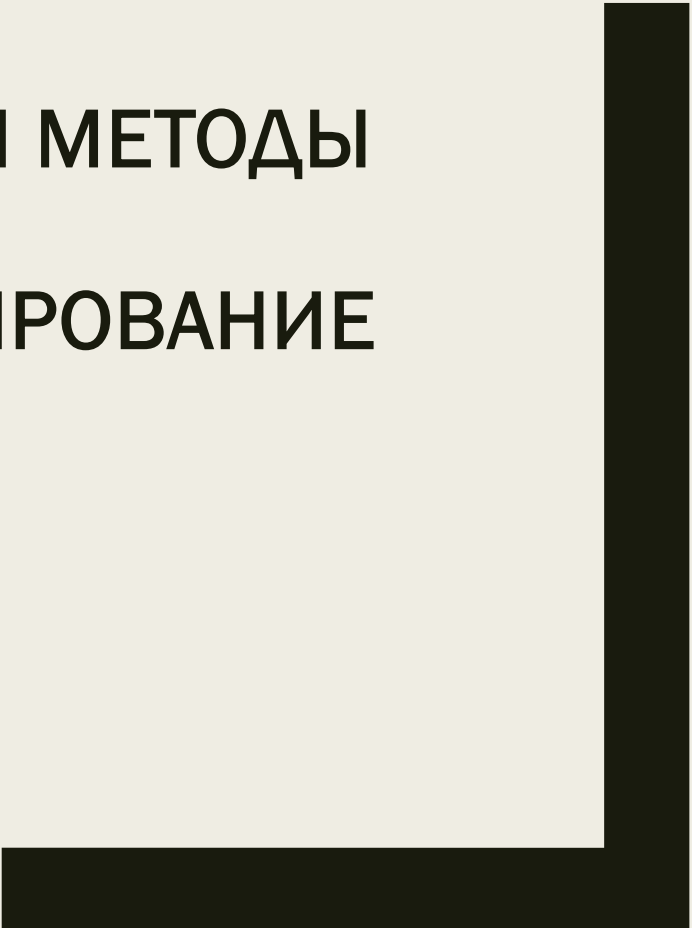
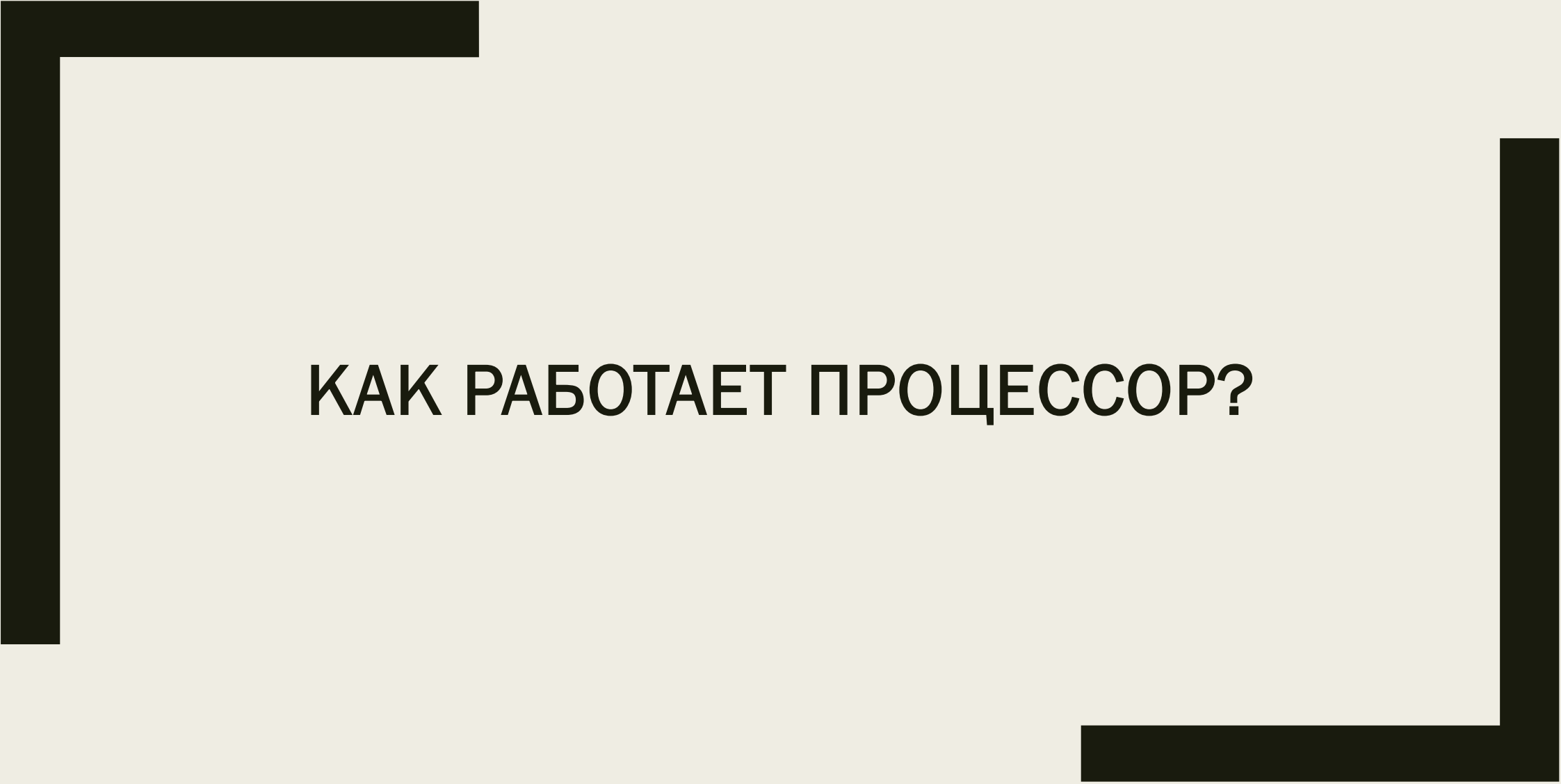




ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ/ НИЗКОУРОВНЕВОЕ ПРОГРАММИРОВАНИЕ

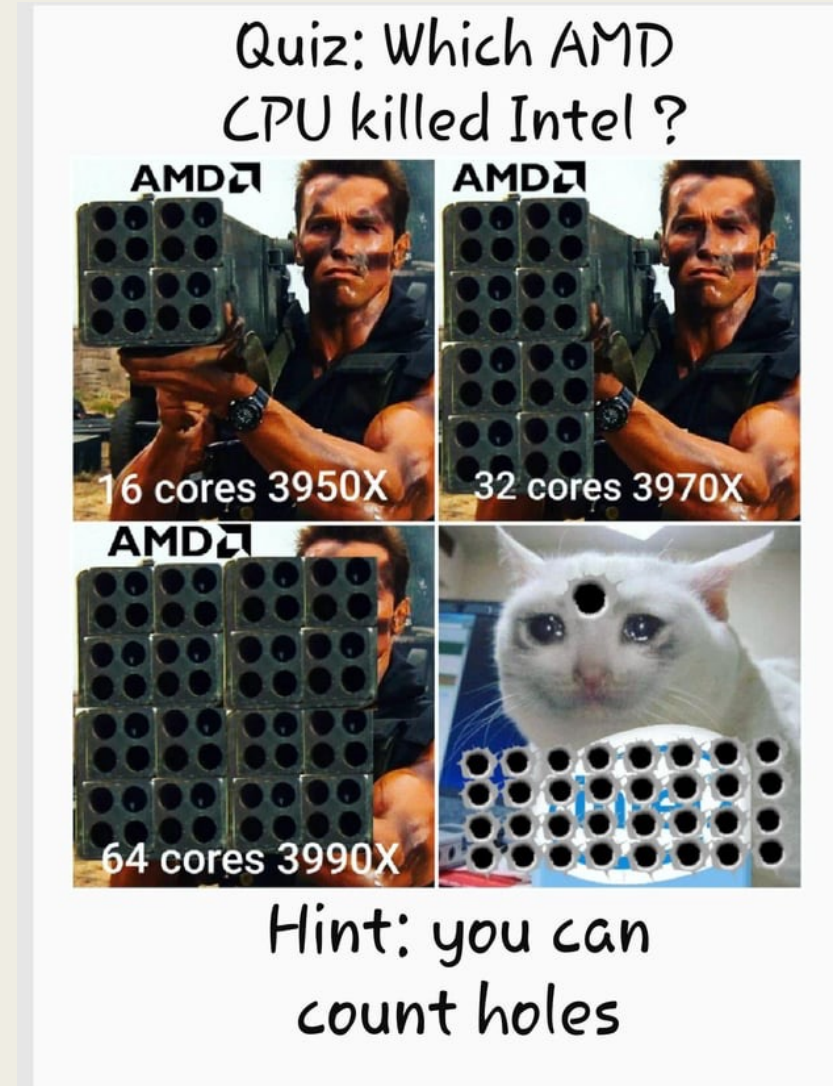
ЛЕКЦИЯ 2



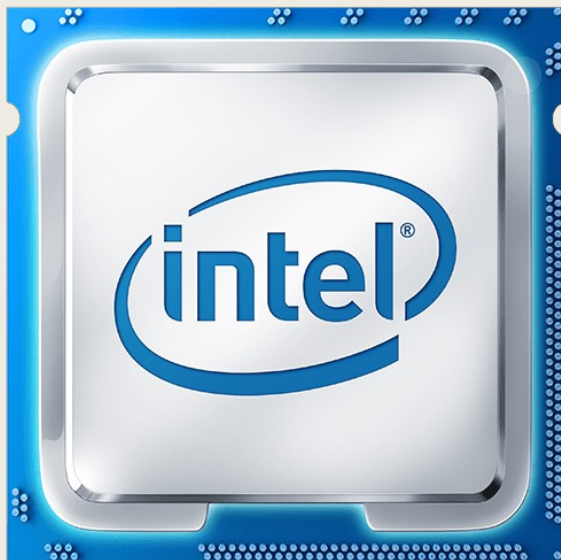
КАК РАБОТАЕТ ПРОЦЕССОР?

Хьюстон, у нас проблемы

- Многообразие языков программирования
- Обратная совместимость
- Необходимость стандартизации архитектуры



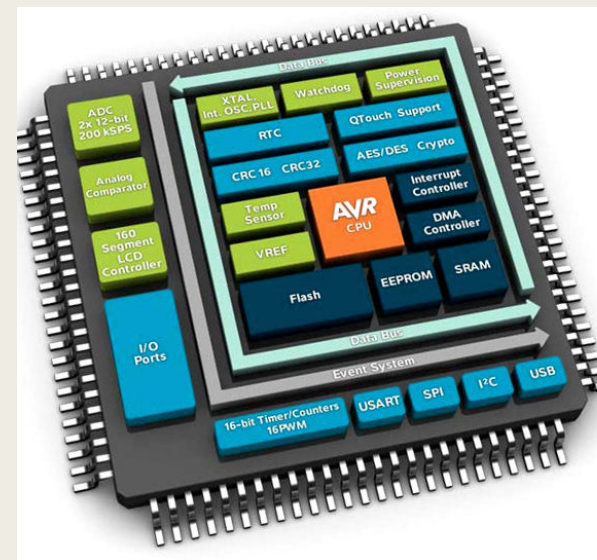
Виды архитектур



x86



ARM



AVR

Процессоры архитектуры x86

8086	80186	80286	80386	80486	...	CORE i7	...
------	-------	-------	-------	-------	-----	------------	-----

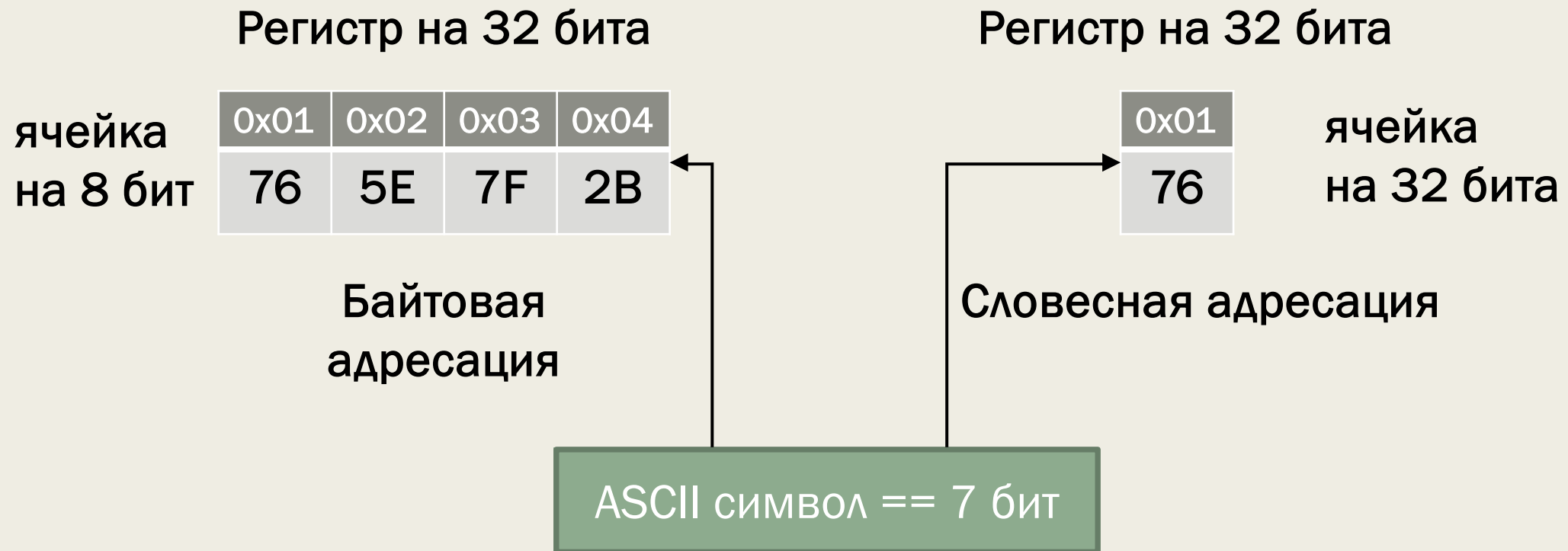
#	Модель/технология	Разрядность	Число (адрес ОЗУ)
1.	с 8086 по 80286	16 бит	от 0 до 65535
2.	с 80386 ... x86	32 бита	от 0 до 4294967295
3.	AMD64 == EM64T x86-64	64 бита	от 0 до 18446744073709551615

Регистры

- Размер должен быть кратен размеру ячейки оперативной памяти (8, 16, 32, 64 бита)
- Два вида адресации: байтовая и словесная



Почему байтовая адресация – это хорошо



И снова регистры

Названия регистров происходят от их назначения:

- EAX/AX/AH/AL (*accumulator register*) – аккумулятор;
- EBX/BX/BH/BL (*base register*) – регистр базы;
- ECX/CX/CH/CL (*counter register*) – счётчик;
- EDX/DX/DH/DL (*data register*) – регистр данных;
- ESI/SI (*source index register*) – индекс источника;
- EDI/DI (*destination index register*) – индекс приёмника (получателя);
- ESP/SP (*stack pointer register*) – регистр указателя стека;
- EBP/BP (*base pointer register*) – регистр указателя базы кадра стека.

Переполнение регистра

-128 ... 127

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

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

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

-126

$$120 + 10 = 120 + 7 + 3$$

$$120 + 7 = 127$$

$$127 + 1 = -128$$

$$-128 + 2 = -126$$

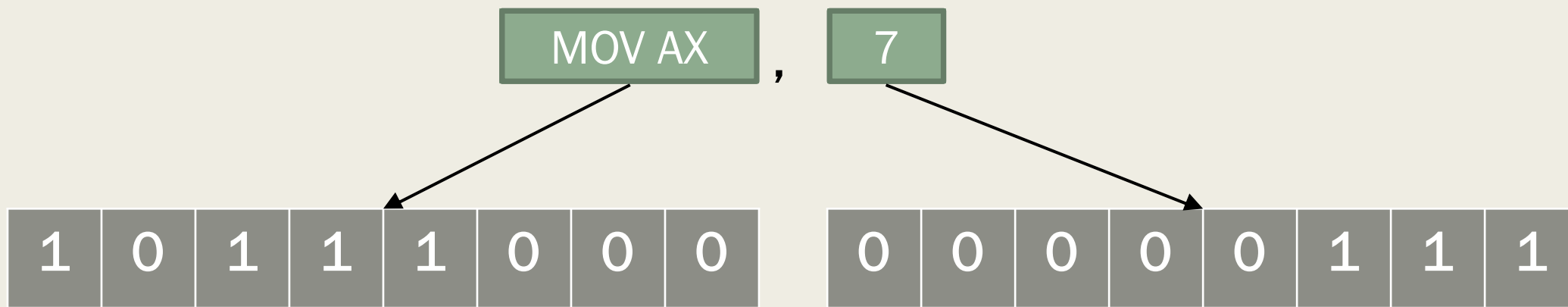
MOV AH, 120

MOV AL, 10

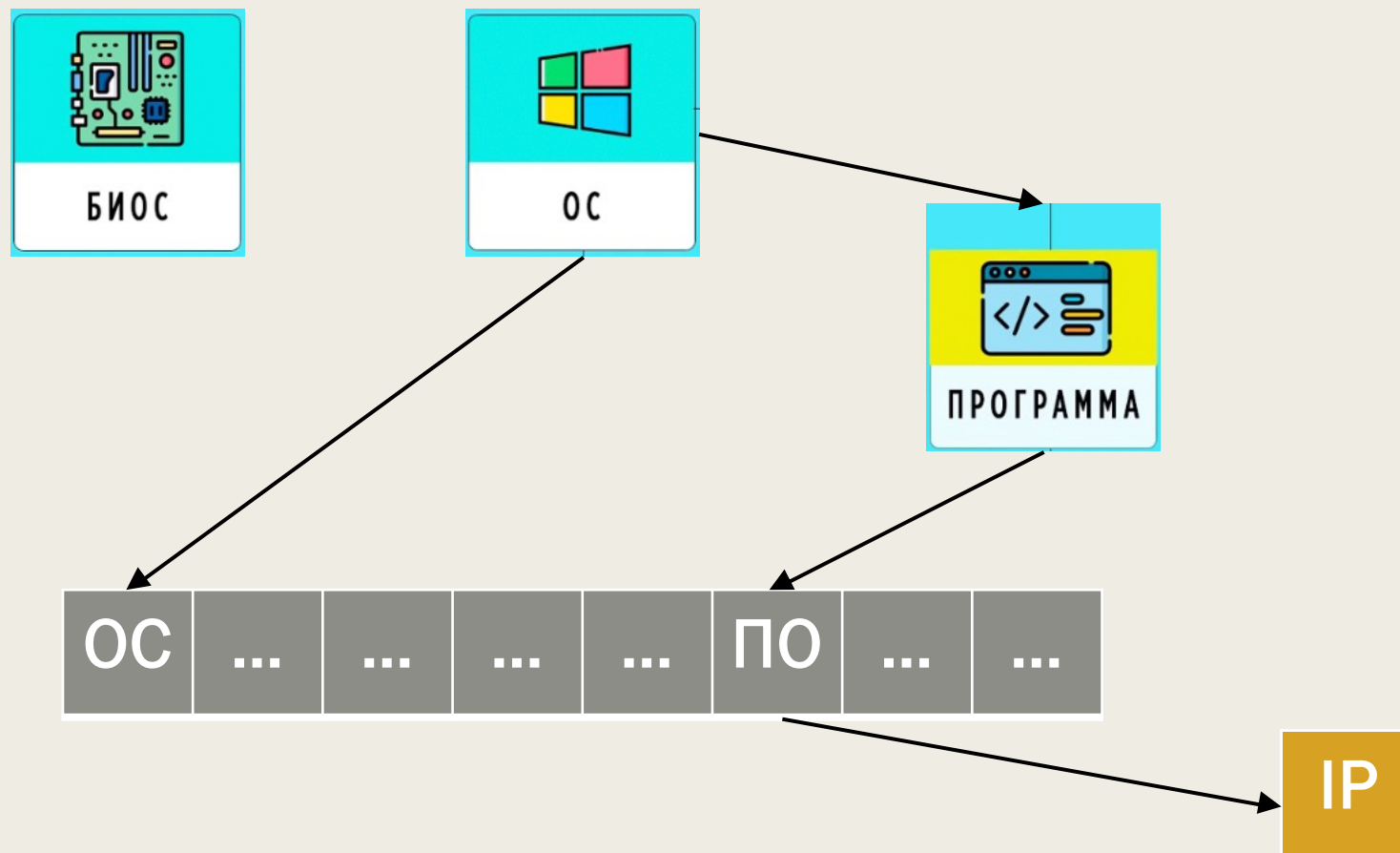
ADD AH, AL

Опкоды

VAR X = 7;
MOV AX, 7



Указатель на следующую команду



Режимы работы процессора

- Режим реальных адресов
- Защищенный режим
- 64-разрядный режим

Защищенный режим

- Данные операционной системы защищены от прикладных программ
- Данные прикладных программ защищены от других прикладных программ
- Используются разные уровни привилегий

Уровни привилегий



Адресация в защищенном режиме

4 ГБ

Виртуальная
память



трансляция виртуального
адреса в физический

...	0x05	0x06	...	0x51	0x52	...	0xA1	0xA2	...
...	2B	8A	...	22	1C	...	E3	45	...

сегмент

16 бит

32 бита

Селектор дескриптора : смещение в сегменте

Уровень привилегий

Таблица

Индекс

#	Начальный адрес сегмента	Уровень привилегий	Размер сегмента
1			
...	...	...	...

адрес начала сегмента + смещение в сегменте
= виртуальный адрес

Страничная организация памяти

Селектор	Дескриптор	Виртуальный адрес
----------	------------	-------------------

1. Индекс в каталоге страниц

2. Индекс в таблице страниц

3. Смещение в странице

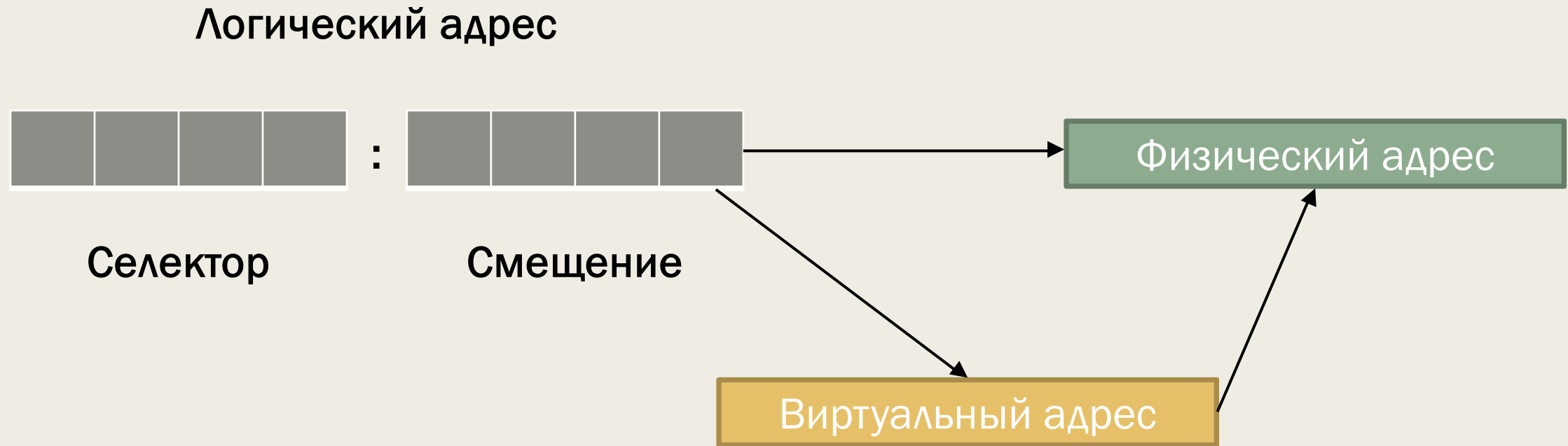
...	0x0 5	0x0 6	...	0x5 1	0x5 2	...	0xA 1	0xA 2	...
...	2B	8A	...	22	1C	...	E3	45	...
страница			страница			страница			

Таблица страниц	
#1	Страница
#2	Страница
#3	Страница

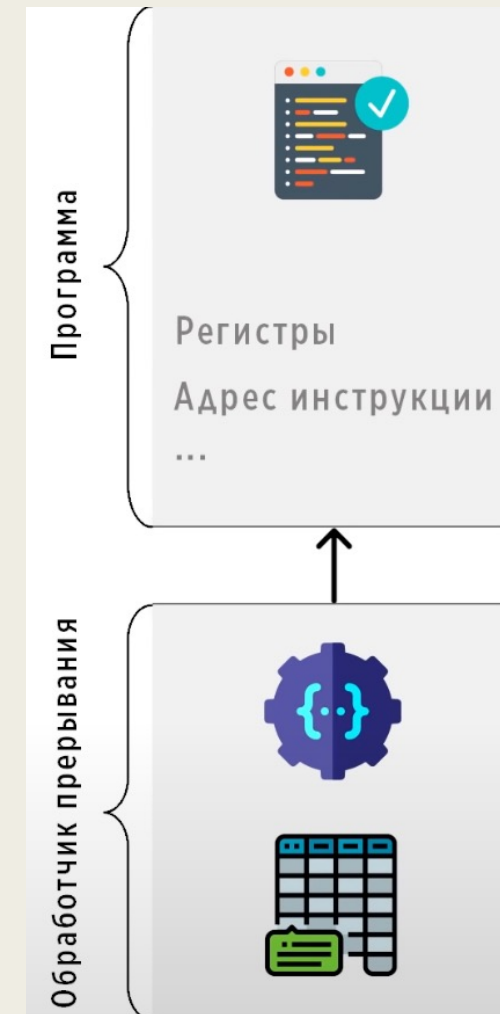
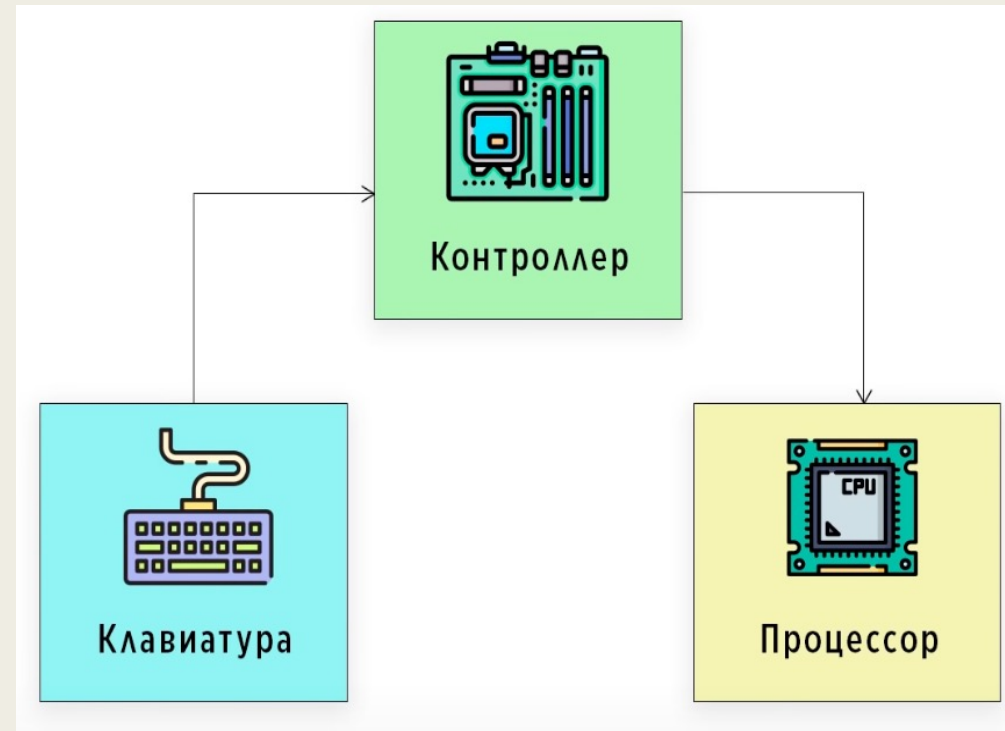
Таблица страниц	
#1	Страница
#2	Страница
#3	Страница

Каталог страниц	
#1	Таблица
#2	Таблица
#3	Таблица

Резюмируем про адресацию

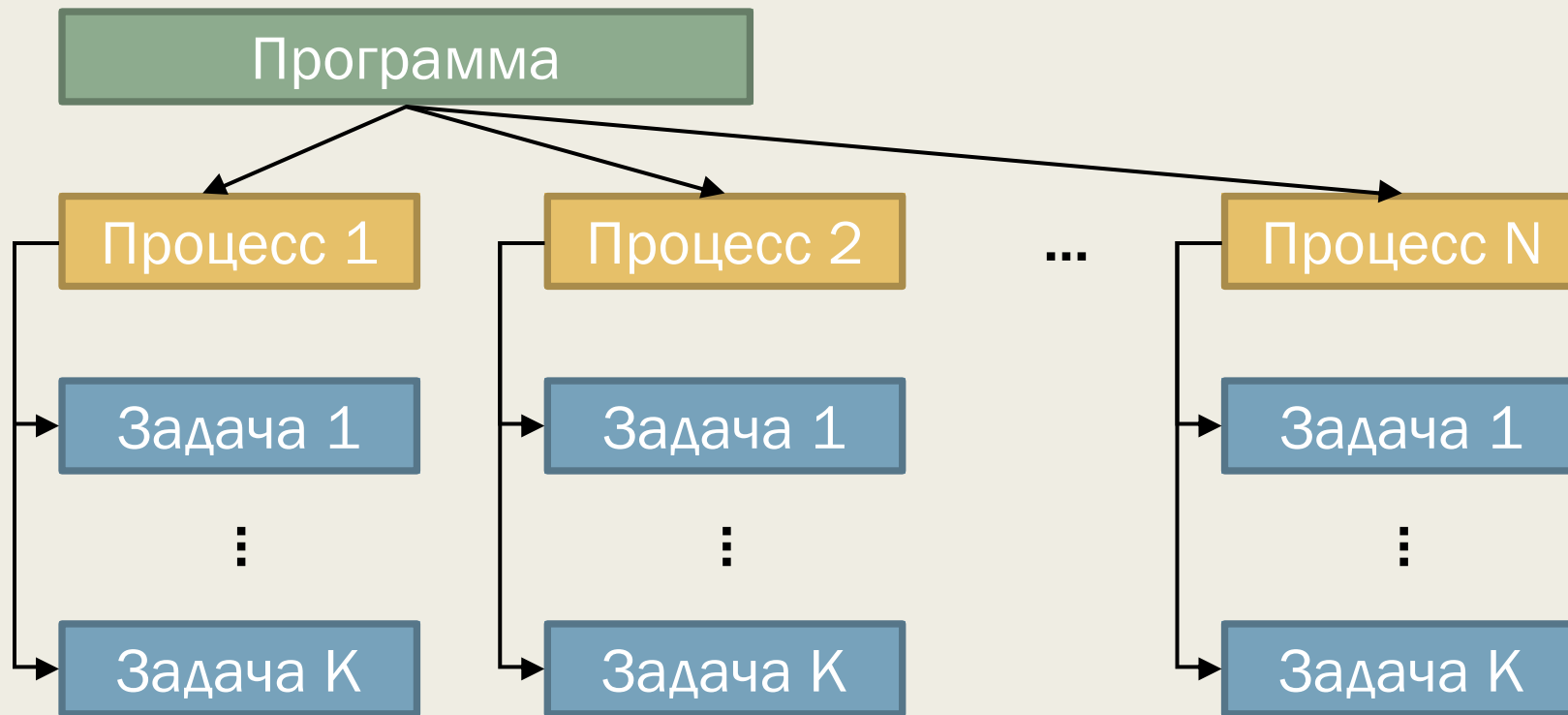


Прерывания

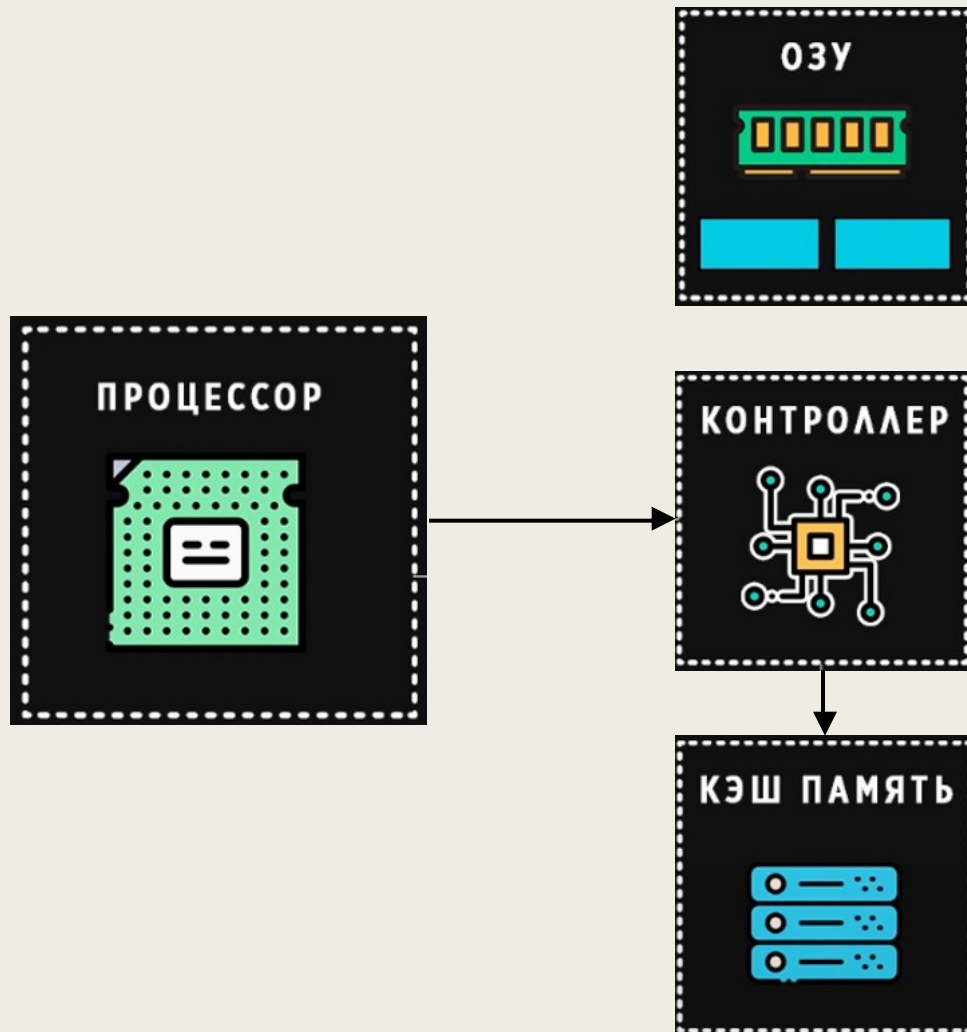


Прерывание – сигнал процессору о том, чтобы он прервал выполнение текущей программы и передал управление специальной функции-обработчику прерывания.

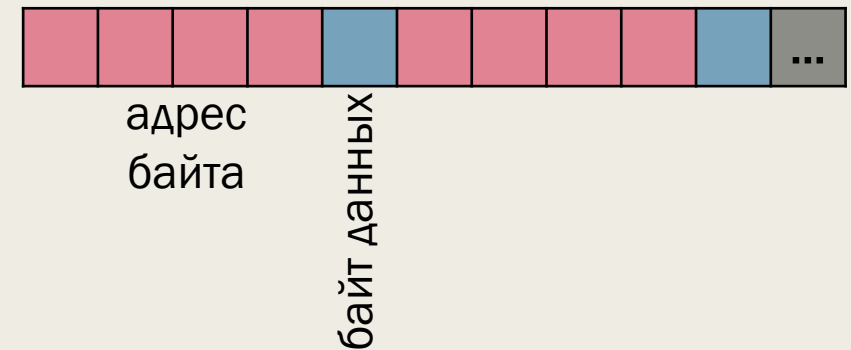
Многозадачность и многопроцессорность



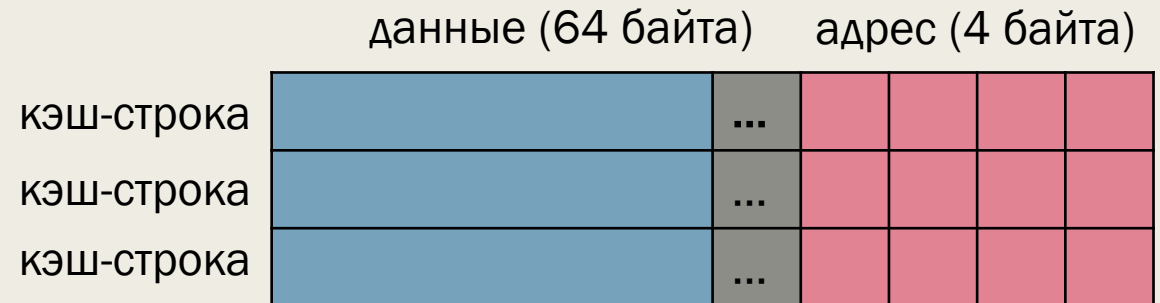
Как работает кэш процессора



Плохой вариант хранения данных в кэше:



Нормальный вариант хранения данных в кэше:



Размер кэш-памяти

- Должен быть кратен степени числа 2
- Полный размер кэша = полезный размер кэша (размер данных) + размер служебных данных

- **Пример:**

Есть кэш размером 128 байт, состоящий из 4 строк по 32 байта.

Полный размер такого кэша будет $32 * 4 + 4 * 4 = 144$ байта.

Кэш прямого отображения

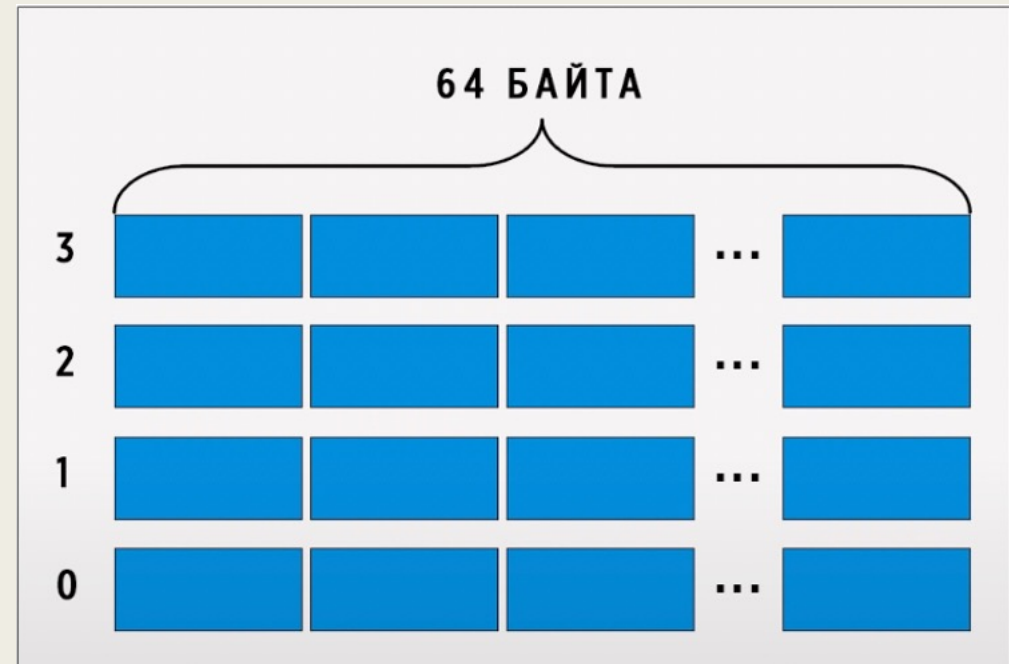
3	960	961	962	...	1023	15
	896	897	898	...	959	14
	832	833	834	...	895	13
	768	769	770	...	831	12

2	704	705	706	...	767	11
	640	641	642	...	703	10
	576	577	578	...	639	9
	512	513	514	...	575	8

1	448	449	450	...	511	7
	384	385	386	...	447	6
	320	321	322	...	383	5
	256	257	258	...	319	4

0	192	193	194	...	255	3
	128	129	130	...	191	2
	64	65	66	...	127	1
	0	1	2	...	63	0

Кэш прямого отображения: 256 байт
Оперативная память: 1 Кб



Наборно-ассоциативный и полностью ассоциативный кэш

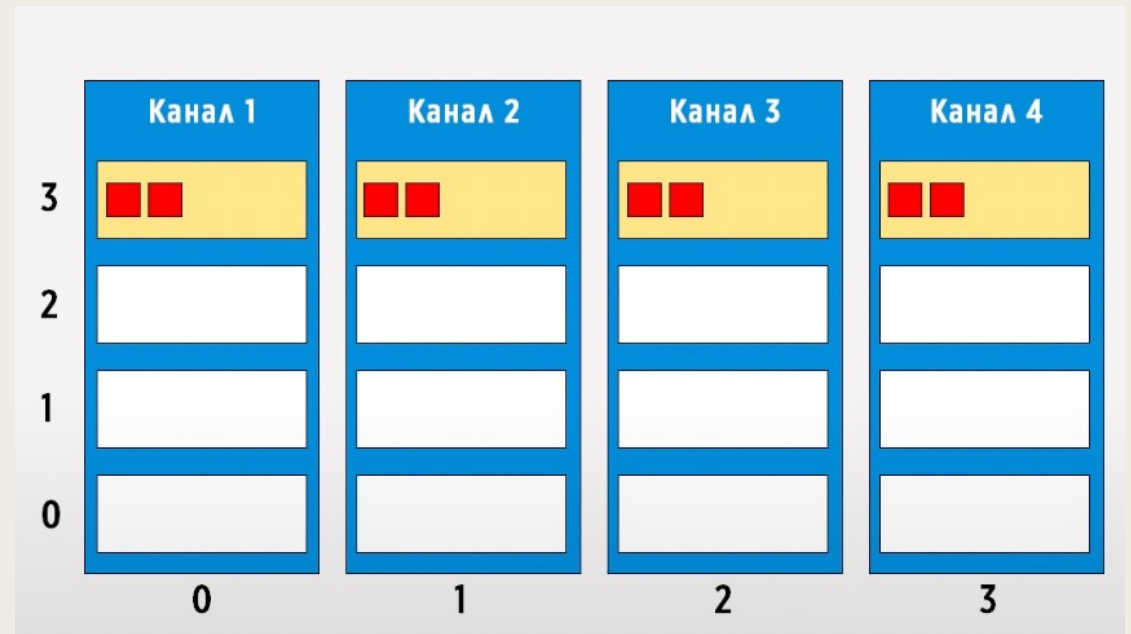
3	960	961	962	...	1023	15
	896	897	898	...	959	14
	832	833	834	...	895	13
	768	769	770	...	831	12

2	704	705	706	...	767	11
	640	641	642	...	703	10
	576	577	578	...	639	9
	512	513	514	...	575	8

1	448	449	450	...	511	7
	384	385	386	...	447	6
	320	321	322	...	383	5
	256	257	258	...	319	4

0	192	193	194	...	255	3
	128	129	130	...	191	2
	64	65	66	...	127	1
	0	1	2	...	63	0

Наборно-ассоциативный кэш



Как замещаются данные в кэше

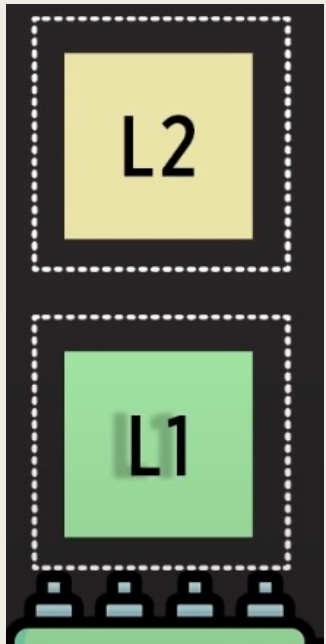
- **LRU** (least recently used) – замещаются те данные, к которым дольше всего не было обращений, предполагается, что к ним не будет обращений в ближайшее время.
- **MFU** (most frequently used) – замещаются последние используемые данные.
- **LFU** (least frequently used) – замещаются данные, которые использовались реже всех.

Многоуровневый кэш



- **L1** – кэш 1-го уровня, самый быстрый, принадлежит конкретному ядру, содержит данные и инструкции
- **L2** – кэш 2-го уровня, медленнее, чем L1, также принадлежит конкретному ядру, содержит данные и инструкции
- **L3** – кэш 3-го уровня, самый большой и медленный, общий для всех ядер

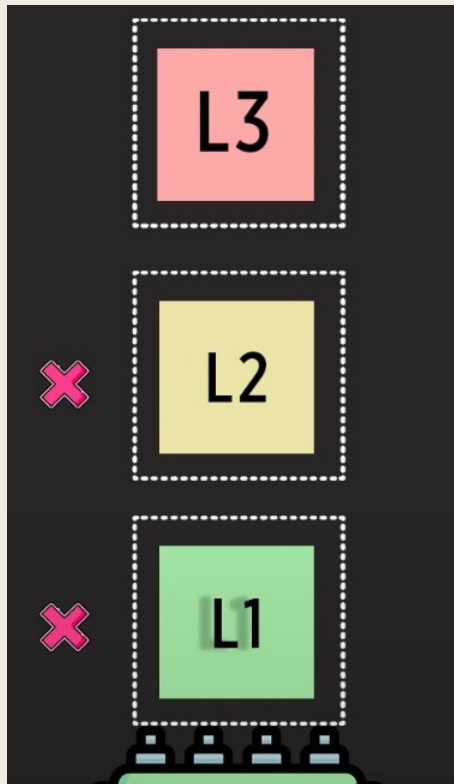
Двухуровневый кэш



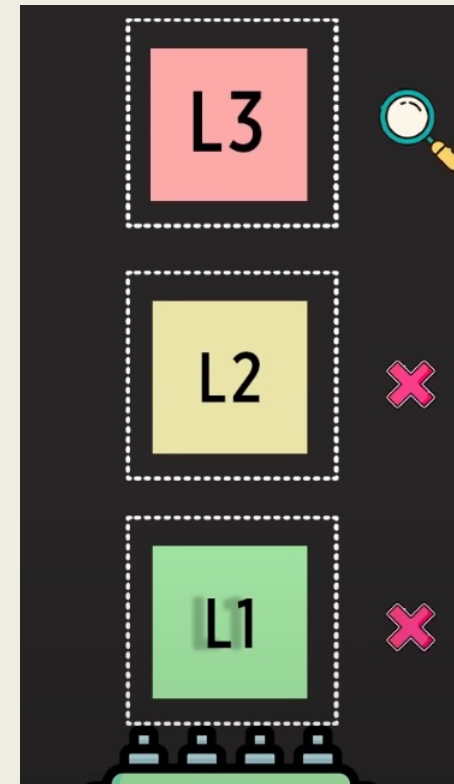
- Инклюзивная (включающая) архитектура – L2 дублирует содержимое L1
- Эксклюзивная (исключающая) архитектура – L2 не дублирует содержимое L1

Трехуровневый кэш

Поиск данных при
включающей архитектуре

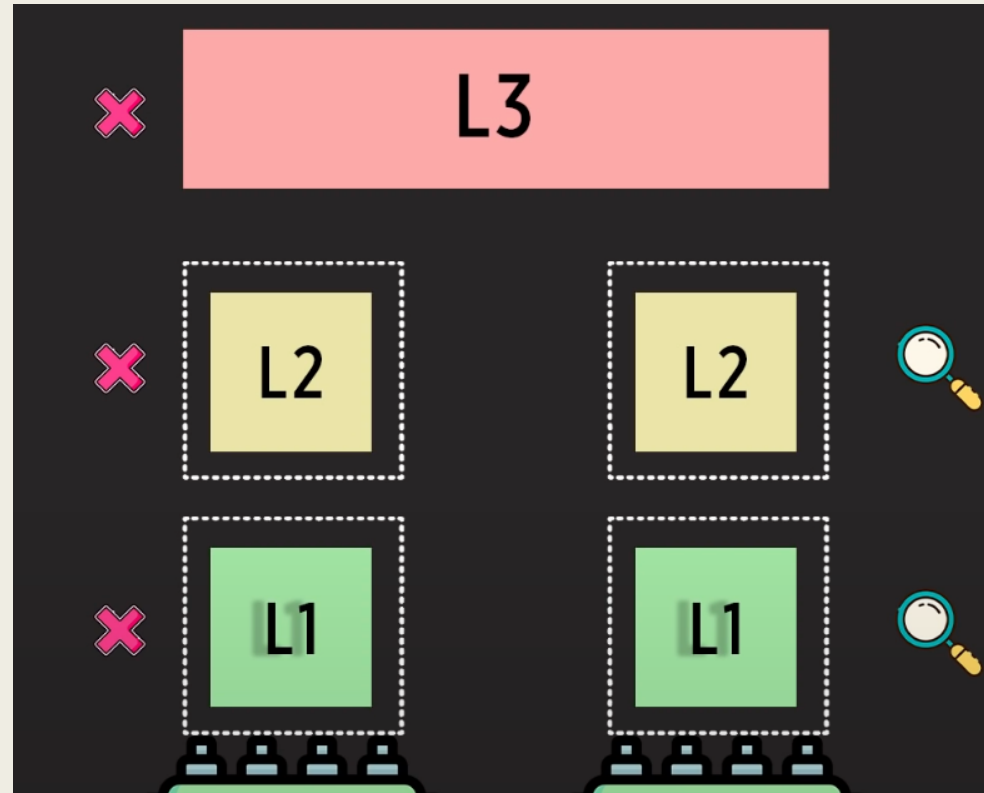


Поиск данных при
исключающей архитектуре

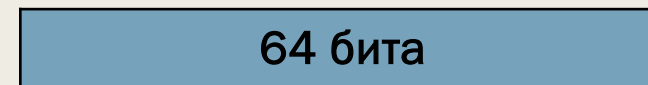
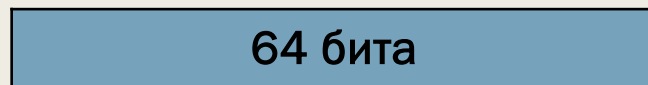
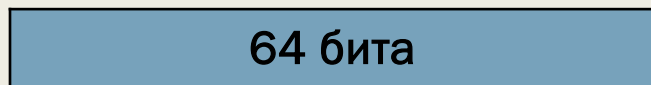
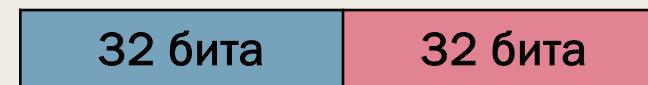
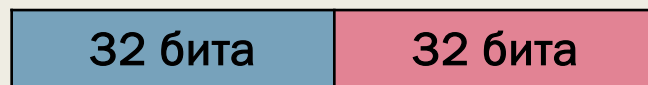
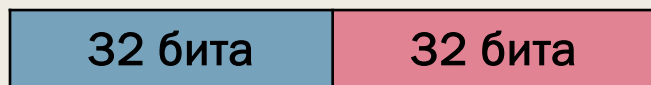
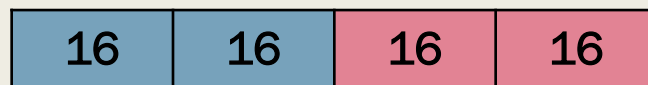
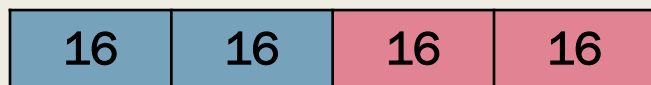
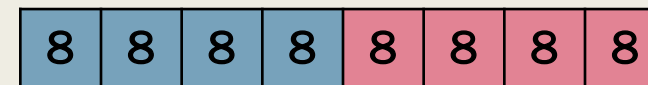
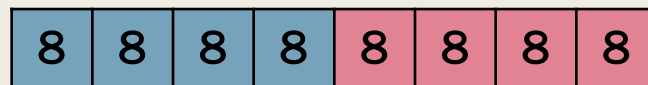
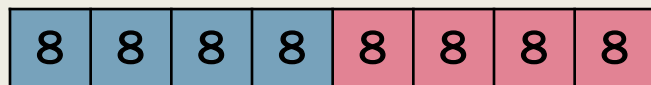


Кэш в многоядерном процессоре

Поиск данных при
исключающей архитектуре



Кэш и типы данных

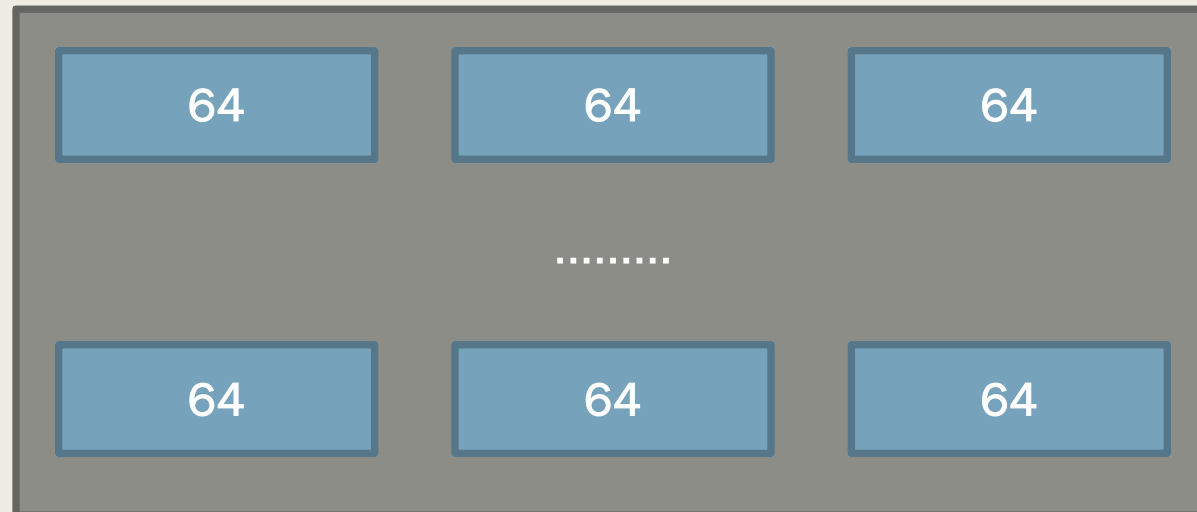


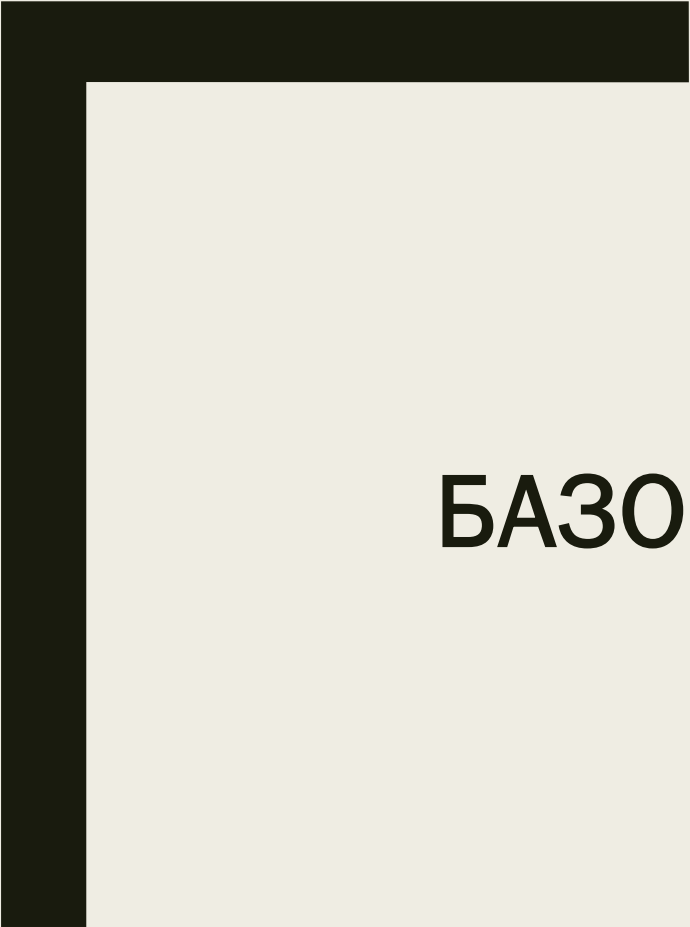
Кэш и типы данных

массив [] 1000 чисел типа `int`
4000 байт

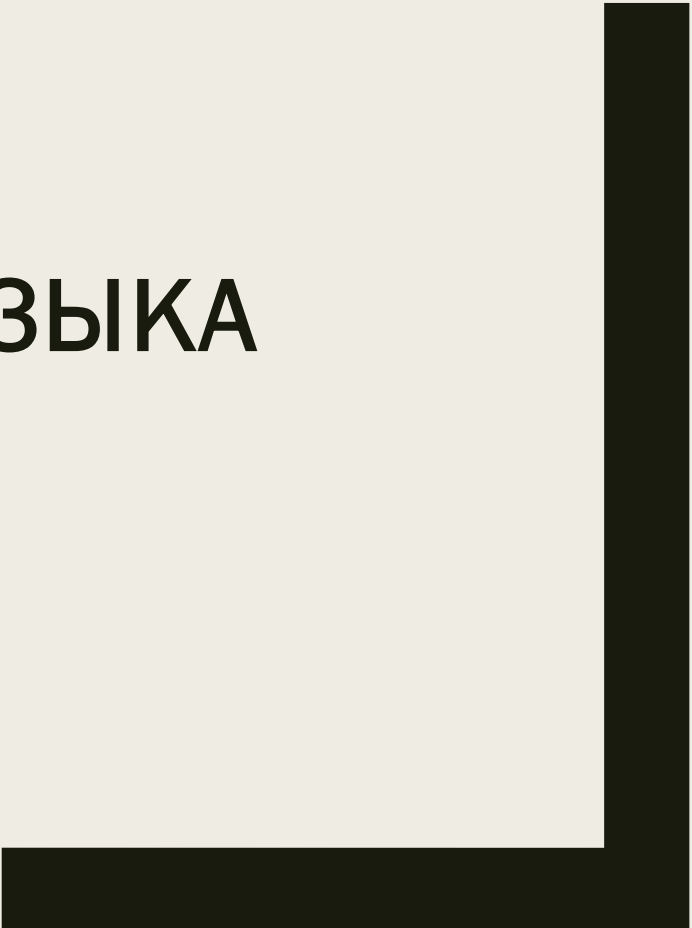
массив [] 1000 чисел типа `long`
8000 байт

кэш-память на 4 кб с кэш-строками на 64 б





БАЗОВЫЕ КОМАНДЫ ЯЗЫКА АССЕМБЛЕР



Команды пересылки данных

- Команда **MOV** – пересылает байт, слово или двойное слово между регистром и ячейкой памяти или между двумя регистрами.

MOV приемник, источник

- Команда **MOV** не умеет пересылать данные непосредственно из одной ячейки памяти в другую. Чтобы присвоить значение одной переменной другой переменной ($A = B$), придется делать так:

MOV AX, A

MOV B, AX

Команды пересылки данных

- Команда **XCHG** – меняет местами содержимое двух операндов.

XCHG приемник, источник

- Команда **XCHG** не может работать с непосредственно заданными значениями.

XCHG VAL, BX ; Обмен содержимого 16-разрядного операнда
; в памяти и регистра **BX**

XCHG EAX, EBX ; Обмен содержимого 32-разрядных регистров

Команды пересылки данных

- Чтобы поменять местами значения двух переменных в памяти, придется делать так:

```
.DATA
    VAL1      DW 1
    VAL2      DW 2
.
.
.
.CODE
    MOV       EAX, VAL1
    XCHG      EAX, VAL2
    MOV       VAL1, EAX
```

Арифметические команды

- Могут выполняться над двоичными числами со знаком или без знака
- Есть средства для отслеживания и обработки ситуаций переполнения (флаги процессора)

Команды сложения ADD и XADD

- Команда **ADD** – обычное сложение (8-, 16-, 32-битовые операнды)

ADD приемник, источник

ADD AX, CX

- Команда **XADD** – сначала производит обмен данными, потом выполняет команду **ADD**

XADD приемник, источник

Команда сложения ADC

- Команда **ADC** – то же самое, что **ADD**, но использует флаг переноса **CF**
приемник = приемник + источник + перенос

- Пример: 250 + 10 с использованием 8-битовых регистров

	7	6	5	4	3	2	1	0
	1	1	1	1	1	0	1	0
+	0	0	0	0	1	0	1	0
=	1	0	0	0	0	1	0	0

↓
CF = 1

Сложение чисел повышенной точности

- **Пример:** сложение чисел повышенной точности. Пусть складываются 64-битовое число, находящееся в регистрах ECX и EDX, с 64-битовым числом, находящимся в регистрах EAX и EBX.

ADD EAX, ECX ; Сначала складываются младшие 32 бита,

ADC EBX, EDX ; а затем старшие 32 бита

- Команда ADC добавляет к (EDX)+(EBX) любой перенос от сложения (ECX)+(EAX).

Команды сложения и флаги

- $CF = 1$ (флаг переноса) – результат сложения не помещается в операнде-приемнике.
- $PF = 1$ (флаг четности) – 8 младших битов результата имеют четное число битов со значением 1.
- $ZF = 1$ (флаг нуля) – результат равен 0.
- $SF = 1$ (флаг знака) – результат отрицателен (старший бит равен 1).

Переполнение

- $OF = 1$ (флаг переполнения) – сложение двух чисел одного знака (оба положительные или оба отрицательные) приводит к результату, который превышает диапазон допустимых значений приемника в обратном коде, а сам приемник при этом меняет знак.
- Переполнение определяется чисто механически: сравниваются два бита переноса (флаг переноса CF и бит переноса в знаковый разряд). Если они не равны, устанавливается флаг переполнения.

Переполнение

	7	6	5	4	3	2	1	0
CF = 1 ←	1	0	0	0	0	0	0	0
+	1	1	1	1	1	1	1	0
=	0	1	1	1	1	1	1	0

Команды вычитания SUB и SBB

- Команда **SUB** – обычное вычитание (8-, 16-, 32-битовые операнды)

SUB приемник, источник

приемник = приемник – источник

SUB AX, CX

- Команда **SBB** – дополнительно вычитает значение флага переноса

приемник = приемник – источник – перенос

Вычитание чисел повышенной точности

- **Пример:** Вычтем 64-битовое число, помещенное в регистры ECX и EDX, из 64-битового числа, помещенного в регистры EAX и EBX.

SUB EAX, ECX ; Вычесть младшие 32 бита

SBB EBX, EDX ; а затем – старшие 32 бита

Команды вычитания и флаги

- $CF = 1$ (флаг переноса) – требуется заем.
- $PF = 1$ (флаг четности) – 8 младших битов результата вычитания имеют четное число битов со значением 1.
- $ZF = 1$ (флаг нуля) – результат равен 0.
- $SF = 1$ (флаг знака) – результат отрицателен (старший бит равен 1).
- $OF = 1$ (флаг переполнения) – при вычитании чисел, имеющих разные знаки, результат превышает диапазон значений приемника в обратном коде, а сам приемник изменяет знак.

Инкремент и декремент

- Команда **INC** добавляет 1 к содержимому регистра или ячейки памяти, но не воздействует на флаг переноса CF.
- Команда **DEC** вычитает 1 из содержимого регистра или ячейки памяти, но при этом не воздействует на флаг переноса CF.

Обращение знака

- Команда **NEG** вычитает значение операнда-приемника из нулевого значения. Оказывает на флаги то же действие, что и команда **SUB**.
- Полезна для вычитания значения регистра/ячейки памяти из непосредственного значения:

NEG AX

ADD AX, 100

Команды умножения

- Команда **MUL** – умножение чисел без знака.
- Команда **IMUL** – умножение чисел со знаком.

MUL/IMUL источник

- Второй операнд – регистр AL/AX/EAX
- Произведение имеет **двойной размер**. Результат в:
 - регистрах AH (старший байт) и AL (младший байт) для умножения байтов
 - регистрах DX (старшее слово) и AX (младшее слово) для умножения слов
 - в регистрах EDX (старшее слово) и EAX (младшее слово) для умножения двойных слов

Особенности команд умножения

- После исполнения команды **MUL** флаги CF и OF равны 0, если старшая половина произведения равна 0.
- После исполнения команды **IMUL** флаги CF и OF равны 0, если старшая половина произведения представляет собой лишь расширение знака младшей половины.
- Команды **MUL** и **IMUL** не позволяют в качестве операнда использовать непосредственное значение:

MOV DX, 10

MUL DX

Команды деления

- Команда **DIV** – деление чисел без знака.
- Команда **IDIV** – деление чисел со знаком.

DIV/IDIV источник (делитель)

- Второй операнд (делимое) имеет **двойной размер**. Извлекается из:
 - регистров AH и AL (при делении на 8-битовое число)
 - из регистров DX и AX (при делении на 16-битовое число)
 - из регистров EDX и EAX (при делении на 32-битовое число)

Команды деления

- Результаты в:
 - в регистре AL (частное) и в регистре AH (остаток) – если источник занимает байт
 - в регистре AX (частное) и в регистре DX (остаток) – если источник занимает слово
 - в регистре EAX (частное) и в регистре EDX (остаток) – если источник занимает двойное слово
- Команды DIV и IDIV не позволяют прямо разделить на непосредственное значение:

```
MOV  BX, 20
```

```
DIV  BX
```