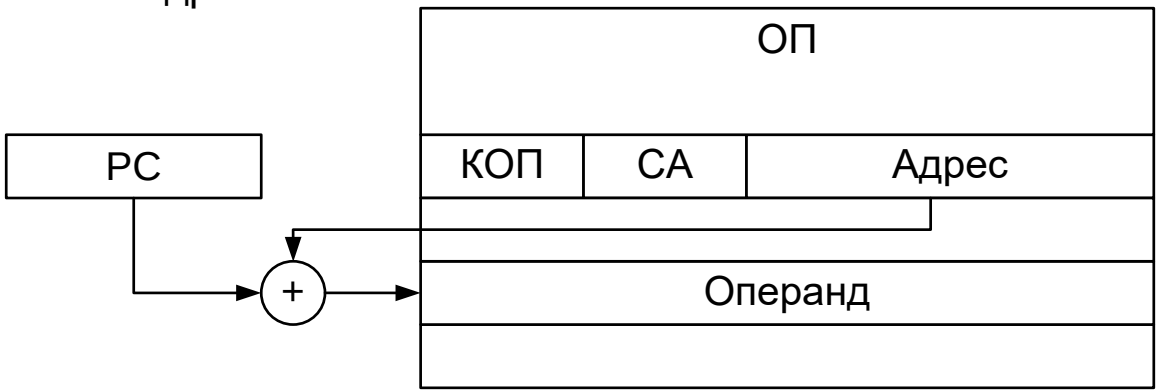


В команде содержится номер регистра, в котором содержится адрес операнда.
(+) быстрее косвенной адресации
(-) для перемещения по ОП нужно менять содержимое регистра

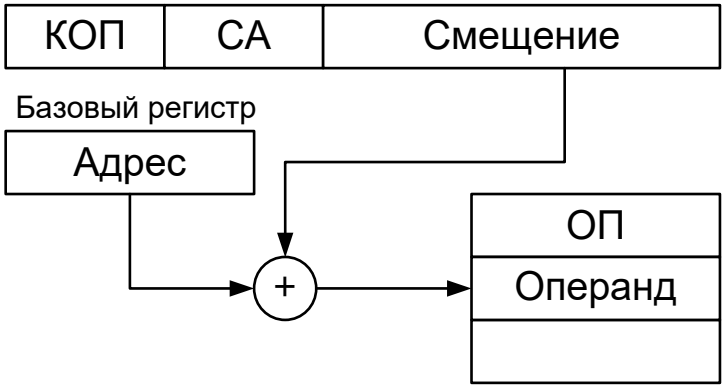
Относительная адресация

Адрес вычисляется относительно счётчика команд

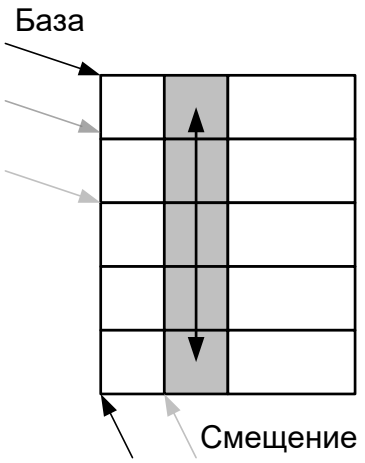
(+) Код переносим, команды занимают мало места
(-) Может понадобиться длинный адрес



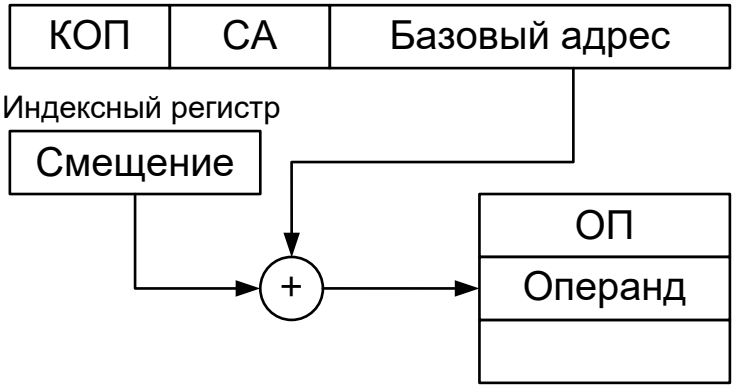
Базовая регистровая адресация

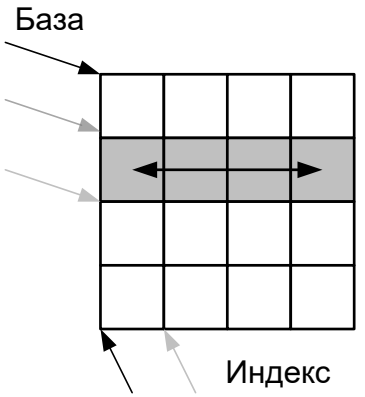


Адрес в команде представляет собой смещение, которое складывается со значением в базовом регистре для получения адреса операнда
(+) Удобна для работы со структурами данных, размещаемых динамически.
(-) Переносимость меньше, чем у относительной адресации



Индексная регистровая адресация





В поле адреса команды содержится базовый адрес, складываемый со значением смещения в индексном регистре.
(+) Удобна для работы со структурами данных, размещаемых динамически.
(-) Переносимость меньше, чем у относительной адресации

Автоинкрементная/автодекрементная адресация

Разновидность регистровой индексной или базовой адресации. До или после выполнения команды значение базового или индексного регистра увеличивается/уменьшается на единицу или масштаб.

(+) Способ адресации удобен для команд обработки строк.

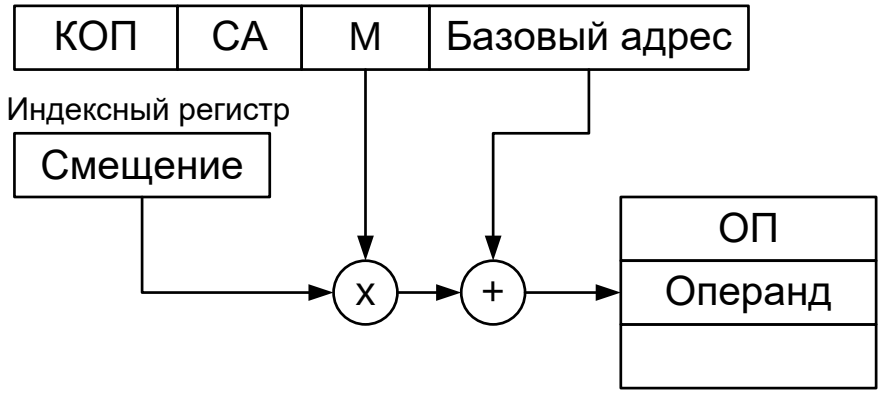
(-) Автоматическое изменение часто требуется выполнять на величину, большую единицы.

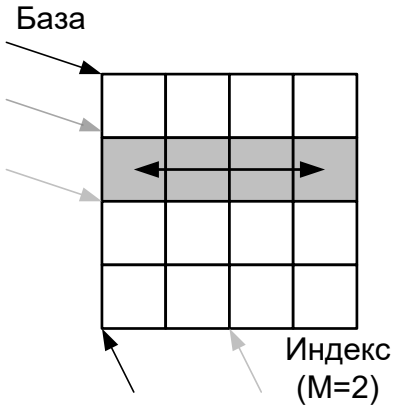
Индексная адресация с масштабированием

Индексный регистр умножается на масштаб M и суммируется с базовым адресом из команды.

(+) Удобен для модификации адреса на величину M.

(-) Вычисление адреса замедляется, т.к. требуется выполнять умножение.





Базовая индексная адресация с масштабированием

Адрес определяется по формуле Адрес=Индекс*Масштаб+База+Смещение.

(+) Базовая индексная адресация с масштабированием часто используется при обращении к системным таблицам, находящимся в ОП (таблица дескрипторов, таблицы страниц, таблица векторов прерываний и т.д.)
(-) Ограниченное на величину M (M=1,2,4,8).