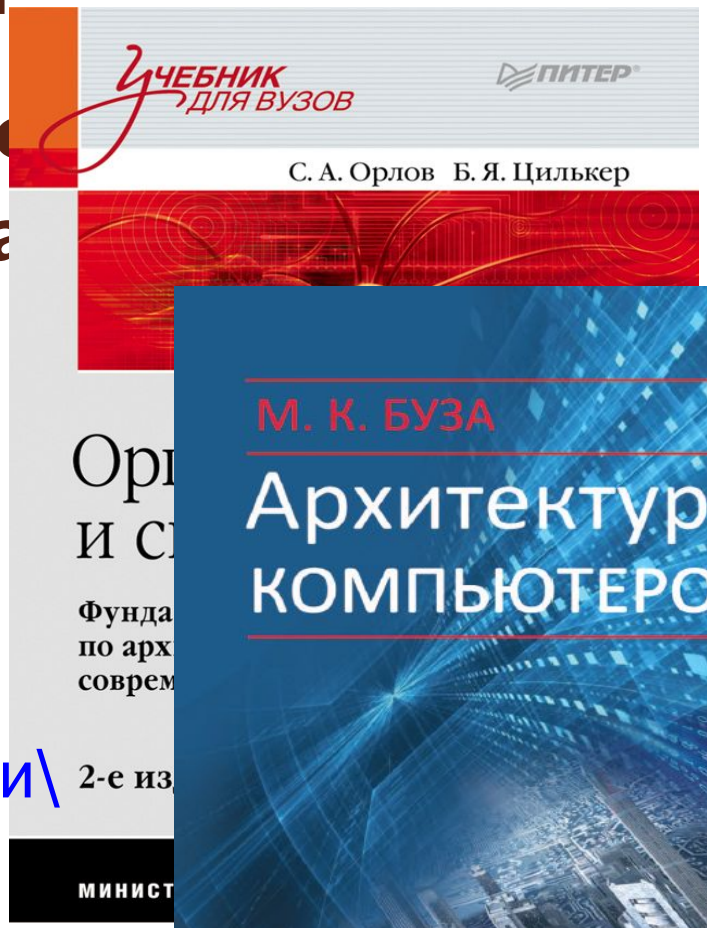
Архитектура ЭВМ и ВС

- **Лекции : 32 + 4 час**
  Практика: 16 +2 ча
  СРС: 18 час
  Зачет

Литература:



H:\Учебные материалы-\Книги\ Архитектура ЭВМ

1. Орлов С.А., Цилькер Б.Я. Организация ЭВМ и систем.

2. Буза М.К. Архитектура компьютеров

Лекции:

- Многоуровневая организация компьютера
- Функциональная организация компьютера
- Архитектура системы команд
- Архитектура памяти
- Операционные устройства вычислительных машин
- Системы ввода/вывода
- Основные направления развития архитектуры вычислительных машин
- Многоядерные процессоры
- Законы параллельных вычислений
- Топология вычислительных систем
- Вычислительные системы класса SIMD
- Вычислительные системы класса MIMD

Практикум:

- Арифметические основы вычислительных машин
- CISK и RISK-системы команд

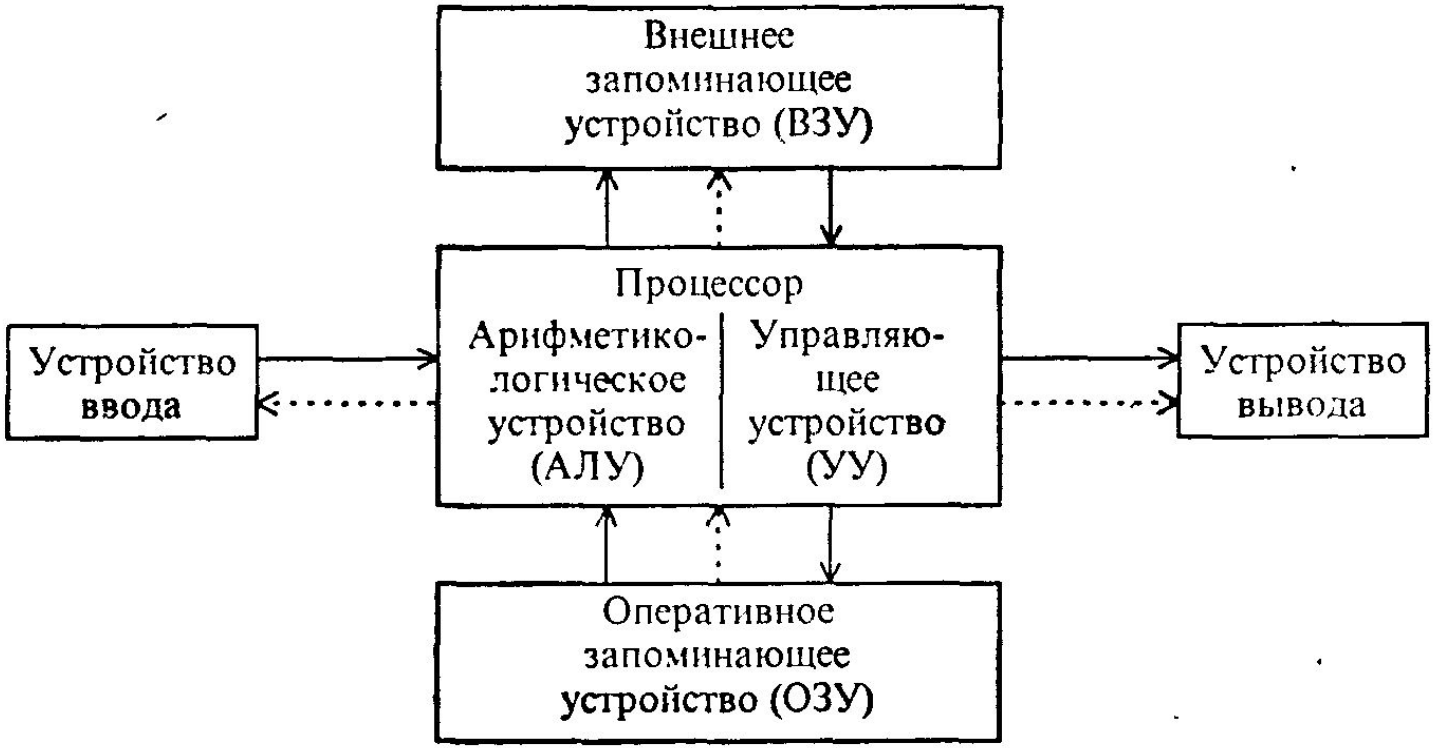
Базовые принципы работы компьютера

"Архитектура - это наиболее общие принципы построения ЭВМ, реализующие программное управление работой и взаимодействием основных ее функциональных узлов"

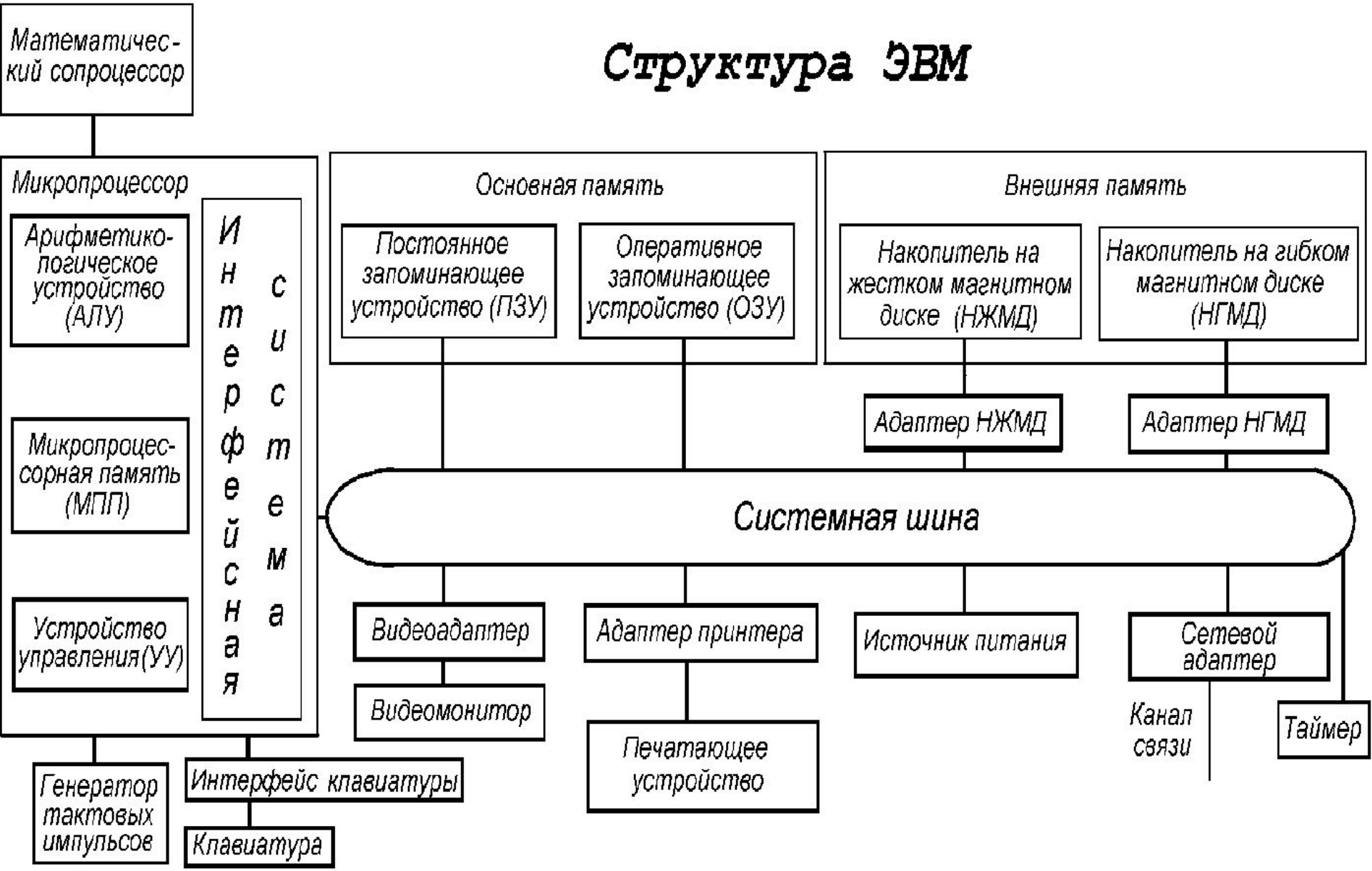
Принципы Фон-Неймана

Классические принципы построения архитектуры ЭВМ были предложены в работе Дж. фон Неймана, Г.Голдстейга и А. Беркса в 1946 году и известны как " принципы фон Неймана".

- **Принцип двоичного кодирования**. Для представления данных и команд используется двоичная система счисления.
- **Принцип программного управления.** Программа состоит из набора команд, которые выполняются процессором друг за другом в определённой последовательности.
- **Принцип однородности памяти.** Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления, чаще всего – двоичной). Над командами можно выполнять такие же действия, как и над данными.
- **Принцип адресуемости памяти.** Структурно основная память состоит из пронумерованных ячеек, процессору в произвольный момент времени доступна любая ячейка.
- **Принцип последовательного программного управления.** Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.
- **Принцип условного перехода.** Команды из программы не всегда выполняются одна за другой. Возможно присутствие в программе команд условного перехода, которые изменяют последовательность выполнения команд в зависимости от значений данных. (Сам принцип был сформулирован задолго до фон Неймана Адой Лавлейс и Чарльзом Бэббиджем, однако он логически включен в фон-неймановский набор как дополняющий предыдущий принцип).

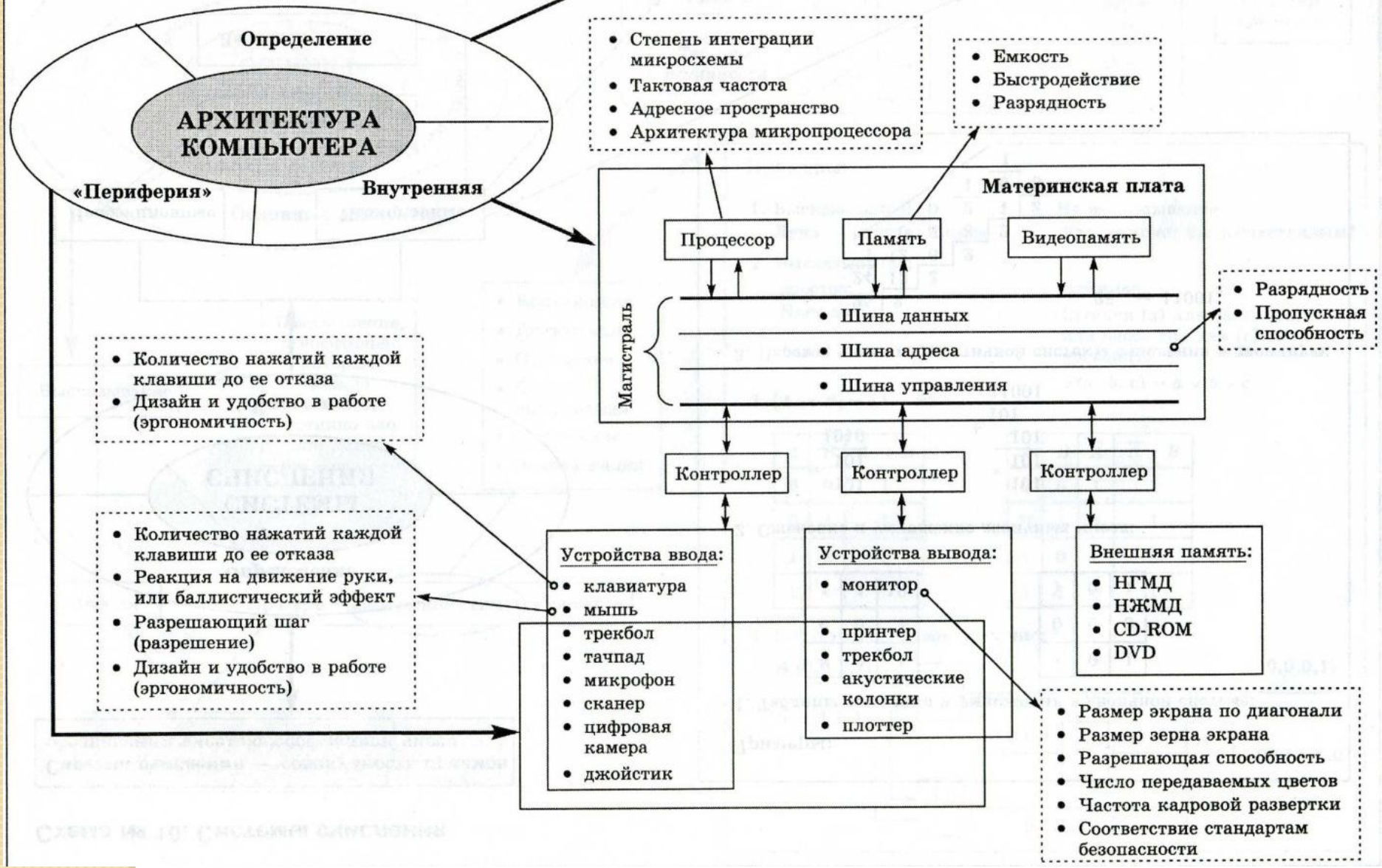

Внешнее запоминающее устройство (ВЗУ)
Процессор
Арифметико-логическое устройство (АЛУ)
Управляю-щее устройство (УУ)
Устройство ввода
Устройство вывода
Оперативное запоминающее устройство (ОЗУ)

Структура ЭВМ



Архитектура компьютера

Архитектура компьютера — это описание наиболее общих принципов построения компьютера, реализующих программное управление его работой и взаимодействие его основных узлов



Принстонская архитектура (фон Неймана)

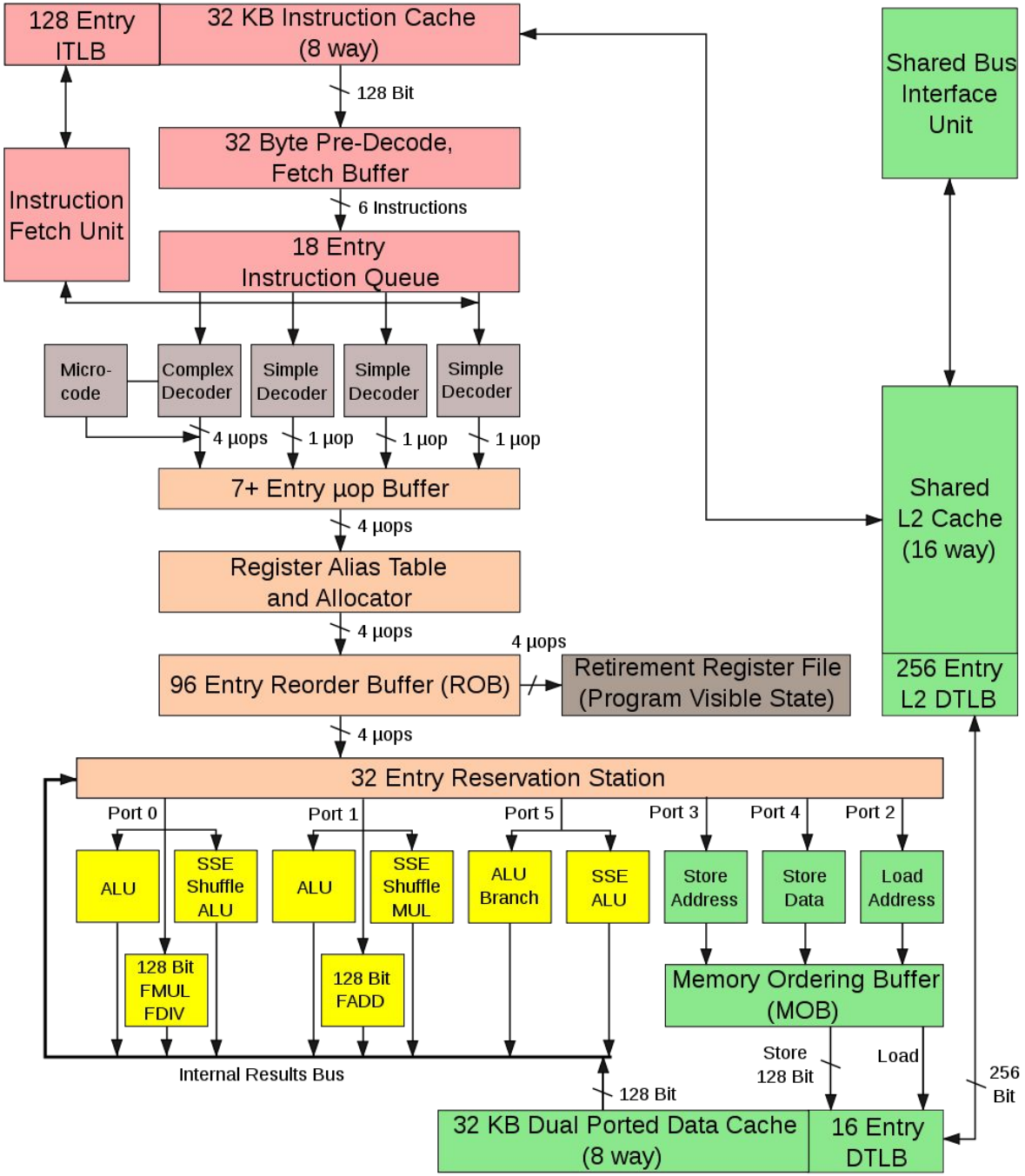
Достоинства:

1) Наличие общей памяти позволяет операти
объем для хранени отдельных массивов ко
стека в зависимост
обеспечивается воз
имеющегося объем
применения.
2) Использование о
значительно упрощ
функционирования

Универсальные ЭВМ предн
решения широкого кру
Характер
оп

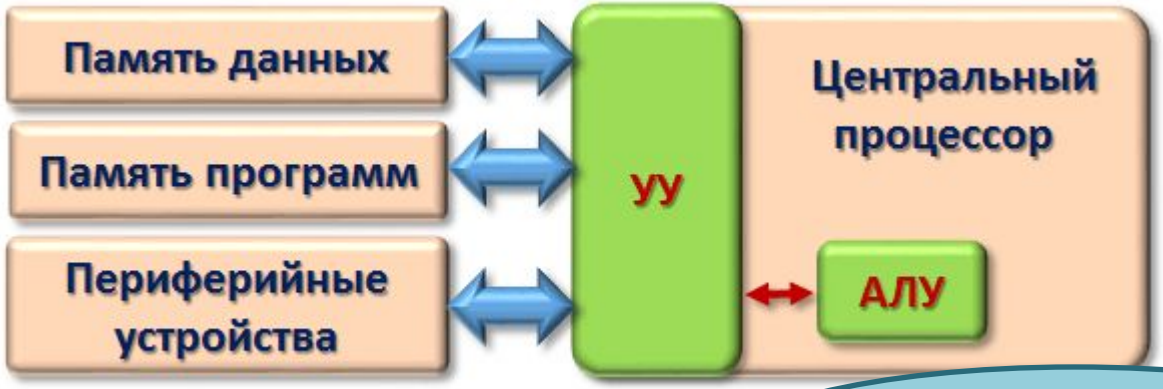
Недостатки:

1) Общая шина ста рое ограничивает производительн
2) Благоприятная сред



Intel Core 2 Architecture

Гарвардская архитектура

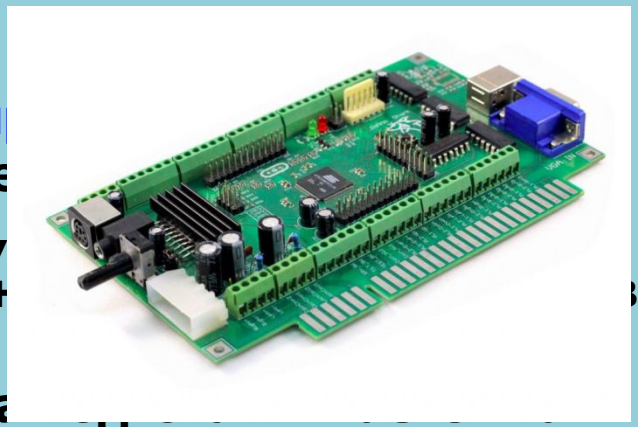


Недостатки:

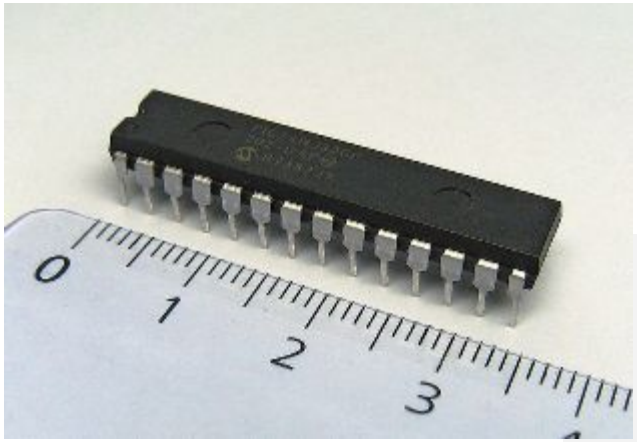
1. Б… число шин, сложн… …я
2. С… …изи… …х, н… …влe… …в с… …ов, дат… …ван… …др.

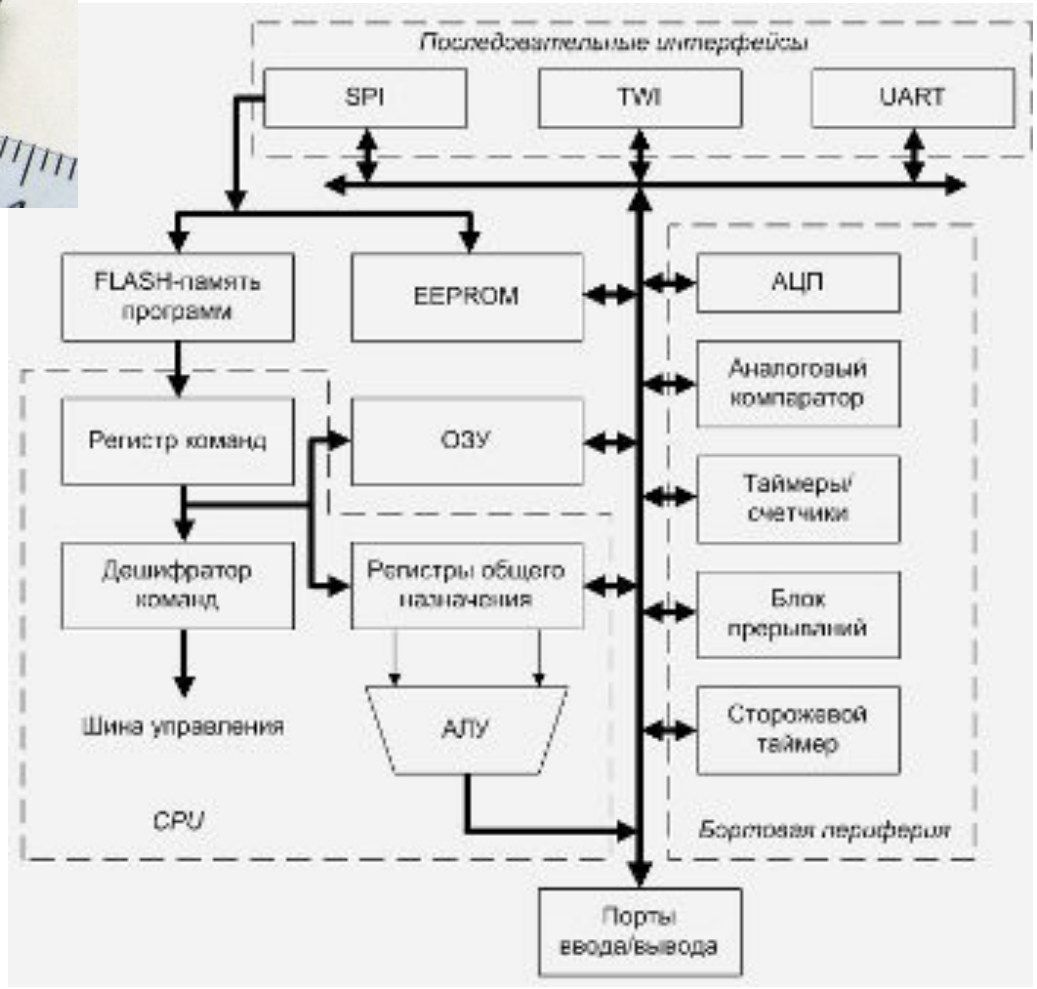
Досто…

1. В… …ера…
2. П… …осле загрузки в память пр… …ащелкнута» (защищена от чтения и записи).

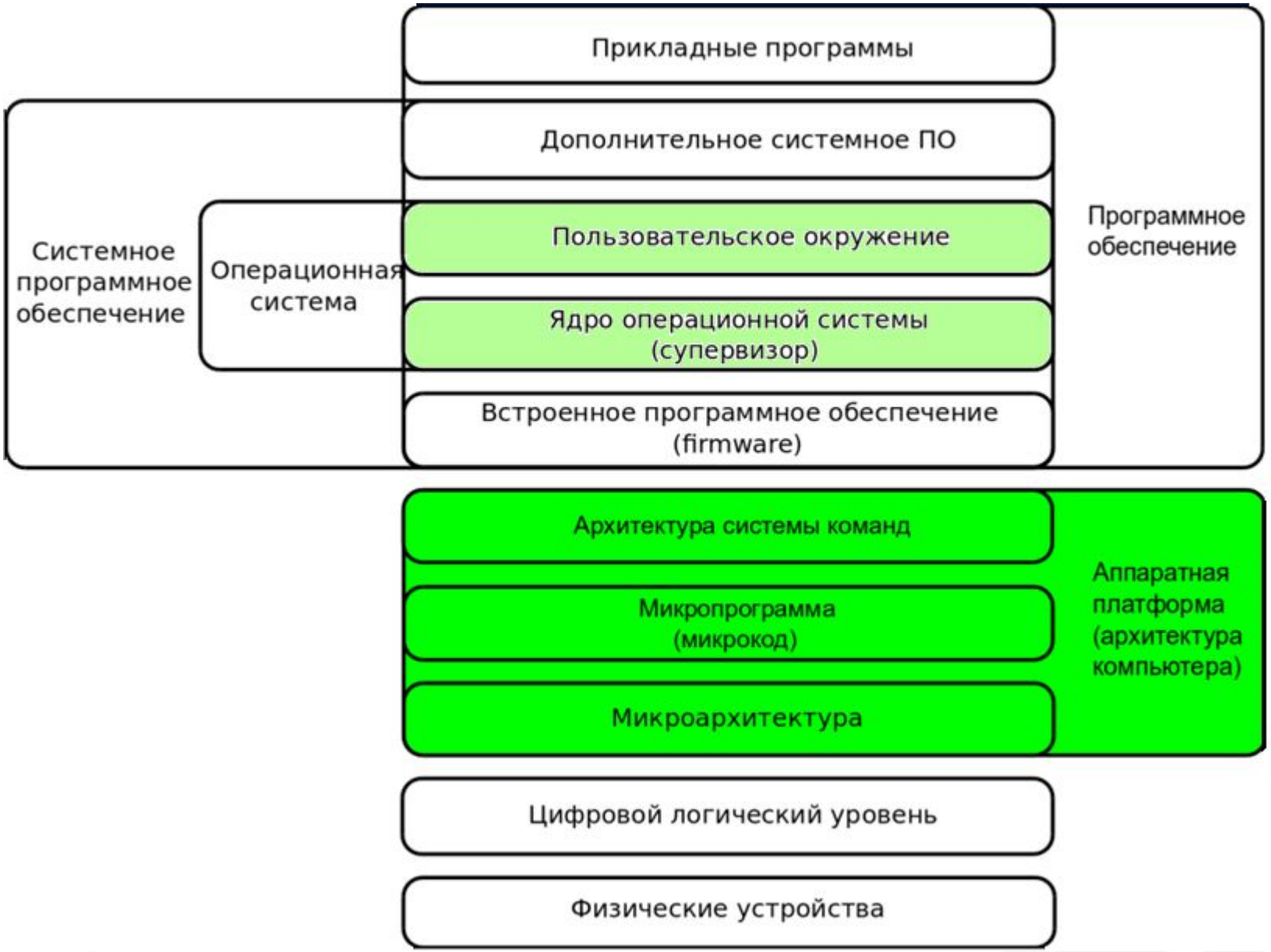


Микроконтроллер Atmel AVR ATmega8

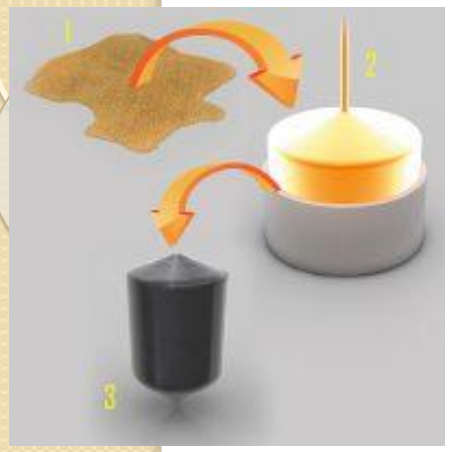


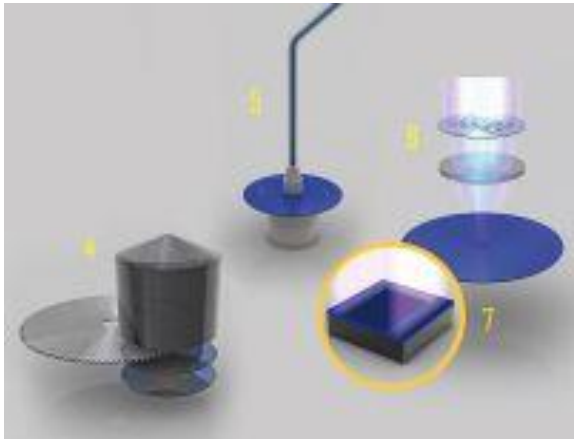


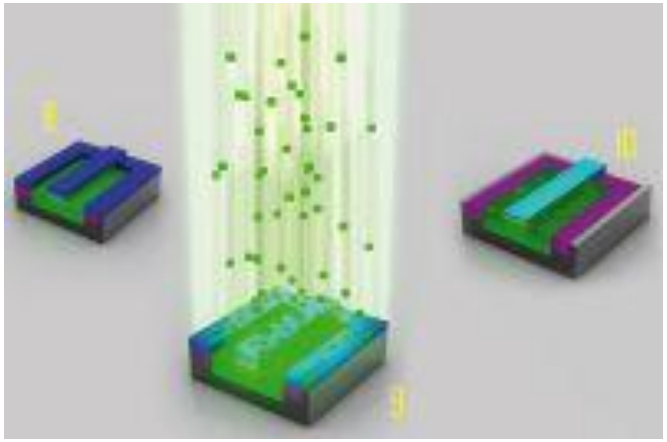
Многоуровневая организация компьютера

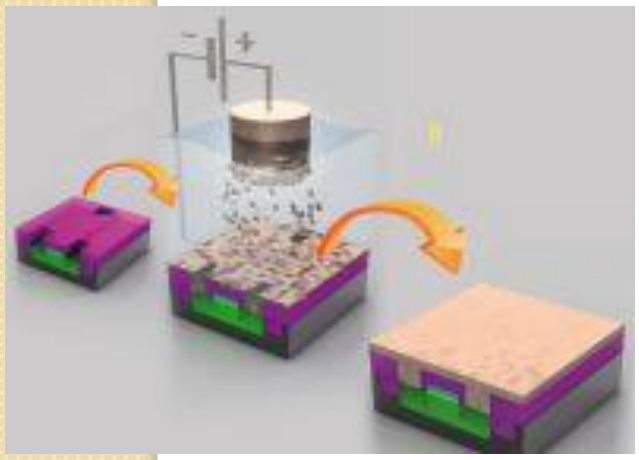


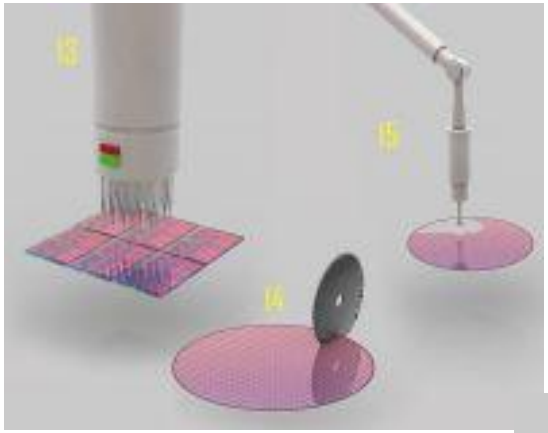
Производство процессоров

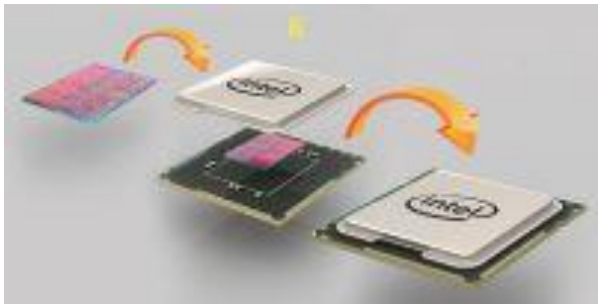








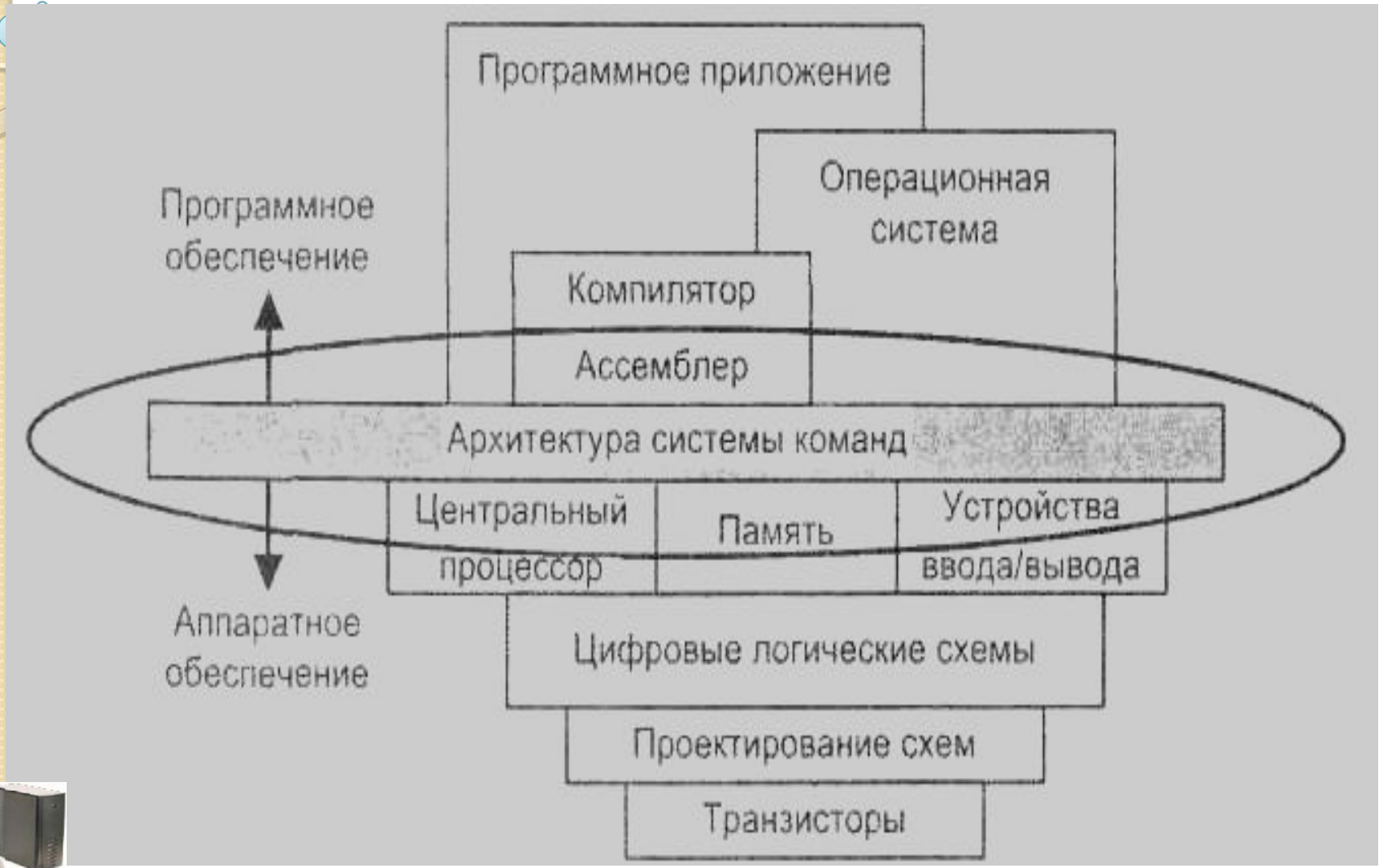




Архитектура системы команд компьютера

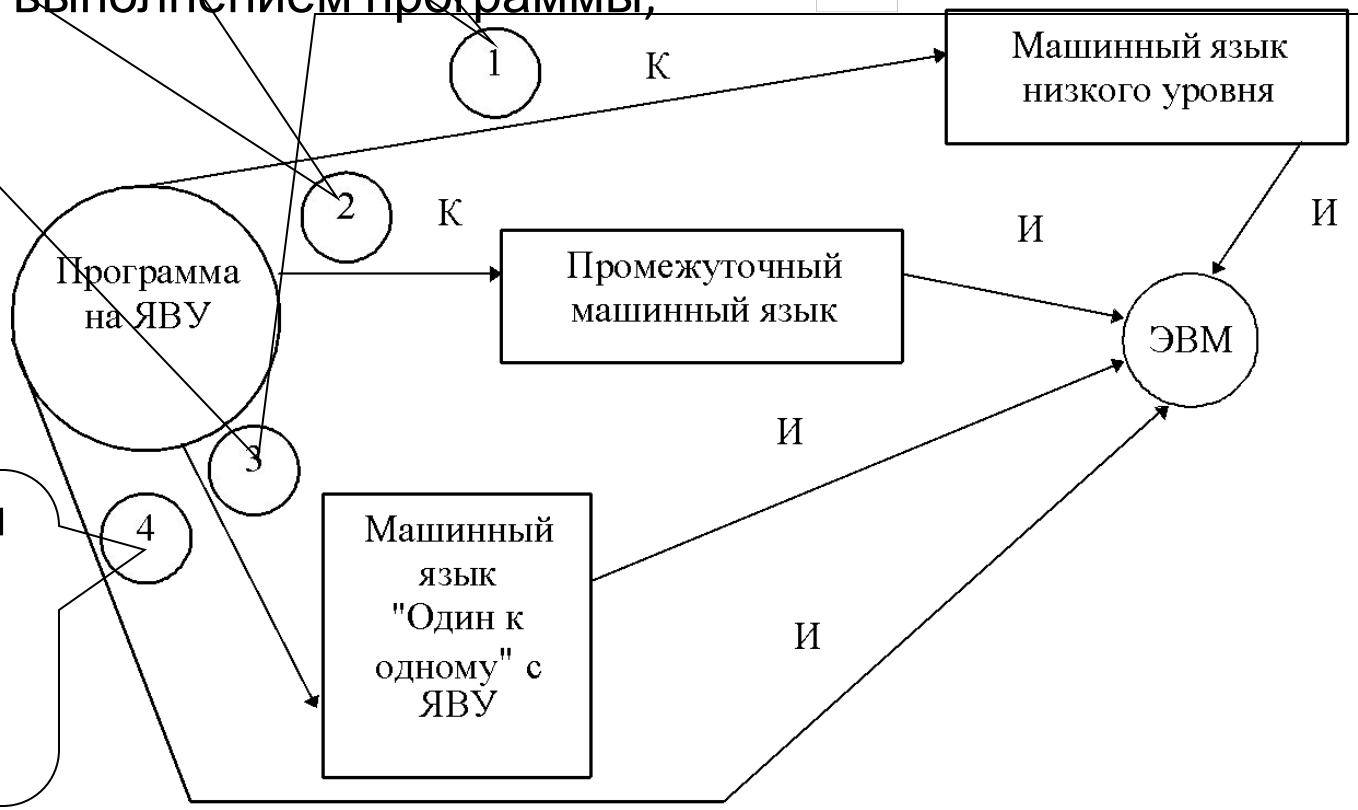
Архитектура системы команд

Архитектура системы команд как интерфейс между программным и аппаратным обеспечением



языке

Здесь ЯВУ можно рассматривать как язык ассемблера, т.е. имеется взаимно однозначное соответствие между типами операторов и знаков операций ЯВУ с командами машинного языка. Здесь идет ассемблирование, а не компилирование, во время которого удаляются комментарии и пробелы исходной программе, преобразуются разделители, ключевые слова и знаки операций в машинные коды, имена – в машинные адреса; таким образом, многих привычных функций компилятора нет, а остальная привязка программы к ЭВМ происходит перед выполнением программы;



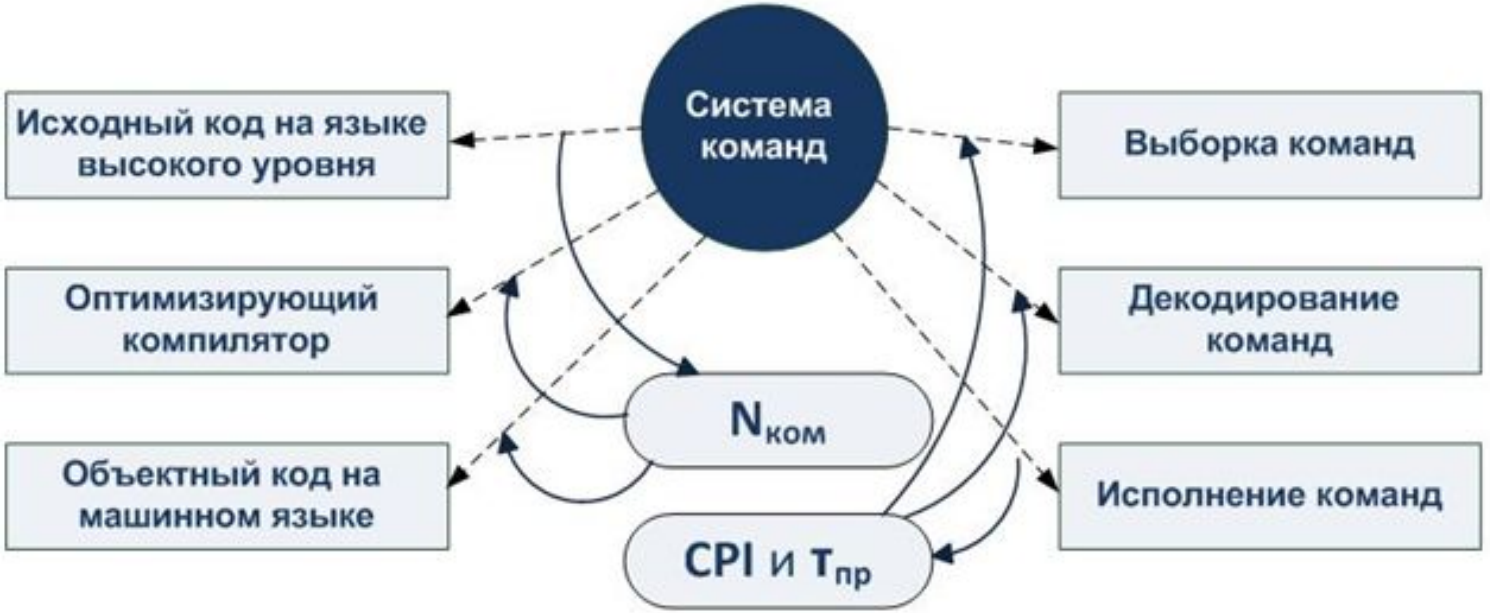
Здесь машинный язык является ЯВУ и идет процесс интерпретации программы на компьютере

Требования ЯВУ к архитектуре ЭВМ :

- память состоит из набора дискретных именуемых переменных.
- ЯВУ наряду с линейными данными оперируют и с многомерными: массивами, структурами, списками;
- в ЯВУ четко разграничены операции и данные;
- данные определяют и операции над ними.

Взаимосвязь между системой команд и эффективностью вычислений

$$T_{выч} = N_{ком} \times CPI \times \tau_{пр}$$



$T_{выч}$ - время выполнения программы;

$N_{ком}$ - число команд в программе;

CPI- среднее количество тактов процессора на одну команду;

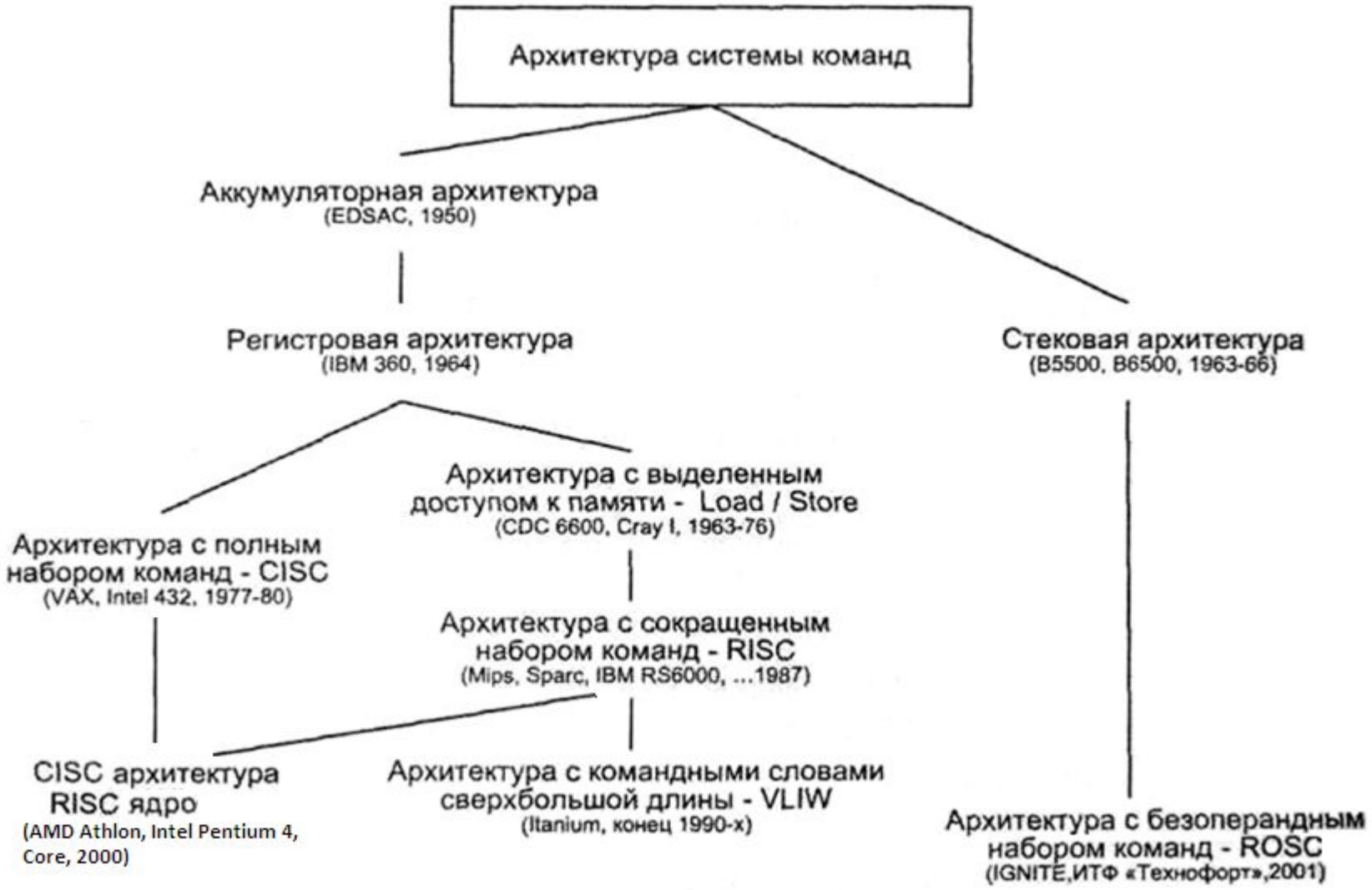
- *Система команд* вычислительной машины - полный перечень команд, которые способна выполнять данная ВМ.
- *Архитектура системы команд* (АСК) - те средства вычислительной машины, которые видны и доступны программисту.

Архитектура системы команд (англ. *instruction set architecture, ISA*) — часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора.

На этом уровне определяются реализованные в **микропроцессоре** конкретного типа:

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

Классификация архитектур системы команд



Хронология развития архитектур системы команд

Классификация архитектур системы команд

- По составу и сложности команд (CISC, RISC, VLIW, ROSC)
- По типу выполняемых операций (общего назначения, специализированные, дополненной системой команд)
- По месту хранения операндов (тип адресуемой памяти)

Классификация по составу и сложности команд

- архитектура с полным набором команд: CISC (Complex Instruction Set Computer);
- архитектура с сокращенным набором команд: RISC (Reduced Instruction Set Computer);
- архитектура с командными словами сверхбольшой длины: VLIW (Very Long Instruction Word):
- безоперандная (стековая) архитектура: ROSC (Removed Operand Set Computer)

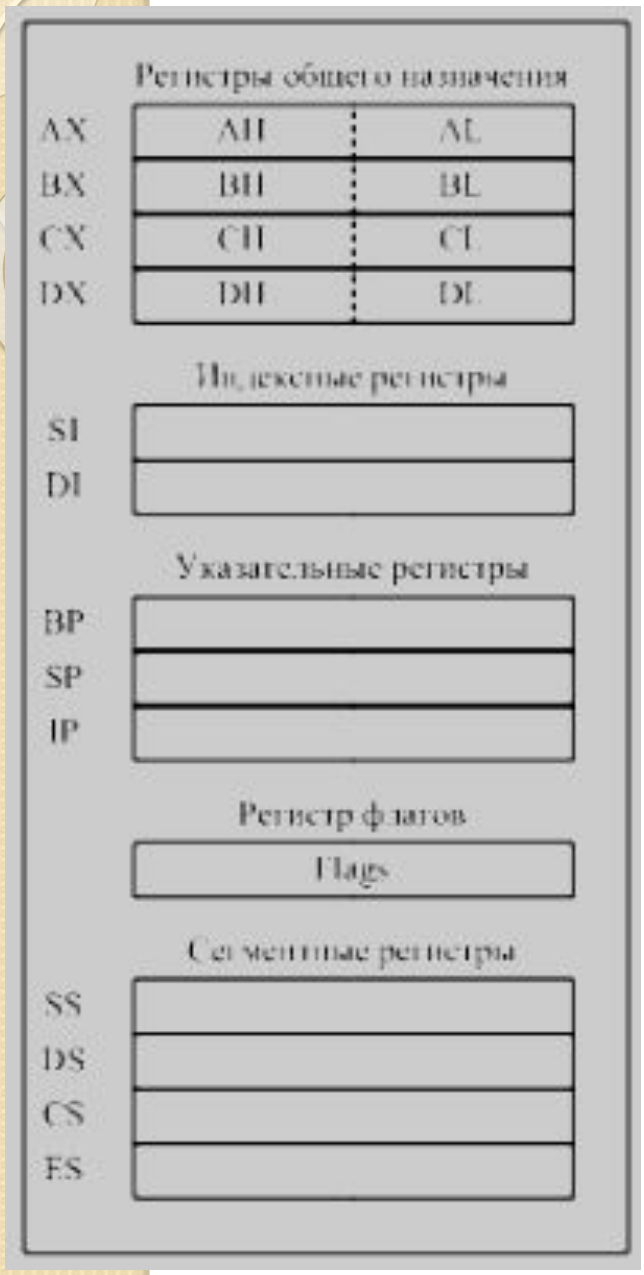
Программная модель Intel 8086

Процессор интересует нас, прежде всего, как набор регистров

Регистр – быстродействующее запоминающее устройство, реализованное на электронных компонентах.

Все регистры имеют размер слова (16 разрядов), за каждым из них закреплено определенное имя. По назначению и способу использования регистры можно разбить на следующие группы:

- регистры общего назначения (AX, BX, CX, DX, SI, DI, BP, SP);
- сегментные регистры (CS, DS, SS, ES);
- указатель команд (IP);
- регистр флагов (Flags).



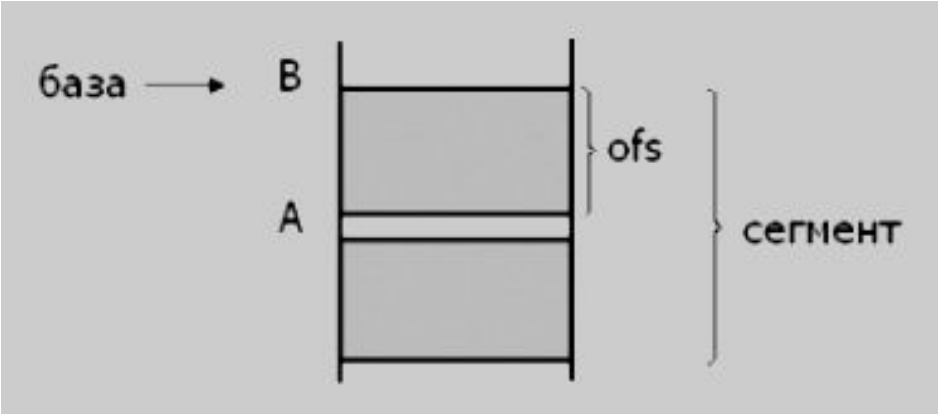
AX accumulator, аккумулятор;
BX base, база;
CX counter, счетчик;
DX data, данные;
(буква X - от слова eXtended, расширенный: в процессоре 8080 были байтовые регистры A, B, C и D, но затем их расширили до размера слова)

SI source index, индекс источника;
DI destination index, индекс приемника;
BP base pointer, указатель базы;
SP stack pointer, указатель стека;

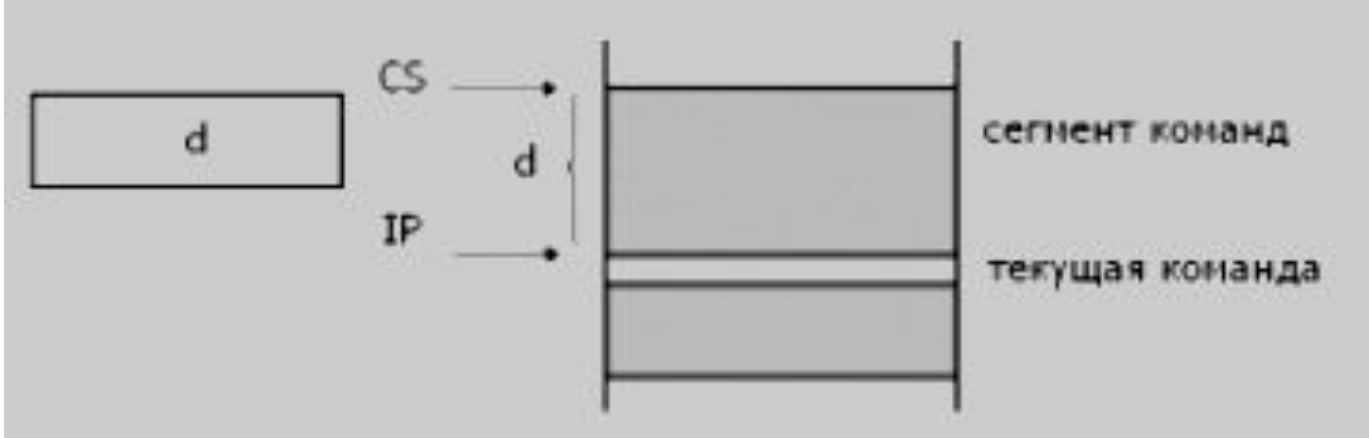
IP instruction pointer, указатель команд;

SS stack segment, сегмент стека;
DS data segment, сегмент данных;
CS code segment, сегмент команд;
ES extra segment, дополнительный сегмент.

Сегментные регистры



Указатель команд



Регистр флагов

Флаг - это бит, принимающий значение 1 ("флаг установлен") или значение 0 ("флаг сброшен"). В i8086 используется 9 флагов, собранных в один 16-разрядный регистр, называемый регистром флагов (Flags).

Флаги условий:

CF (carry flag) - флаг переноса.
OF (overflow flag) - флаг переполнения
ZF (zero flag) - флаг нуля
SF (sign flag) - флаг знака
PF (parity flag) - флаг четности
AF (auxiliary carry flag) - флаг дополнительного переноса

Флаги состояний:

DF (direction flag) - флаг направления.
IF (interrupt flag) - флаг прерываний
TF (trace flag) - флаг трассировки

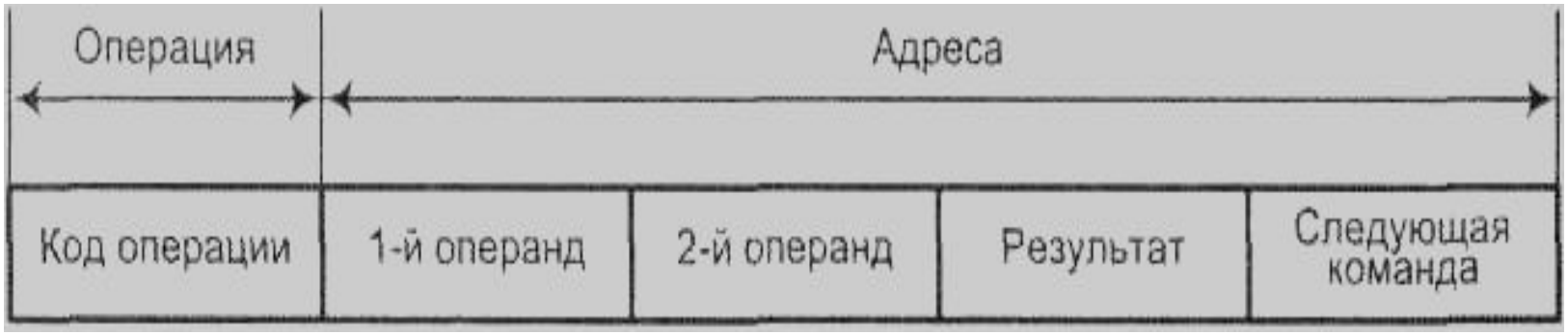
Форматы команд

- Длина команды
- Разрядность полей команды
- Количество адресов в команде
- Способы адресации операндов
- Способы адресации в командах управления потоком команд
- Система операций

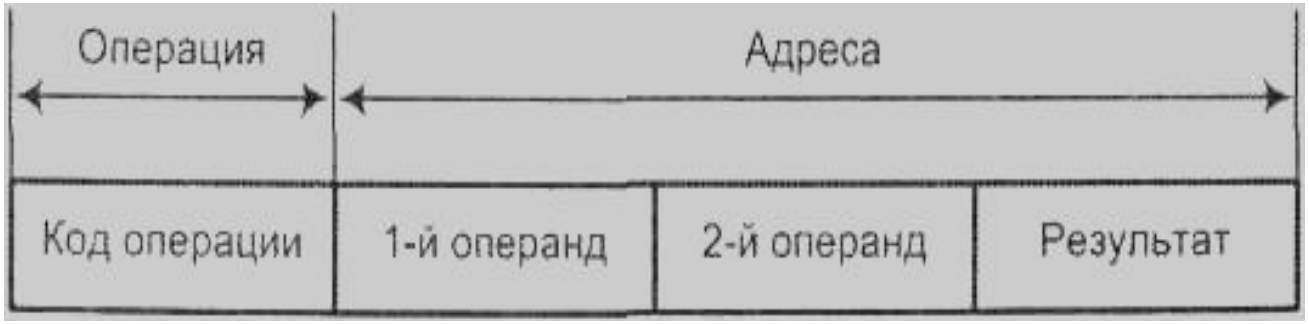
Форматы команд

- Количество адресов в команде

Четырехадресный формат команды



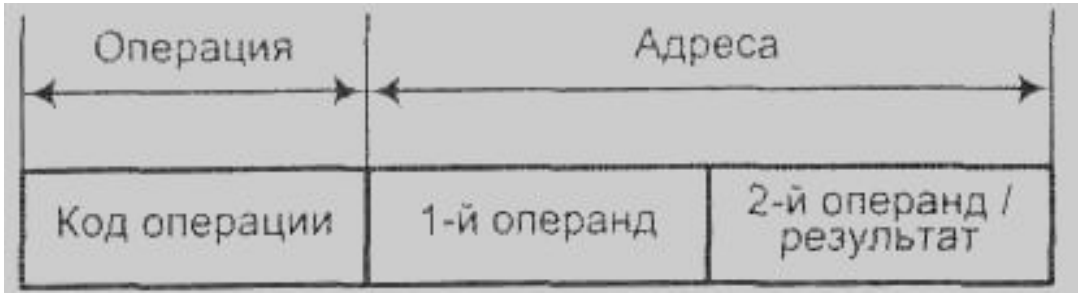
Трехадресный формат команды



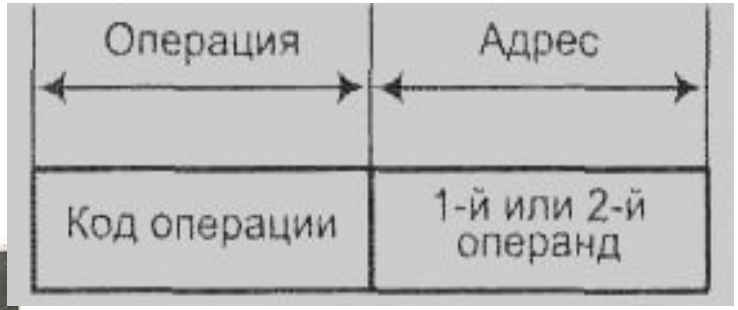
Форматы команд

- Количество адресов в команде

Двухадресный формат команды



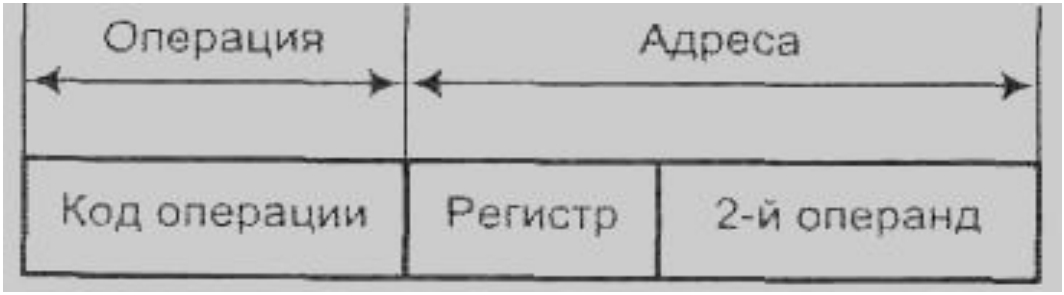
Одноадресный формат команды



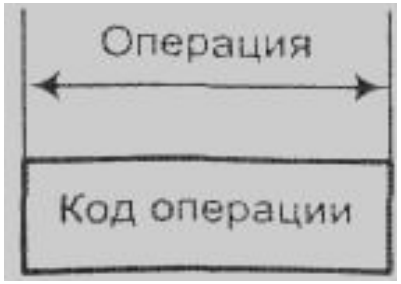
Форматы команд

- Количество адресов в команде

Полутораадресный формат команды



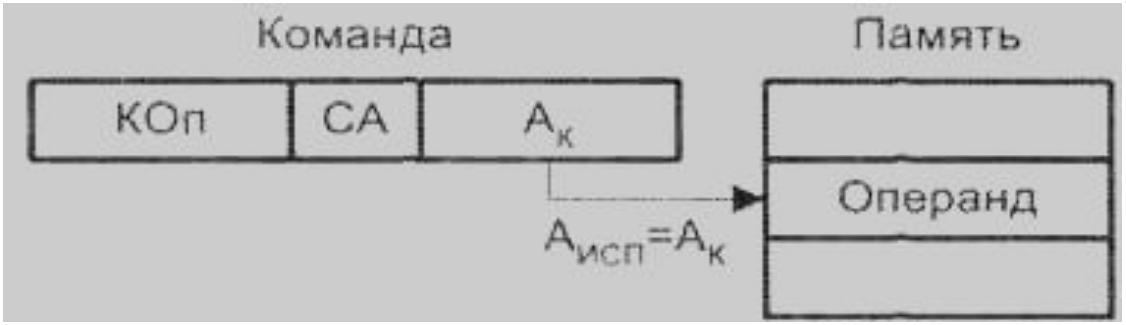
Нульадресный формат команды



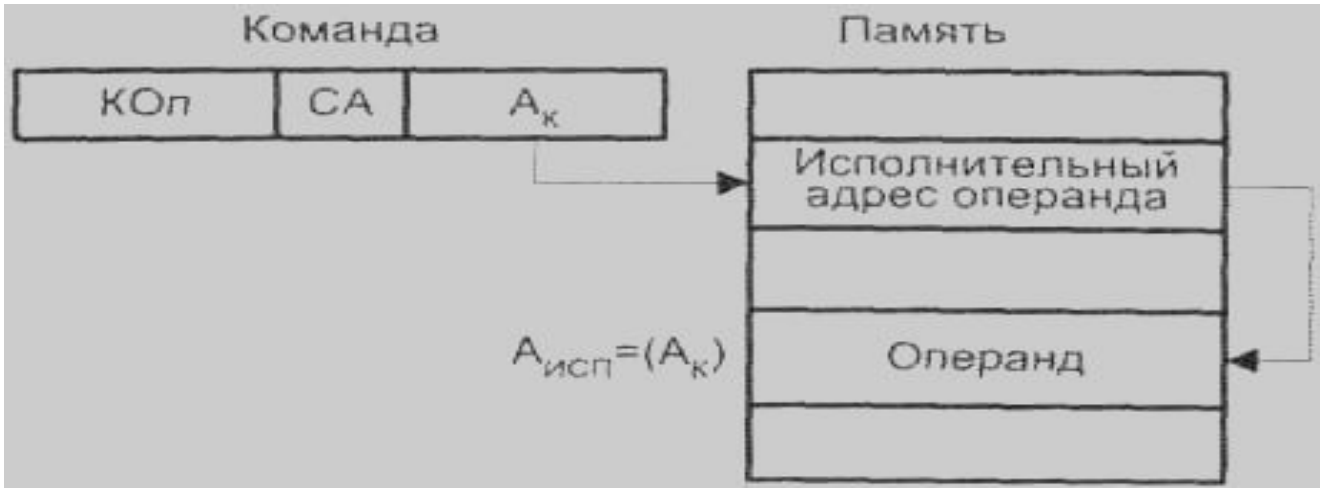
Форматы команд

Способы адресации операндов

Прямая адресация



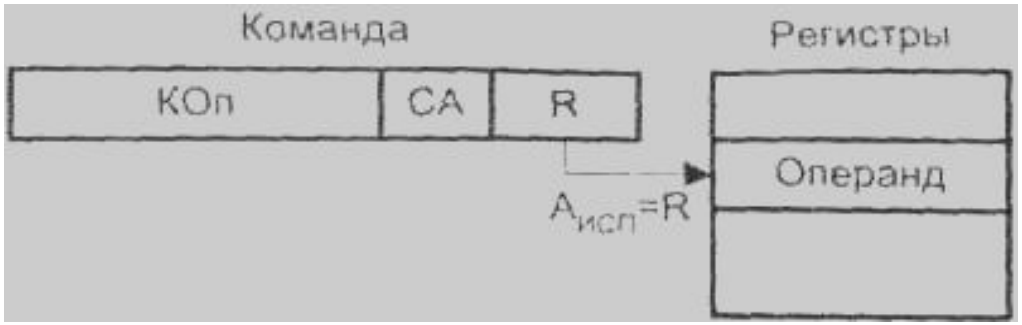
Косвенная адресация



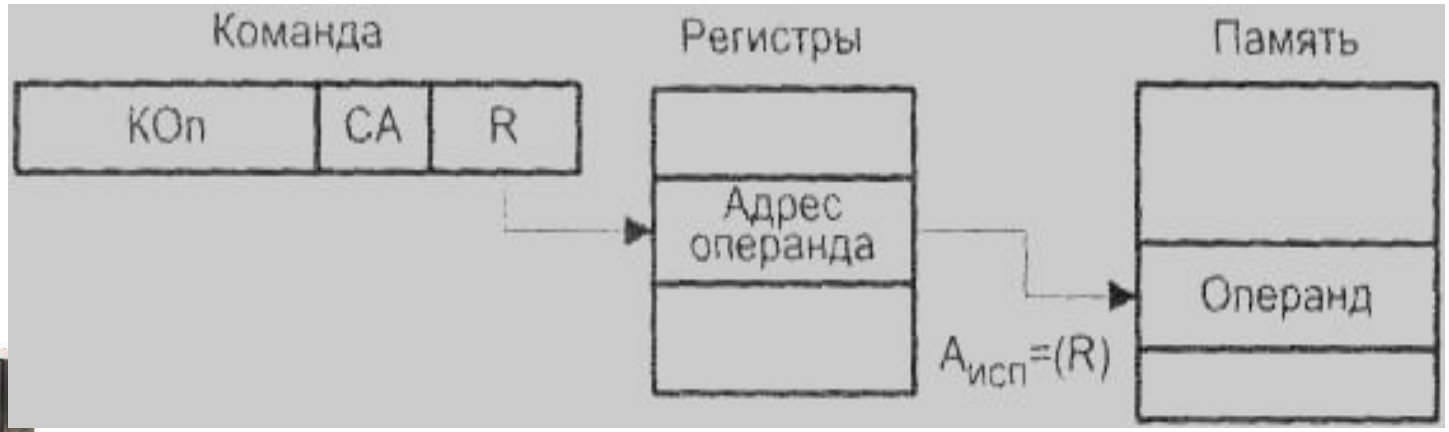
Форматы команд

Способы адресации операндов

Регистровая адресация



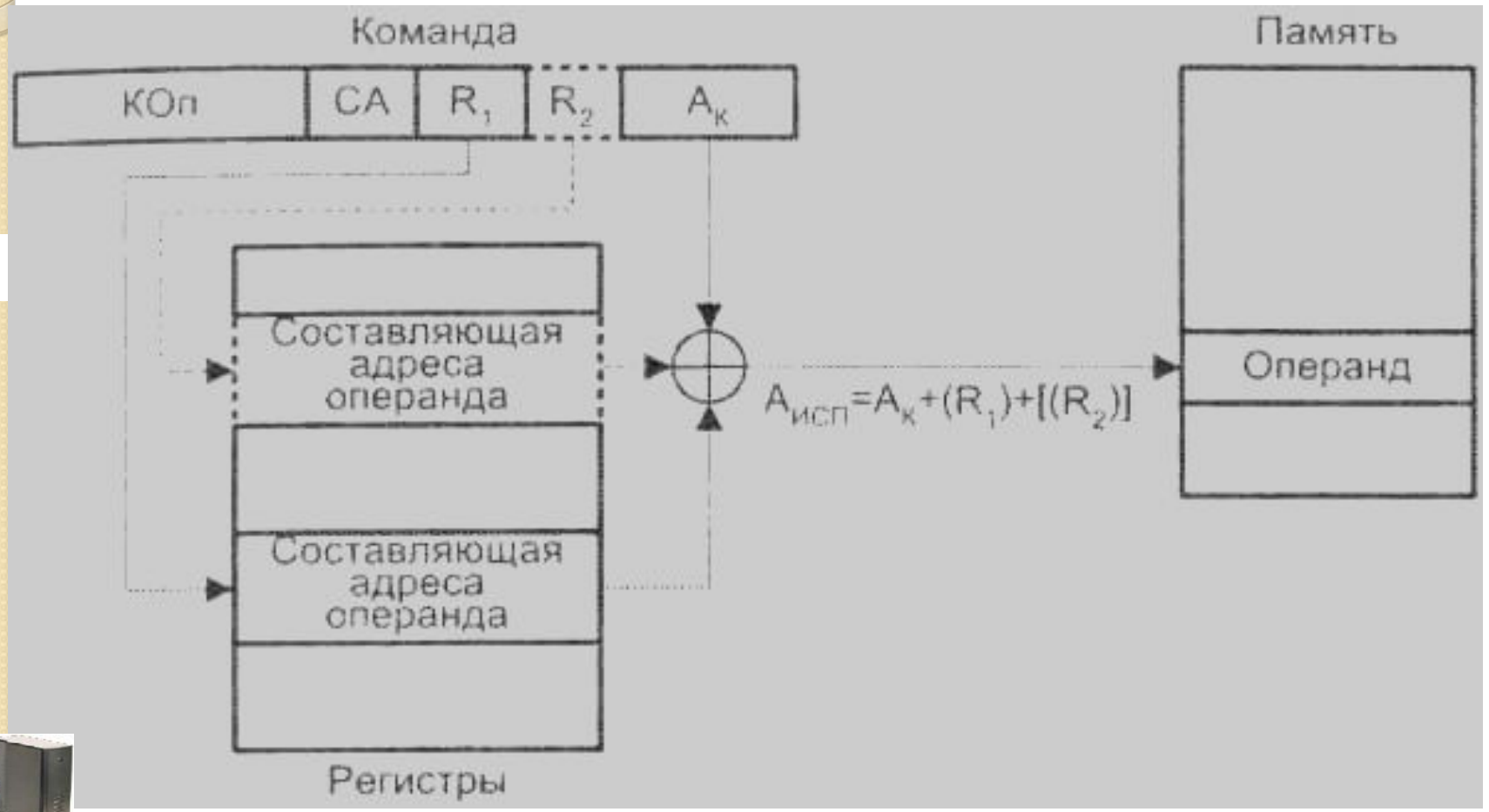
Косвенная регистровая адресация



Форматы команд

- **Способы адресации операндов**

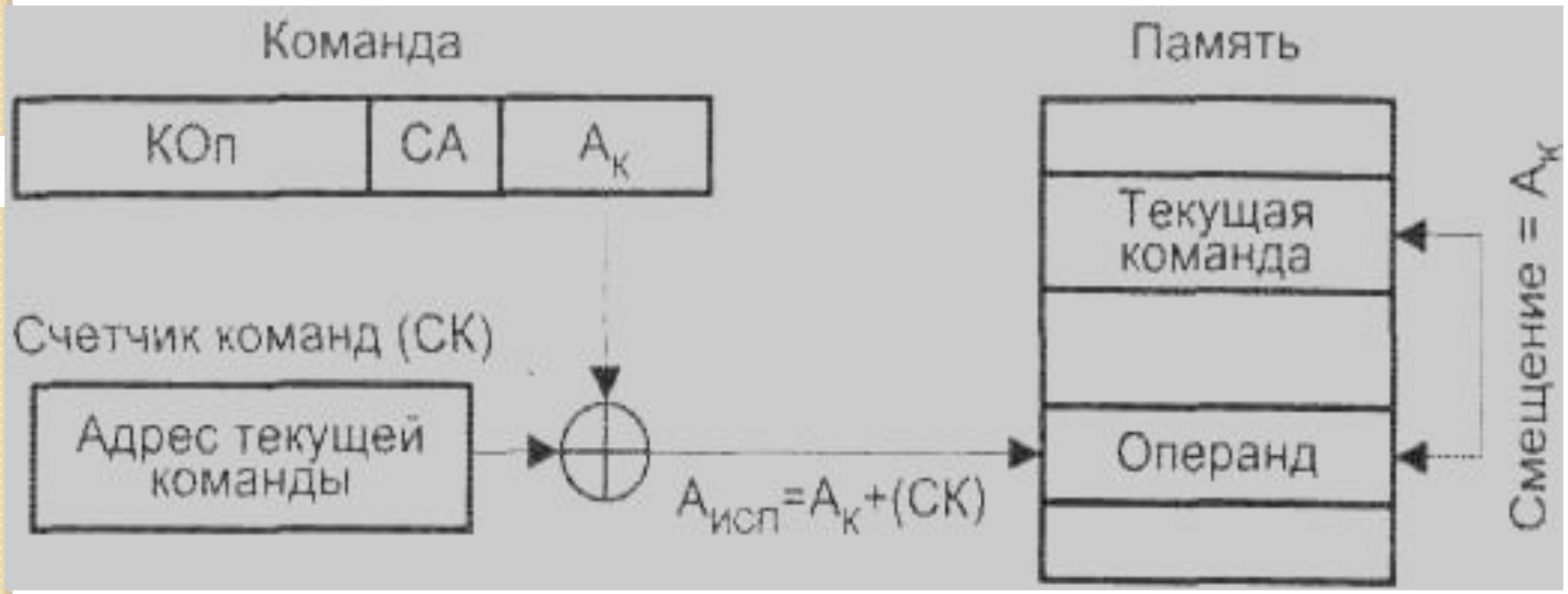
Адресация со смещением



Форматы команд

- **Способы адресации операндов**

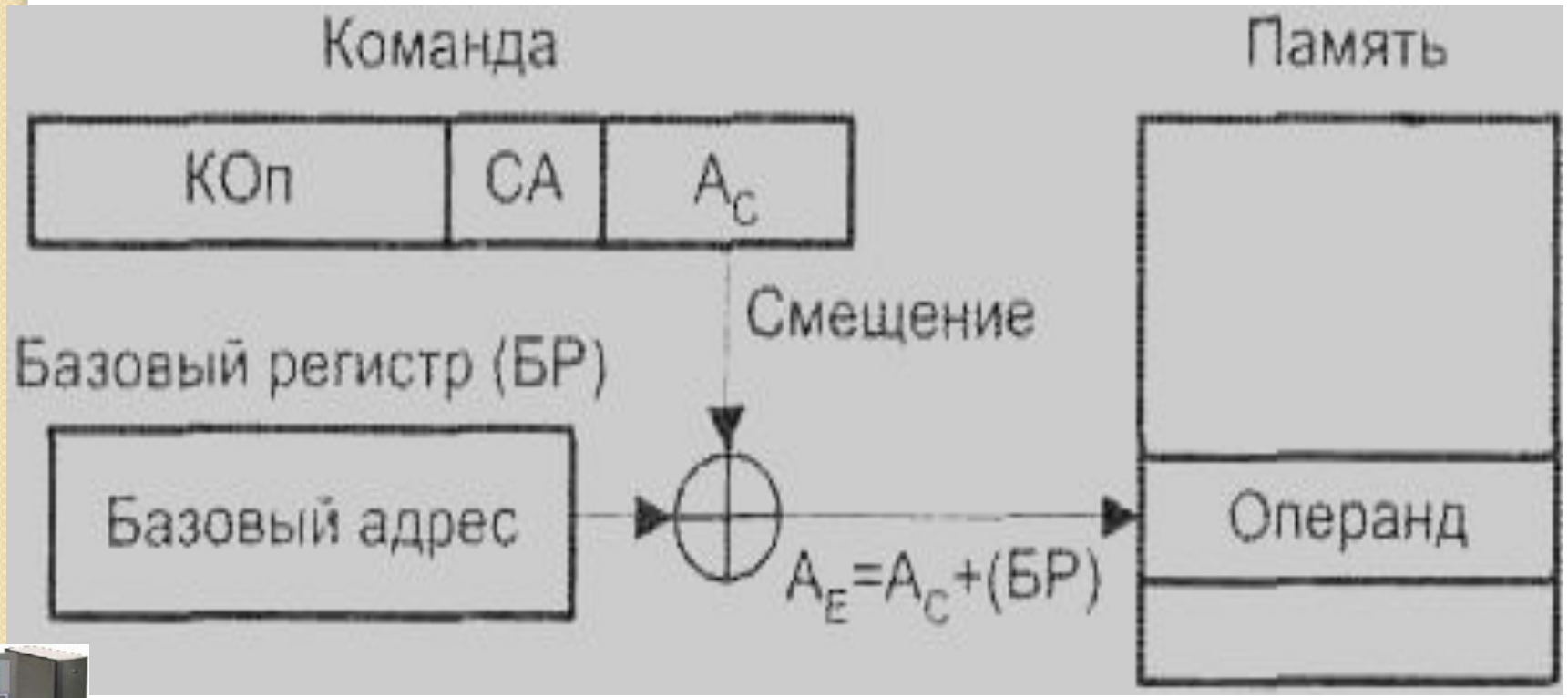
Относительная адресация



Форматы команд

- **Способы адресации операндов**

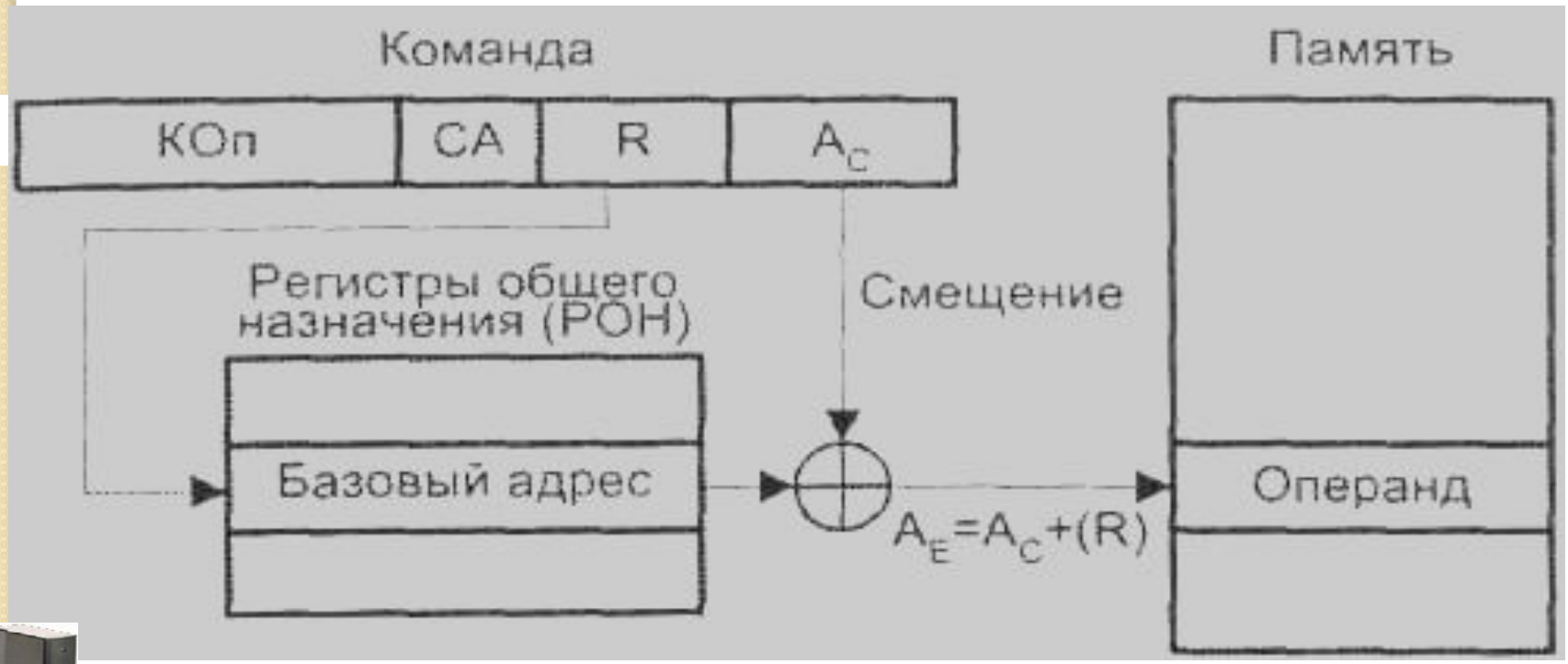
Базовая регистровая адресация с базовым регистром



Форматы команд

- **Способы адресации операндов**

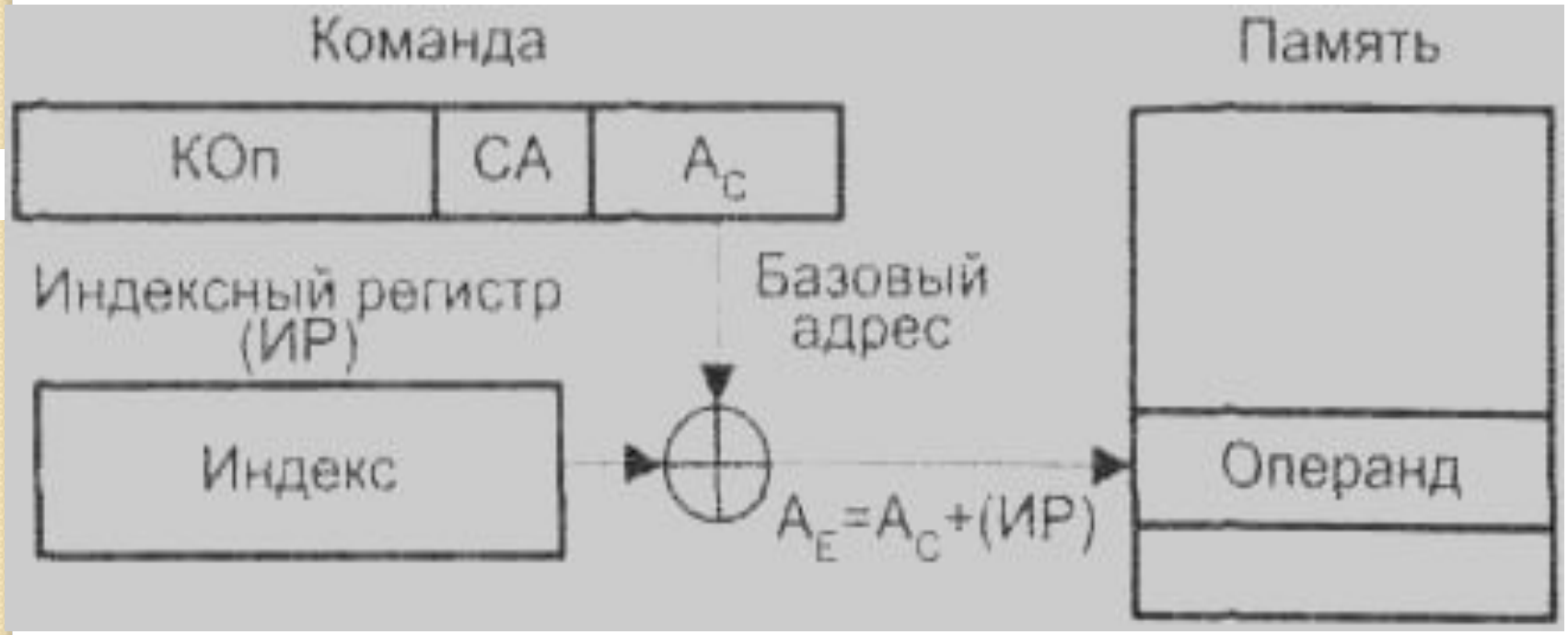
Базовая регистровая адресация с использованием одного из РОН



Форматы команд

- **Способы адресации операндов**

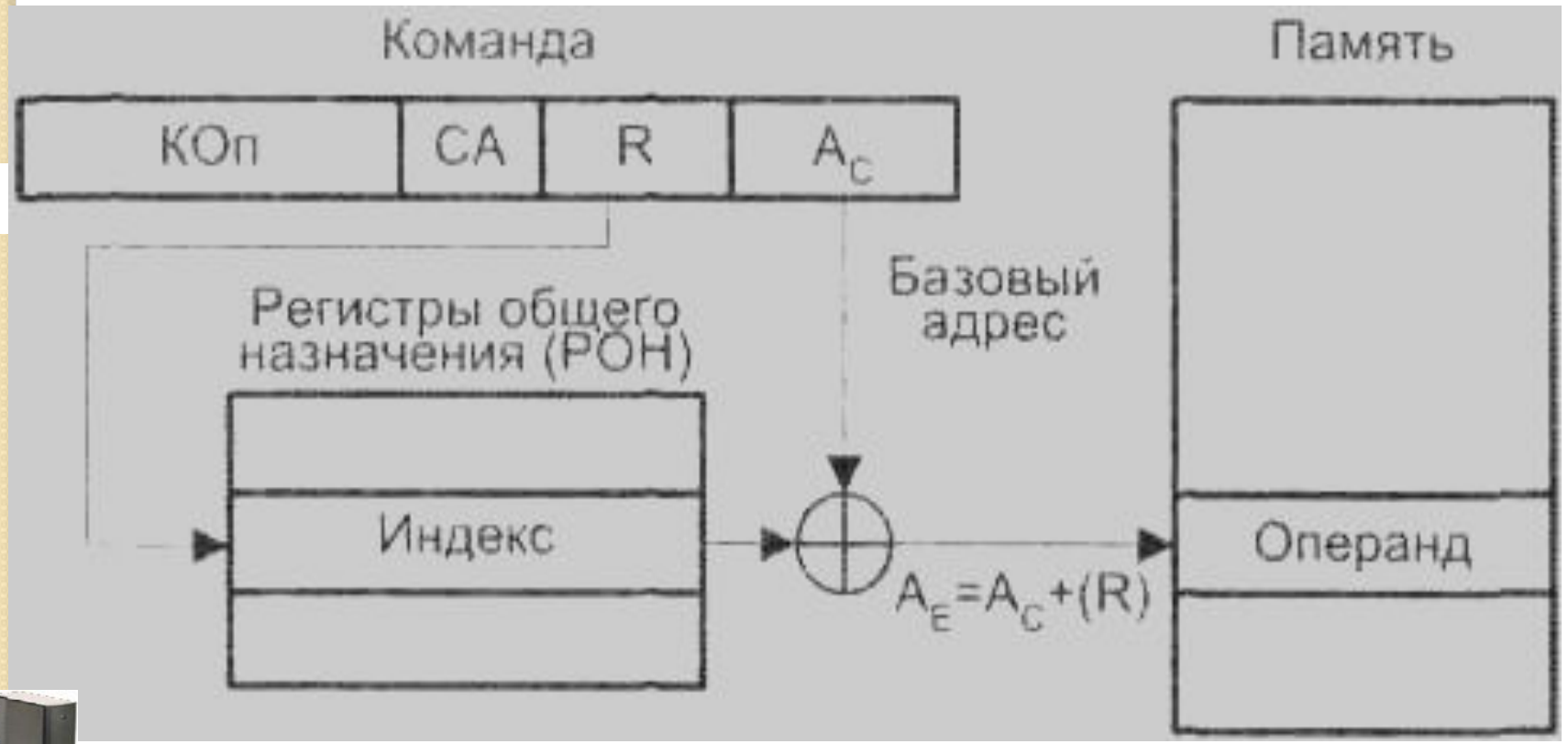
Индексная адресация с индексным регистром



Форматы команд

- **Способы адресации операндов**

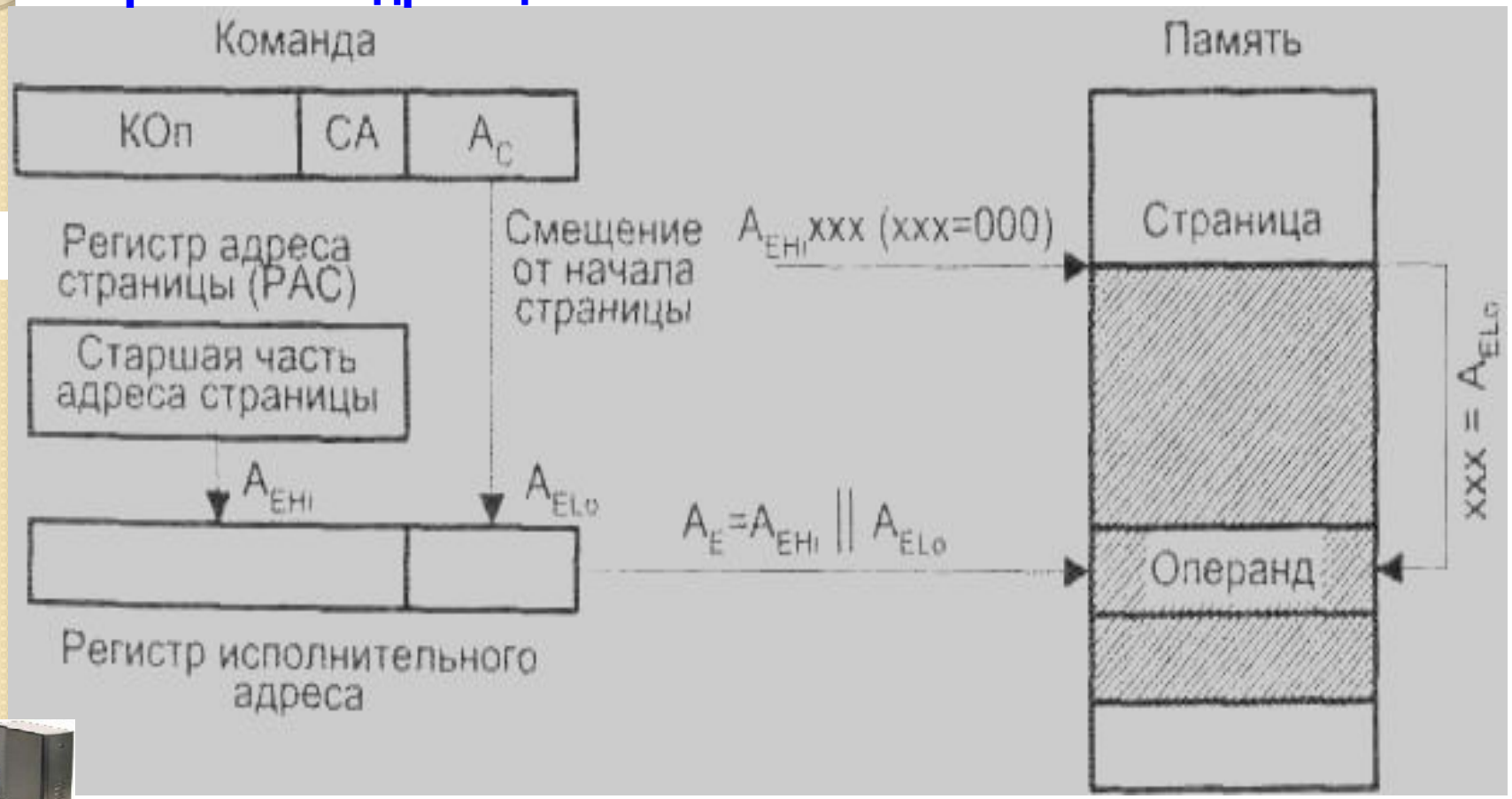
Индексная адресация с использованием одного из РОН



Форматы команд

- **Способы адресации операндов**

Страничная адресация



CISC архитектура команд

- большое количество машинных команд, часть из которых аппаратно реализуют сложные операторы ЯВУ;
- наличие в процессоре сравнительно небольшого числа регистров общего назначения;
- разнообразие способов адресации операндов;
- множество форматов команд различной разрядности;
- наличие команд, где обработка совмещается с обращением к памяти;
- команда может выполняться за несколько машинных циклов

CISC архитектура команд

Достоинства:

Система команд процессоров с CISC–архитектурой может содержать несколько сотен команд разного формата (от 1 до 15 байт), или степени сложности, и использовать более 10 различных способов адресации, что позволяет программисту реализовать наиболее эффективные алгоритмы решения различных задач.

Недостатки:

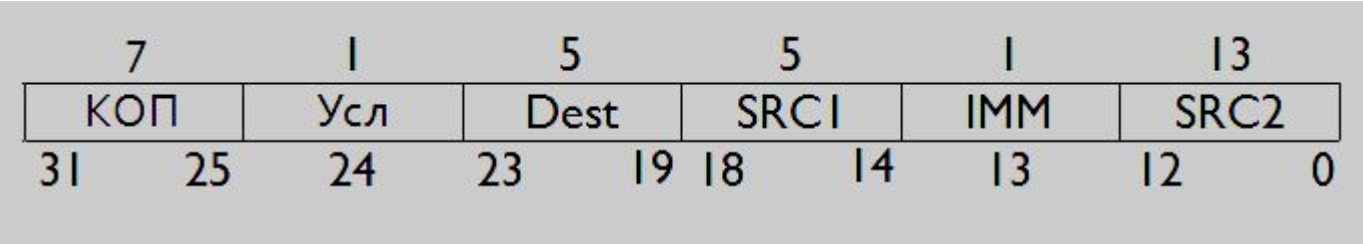
- усложнение аппаратной части АЛУ и УУ
- увеличение сроков разработки процессора
- снижение быстродействия выполнения программ так как большинство команд выполняется за несколько тактов

RISC архитектура команд

- каждая команда независимо от ее типа выполняется за один машинный цикл, длительность которого должна быть максимально короткой;
- все команды должны иметь одинаковую длину и использовать минимум адресных форматов, что резко упрощает логику центрального управления процессором;
- обращение к памяти происходит только при выполнении операций записи и чтения, вся обработка данных осуществляется исключительно в регистровой структуре процессора;
- система команд должна обеспечивать поддержку языка высокого уровня. (Имеется в виду подбор системы команд, наиболее эффективной для различных языков программирования.)

Команда в RISC архитектуре

В большинстве RISC-процессоров команды являются трех адресными.



КОП - код операции.
Усл - бит условия (для команд переходов).
Dest - номер регистра назначения (длина пять бит - $N_{РОН} = 32$).
SRC1 - номер регистра-источника 1
SRC2 - номер регистра или непосредственного значения источника 2:

Если IMM = 1, то SRC2 – непосредственное данное
Если IMM = 0 , то SRC2 – регистр.

Используется два вида формата команды.

1. $R_{Dest} = R_{SRC1}$ oper S_2 - выполнение операции обработки;
2. $R_{Dest} = Mem$; $Mem = R_{SRC1}$ - чтение/запись в память

RISC архитектура команд

Достоинства:

- Повышение производительности обработки программ вычислительных задач.
- Благодаря использованию простых команд и минимума их форматов сокращается время разработки RISC-процессора.
- Улучшение технологичности RISC-процессоров благодаря большей свободе в размещении их элементов на кристалле интегральной схемы.

RISC архитектура команд

Недостатки:

- Нарушение основных принципов программирования:
  - минимум длины исполняемого кода программы
  - снижение семантического разрыва между исходным описанием и машинным кодом
- Сложность построения компилятора, поскольку программа с языка высокого уровня должна транслироваться в микрокод с оптимизацией использования регистров.
- Высокие требования к быстродействию памяти.

CISC система команд и RISC ядро

В последних микропроцессорах фирмы Intel и AMD широко используются идеи, свойственные RISC-архитектуре, начиная с Intel Pentium Pro, появились CISC-процессоры с RISC-ядром.

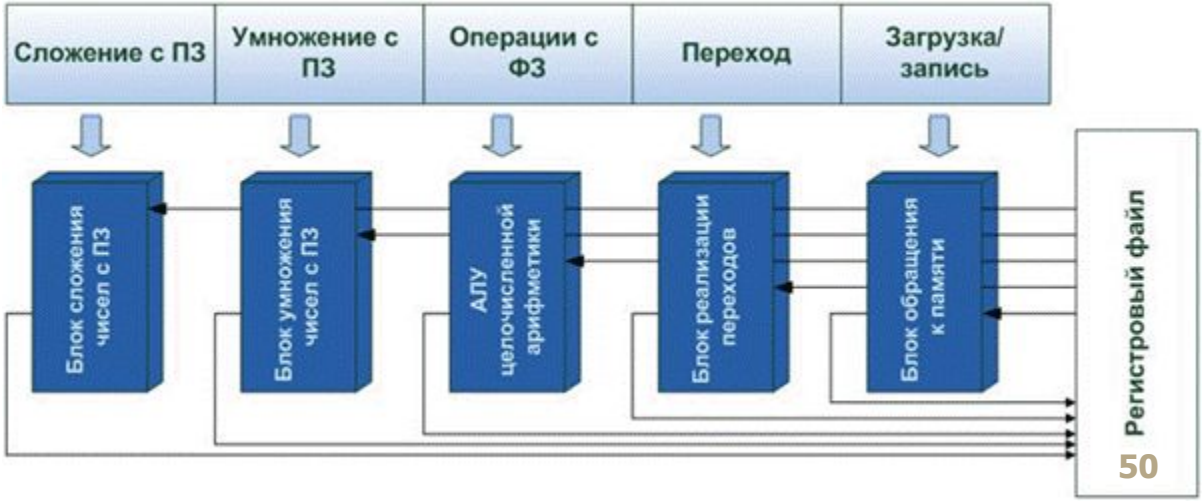
Они непосредственно перед исполнением преобразуют CISC-инструкции в более простой набор внутренних инструкций RISC.

В микропроцессор встраивается аппаратный транслятор, превращающий CISC-команды в команды внутреннего RISC-процессора. При этом одна CISC-команда может порождать несколько RISC-команд.

Исполнение команд происходит на суперскалярном конвейере одновременно по несколько штук. В итоге, такой подход и позволил поднять производительность CPU.

VLIW архитектура команд

- количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
- в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.



VLIW архитектура команд

Преимущества

- Подход VLIW сильно упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор.
- Поскольку отсутствуют большие и сложные узлы, сильно снижается энергопотребление.

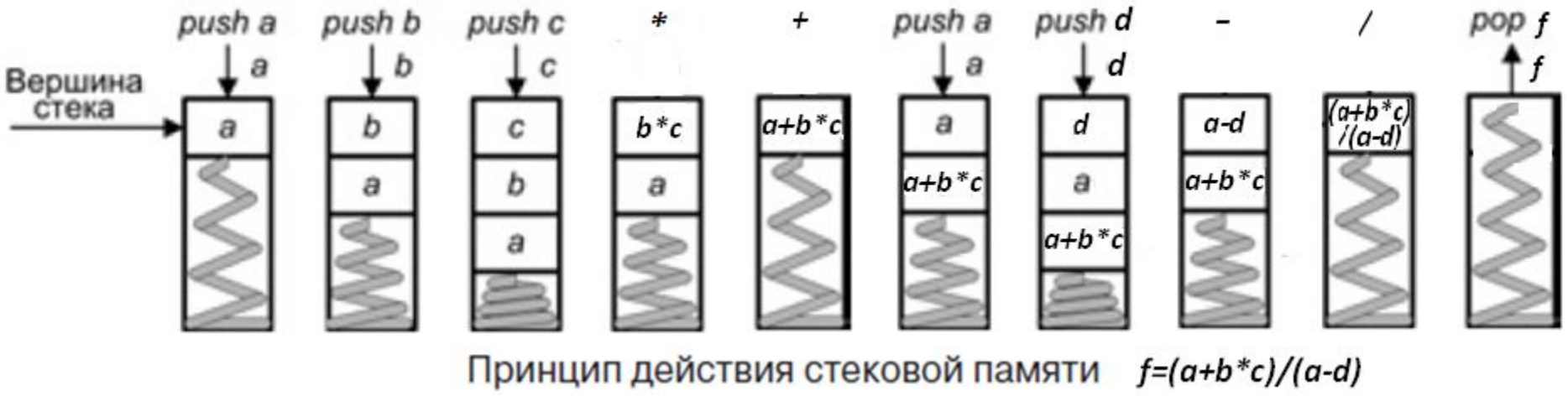
Недостатки

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

Сравнительная оценка CISC-, RISC- и VLIW-архитетур

Характеристика	CISC	RISC	VLIW
Длина команды	Варьируется	Единая	Единая
Расположение полей в команде	Варьируется	Неизменное	Неизменное
Количество регистров	Несколько (часто специализированных)	Много регистров общего назначения	Много регистров общего назначения
Доступ к памяти	Может выполняться как часть команд различных типов	Выполняется только специальными командами	Выполняется только специальными командами

ROSC (стековая) архитектура команд



Стек - LIFO, Last In First Out

При описании вычислений с использованием стека обычно используется форма записи математических выражений, известная как обратная польская, которую предложил польский математик Я. Лукашевич.

f=(a+b*c)/(a-d) f= abc*+ad-/

Сравнение выполнения программы на RISC процессоре и на ROSC процессоре IGNITE

g5 = g1–(g2+1)+g3–(g4*2)

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**

ROSC-архитектура команд

Достоинства

- Сокращение адресной части команд, поскольку все операции производятся через вершину стека (не нужно указывать адреса операндов и результата в командах арифметической и логической обработки информации).
- Компактный код программы.
- Простое декодирование команд.

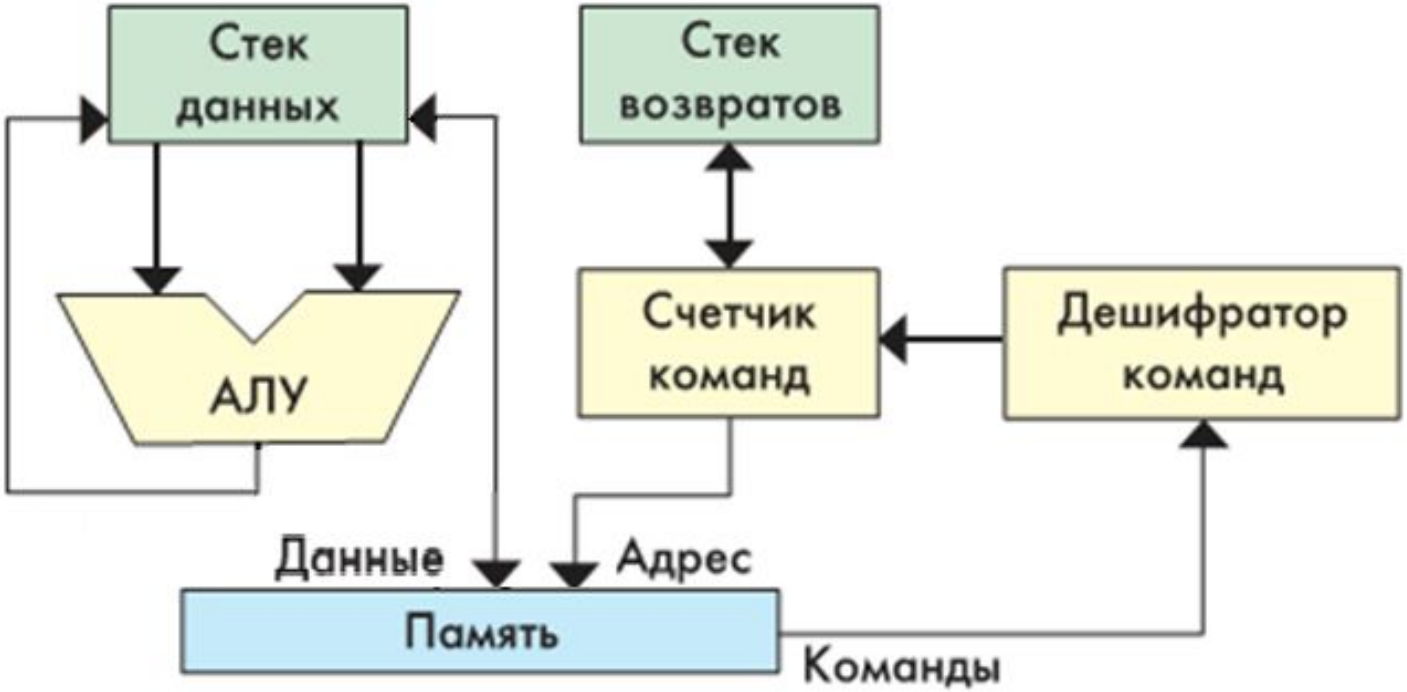
Недостатки

- Стековая архитектура не предполагает произвольного доступа к памяти, из-за чего компилятору трудно создать эффективный программный код, хотя создание самих компиляторов упрощается.
- Стек становится «узким местом» ВМ в плане повышения производительности.

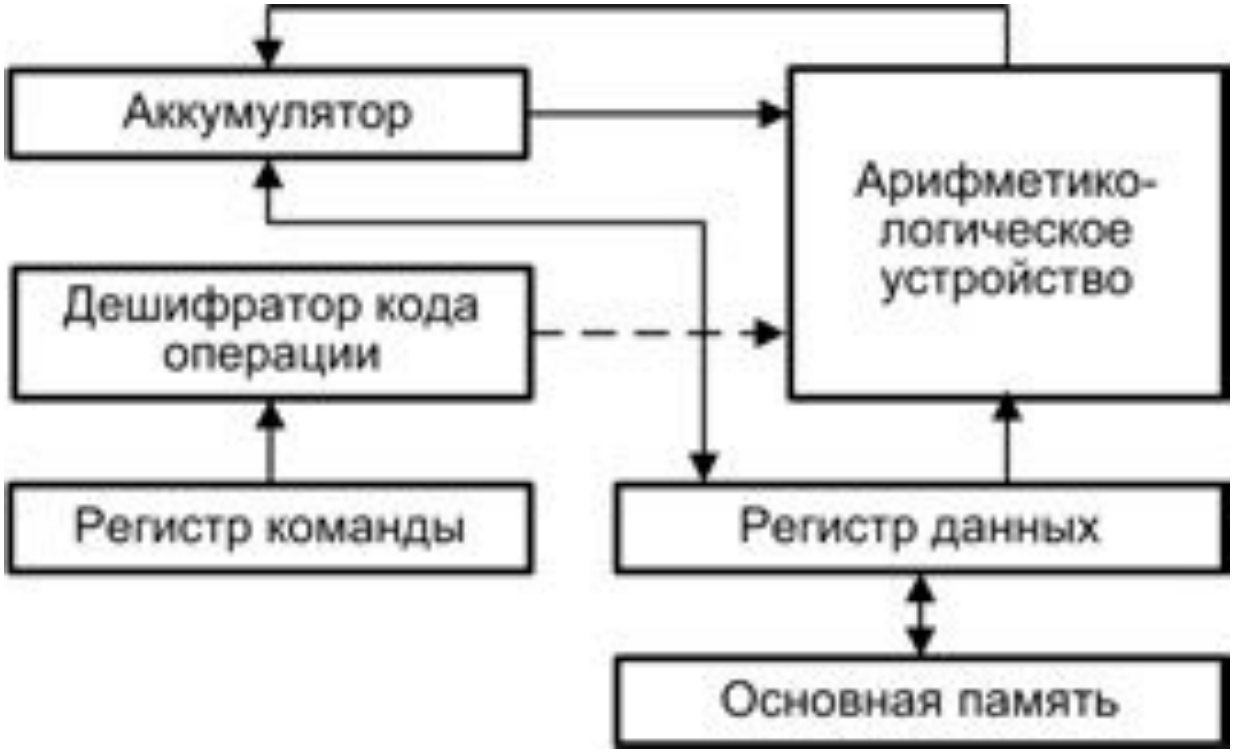
Классификация по месту хранения операндов

- стековая;
- аккумуляторная;
- регистровая;
- с выделенным доступом к памяти.

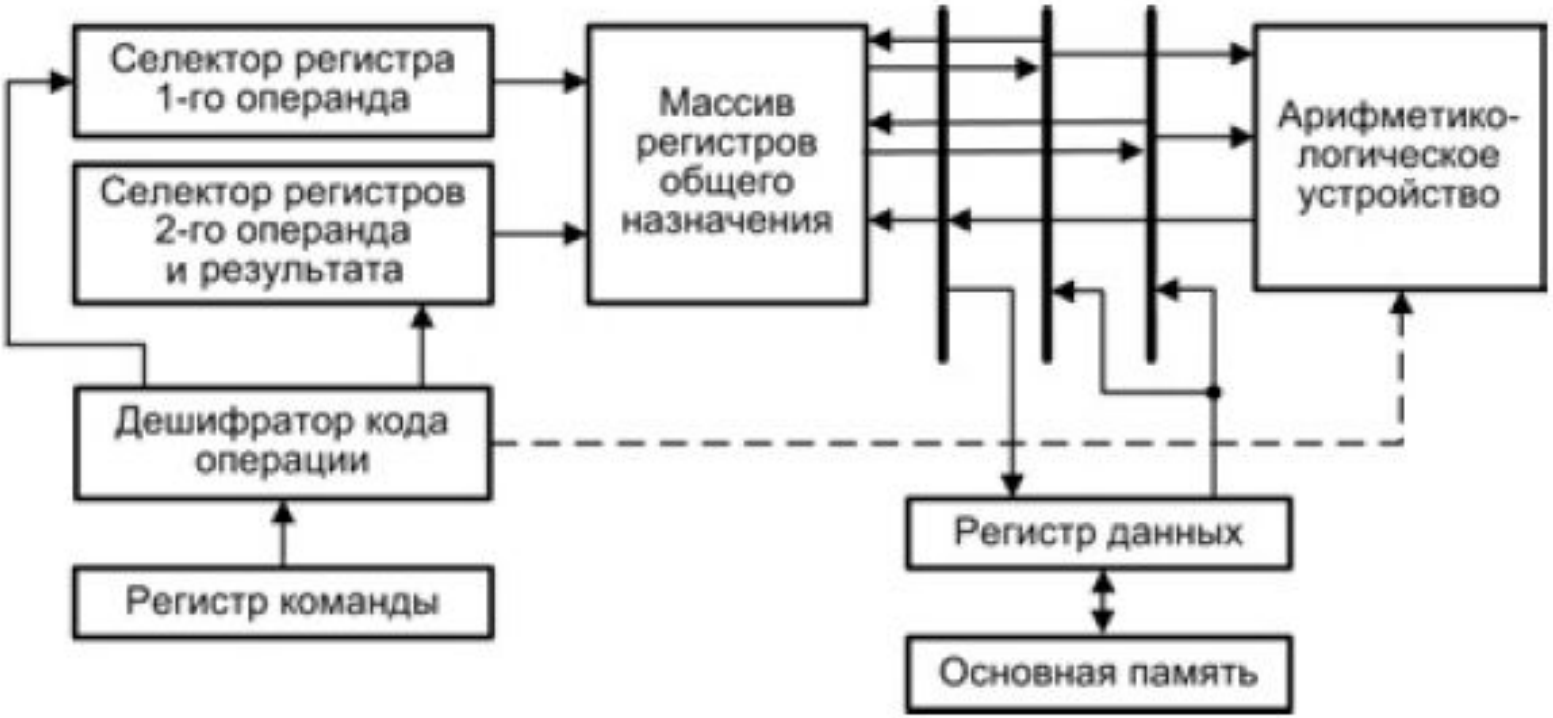
Стековая архитектура



Аккумуляторная архитектура



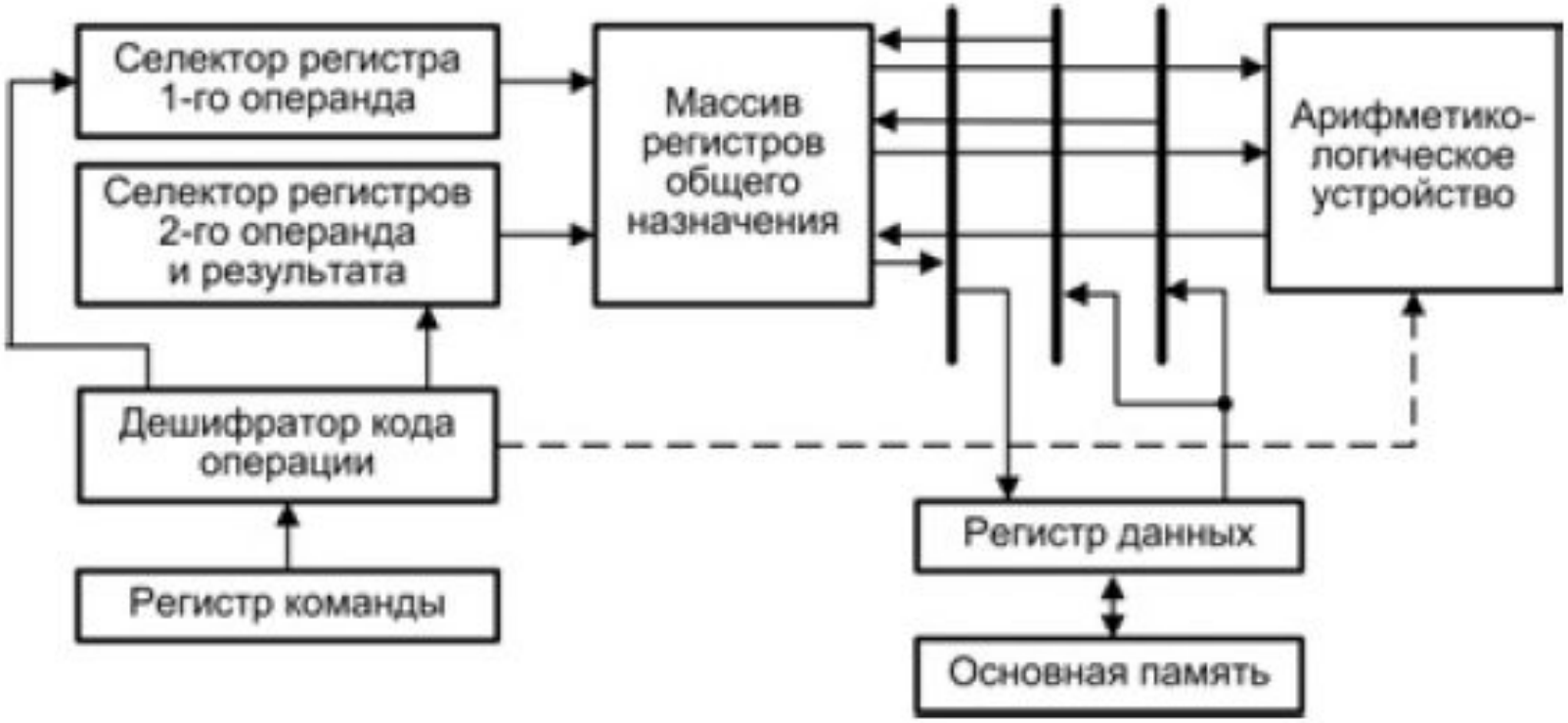
Регистровая архитектура



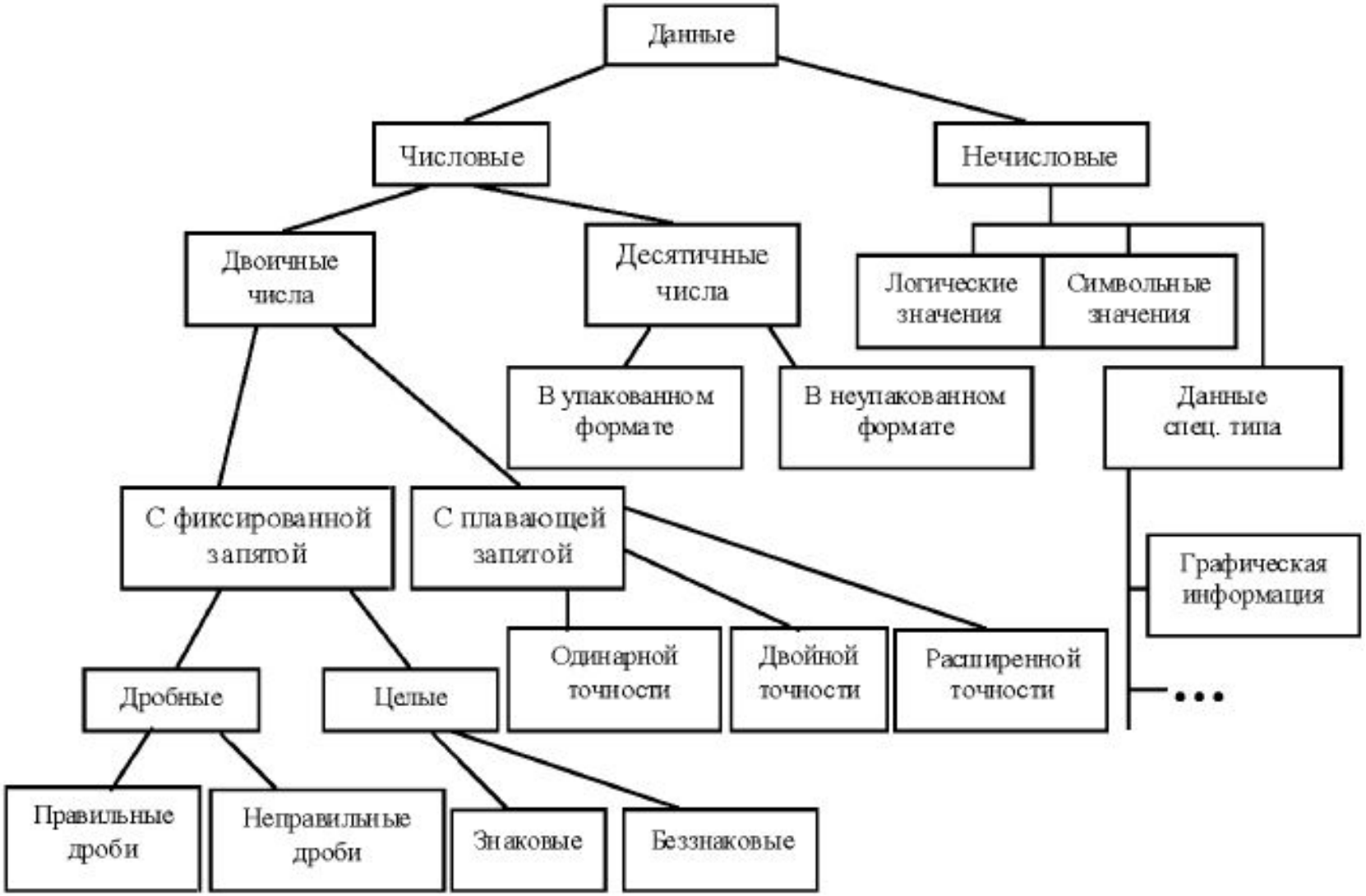
Сравнительная оценка вариантов размещения операндов

Вариант	Достоинства	Недостатки
Регистр-регистр (0, 3)	Простота реализации, фиксированная длина команд, простая модель формирования объектного кода при компиляции программ, возможность выполнения всех команд за одинаковое количество тактов	Большая длина объектного кода, из-за фиксированной длины команд часть разрядов в коротких командах не используется
Регистр-память (1, 2)	Данные могут быть доступны без загрузки в регистры процессора, простота кодирования команд, объектный код получается достаточно компактным	Потеря одного из операндов при записи результата, длинное поле адреса памяти в коде команды сокращает место под номер регистра, что ограничивает общее число РОН. CPI зависит от места размещения операнда
Память-память (3, 3)	Компактность объектного кода, малая потребность в регистрах для хранения промежуточных данных	Разнообразие форматов команд и времени их исполнения, низкое быстродействие из-за обращения к памяти

Архитектура с выделенным доступом к памяти



Классификация данных



Типы команд

- **команды пересылки данных (*регистр-регистр , регистр-память, память-память*);**
- **команды арифметической и логической обработки;**
- **команды работы со строками;**
- **команды SIMD;**
- **команды преобразования;**
- **команды ввода/вывода;**
- **команды управления потоком команд:**
  - **безусловные переходы;**
  - **условные переходы (ветвления);**

Тип выполняемых операций

- Команды пересылки и загрузки данных (память – регистр)
- Команды арифметической и логической обработки
- Команды ввода/вывода
- Команды управления
- Системные команды

Система прерывания программ

Система прерывания программ — это совокупность аппаратных и программных средств, позволяющая ВМ (при получении соответствующего запроса) на время прервать выполнение текущей программы, передать управление программе обслуживания поступившего запроса, а по завершении последней продолжить прерванную программу с того места, где она была прервана


События
Аппаратные прерывания
Программные прерывания
Исключения
Маскируемые
Немаскируемые
Ошибки
Ловушки
Аварии

Цикл команды с учетом прерываний

1. Установка запрета на прием запросов прерывания.
2. Сохранение всей информации прерванной программы, которая необходима для возобновления выполнения этой программы (контекста программы) после завершения обработки прерывания.
3. Снятие запрета на прием запросов прерывания.
4. Идентификация источника ЗП, определение нужного обработчика прерывания и его запуск.
5. Завершение программы обработки прерывания.
6. Установка запрета на прием запросов прерывания.
7. Восстановление контекста прерванной программы (возврат к состоянию на момент прерывания).
8. Снятие запрета на прием запросов прерывания.
9. Возврат к выполнению прерванной программы.

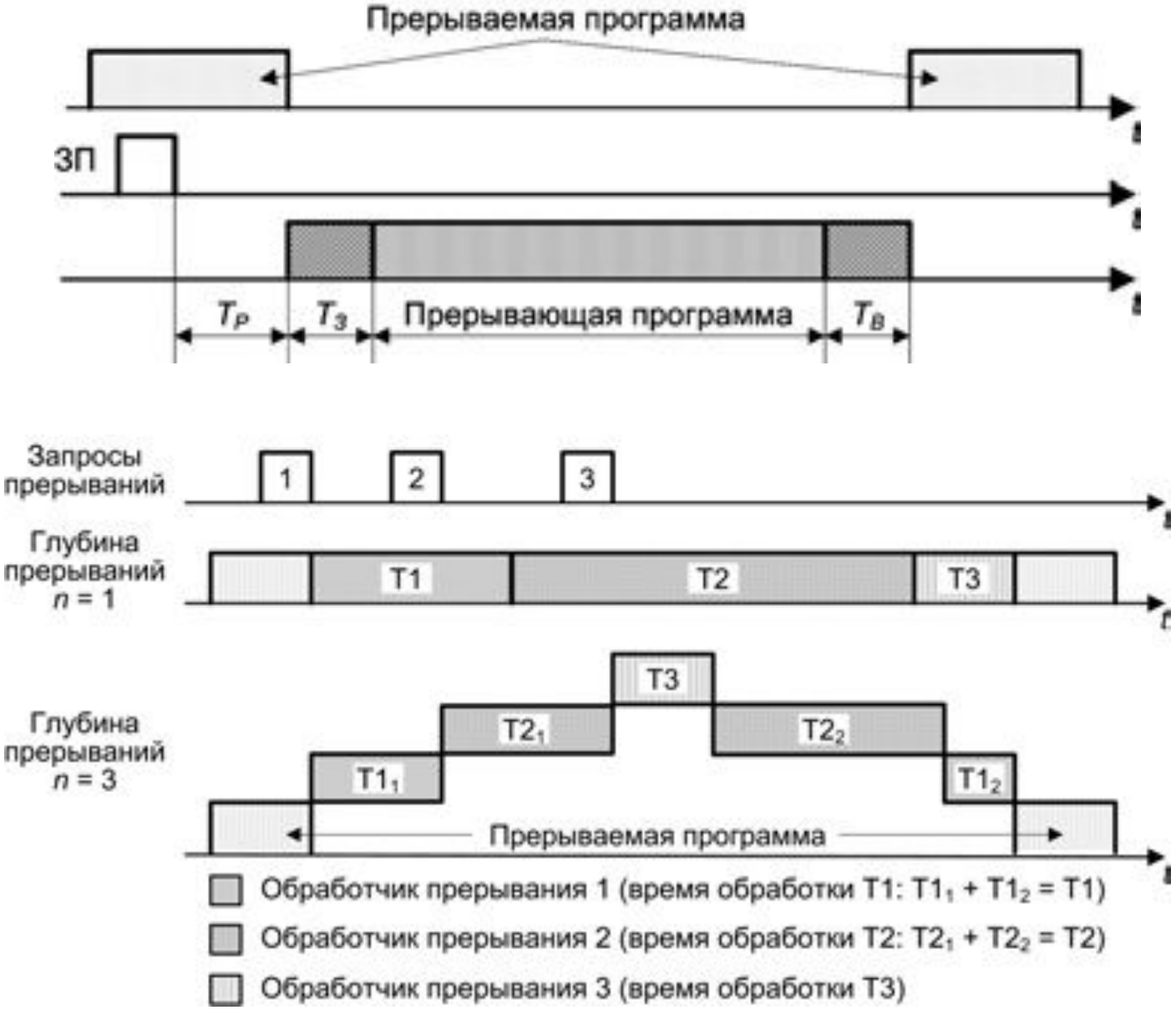
Характеристики систем прерывания

- *время реакции* T_P — время между появлением запроса прерывания и началом выполнения первой команды обработчика прерывания;
- *затраты времени на переключение программ* — суммарный расход времени на запоминание $T_З$ и восстановление $T_В$ состояния программы ($T_{обсл} = T_з + T_в$);
- *эффективность прерывания* η — отношение времени выполнения прерывающей программы к общему времени, необходимому для обслуживания прерывания ($\eta = T_п / T_{пр}$);
- *глубина прерываний* — максимальное число программ, которые могут последовательно прерывать друг друга.

С позиций глубины прерывания можно рассматривать три варианта СПП:

1. **СПП способна воспринимать только один запрос;**
2. **глубина прерываний ограничена некоторым значением *n;***
3. **программы могут неограниченно прерывать друг друга.**

временная диаграмма процесса прерывания



Допустимые моменты прерывания программ

Метод помеченного оператора (метод опорных точек) – наличие в коде команд специального бита, единичное значение которого означает разрешение прерывания по завершении данной команды, а нулевое — запрет. Сокращается время обслуживания $T_{обсл}$, но время реакции T_P увеличивается

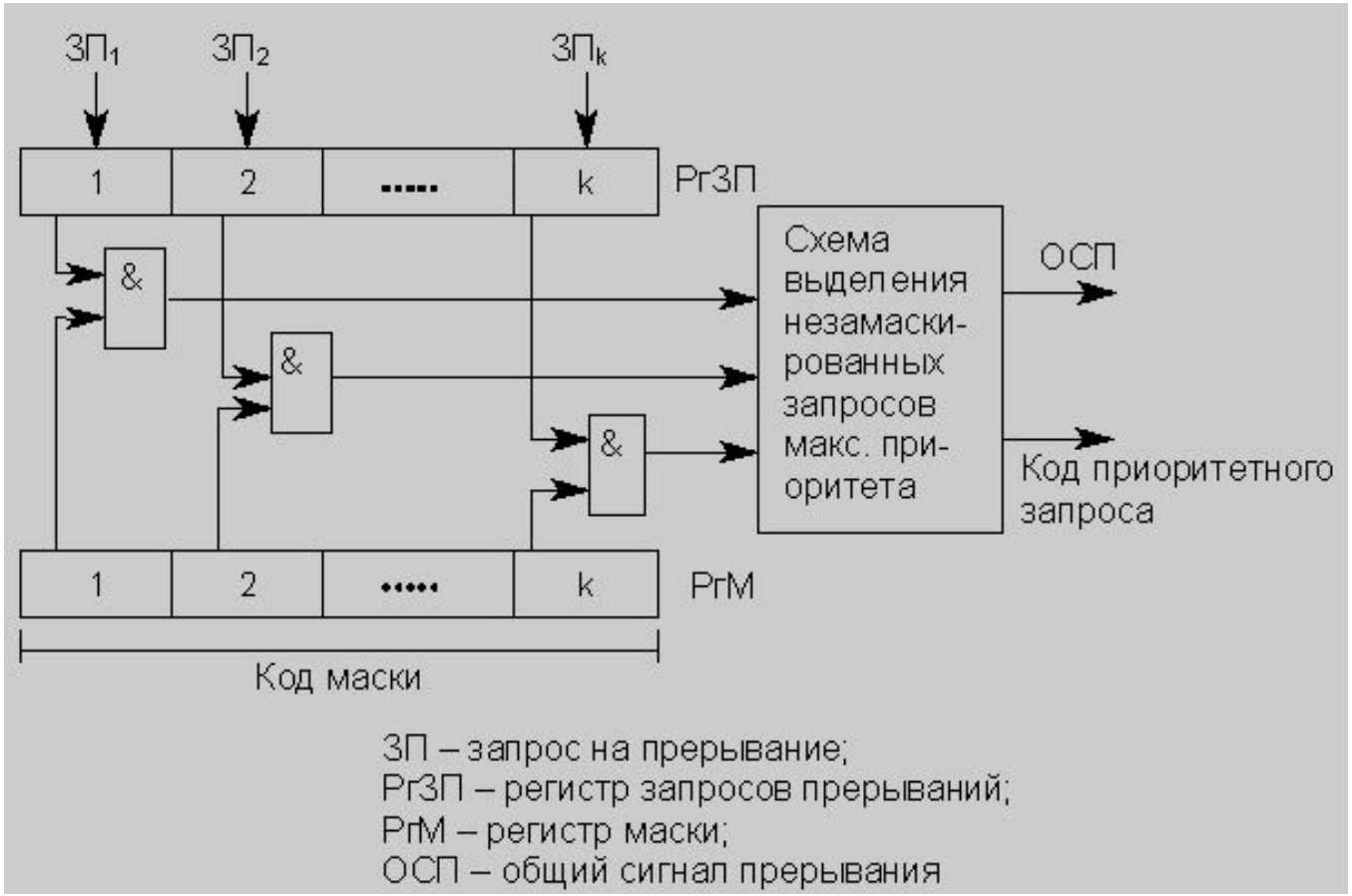
Покомандный метод - прерывание допускается после завершения любой текущей команды. Уменьшается времени реакции T_P, *но* возрастает время обслуживания $T_{обсл}$

Метод быстрого реагирования - допускает прерывание после любого такта выполнения команды. Минимальное время реакции T_P, но возрастает объем запоминаемого контекста программы и увеличивается $T_{обсл}$

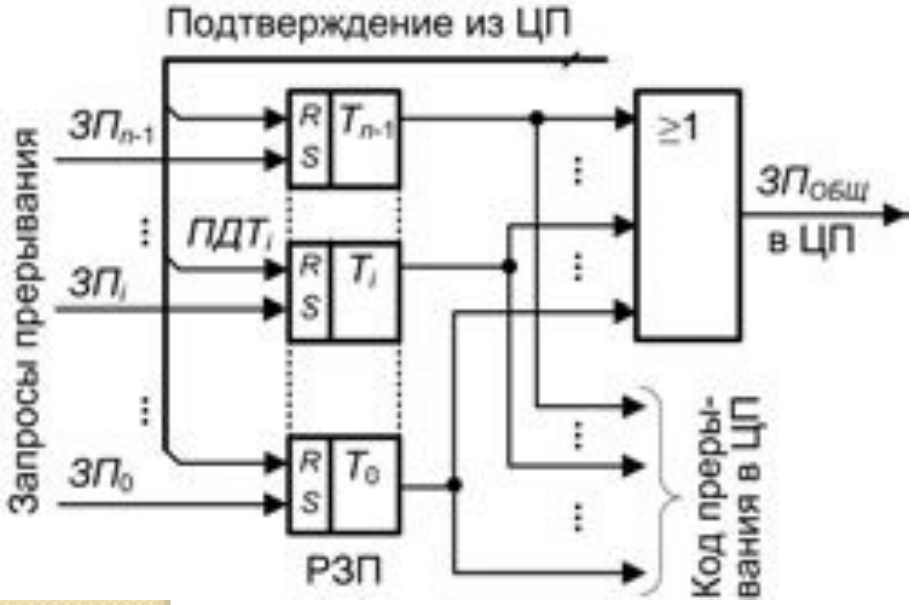
Дисциплины обслуживания прерываний

$P = p_{n-1} \ldots p_1 p_0$ - код прерывания (p_i – запрос от *i*-го источника)

$M = m_{n-1} \ldots m_1 m_0$ – маска прерываний ($m_i = 1$ — *прерывание* от *i*-го источника *допустимо*)

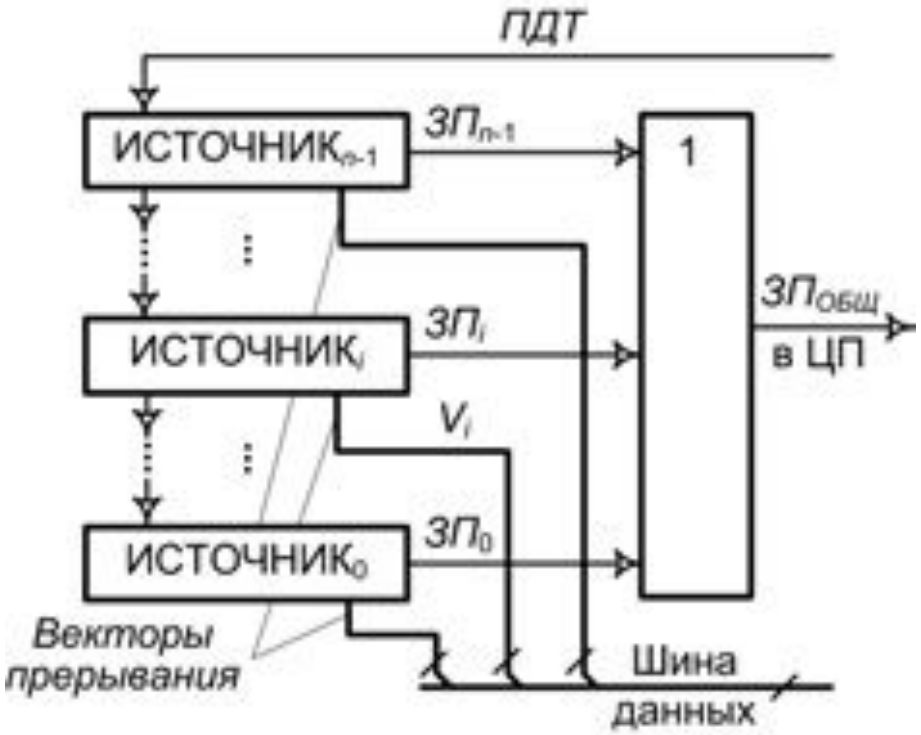


Идентификация источника запроса прерывания



⇐ *обзорная СПП*

СПП с векторными прерываниями ⇒



Система приоритетов

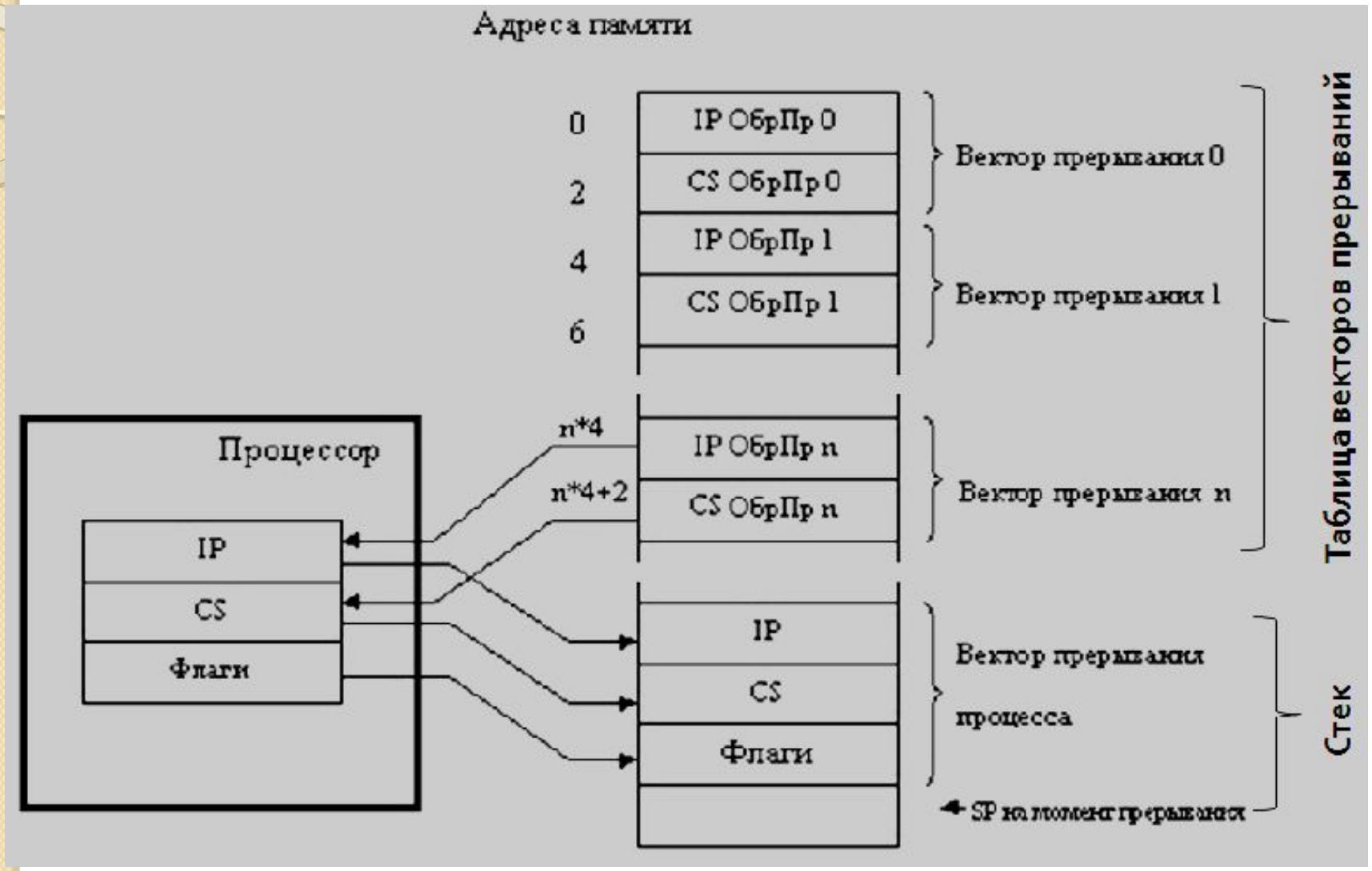
Система приоритетов позволяет определить: имеет ли право поступивший запрос прерывания прервать выполняемую в данный момент программу. В случае одновременного поступления нескольких ЗП эта система дает возможность выбрать тот из них, который должен быть обслужен в первую очередь.

Различают абсолютный и относительный приоритеты.

Запрос, имеющий абсолютный приоритет, прерывает выполняемую программу и запускает выполнение соответствующей прерывающей программы.

Запрос с относительным приоритетом является первым кандидатом на обслуживание после завершения выполнения текущей программы.

Обслуживание прерывания



Спасибо за внимание