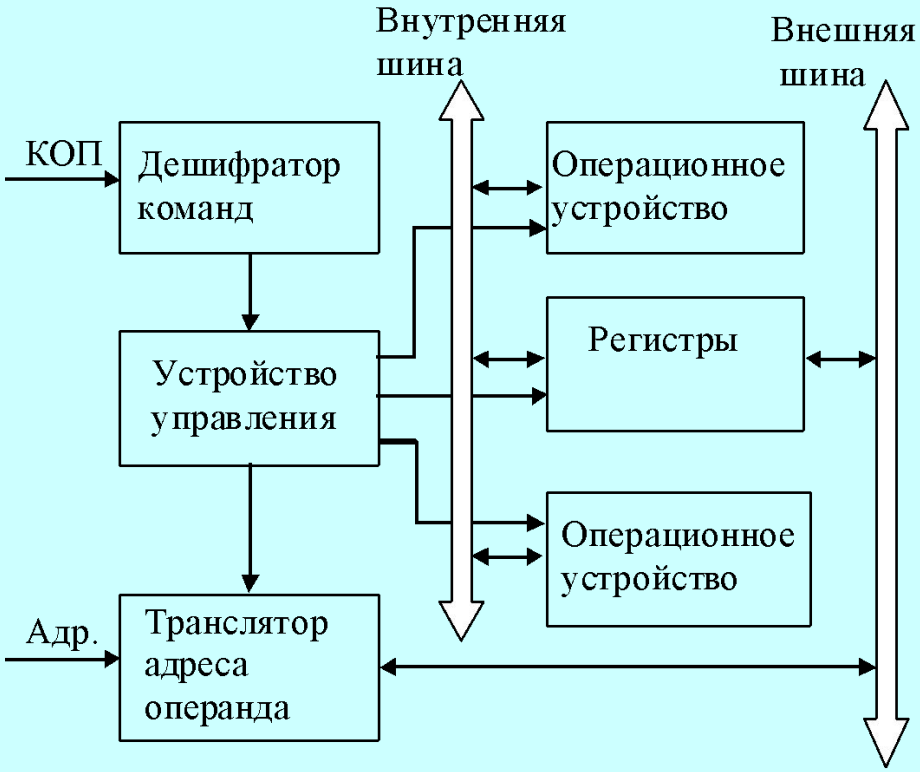
План лекции
Процессор

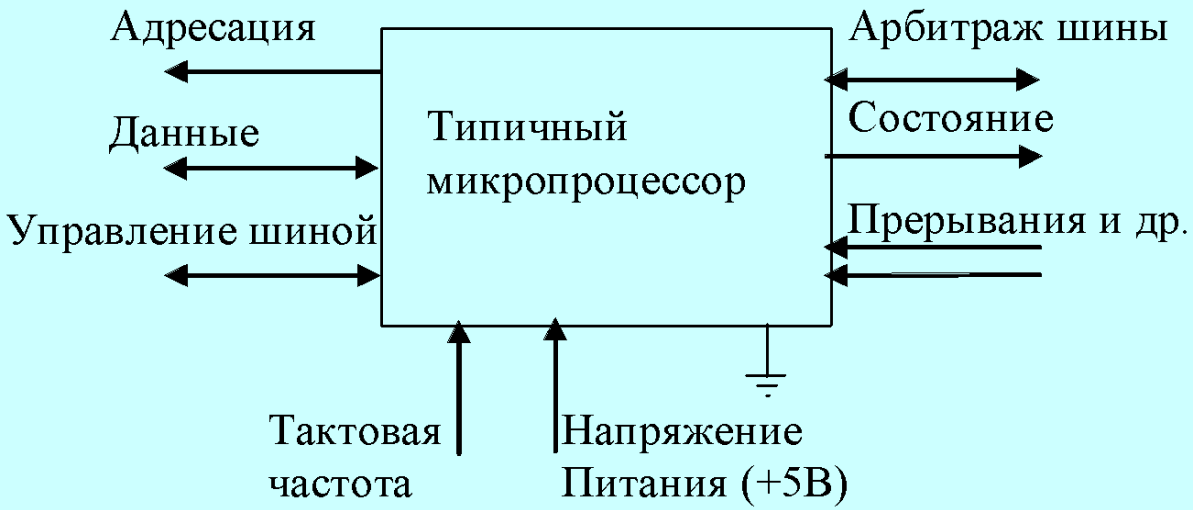
1. **Элементы процессора. Аппаратный уровень. Операционные устройства.**
2. **Устройство управления. Микропрограммный автомат. Микропрограмма.**
3. **Процессор Intel 8086. Сегментная организация памяти.**
4. **Команды. Кодирование команд.**
5. **Подпрограммы.**

Элементы процессора

- **Дешифратор команд**
- **Операционные устройства (Арифметико - логическое устройство (АЛУ))**
- **Устройство управления**
- **Регистры**
- **Транслятор адресов операндов**
- **Шины**



Типичный микропроцессор



В 1971 году появился первый микропроцессор Intel 4004. Он обладал такой же вычислительной мощностью, что и ENIAC. Но стоил всего 200 долларов. Вмещал 2.300 транзисторов на кристалле размерами не больше ногтя и потреблял несколько десятков ватт. Его тактовая частота равнялась 108 КГц.

Операционные устройства

- целочисленной арифметики
- логических операций
- операций сдвига
- десятичной арифметики
- чисел с плавающей запятой
- графических операций ...

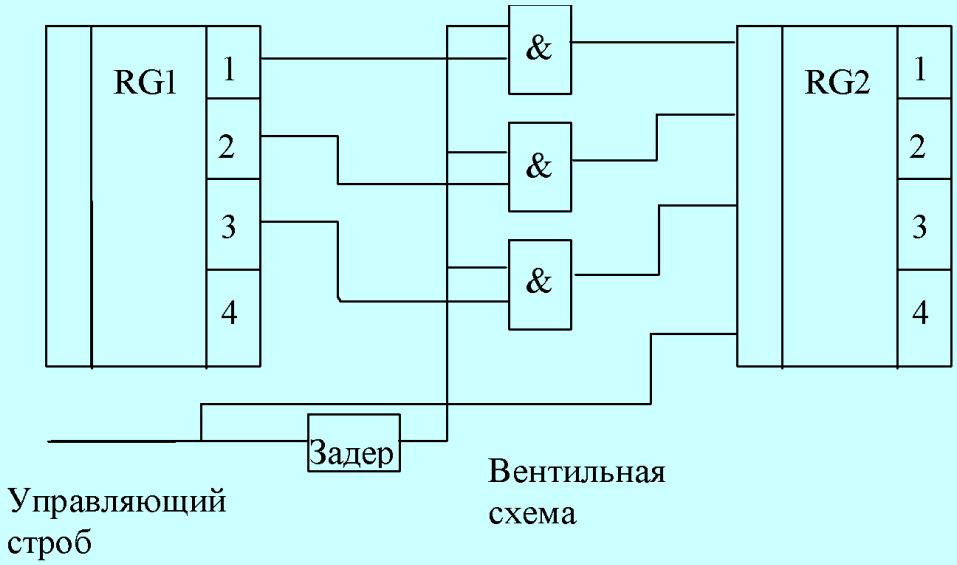
Структурный базис - набор элементов, на основе которых строятся структуры операционных устройств.

- **регистры** - обеспечивают хранение слов данных;
- **шины -** связывают регистры и предназначены для передачи слов данных;
- **комбинационные схемы** - реализуют вычисления по управляющим сигналам от устройства управления .

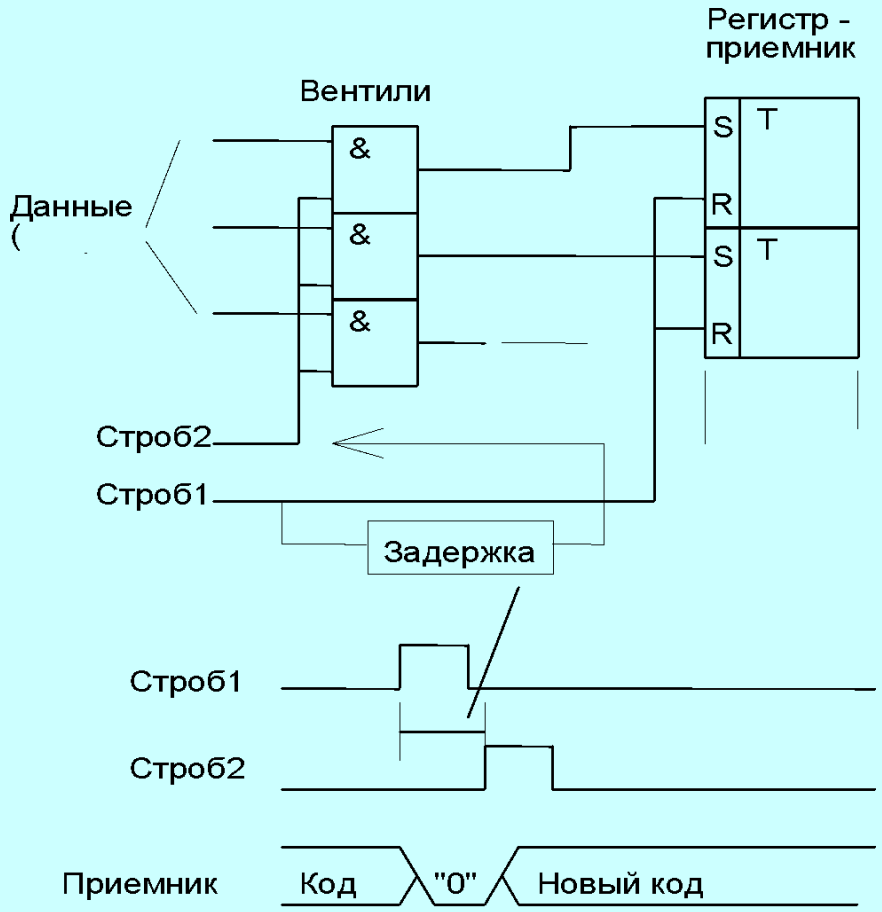
Операционные устройства. Регистры

Регистр - линейка триггеров, имеющих входы для изменения состояния, которое влияет на выходы. Регистры могут быть:

- программно-видимые явно
- видимые косвенно, такие как "теневые" регистры дескрипторов сегментов
- внутренние для специальных целей, например регистр адреса памяти и регистр данных для обмена с памятью, про них по крайней мере известно, для чего они...
- внутренние для хранения внутренних промежуточных результатов...

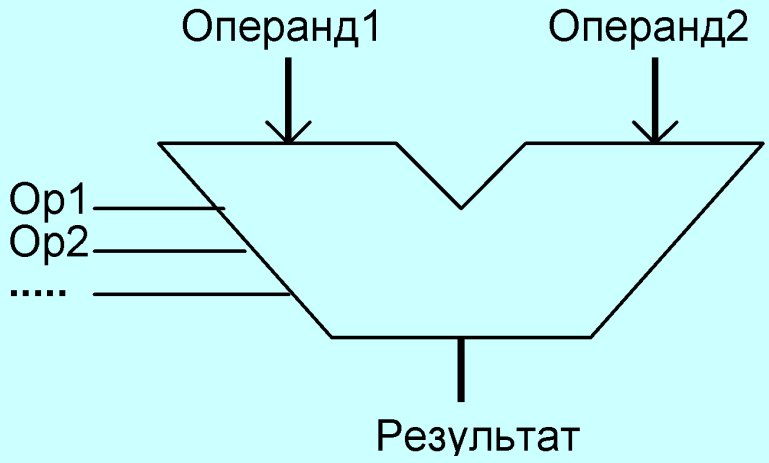


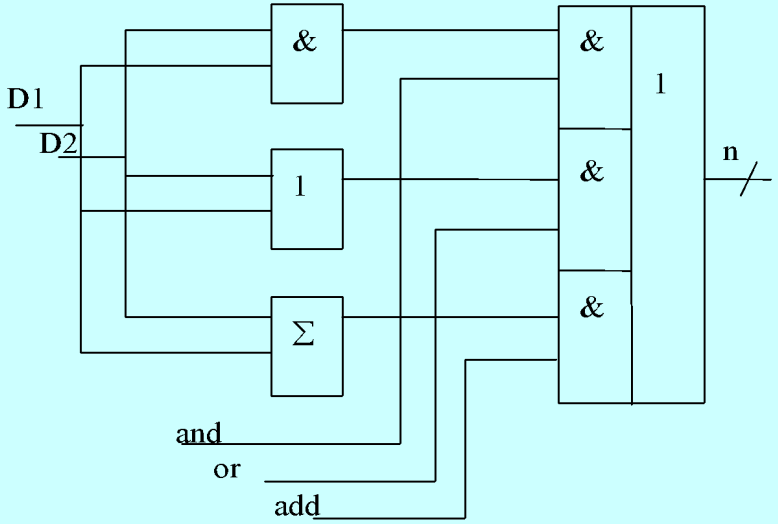
Простейший мультиплексор



Операционные устройства. АЛУ

АЛУ - комбинационная схема, (т.е. она не содержит внутри элементов памяти), принимающая на два входа два операнда (например одержимые двух регистров) и формирующая на выходе результат операции.

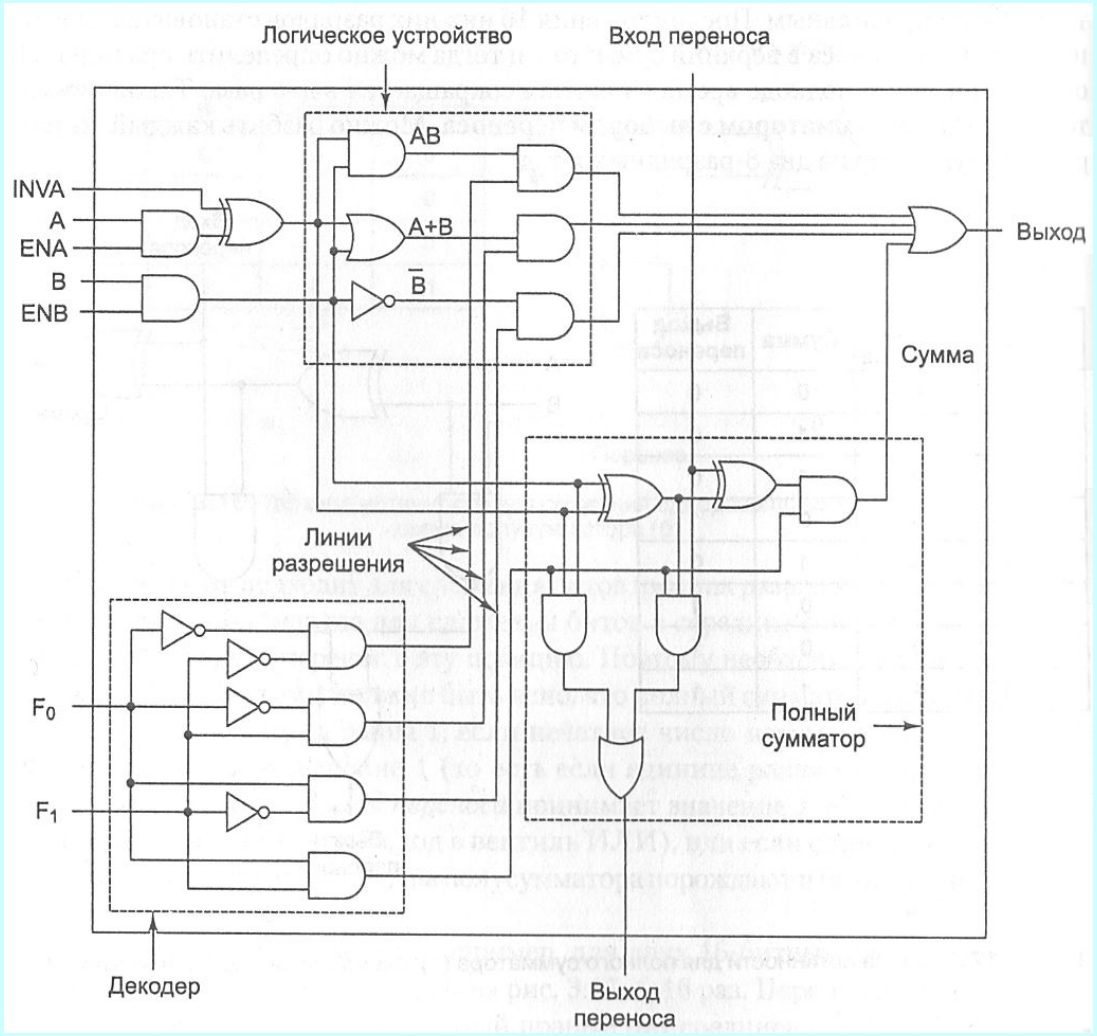




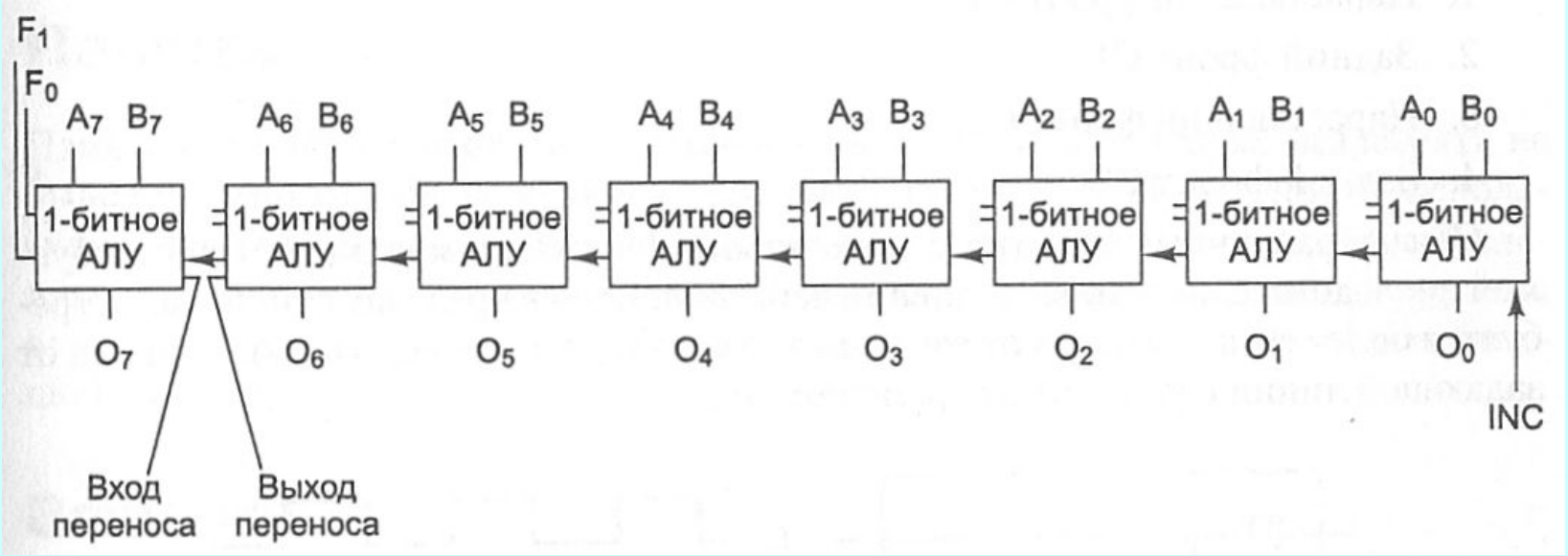
АЛУ. Принципиальная схема.

Состав

- Логическое устройство
- Полный сумматор
- Декодер операции

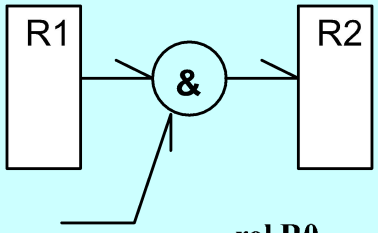


Восьмиразрядное АЛУ.

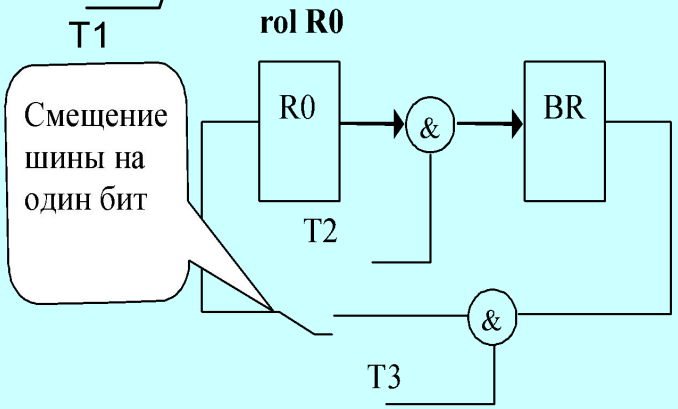


Простейшие операционные устройства

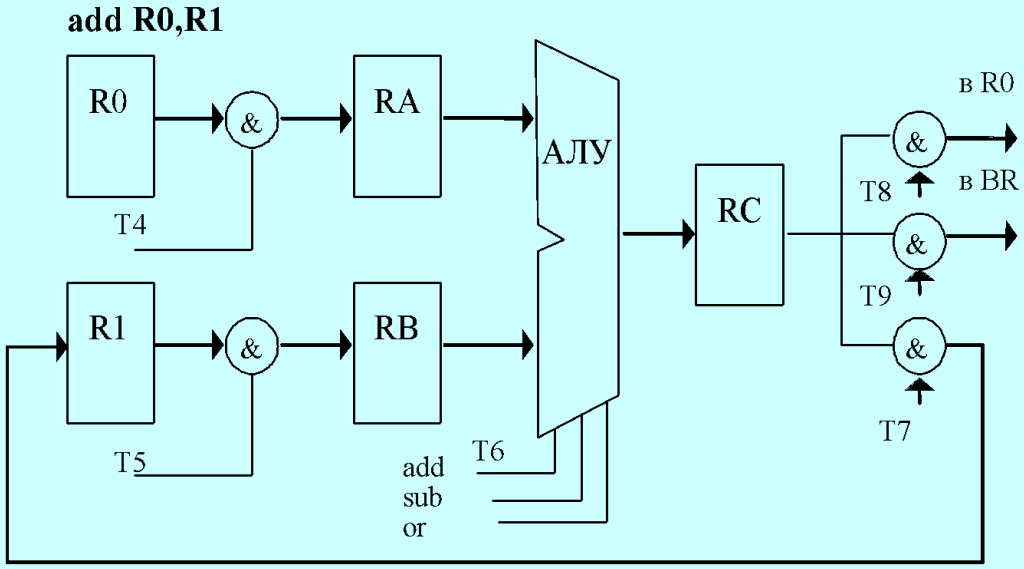
Пересылка mov R1,R2

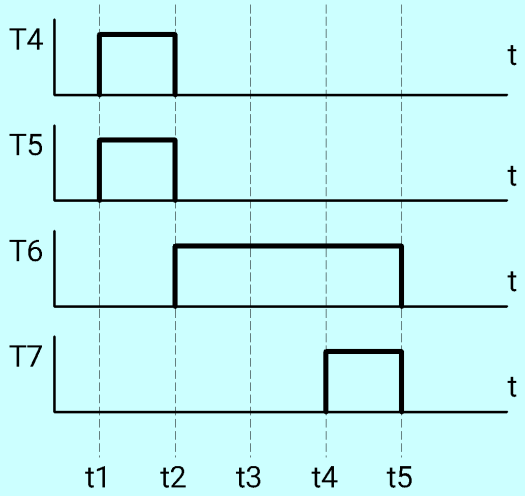


Сдвиг rol R0

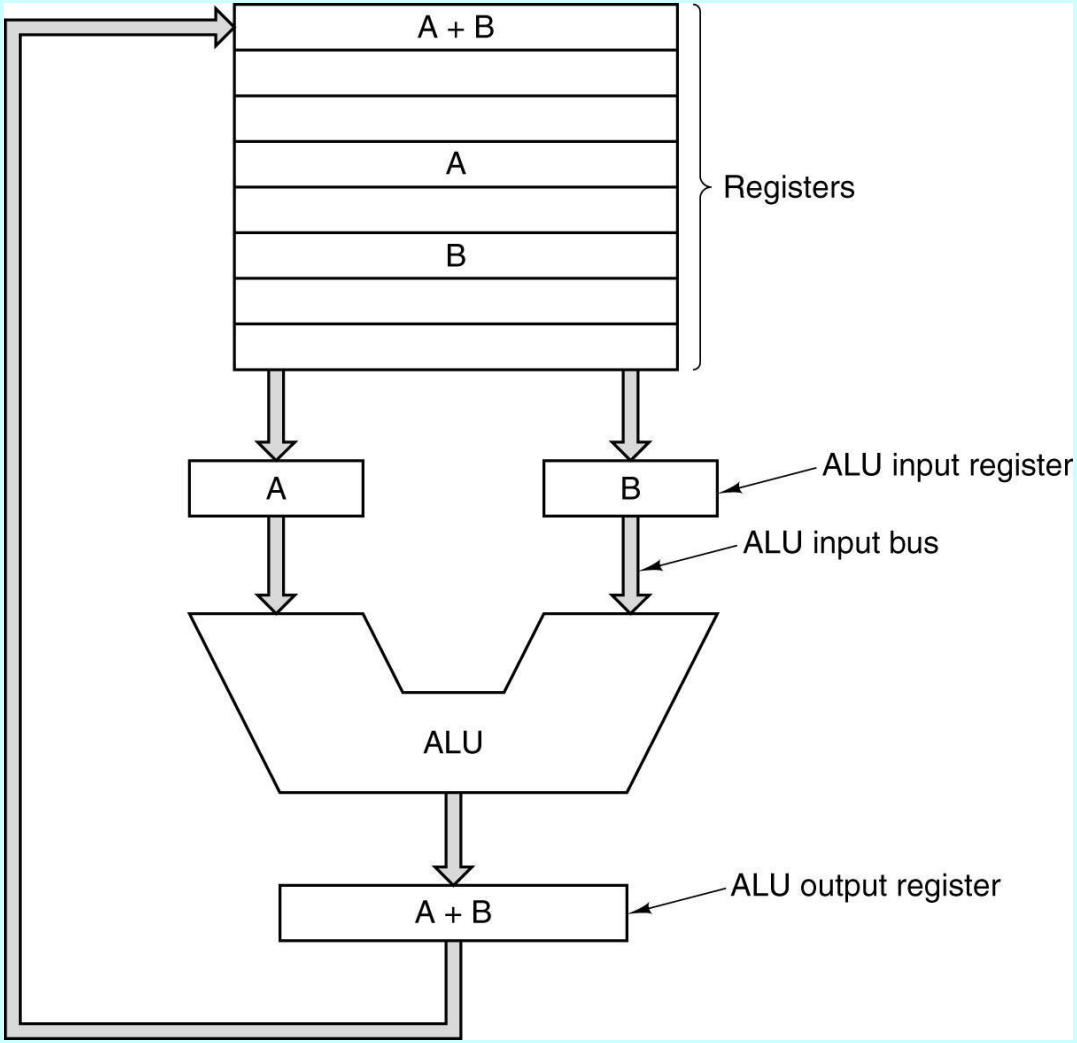


Сложение add R0, R1.

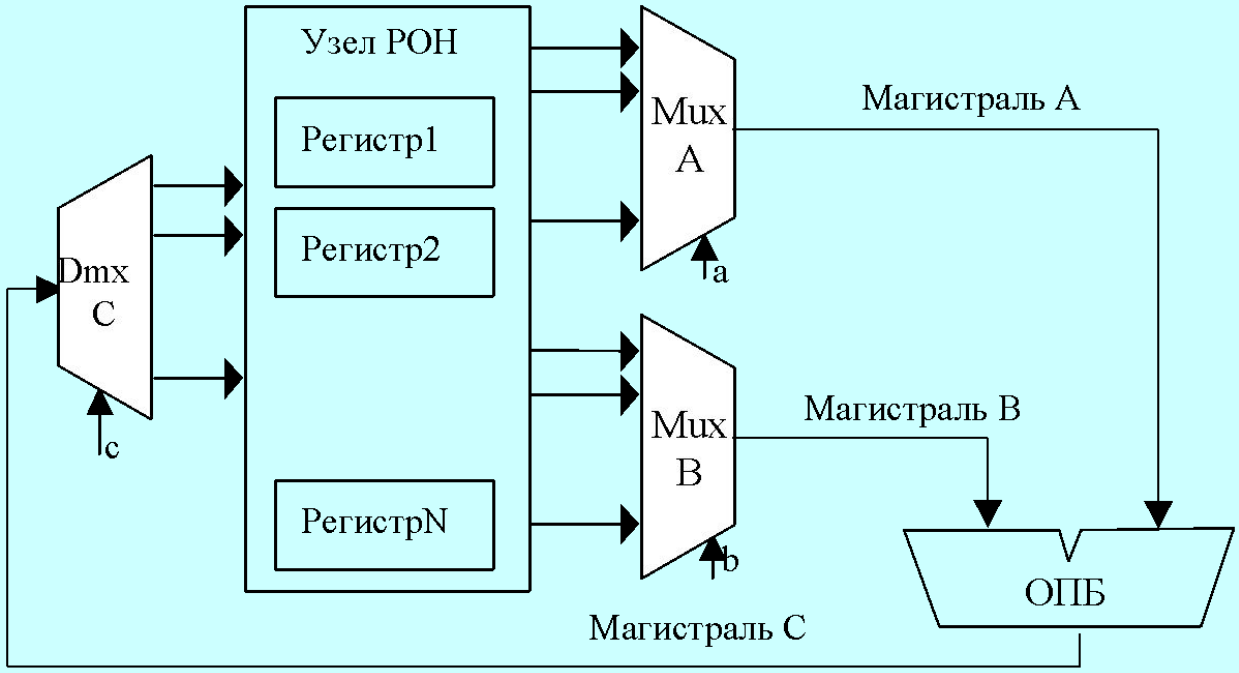




Как связать операционные устройства с шиной ?
The data path of a typical Von Neumann machine.



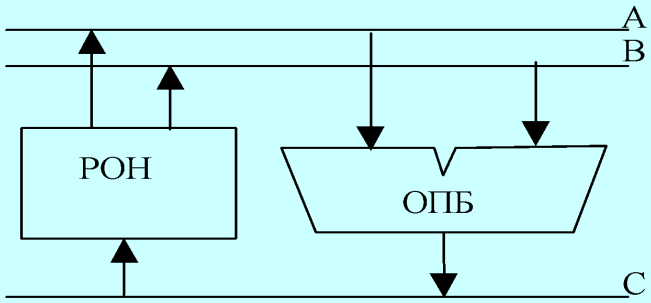
Операционное устройство с магистральной структурой



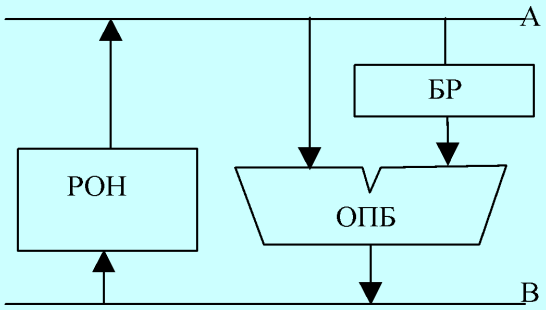
Магистральное операционное устройство

ОПБ - операционный блок

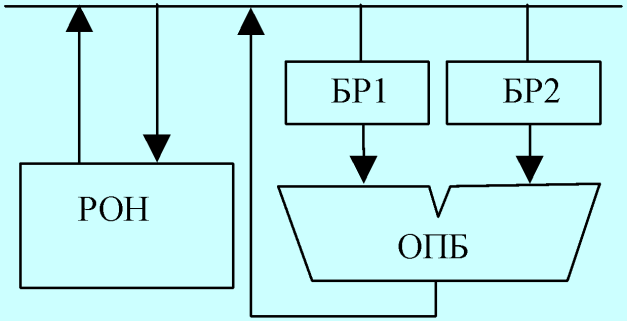
Типы операционных устройств с магистральной структурой



Трехмагистральное ОПУ



Двухмагистральное ОПУ



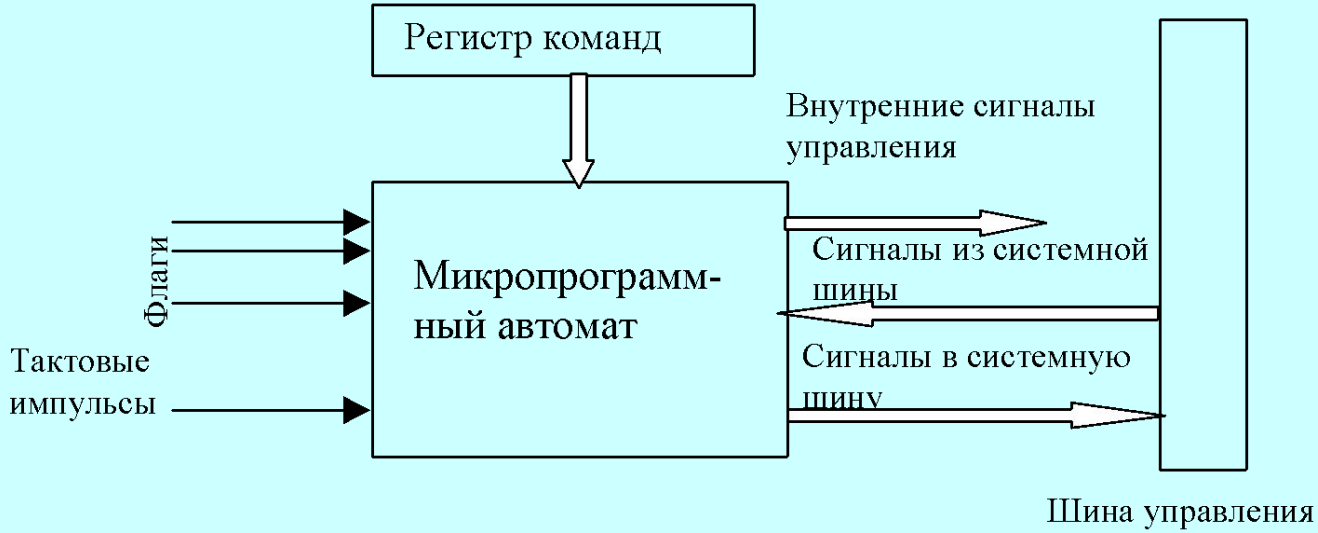
Одномагистральное ОПУ

Функции устройства управления

Устройство управления ЭВМ реализует функции управления ходом вычислительного процесса, обеспечивая автоматическое выполнение команд программы.

- формирует необходимые управляющие сигналы для выборки очередной команды из ОЗУ,
- дешифрации кода операции,
- формирование адресов операндов,
- выборки операндов из ОЗУ,
- передача операндов в операционное устройство,
- выполнение операций операционным устройством,
- передача результата из операционного устройства в ОЗУ,
- инициирование операции ввода/вывода,
- организация реакции процессора на запросы прерывания.

Устройство управления. Микропрограммный автомат



Модель устройства управления

Микрооперация - элементарные действия, выполняемые в течение одного такта сигналов синхронизации.

Микрокоманда - совокупность одновременно выполняемых микроопераций.

Микропрограмма - последовательность микрокоманд, определяющая порядок реализации машинного цикла.

Состав устройства управления

Состав управляющей части УУ (на основе декодирования команды вырабатывает определенную последовательность микрокоманд):

- регистр команды,
- микропрограммный автомат (+ дешифратор команд),
- узел прерываний и приоритетов.

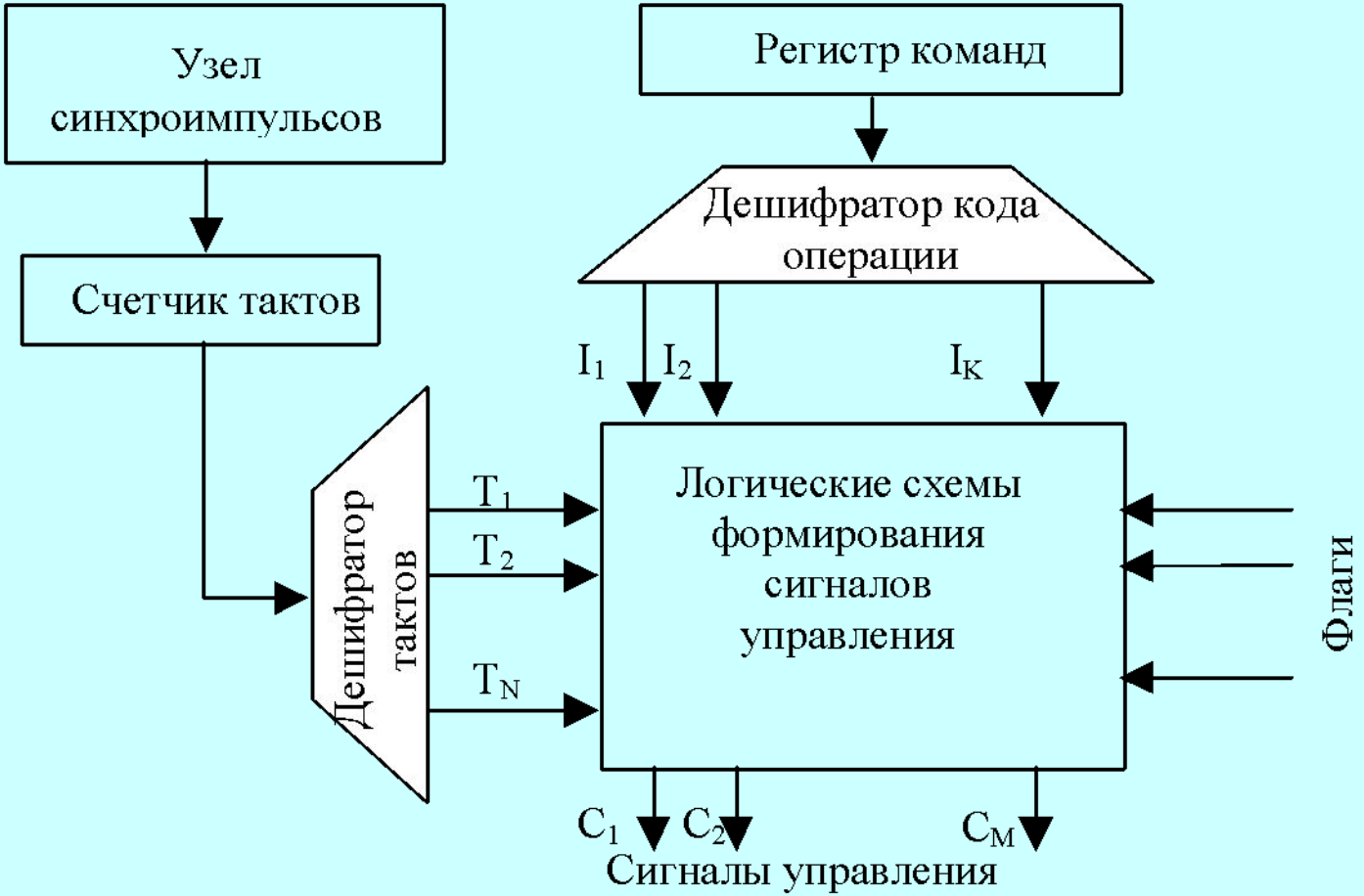
Состав адресной части УУ (обеспечивает формирование адресов команд и адресов операндов в основной памяти):

- операционный узел устройства управления,
- регистр адреса,
- счетчик команд.

Микропрограммный автомат с жесткой логикой.

- Выходные сигналы управления реализуются за счет однажды соединенных логических схем.
- Код операций (КОП), определяет какие сигналы управления и в какой последовательности должны формироваться.
- Сигналы управления, по которым выполняется микрооперация, вырабатываются в строго определенные моменты времени, “привязаны” к импульсам синхронизации.
- Счетчик тактов сбрасывается (устанавливается в состояние T_1) по окончании цикла очередной команды.
- Цикл команды может потребовать разного количества тактов. На каждом такте вырабатывается своя микрокоманда, состоящая из нескольких сигналов управления.
- Дополнительным фактором, влияющим на выработку сигналов управления, являются флаги.

Схема микропрограммного автомата с жесткой логикой



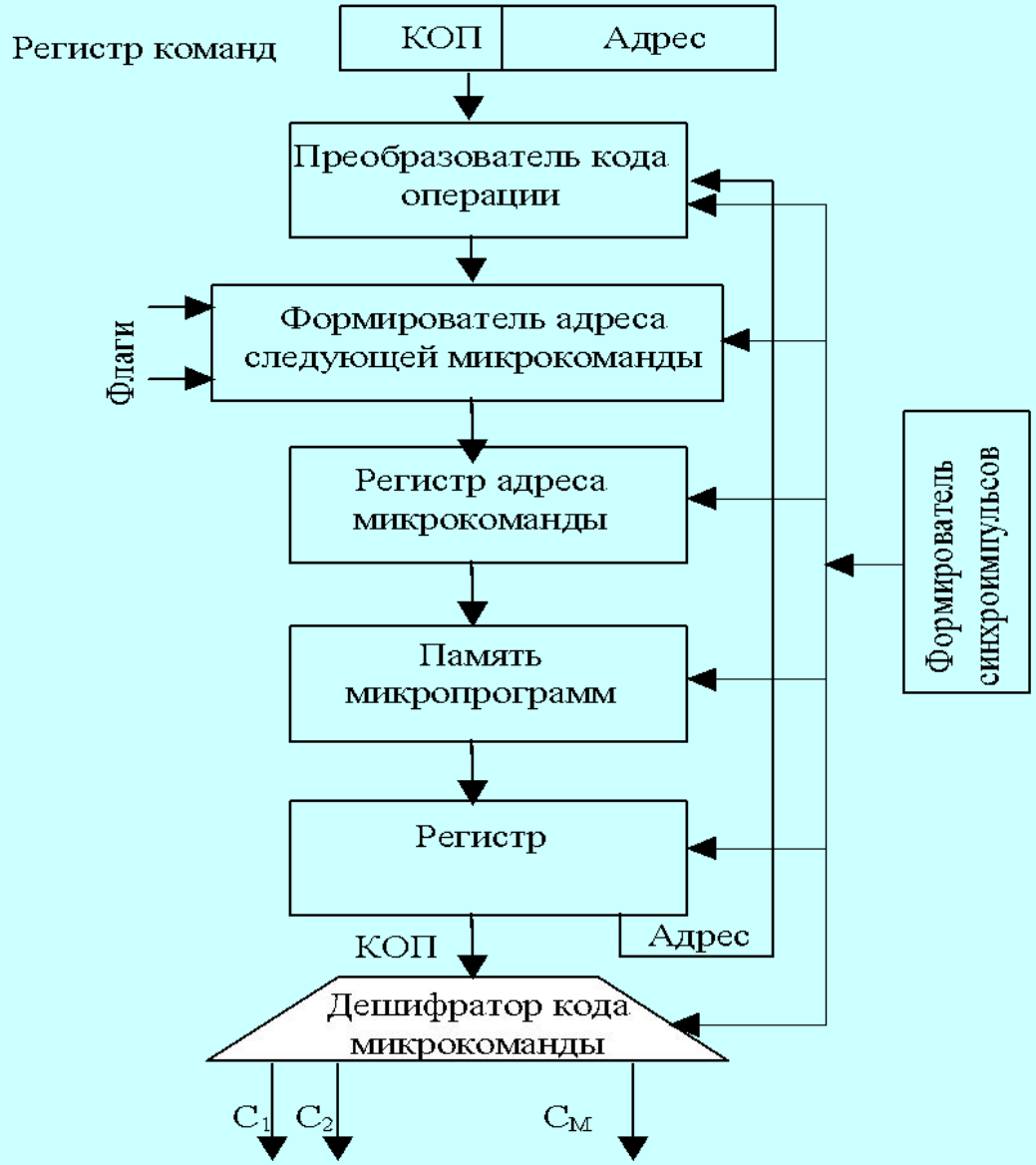
Микропрограммный автомат с жесткой логикой

Микропрограммный автомат с программируемой логикой.

- Идею микропрограммирования сигналов управления предложил в 1951 г. Морис Уилкс (Кембриджский университет, Британия). ЭВМ стала иметь три уровня выполнения команд: между командами и сигналами управления появилась микропрограмма. Команда ЭВМ интерпретировалась в микропрограмму. Аппаратное обеспечение должно было выполнять только микропрограммы с ограниченным набором микрокоманд, отсюда существенно уменьшались аппаратные затраты.

- Отличительной особенностью является наличие памяти микропрограмм.
- Каждой команде соответствует микропрограмма.
- К 70-м годам идея о том, что написанная программа сначала должна интерпретироваться микропрограммами, стала преобладающей. Однако в современных процессорах, когда аппаратные затраты стали менее существенны, отказались от идеи микропрограммирования, так как она стала сдерживать рост производительности.

Микропрограммный автомат с программируемой логикой.



Пример процессора с тремя шинами и его микропрограммирования

Микрокоманда GATE

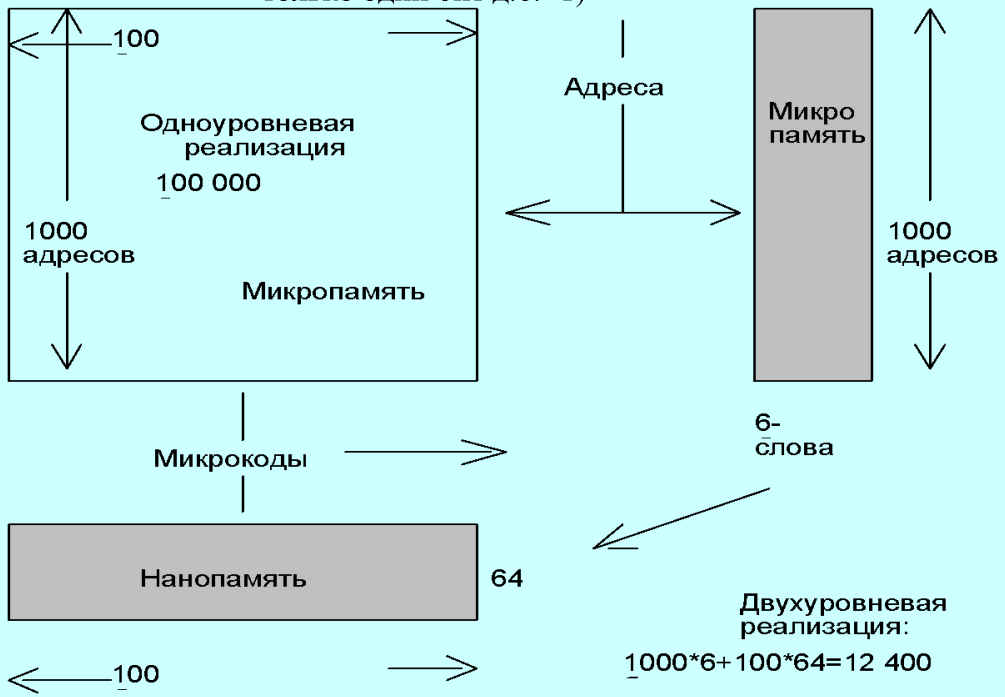
1	0	0	1	1	.....	1	1	1	0	0	1

Биты соответствуют стробам (вентилям) — Признак GATE

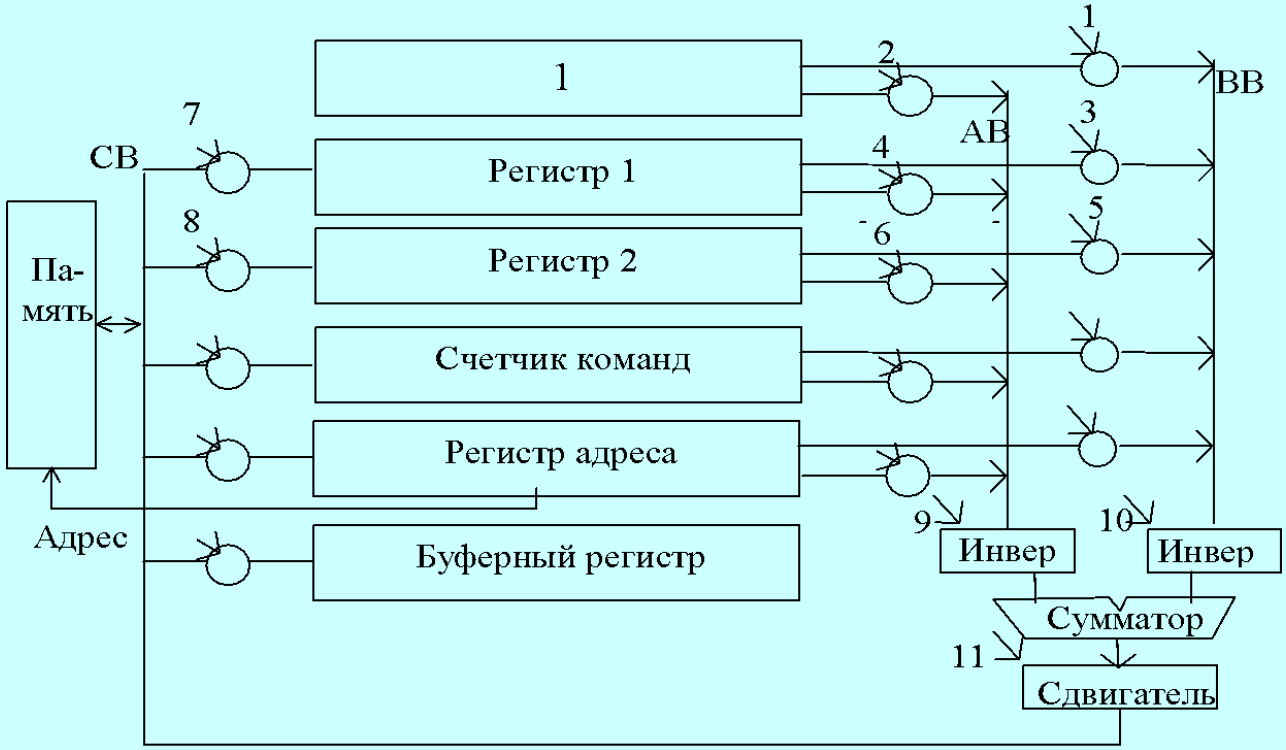
Микрокоманда TEST

Адрес	0/1	000...010...000	N регистра	0

На какой адрес микрокода перейти — В — Какой бит проверить на = В (только один бит д.б.=1) — В каком регистре — Признак TEST



Пример процессора с тремя шинами



Процессор с тремя внутренними шинами

	Вент1	Вент2	Вент3	Вент4	Вент5	Вент6	Вент7	Вент8	Вент9	Вент10
add1:	0	0	1	0	0	1	0	0	0	0
	0	0	0	0	0	0	1	0	0	0
add2:	0	0	1	0	0	1	0	0	0	0
	0	0	0	0	0	0	0	1	0	0
not r1:	1	0	0	1	0	0	0	0	1	1
	0	0	0	0	0	0	1	0	0	0

Команды

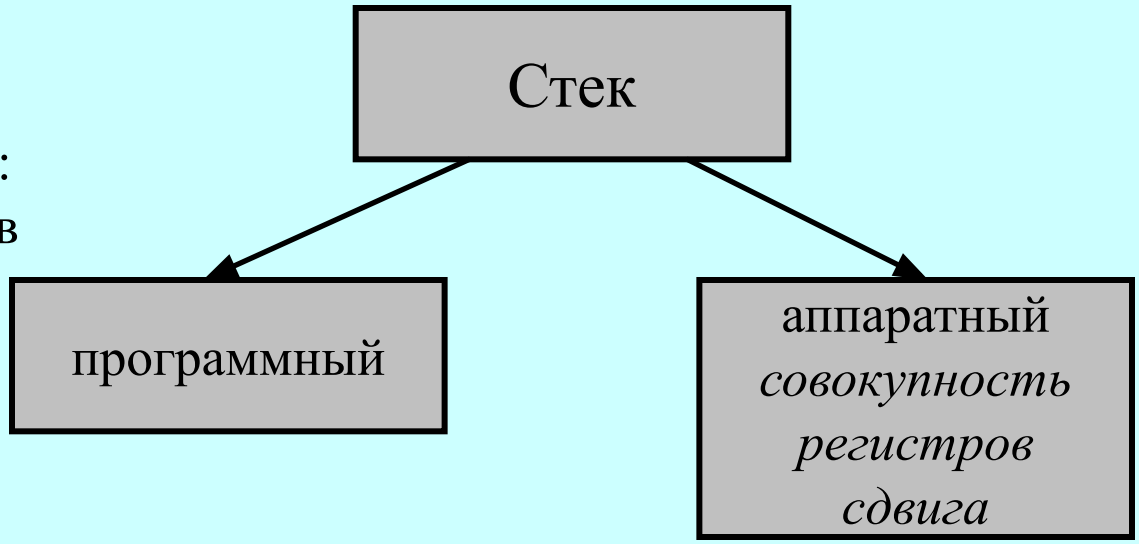
Важную роль в выборе архитектуры системы команд играет ответ на вопрос о том, где могут храниться операнды и каким образом к ним осуществляется доступ. С этих позиций различают следующие виды архитектур системы команд:

- **стековую;**
- **аккумуляторную;**
- **регистровую;**
- **с выделенным доступом к памяти.**

Стековая архитектура

Стек - память, состоящая из взаимосвязанных ячеек, взаимодействующих по принпипу “последним вошел, первым вышел” (LIFO, Last In First Out).

Для работы со стеком предусмотрены две операции: push (проталкивание данных в стек) и pop (выталкивание данных из стека). Запись возможна только в верхнюю ячейку стека.

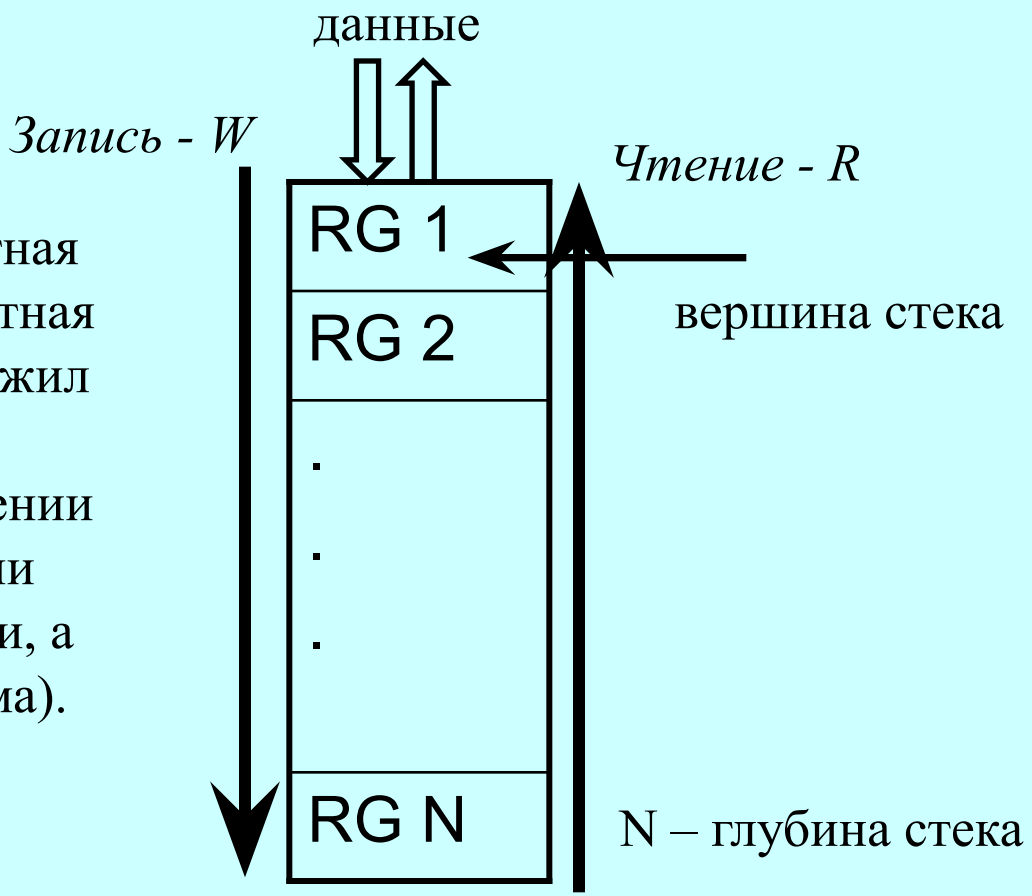


Механизм стеков снижает издержки на адресацию.

При стековой организации все данные идентифицируются вершиной стека.

Стековая архитектура

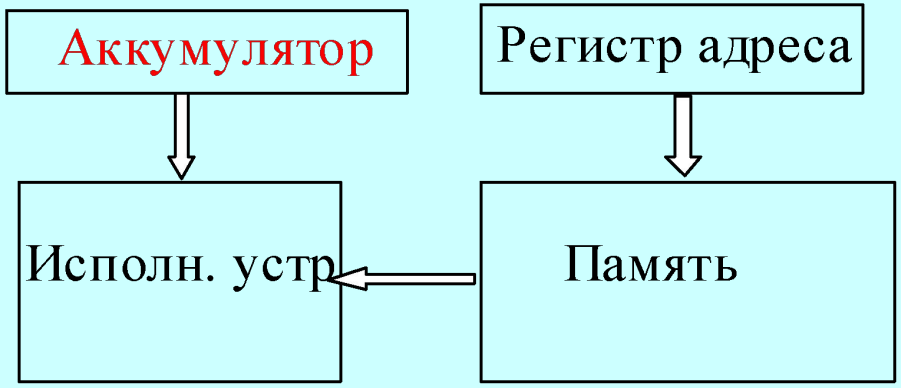
При описании вычислений с использованием стека обычно используется иная форма записи математических выражений, известная как обратная польская запись (обратная польская нотация), которую предложил польский математик Я. Лукашевич. Особенность ее в том, что в выражении отсутствуют скобки, а знак операции располагается не между операндами, а следует за ними (постфиксная форма). Последовательность операций определяется их приоритетами. Выражение *a =a+b + a x c* в постфиксной форме будет записано в виде: *a = a b + a c x+.*



Аккумуляторная архитектура

Аккумулятор - регистр для хранения результата и одного из операндов арифметической или логической операции в процессоре.

Для выполнения операции в АЛУ производится считывание одного из операндов из памяти в регистр данных. Второй операнд находится в аккумуляторе. Выходы регистра данных и аккумулятора подключаются к соответствующим входам АЛУ. По окончании предписанной операции результат с выхода АЛУ заносится в аккумулятор.



Аккумуляторная архитектура 2

Для загрузки в аккумулятор содержимого ячейки *x* предусмотрена команда загрузки *load x*.

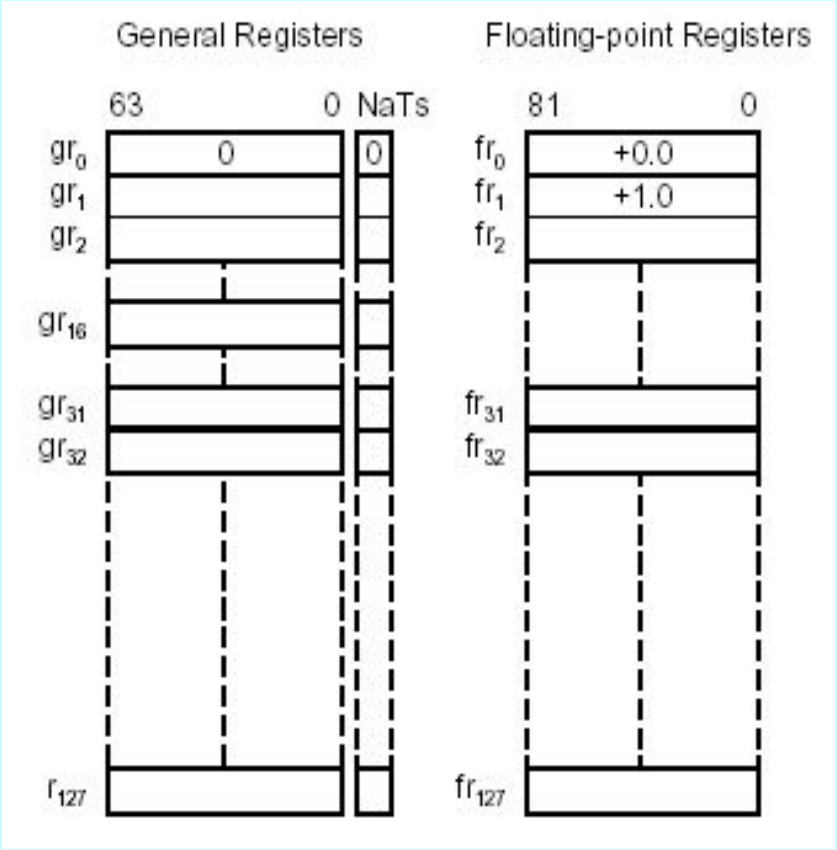
Запись содержимого аккумулятора в ячейку *x* осуществляется командой сохранения *store x*.

Достоинствами аккумуляторной АСК можно считать короткие команды и простоту декодирования команд. Однако наличие всего одного регистра порождает многократные обращения к основной памяти.

Регистровая архитектура

Процессор включает в себя массив регистров (регистровый файл), известных как регистры общего назначения (РОН). Эти регистры, в каком-то смысле, можно рассматривать как явно управляемый КЭШ для хранения недавно использовавшихся данных.

Размер регистров обычно фиксирован и совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер.



Регистровая архитектура

Количество РОН в архитектурах типа CISC обычно невелико. RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен).

Регистровая архитектура допускает расположение операндов в одной из двух запоминающих сред: основной памяти или регистрах. Выделяют три подвида команд обработки: регистр-регистр; регистр-память; память-память.

Достоинства регистровых АСК : компактность получаемого кода, высокую скорость вычислений за счет замены обращений к памяти на обращения к регистрам. С другой стороны, данная архитектура требует более длинных инструкций по сравнению с аккумуляторной архитектурой.

Архитектура с выделенным доступом к памяти

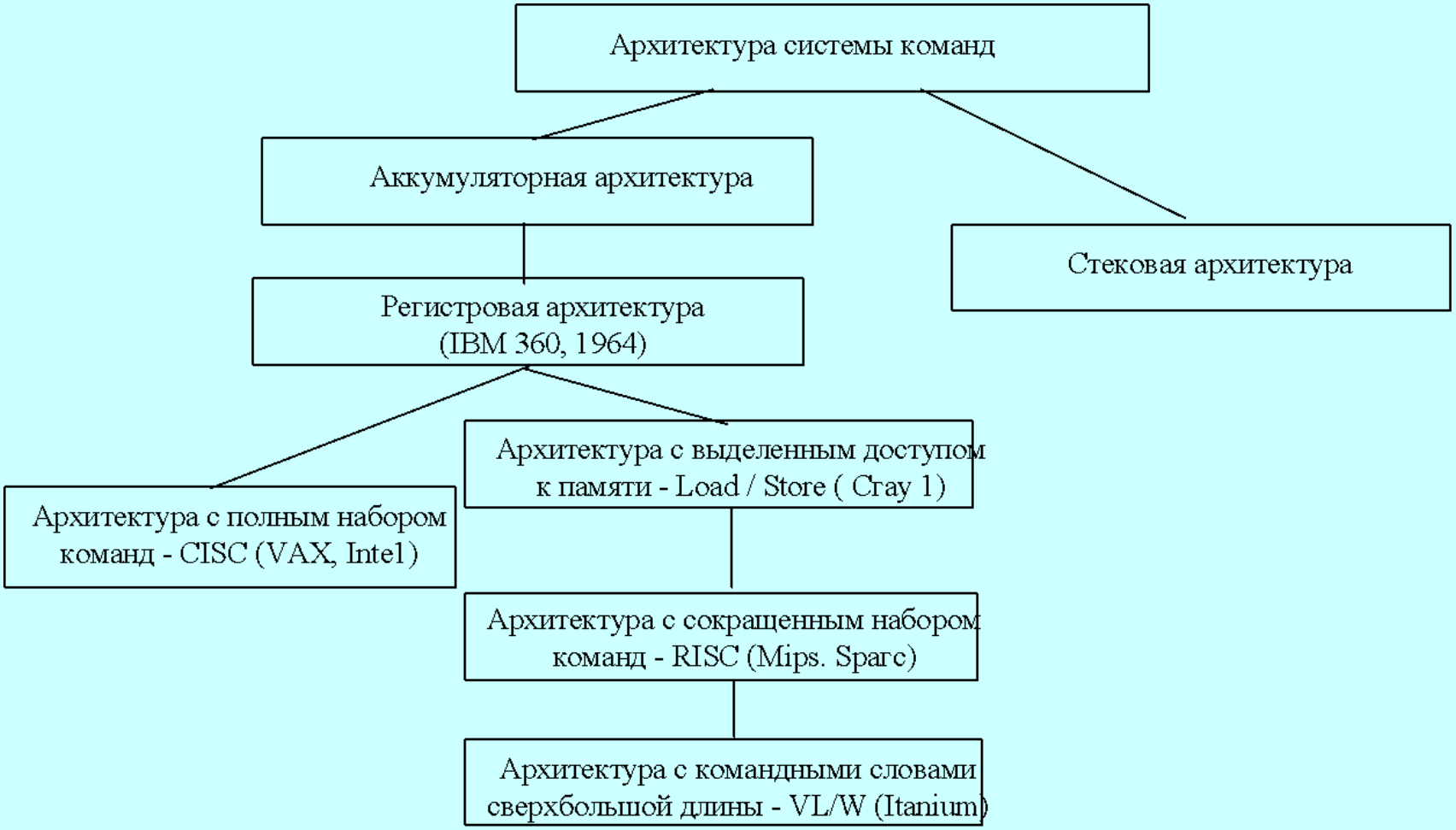
В архитектуре с выделенным доступом к памяти обращение к основной памяти возможно только с помощью двух специальных команд: *load* и *store*. По команде *load* информация считывается из ячейки памяти в регистр процессора. Запись из регистра в память происходит по команде *store*.

Операнды во всех командах обработки могут находиться только в регистрах процессора (в регистрах общего назначения). Результат операции также заносится в регистр.

В архитектуре отсутствуют команды обработки, допускающие прямое обращение к основной памяти.

АСК с выделенным доступом к памяти характерна для всех вычислительных машин с RISC-архитектурой. Команды в таких ЭВМ, как правило, имеют длину 32 бита и трехадресный формат. Примеры процессоров: Sun SPARC, MIPS R10000, DEC Alpha, PowerPC и др.

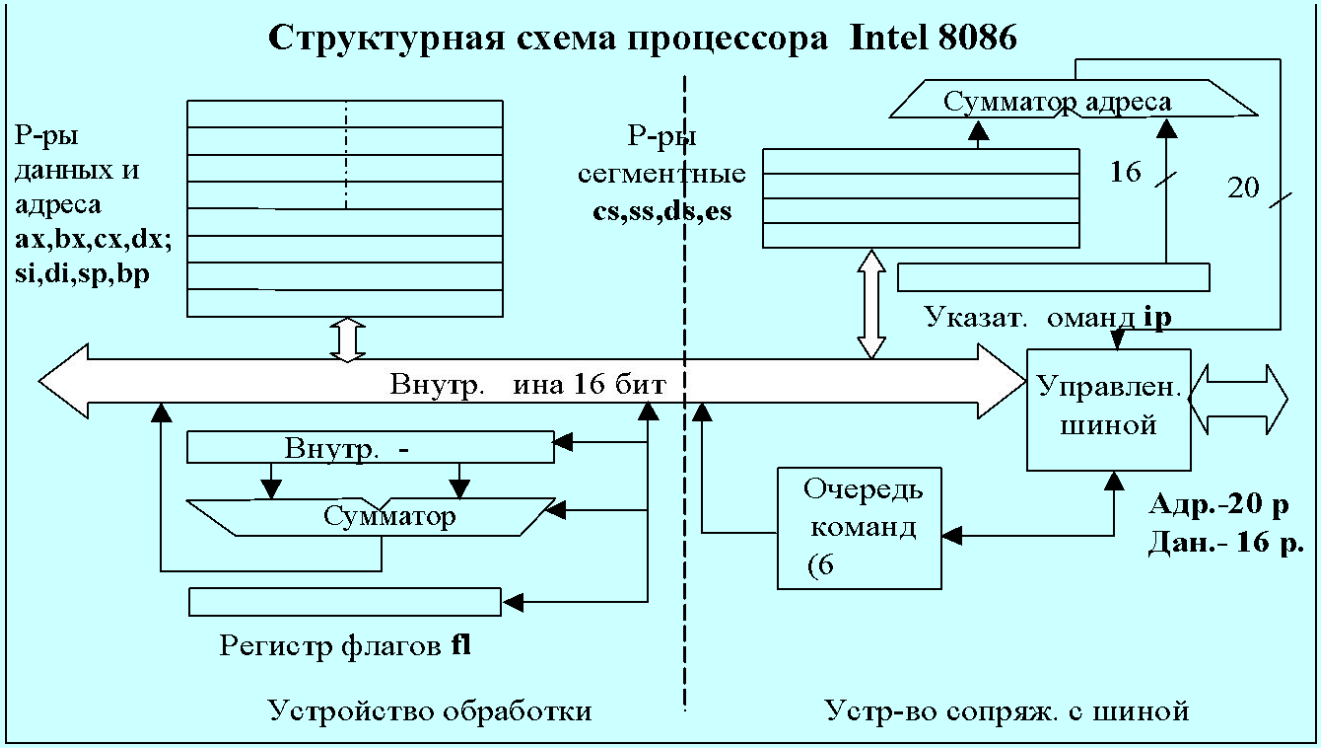
Команды


Архитектура системы команд
Аккумуляторная архитектура
Стековая архитектура
Регистровая архитектура
(IBM 360, 1964)
Архитектура с выделенным доступом
к памяти - Load / Store (Cray 1)
Архитектура с полным набором
команд - CISC (VAX, Intel)
Архитектура с сокращенным набором
команд - RISC (Mips. Sparc)
Архитектура с командными словами
сверхбольшой длины - VL/W (Itanium)

Конфигурируемые процессоры Xtensa фирмы Tensilica

Средство разработки Xenergy компании Tensilica является первым инструментом САПР, которое позволяет реально оценить влияние различных конфигураций процессора на суммарное потребление энергии. В то время как большинство инструментов разработки ориентировано на улучшение производительности и создание программного кода оптимального размера, Xenergy позволяет построить более эффективную с точки зрения потребляемой энергии конфигурацию подсистемы «процессор–память».

Xenergy может быть использован разработчиками для выполнения прикладных программ на процессорах Xtensa или Diamond Standard компании Tensilica с различными конфигурациями ядер, а также для оценки мощности, рассеиваемой процессором, кэш- и локальной памятью, связанной с ядром процессора. Разработчик может модифицировать конфигурацию процессора, добавить расширения команд, регистровые файлы или специализированные программные модули, а также оптимизировать прикладной код с целью снижения общей мощности, потребляемой процессором и памятью.

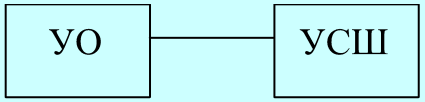


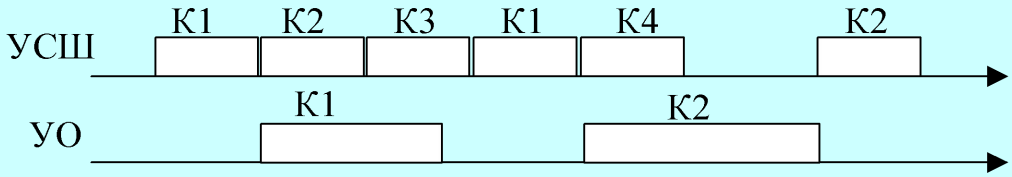
Регистр флагов процессора 8086

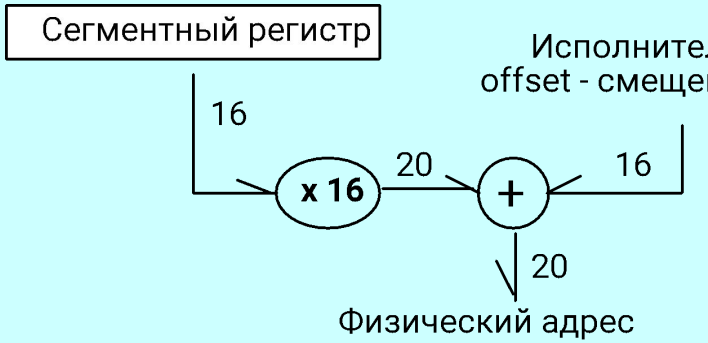
15				11	10	9	8	7					1	0
				OF	DF	IF	TF	SF	ZF	0	AF	0	PF	0	CF

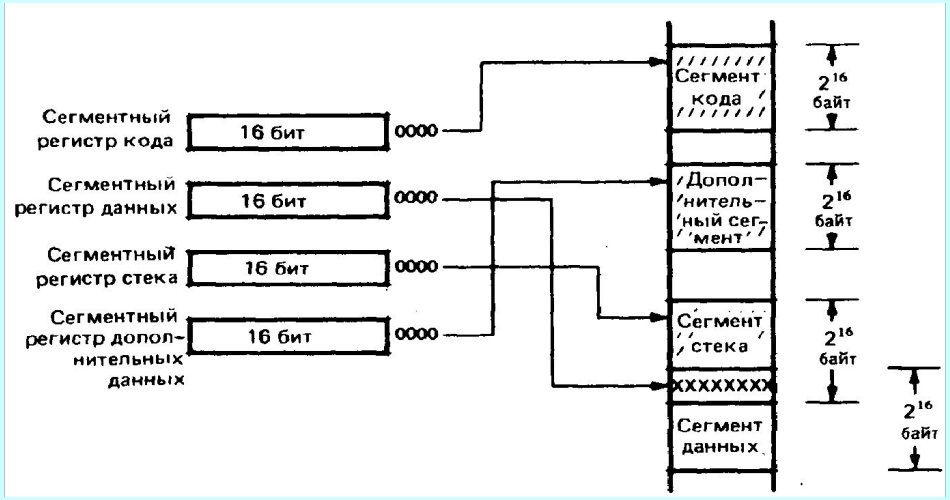
CF (Carry Flag) - флаг переноса;
PF (Parity Flag) - флаг четности;
AF (Auxiliary Carry Flag) - флаг вспомогательного переноса;
ZF (Zero Flag) - флаг нуля;
SF (Sign Flag) - флаг знака;
TF (Trap Flag) - флаг прерывания для отладки
IF (Interrupt-Enable Flag) - флаг разрешения прерывания;
DF (Direction Flag) - флаг направления цепочечных команд
OF (Overflow Flag) - флаг переполнения.

Конвейер. Организация памяти. Сегментация









cs - р-р сегмента кода
ds - р-р сегмента данных
ss - р-р сегмента стека
es - р-р сегмента дополн.
Полный адрес - ds:bx; 0xb800:0

Для операций с адресами используются команды:

int iA; int far *pA=&iA;

lea bx, iA; les bx,pA

Модели организации памяти:

Tiny, Small, Medium, Compact

Large, Huge

Форматы команд

1) 

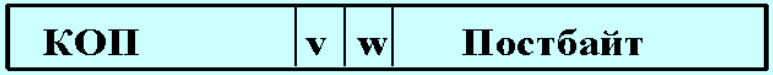
Операнд в регистр

2) 

3) 

4) 

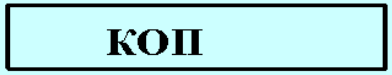
Кор. орм. прям. адресация

5) 

Сдвиги

6) 

Условн. ереход

7) 

Неявн. адес. и управл.

8) 

Ввод-вывод из порта

Особенности команд процессора i8086

Семейство процессоров x86 принято относить к классу процессоров CISC (Complicated Instruction Set Computer – с усложненным набором команд).

Общее количество команд в наборе:

целочисленные	- 210
плавающей точки	- 85
MMX	- 57
MMX Pentium III	- 16
XMM	- 70
Всего:	- 538

- Базовая структура основных команд обработки – двухоперандная, один из операндов всегда располагается в регистре процессора, второй может находиться в регистре либо в памяти, причем операнд-приемник может быть как в регистре, так и в памяти.
- Длина команды может лежать в пределах от 1 до 15 байтов
- Форматы кодирования усложнены, в них исключений не меньше, чем правил
- Используются сложные многокомпонентные способы адресации.
- Структуру регистровой модели, набора команд и способов адресации нельзя назвать вполне ортогональной: многие команды, регистры в сильной степени специализированы.
- Команды пересылки не изменяют содержимого признаков.

1. Команды пересылки.

1.1.Пересылки общего назначения MOV eax, [ebx]

1.2. Пересылки из/в стек: PUSH (втолкнуть в стек), POP (извлечь из стека). push ecx

Используют регистр (E)SP в качестве указателя стека. Указатель стека всегда указывает на последний занятый элемент стека.

1.3. Пересылки двоичных слов, представляющих собой адреса операндов или части (компоненты) адресов. Команды загрузки “длинных адресов”: LDS, LES, LFS, LGS, LSS (Load xxx Segment) . Команда загрузки исполнительного (эффективного) адреса LEA (Load Effective Address)

1.4. Строковые (блочные) пересылки rep movsb

1) Элемент данных из адреса ds:[e]si пересылается по адресу es:[e]di.
2) Оба адреса после этого автоматически увеличиваются (при df=0) либо уменьшаются (при df=1).
3) Содержимое [e]cx автоматически уменьшается на 1,
4) Если содержимое [e]cx не достигло нулевого значения, шаги 1)…3) повторяются.

MOVSB / MOVSW / MOVSD – позволяют осуществлять блочные пересылки строк (цепочек, блоков), содержащих байты, либо двухбайтовые, либо четырехбайтовые слова, из одного участка памяти в другой.
LODSB / LODSW / LODSD - команды загрузки из строки в регистр **al** / **ax** / **eax**. Эту команду имеет смысл использовать только без префикса повторения.
STOSB / STOSW / STOSD – команды сохранения в строку из регистра **al** / **ax** / **eax.** Использованная с префиксом повторения, эта команда позволяет заполнить участок памяти одинаковыми элементами.

1. Команды пересылки. (продолжение)

1.5. Команды условной пересылки CMOVcc – (начиная с процессоров PentiumPro) позволяют совместить в одной команде проверку условия и (не)выполнение действия – пересылка происходит только если выполнено условие, определяемое состоянием флагов. Эти команды относятся к “предикатным” командам, позволяющим реализовать условные конструкции без использования команд условного перехода, и тем самым, избежать возникновения в потоке команд (Control Dependencies),

1.6. Команды обмена BSWAP, XCHG – позволяют поменять порядок следования байтов в операнде, либо обменять содержимое операндов.

1.7. Большая группа специализированных команд для пересылки из/в системные регистры процессора (LGDT, LIDT, LLDT, RDMSR, SGDT, SIDT, SLDT,… и многие другие).

1.8. Пересылки между процессором и периферийными устройствами.

IN - ввести содержимое из порта периферийного устройства в регистр **AL** или **AX** или **EAX**)
OUT - вывести содержимое регистра **AL** или **AX** или **EAX** в порт периферийного устройства, Номер порта может быть указан в команде (в пределах 0…255), либо в регистре DX.

2. Команды обработки. Арифметические.

ADD целочисленное двоичное сложение операндов

ADC целочисленное двоичное сложение с учетом переноса в **cf**

SUB целочисленное двоичное вычитание операндов

SBB целочисленное двоичное вычитание с учетом заема в **cf**

CMP сравнение операндов путем вычитания, по результату устанавливаются флаги, после чего результат вычитания теряется

NEG, смена знака целочисленного операнда в дополнительном коде операнда

ASR,ASL, SAR,SAL арифметические сдвиги операнда (Arithmetic Shift to Right/Left) – эти команды эквивалентны делению/умножению на степень двойки, при сдвиге вправо знаковый бит операнда не изменяется.

INC, DEC увеличение или уменьшение операнда на 1 (эти команды не влияют на флаг **cf**)

MUL, DIV, IMUL, IDIV – умножение и деление без знаковых и знаковых чисел. Разрядность операнда-приемника для произведения в командах умножения и операнда-источника для делимого в командах деления вдвое больше разрядности операндов сомножителей частного и остатка.

Mul iAA (iAA*ax -> DX, AX или AH, AL)

2. Команды обработки. Логические.

OR - . Это команда побитовой установки (т.е. записи “единицы” в заданные биты операнда).

AND - . Это команда побитового сброса (записи в заданные биты “нулей”).

XOR - , eXclusive OR (иногда на русском эту операцию называют “ЛИБО”). Эта двухоперандная команда фактически выполняет выборочное инвертирование битов.
TEST – проверка битовых полей – команда изменяет значения флагов по результату проверки указанных битов в указанном операнде.

NOT операнда (замена значения каждого бита на противоположное).

BSF, BSR - сканирование битов (поиск первого единичного бита в операнде-источнике, с возвратом номера в операнде-приемнике). **BSF -** сканировать к старшему.

BT, BTC, BTR, BTS – команды проверки значения указанного бита в операнде-источнике (копируют значение указанного бита во флаг)

Практически все команды обработки прибодят к изменению состояний (хотя бы некоторых) флагов в регистре [E]FLAGS

2. Команды обработки. Сдвиги.

В процессорах 386+ реализовано 5 видов операции сдвигов, которые различаются тем, что происходит с битами, выходящими за пределы разрядной сетки с одного “конца” операнда, и с освобождающимися позициями на другом его “конце”. Количество сдвигов для каждого из видов может быть либо равно 1, либо задается содержимым регистра **cl**, либо непосредственной константой в команде.

ROR, ROL, RCR, RCL –циклические сдвиги (мнемоника образована от слова *rotate* – вращать). При циклическом сдвиге то, что выходит за границу разрядной сетки операнда, помещается в освобождающуюся позицию на другом конце операнда. В процессорах x86 существует две разновидности циклических сдвигов: во второй разновидности “вращение” операнда происходит с участием флага cf.

SAR,SAL - арифметические сдвиги (мнемоники – от слов *shift arithmetic*). Эта разновидность сдвига осуществляется таким образом, что результат оказывается эквивалентен умножению (при сдвиге влево) или делению (при сдвиге вправо) операнда на основание системы счисления, т.е. на 2. Для этого, при сдвиге вправо, старший (знаковый) разряд операнда сохраняет свое первоначальное значение.

SHL,SHR - логические сдвиги. При выполнении логических сдвигов биты, “выдвигаемые” из разрядной сетки, теряются, а противоположный конец операнда заполняется “нулями”.

SHLD, SHRD - сдвиг двойного слова.

2. Команды обработки. Сдвиги.

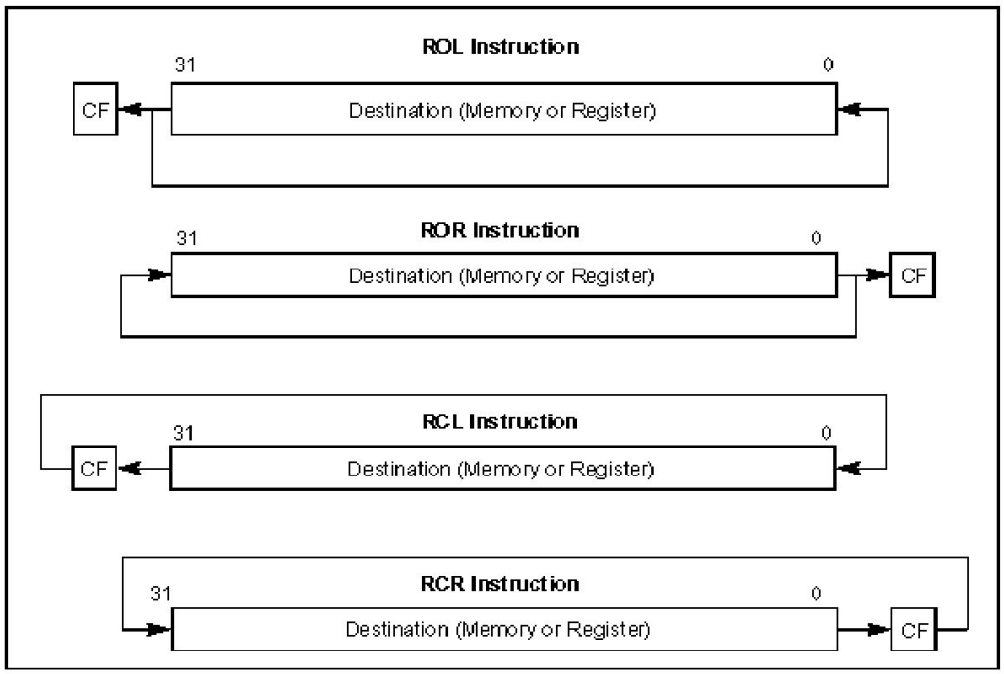


Figure 6-10. ROL, ROR, RCL, and RCR Instruction Operations

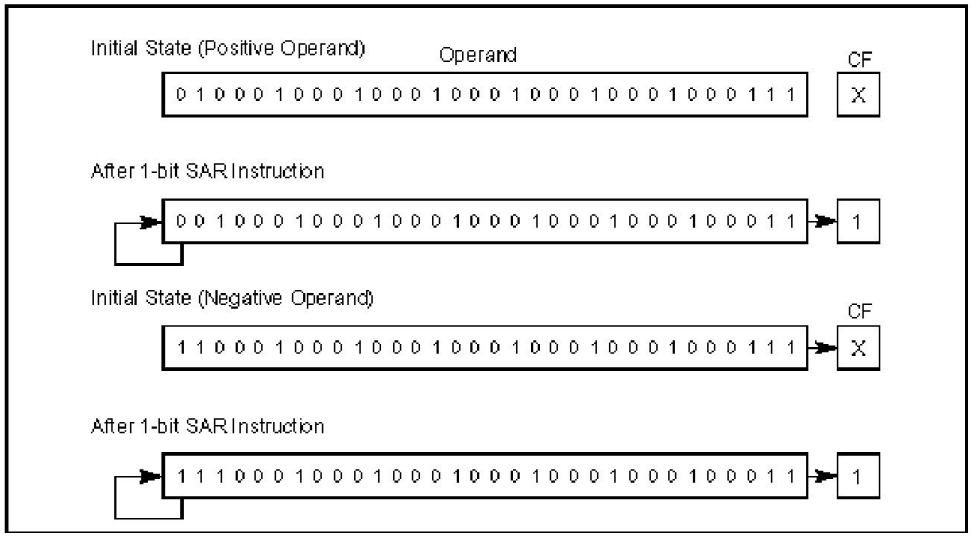


Figure 6-8. SAR Instruction Operation

3. Проверки и передача управления.

3.1. Команды проверки и сравнения

CMP - сравнение операндов путем вычитания. Результат вызывает изменение признаков (флагов), а затем теряется.

CMPS – сравнение строк и изменение флагов по результату сравнения.

SCAS – сканирование строки – сравнение элементов строки с содержимым регистра AL или AX или EAX и изменение флагов по результату сравнения.

3.2. Команды условного ветвления.
Jcc (jump conditional). Их в системе команд x86– 63 штуки.

Мнемонические обозначения команд условного ветвления содержат сокращения анализируемых условий. Для простых условий используются те же обозначения, что и для флагов: **jc** – ветвление, если флаг **cf** установлен, **jnz** – ветвление, если флаг **zf** не установлен. Для сложных условий - “больше”, “больше или равно” “меньше или равно”, “меньше” - используются сокращения: от слов *greater* – “больше” и *less* – меньше - при сравнениях целочисленных операндов со знаком

3.3. Команда организации цикла LOOP* позволяет организовать программный цикл с аппаратным счетчиком в регистре **[e]cx**. Затруднительно организовывать вложенные циклы.

3.4. Команда безусловной передачи управления JMP.

3.5. Команды поддержки модульной структуры программ

CALL вызов процедуры. **ENTER** формирование стекового кадра. **LEAVE**

освобождение стекового кадра **RET** возврат из процедуры

Формат двухоперандной команды

[Префикс] КОП [постбайт адресации] [смещение] [непоср.операнд]

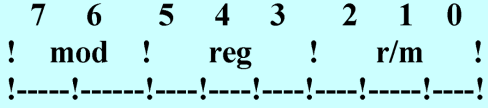
Префикс

Длина 1 байт. (Всего 5 префиксов для X86)

1.	Переназначения сегмента	**Fadd es:[bx],ax**	
2.	Повторения действия для строковых команд	**REP, REPZ, REPNZ, REPNE**	rep movsb
3.	Блокировки	**Lock**	lock rep movsb
4.	Размера адреса		
5.	Размера операнда		

КОП -

Постбайт адресации



Кодирование регистров полями reg и r/m при регистровой адресации (mod=11)

Табл.1

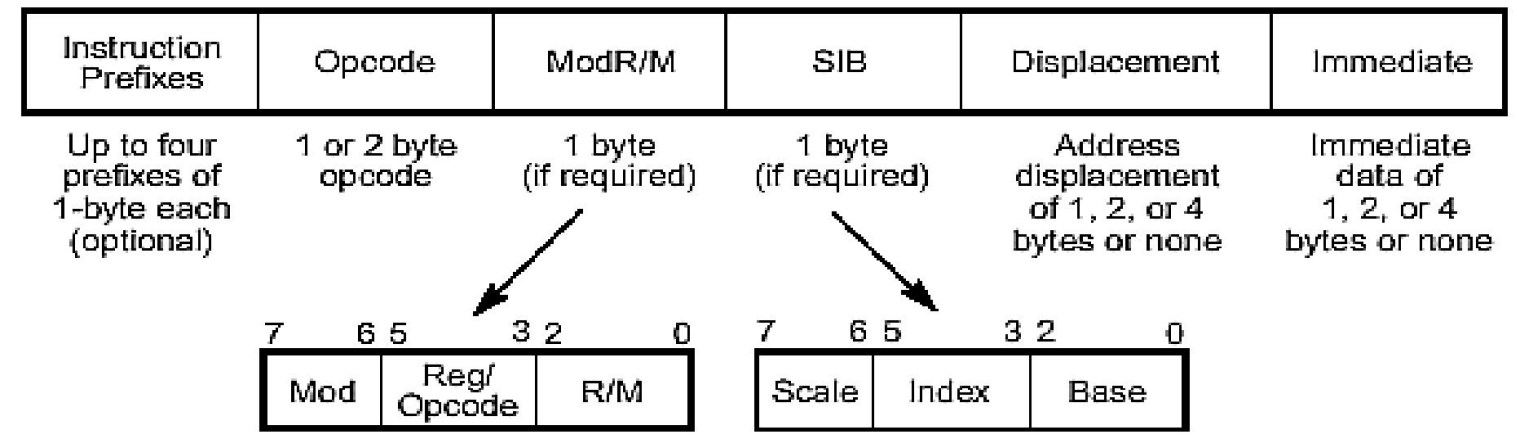
reg или r/m	Байт	Слово		
000	AL	AX		
001	CL	CX		
010	DL	DX		
011	BL	BX		
100	AH	SP		
101	CH	BP		
110	DH	SI		
111	BH	DI		

Кодирование способа вычисления адреса при адресации в память с использованием полей mod и r/m)

Табл.2

mod и r/m	mod=00	mod=01 или 10
000	BX+SI	BX+SI+ смещение
001	BX+DI	BX+DI+ смещение
010	BP+SI	BP+SI+ смещение
011	BP+DI	BP+DI+ смещение
100	SI	SI+ смещение
101	DI	DI+ смещение
110	direct	BP+ смещение
111	BX	BX+ смещение

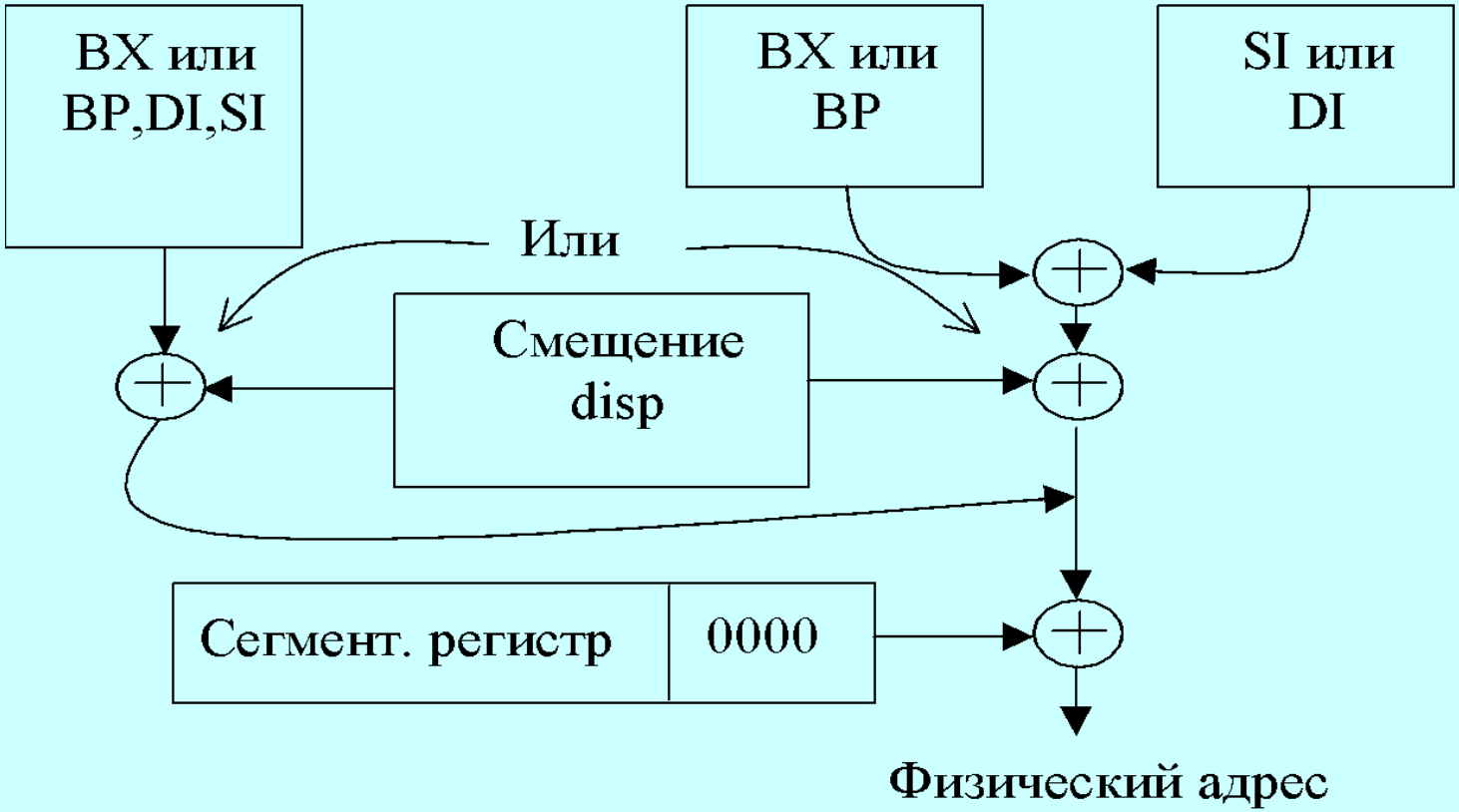
Полный формат двухоперандной команды



Add [eax+2*ebx],ecx

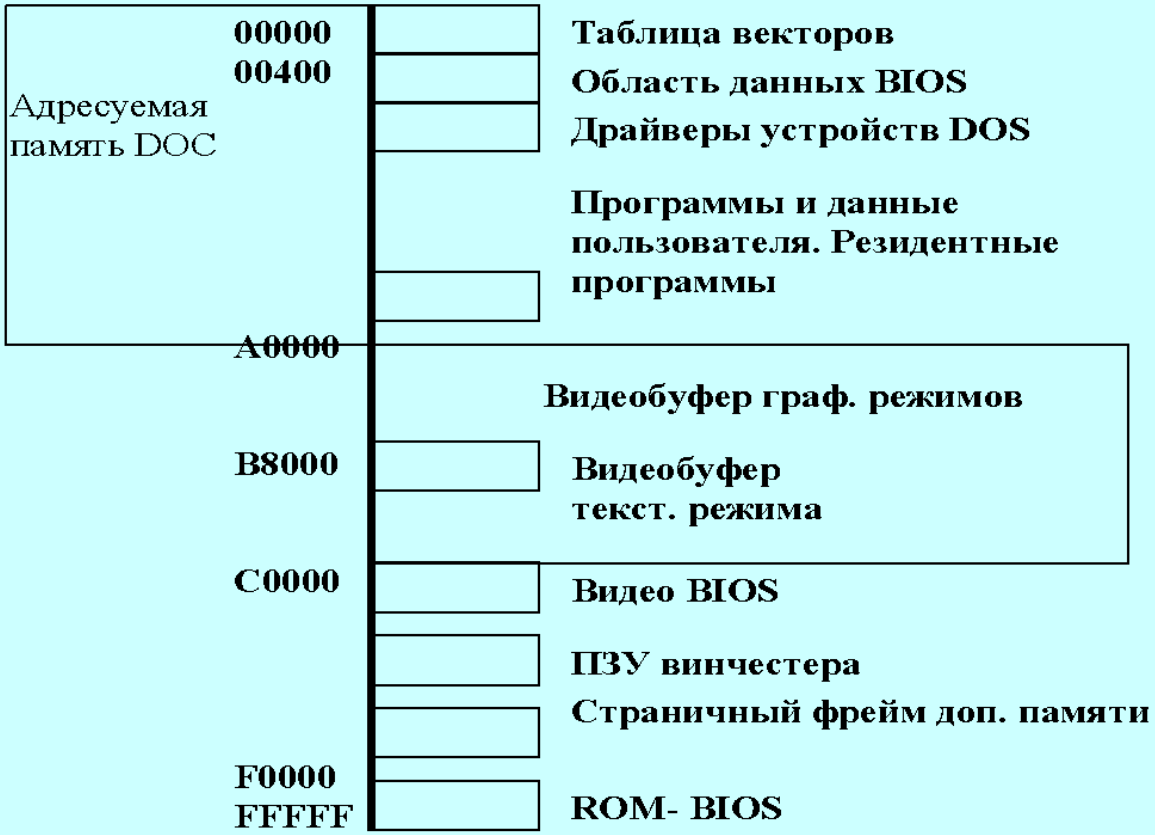
000000 01 00 001 100 01 011 000

Способы вычисления смещения (offset)



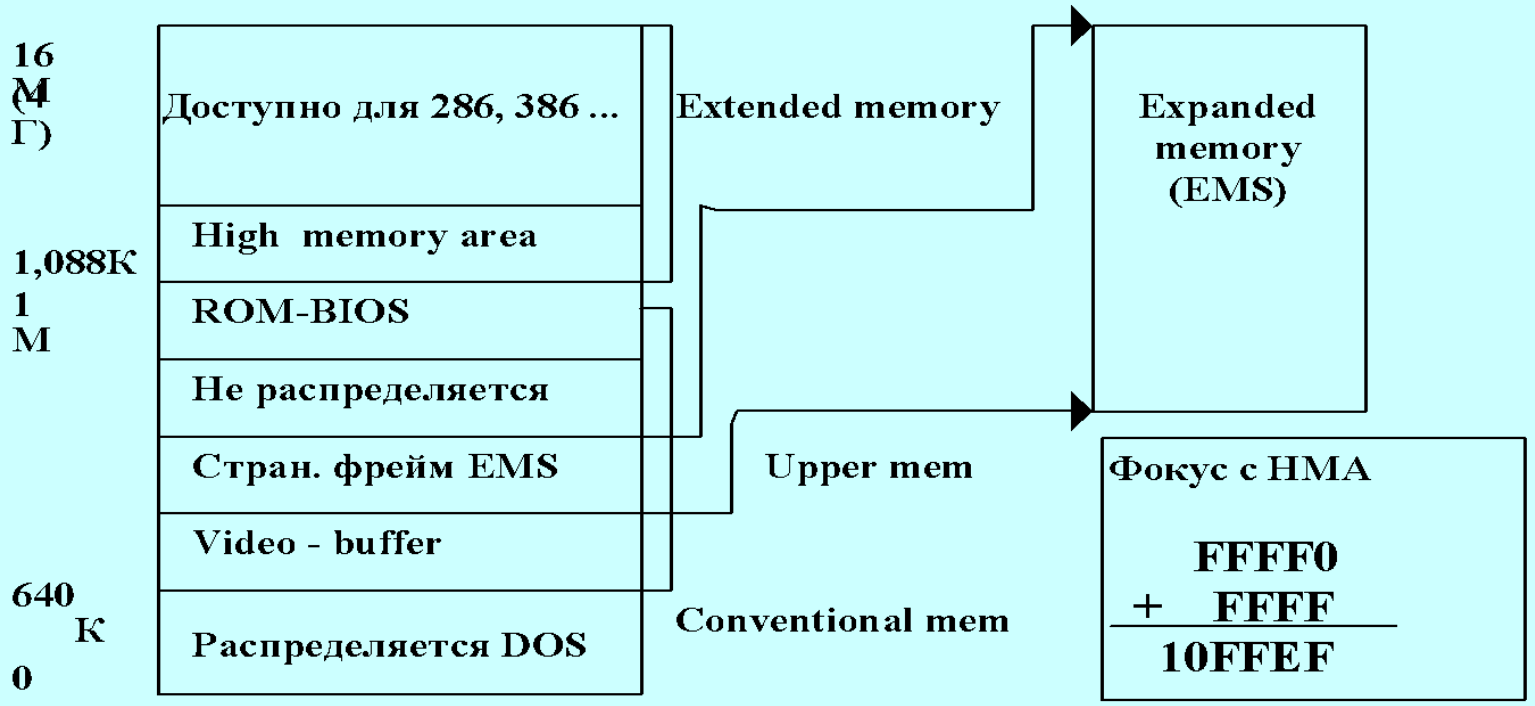
Адресное пространство процессора Intel 8086

( )



Распределение памяти в IBM PC

(



EMS - Expanded memory specification. Поддерживается EMM, QEMM (EMM386.EXE - интелектуальный менеджер памяти)-драйверами при использовании расширенной памяти (extended).

XMS - Extended memory specification - метод/ стандарт управления памятью за барьером 640 Кбайт. Поддерживается HIMEM.SYS – драйвером. (DOS=HIGH - загрузка DOS в “верхнюю” память)

UMB – Upper memory block (DOS=HIGH, UMB)

Shadow RAM - теневая память - предназначена для перезаписи данных из медленных ПЗУ (в быстрое параллельно расположенное ОЗУ).

Организация иерархической структуры программы

Два подхода к структурированию:

макроподстановки - подстановка текста фрагмента на этапе компиляции;

подпрограммы - связь по управлению, по данным, запоминание контекста, динамическое распределение (занятие и освобождение) памяти.

Основное средство структурирования (структурное программирование) на уровне системы команд процессора – это поддержка аппаратным уровнем *организации подпрограмм*.

Контекст программы

содержимое всех переменных элементов (регистров, ячеек памяти), которое требуется установить/восстановить, чтобы стало возможным запустить выполнение программы снова.

- счетчик команд,
- регистр состояния процессора (регистр флагов),
- содержимое остальных регистров процессора,
- указатели на стеки ядра ОС и пользователя,
- указатели на адресное пространство задачи (см. каталог таблиц страниц CR3 для x86),
- промежуточные результаты работы программы.

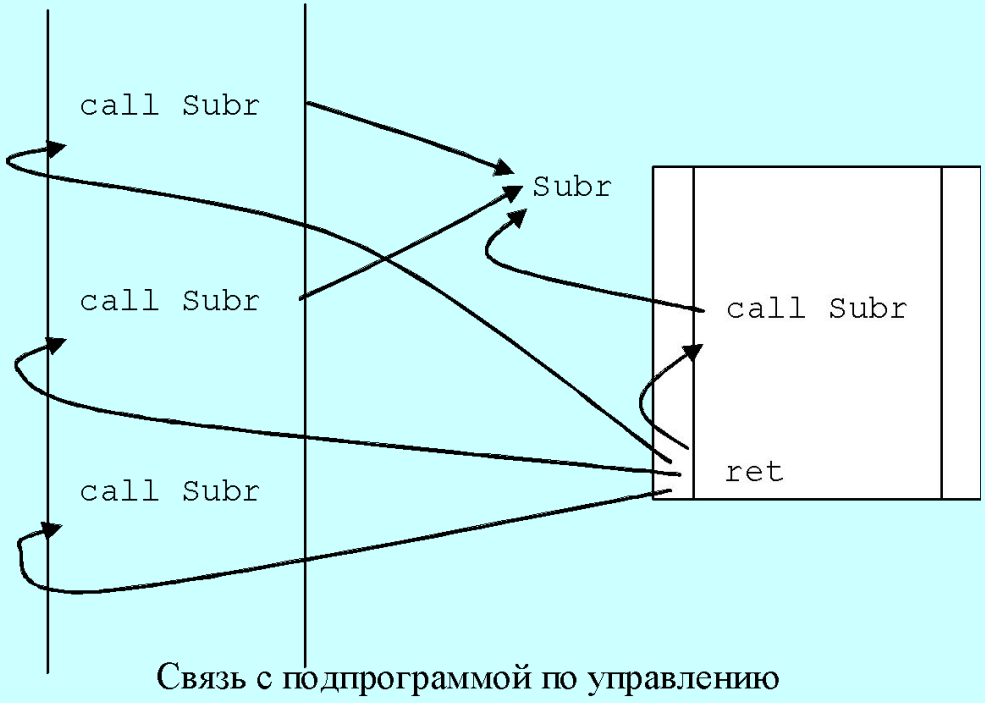
Для облегчения сохранения-восстановления контекста в разных реализациях процессоров могут быть использованы:

1) Специальные команды:
В ix86+ команды ***pusha, popa*** - они сохраняют в стеке - восстанавливают регистры данных и адресные в таком порядке: (e)ax, (e)cx, (e)dx, (e)bx, (e)sp, (e)bp, (e)si, (e)di. При этом значение (e)sp берется то, которое было до начала выполнения команды pusha.
2) Несколько экземпляров наборов регистров
TMS9900 - регистров данных и адресов не было вообще. В качестве регистров использовалось несколько ячеек памяти, положение которых в адресном пространстве указывал специальный регистр - указатель рабочей области (аналогично в **Transputer** фирмы Inmos).
Transputer (семейство процессоров фирмы Inmos) - регистров данных/адресов всего 3, и они образуют стек. Переход (и к подпрограмме) производится только, когда стек пуст, поэтому сохранять надо только адрес возврата.
IA-64 (Itanium) динамическое переименование регистров в регистровом файле при вызове процедуры (см. Intel IA-64 Architecture Software Developer's Manual, vol.2, IA-64 System Architecture, раздел 6 – IA-64 Register Stack Engine).

Обращение к подпрограммам - передача управления

Команда обращения к подпрограмме ОП (для этой команды используются мнемоники **call, jsr**). Эта команда должна автоматически запоминать адрес возврата (т.е. адрес команды, следующей за командой **call**).

Команда возврата из ПП (используются мнемоники **ret**, **rts**).



Обращение к подпрограммам - пример вложенности

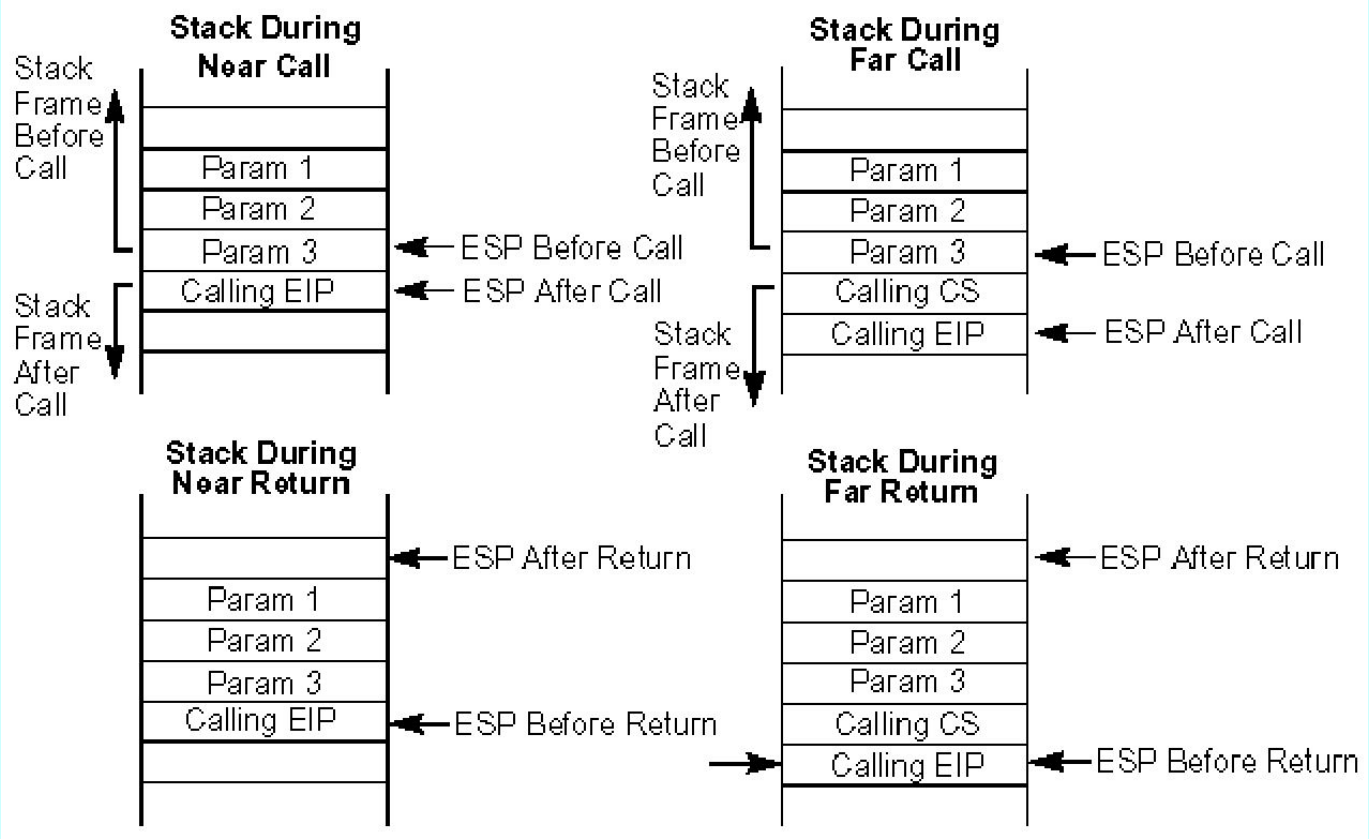
```
int x, y; float z;

void  proc3(int a, int b, int c) {
      - - - - - - - - - - - - - - - - - -
      }
- - - - - - - - - - - - - - - - - - - - - -
void  proc2(int k) {
      int j, q;
      - - - - - - - - - - - - - - - - - -
      proc3(j, 5, q);
      }
- - - - - - - - - - - - - - - - - - - - - -
void  proc1() {
      int i;
      - - - - - - - - - - - - - - - - - -
      proc2(i);
      }
- - - - - - - - - - - - - - - - - - - - - -
main () {
      proc1();
      }
```

Обращение к подпрограммам - сохранение/восстановление контекста

Минимальный набор действий, которые выполняет такая команда call:

- сохранение адреса возврата (т.е. адреса команды, следующей за **call**)
- загрузка в счетчик команд адреса первой исполняемой команды ПП.



Обращение к подпрограммам - обмен данными

При обращении к подпрограмме могут передаваться: сами данные; адрес участка памяти, где находятся данные. Данные могут находиться:
а) в регистре/ax, если данных мало или если в процессоре регистров много;
б) в стеке;
в) в оговоренном известном месте памяти (например в DEC16 - программная память, слова, следующие за командой вызова ПП)

```
void main ( void )
{int a,b;
int sum;
cout<<"\n ,Введите первое слагаем
cin>>a;
cout<<" ,Введите второе слагаемое:  ";
cin>>b;
sum=addition(a,b);
cout<<"a + b =  "<<sum;
getch();
```

Some statistics concerning procedure calls:

- Only 1.25% of called procedures have more than six parameters.
- Only 6.7% of called procedures have more than six local variables.
- Chains of nested procedure calls are usually short and only very seldom longer than 6.

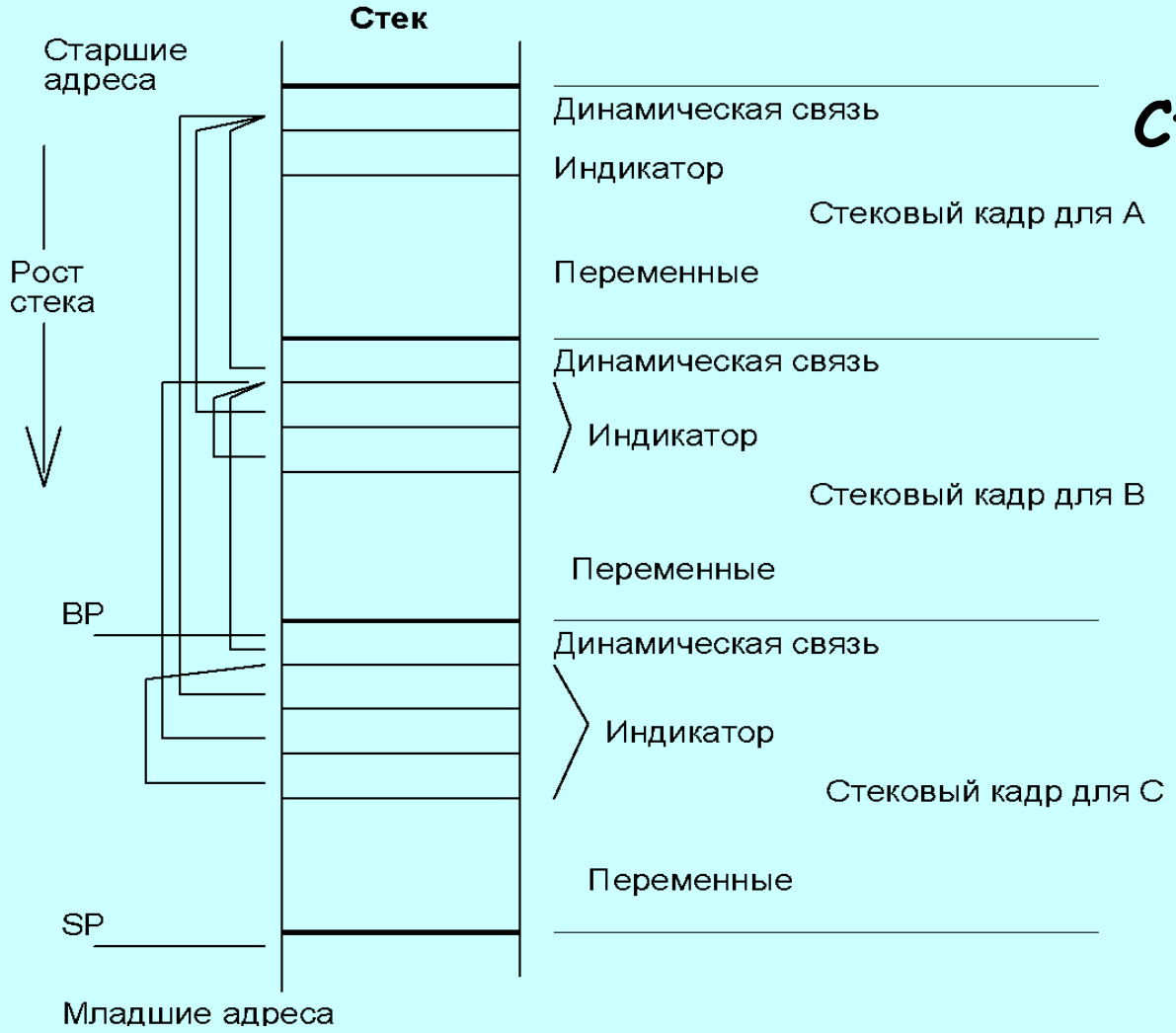
Результат дизассемблирования программы отладчиком TurboDebugger

Пример на Си:

```
int addition (int x, int y) {
        int iX, iY;
        iX=2*x+y;
        iY=3*y-x;
        return iX*iY;
}
```

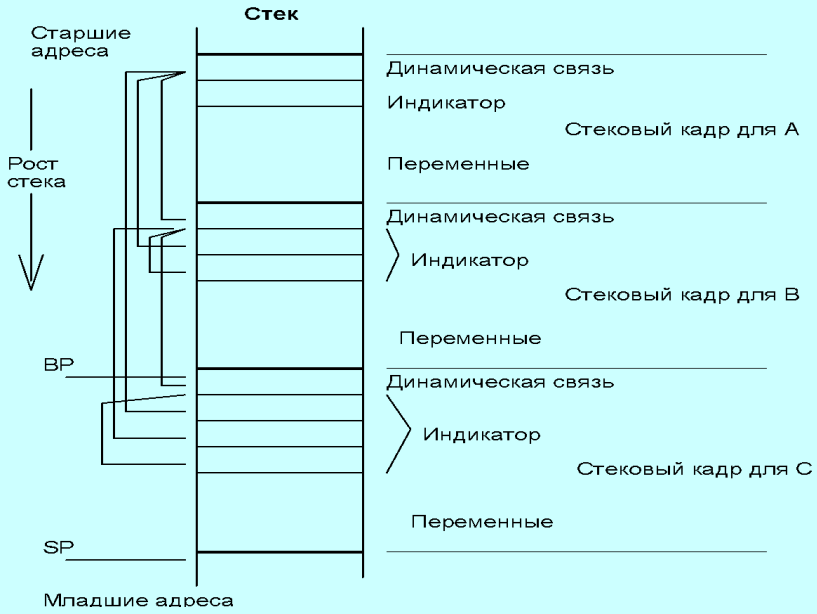
	AB1_3.: begin		
cs:0000 55	push	bp	
cs:0001 89E5	mov	bp,sp	Сохранение контекста В bp базовый адрес стекового кадра
cs:0003 83EC04	sub	sp,0004	
	AB1_3.15: iX:=2*x+y;		Выделение места под локальные переменные
cs:0006 8B460A	mov	ax,[bp+06]	Доступ к первому параметру
cs:0009 D1E0	shl	ax,1	
cs:000B 034608	add	ax,[bp+08]	Доступ ко второму параметру
cs:000E 8946FE	mov	[bp-02],ax	Доступ к локальной переменной
	AB1_3.16: iY:=3*y-x;		
cs:0011 8B4608	mov	ax,[bp+08]	Доступ ко второму параметру
cs:0014 8BF0	mov	si,ax	
cs:0016 D1E0	shl	ax,1	
cs:0018 01F0	add	ax,si	
cs:001A 2B460A	sub	ax,[bp+06]	Доступ к первому параметру
cs:001D□8946FC	mov	[bp-04],ax	Доступ к локальной переменной
	AB1_3.17: z:=iX*iY;		
cs:0020 8B46FE	mov	ax,[bp-02]	Доступ к локальной переменной
cs:0023 F766FC	imul	word ptr [bp-04]	Результат оставляем в AX
cs:0026 C47E04	les	di,[bp+0A]	Берет третий параметр – адрес результата
cs:0029 268905	mov	es:[di],ax	Запись результата процедуры
	AB1_3.18: end;		
cs:002C 89EC	mov	sp,bp	Освобождение локальных переменных
cs:002E 5D	pop	bp	
cs:002F C20800	ret	0008	Восстановление контекста Возврат из процедуры с освобождением ме с-та

Структура стекового кадра в x86



Структура стекового кадра в x86 включает три компонента:
- указатель на начало предыдущего стекового кадра (*динамическая связь*)
- таблица указателей на начала каждого из ранее созданных кадров (*индикатор*)
- локальные переменные.

Структура стекового кадра в x86. Команды ENTER и LEAVE



enter frmsiz, level

Первый параметр команды определяет количество байтов, требуемых в стеке для временных (локальных) переменных. Второй параметр показывает уровень "рекурсивности" (для самого внешнего уровня level=1). Для доступа к элементам стекового кадра используется адресный регистр BP, который указывает на начало (самый старший адрес) стекового кадра.

Выполнение команды ENTER:

1.TmpREG = SP ; Запомнить адрес начала следующего стекового кадра
2.BP -> в стек ; Создание динамической связи
3.Повторить (level-1) раз: ; Создание индикатора
BP -=2
[BP] -> в стек ; Копирование элементов индикатора из предыдущего кадра
4.TmpREG -> в стек ; Добавление в индикатор ссылки на данный кадр
5.BP = TmpREG ; Установить BP на начало создаваемого кадра
6.SP -= frmsiz ; Выделение в стеке блока под временные переменные