Системы счисления

Позиционные:

1. Десятичная (2009_{10})
2. Двоичная (11111011001_2)
3. Восьмеричная (3731_8)
4. Шестнадцатеричная($7D9_{16}$)

Непозиционные:

1.Римская (MMIX)

В позиционных системах счисления **количественный эквивалент числа** ***A***, состоящего из ***n*** цифр a_k ($k = 0,\ldots,n-1$) в системе счисления с основанием ***p*** записывается в виде последовательности цифр

$A_{(p)}=a_{n-1}a_{n-2}\ldots a_1a_0$, где $a_k<p$.

Значение количественного эквивалента числа в позиционной системе счисления с основанием p можно представить в виде многочлена:

$A_{(p)}=a_{n-1}*p^{n-1}+a_{n-2}*p^{n-2}+\ldots+a_1*p^1+a_0*p^0$, где $a_k<p$, $p>0$

Переводы натуральных чисел между системами счисления I

восьмеричная <->двоичная<->шестнадцатеричная

Если основание системы счисления можно представить как степень 2,

$p=2^m,\ m=1,2,\ldots,k$

то перевод осуществляется через двоичную систему.

1. Сначала число в восьмеричной или шестнадцатеричной системе записываем в двоичном представлении:
   $7D9_{16}= 0111\ 1101\ 1001_2$
   $3731_8= 011\ 111\ 011\ 001_2$
2. разряды согласно основанию ***p*** системы счисления, в которую выполняем перевод числа:
   $0111\ 1101\ 1001_2 = 011\ 111\ 011\ 001_2$
   $011\ 111\ 011\ 001_2 = 0111\ 1101\ 1001_2$
3. Записываем число в представлении с основанием ***p*** системы счисления, в которую выполняем перевод числа:
   3731_8
   $7D9_{16}$

Переводы натуральных чисел между системами счисления II

десятичная ->двоичная

двоичная -> десятичная

$11111011001_2 =$

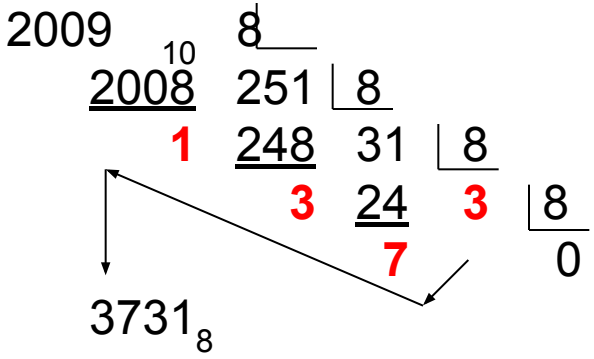
$= 1*2^{10} +1*2^9 +1*2^8 +1*2^7 +1*2^6+0*2^5+1*2^4+1*2^3+0*2^2+0*2^1+1*2^0 =$

$= 1024_{10}+512_{10}+256_{10}+128_{10}+64_{10}+0 +16_{10}+8_{10} +0 +0 + 1_{10}=$

$= 2009_{10}$

Переводы натуральных чисел между системами счисления III

десятичная -> восьмеричная



восьмеричная -> десятичная

$3731_8 =$

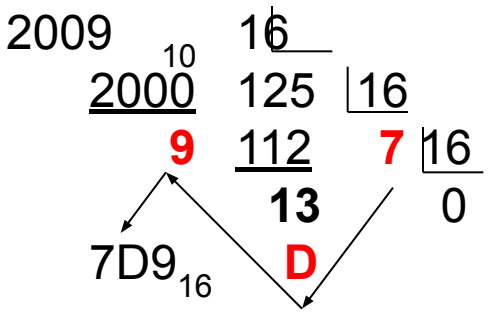
$= 3*8^3 + 7*8^2 + 3*8^1 + 1*8^0 =$

$= 1536_{10} + 448_{10} + 24_{10} + 1_{10} =$

$= 2009_{10}$

Переводы натуральных чисел между системами счисления IV

десятичная -> шестнадцатеричная



шестнадцатеричная -> десятичная

$7D9_{16} =$

$= 7*16^2 + 13*16^1 + 9*16^0 =$

$= 1792_{10} + 208_{10} + 9_{10} =$

$= 2009_{10}$

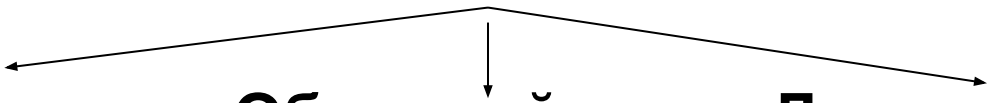
Отрицательные числа

Старший разряд - знаковый ,

1 в знаковом разряде означает, что число отрицательное

4 разряда

$0101_2 = 5_{10}$



Прямой код

0101

1101 $= -5_{10}$

Обратный код

0101

1010 $= -5_{10}$

Дополнительный код

0101

1011 $= -5_{10}$

-1	1001		1110		1111		
+- 0	0000	1000	0000	1111	0000		
1	0001		0001		0001		

Ассемблер

- *Язык ассемблера* — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

  Часто для краткости его называют просто *ассемблером*, что, строго говоря, не верно.
- *Ассемблер* — также это программа-компилятор для языка ассемблера.

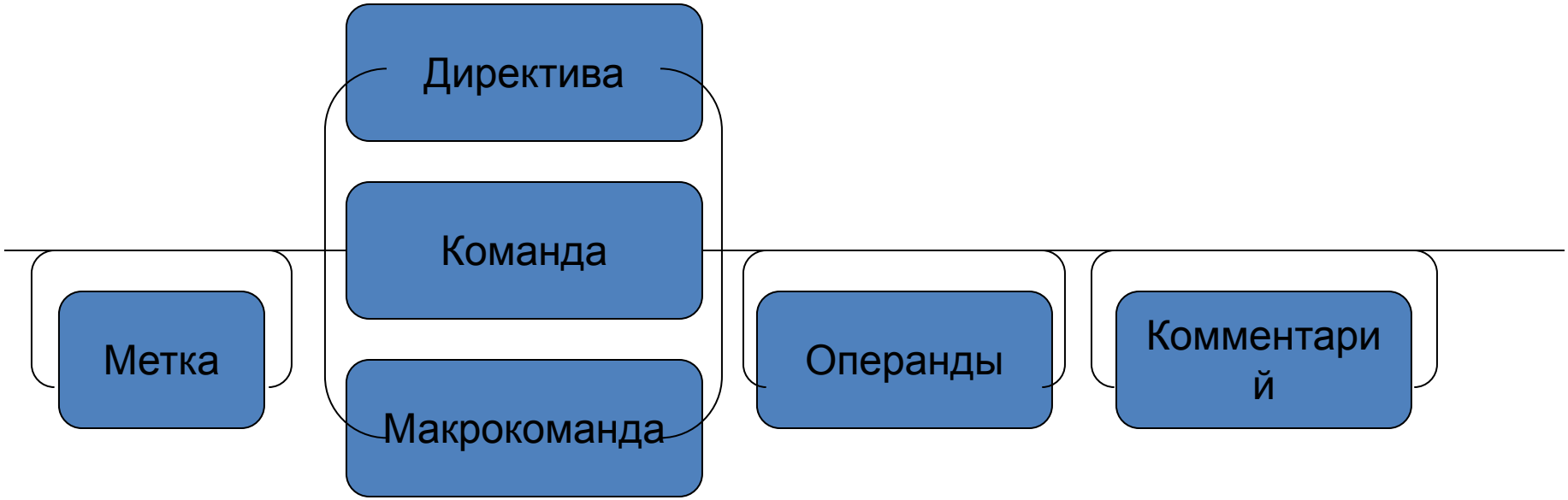
Команды микропроцессора

- *Команда* – указание процессору выполнить определенное действие.
- *Мнемоника команды* – удобная символьная запись команды.

Примеры мнемоники команд:

`mov`	**`loop`**	**`cmp`**	**`ja`**
`add`	**`sub`**	**`mul`**	**`div`**
`jmp`	**`and`**	**`xor`**	**`in`**

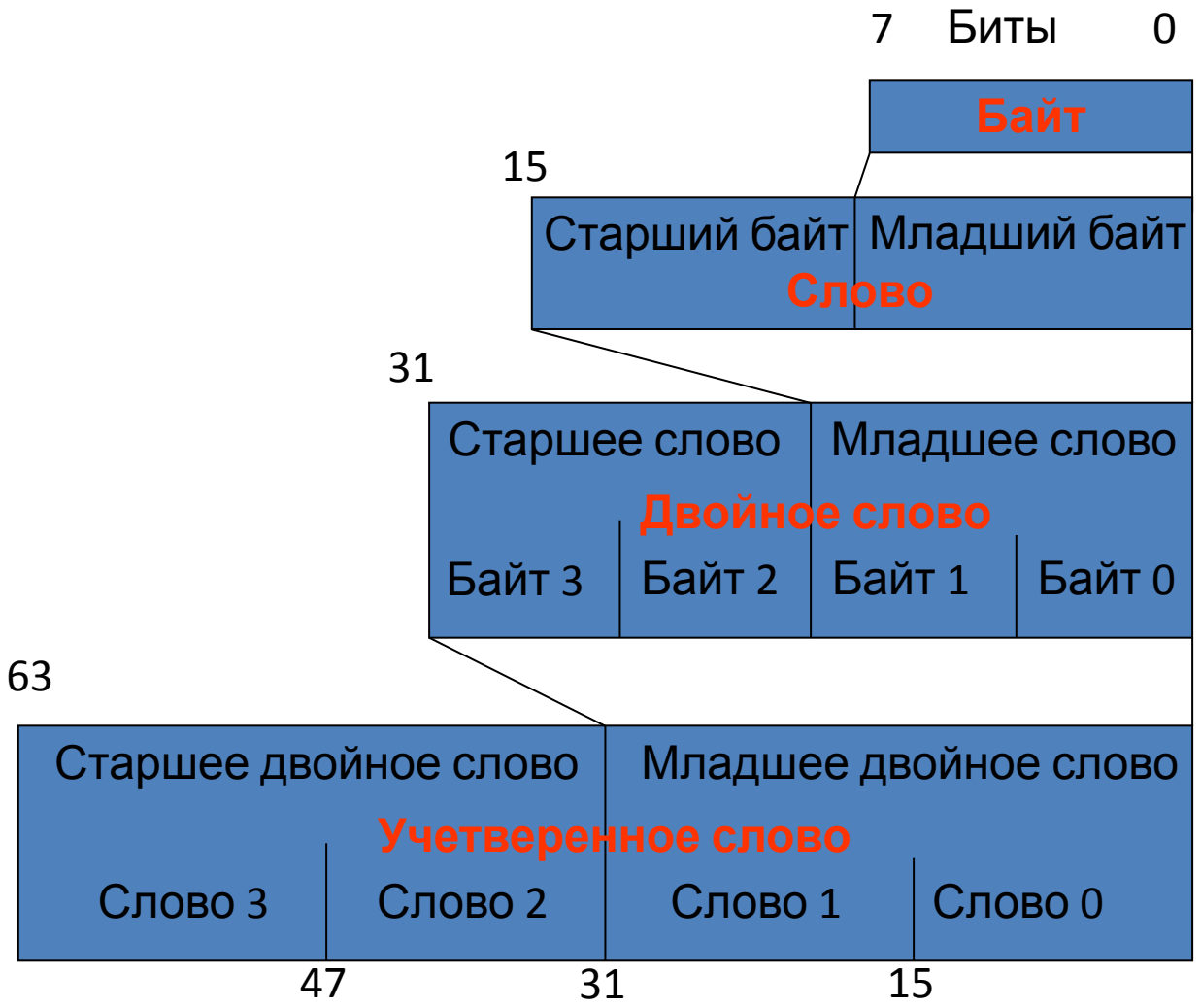
Формат инструкции на языке ассемблера



Примеры инструкций:

```
     mov cx,10
start: mov ax,bx
       loop start ; возврат на
       ; начало цикла
```

Типы данных по размеру (разрядности)



Типы данных по представлению (логической интерпретации)

1. **Беззнаковый целый тип** – двоичное значение без знака.
Диапазон значений определяется разрядностью:
Байт без знака – [0, 255];
Слово без знака – [0, 65535];
Двойное слово без знака – [0, 2^{32}-1 = 4 294 967 295]

2. **Знаковый целый тип** – двоичное значение со знаком.
Знак записывается в старший бит.
Отрицательные числа представляются в дополнительном коде.
Диапазон значений определяется разрядностью:
Байт со знаком – [-128, 127];
Слово со знаком – [-32768, 32767];
Двойное слово со знаком – [-2^{31}=2 147 483 648, 2^{31}-1=2 147 483 647].

3. **Битовое поле** – битовая последовательность, содержащая до 32 независимых битов (флагов).

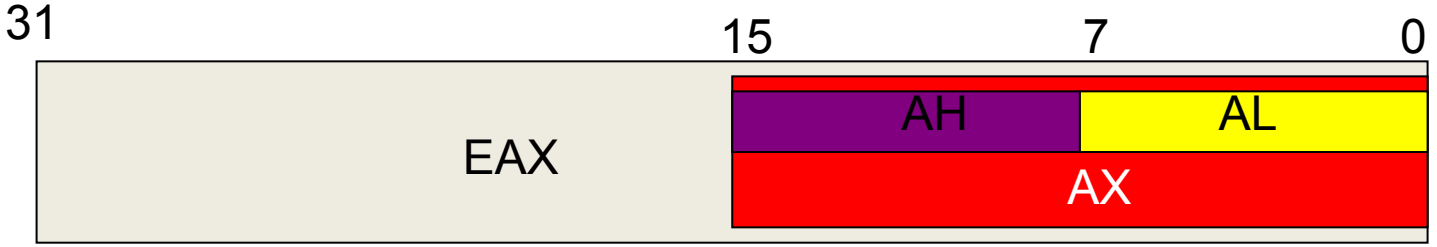
Регистры процессора

Регистры – специальные ячейки памяти, конструктивно расположенные внутри процессора, предназначенные для кратковременного хранения и обработки данных

Регистры общего назначения.

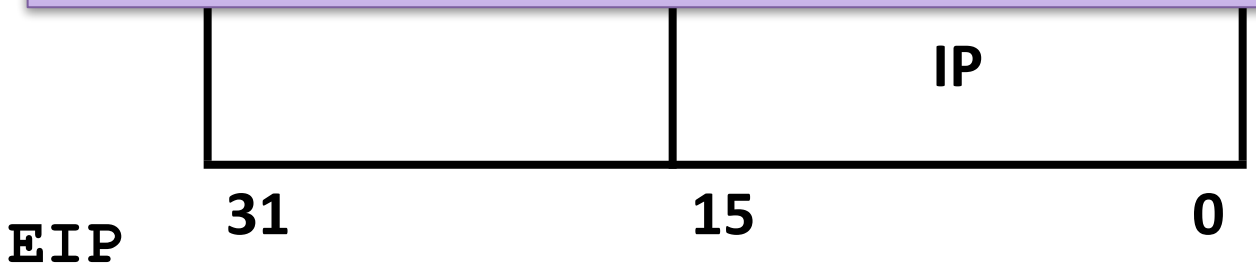
Предназначены для хранения данных и адресов.

- EAX/AX/AH/AL – accumulator register для проведения арифметических операций
- EBX/BX/BH/BL – base register для хранения базового адреса объекта
- ECX/CX/CH/CL – count register для организации циклов
- EDX/DX/DH/DL – data register для хранения промежуточных данных
- ESI/SI – source index register для текущего адреса элемента источника
- EDI/DI – destination index register для текущего адреса элемента приёмника
- ESP/SP – stack pointer register - указатель вершины стека в текущем сегменте стека
- EBP/BP – base pointer register для доступа к данным в стеке



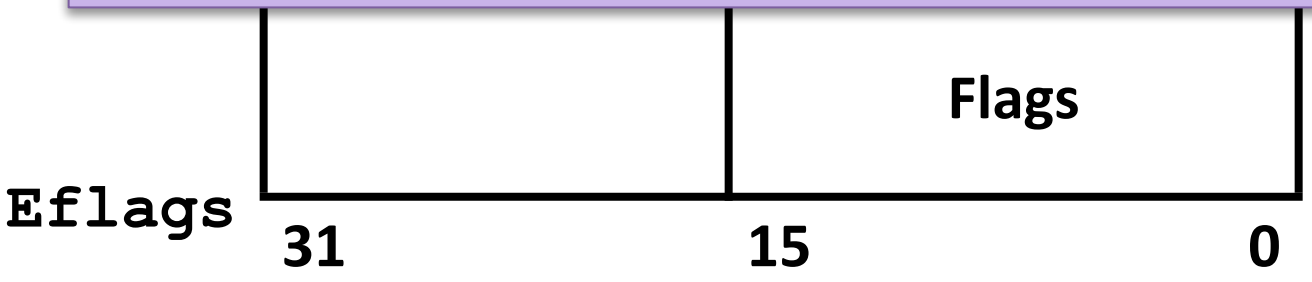
Регистры состояния

Содержит смещение следующей команды относительно базисной точки сегмента команд



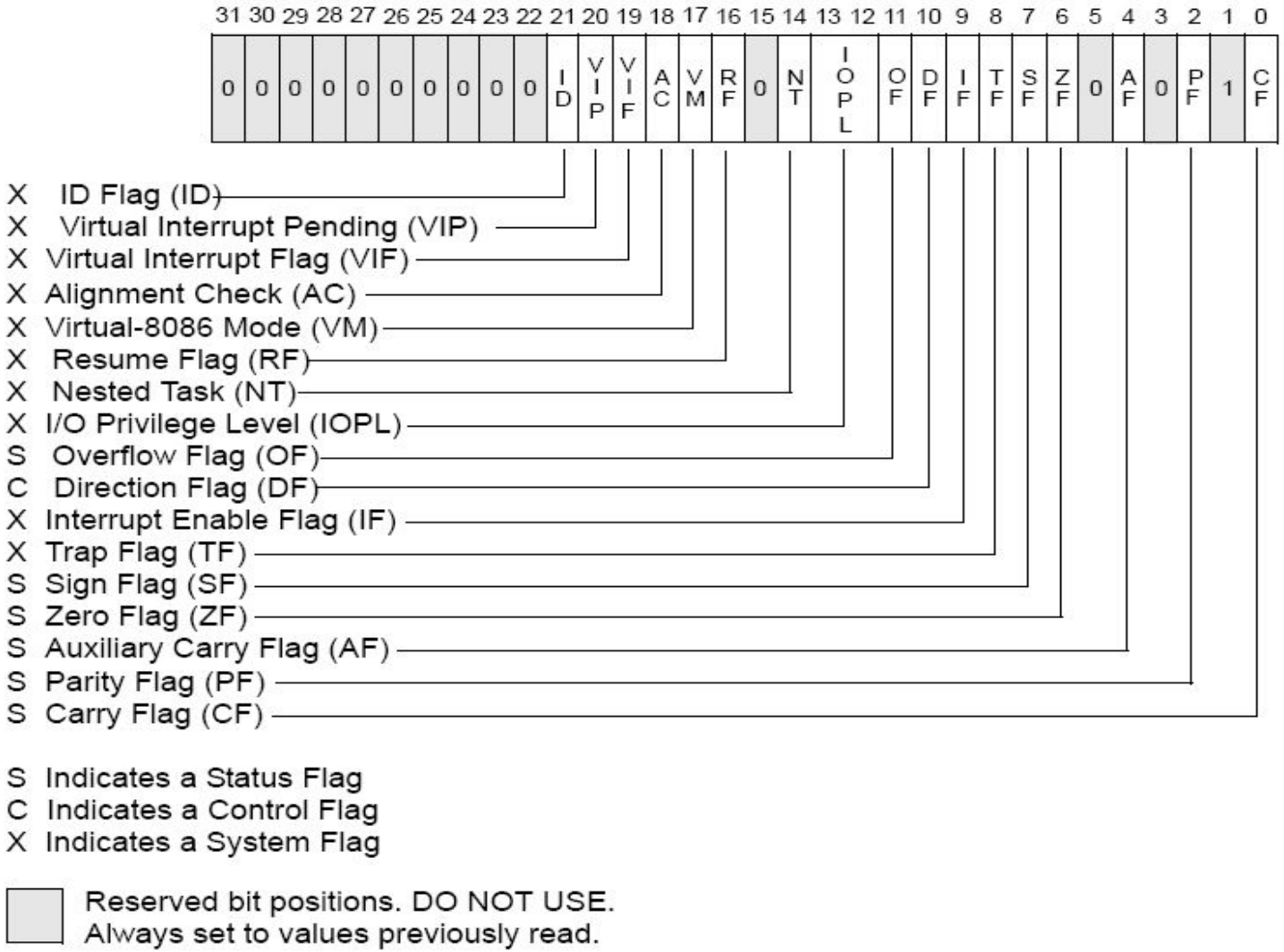
Указатель команд

Значения битов характеризуют статус текущего состояния процессора или результата выполненной арифметической операции

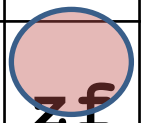
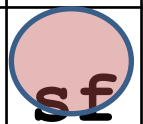


Регистр флагов

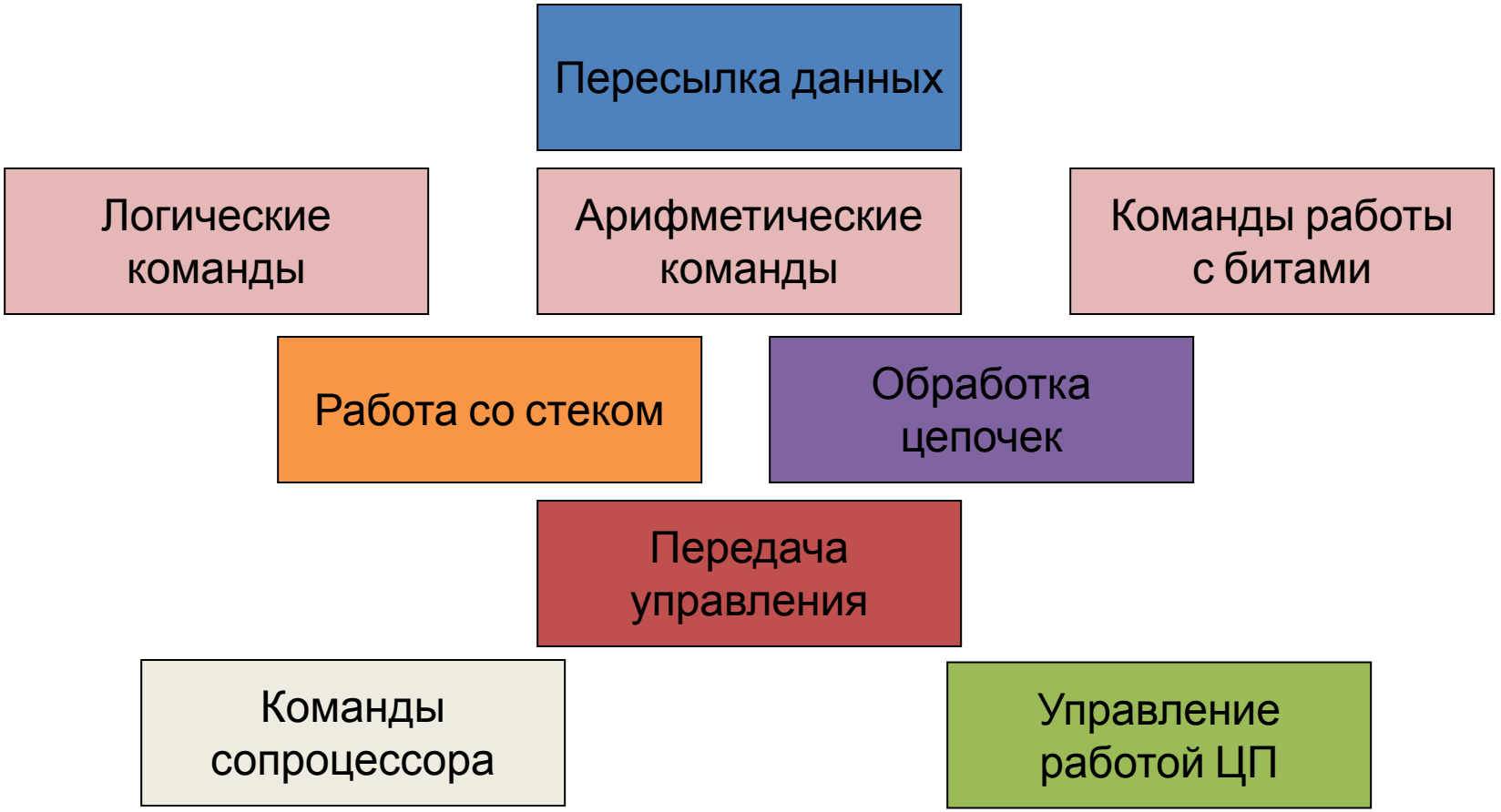
Регистр флагов



Флаги состояния

№ бита	Мнемоника	Флаг	Содержание и назначение
0		Carry Flag Флаг переноса	1 - арифметическая операция произвела перенос из старшего бита результата. Старшим является 7-й, 15-й или 31-й бит в зависимости от размерности операнда; 0 - переноса не было.
6		Zero Flag Флаг нуля	1 - результат нулевой; 0 - результат ненулевой.
7		Sign Flag Флаг знака	отражает состояние старшего бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно).
11		Overflow Flag Флаг переполнения	фиксирует факт потери значащего бита при арифметических операциях. 1 - произошел перенос(заем) из(в) старшего или знакового бита; 0 – произошел перенос(заем) из(в) старшего и знакового бита или переноса не было.

Основные команды ассемблера

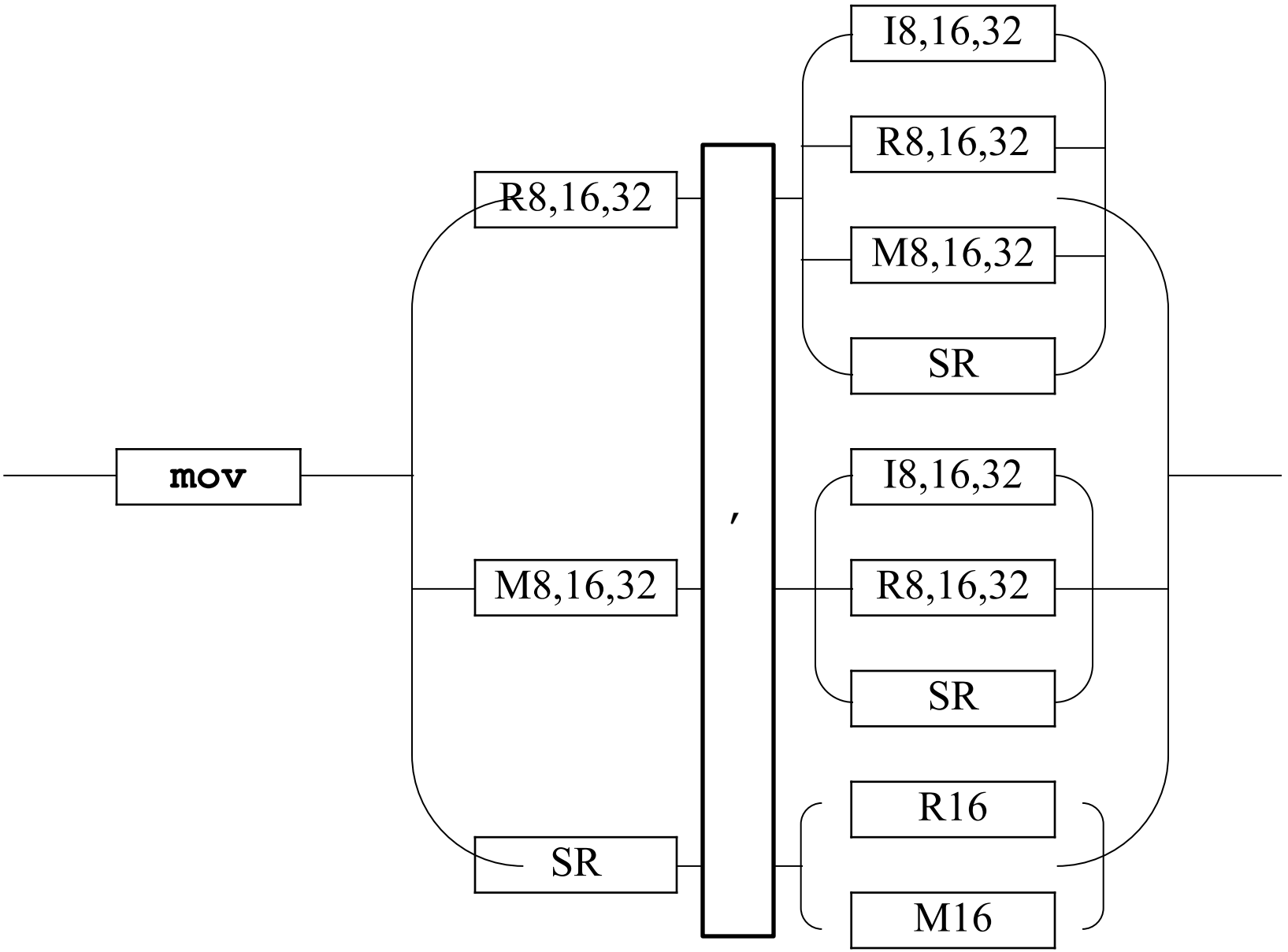


Обозначения

- SR – сегментный регистр
- R8, R16, R32 – регистр общего назначения
- M8, M16,M32 – адрес области памяти
- I8,I16,I32 – непосредственное значение (константа)

Команда mov

Пересылка данных



Использование команд встроенного ассемблера

```
#include <iostream.h>
void main() {
   int mem;
  __asm mov mem,5;
   // то же, что и mem=5;
  cout <<“mem=“<< mem << endl;
  }
```

Результат: mem=5

Использование команд встроенного ассемблера

```
#include <iostream.h>
void main() {
   int mem;
  __asm {
   mov mem,5;
   }
cout <<“mem=“<<mem<<endl;
  }
Результат: mem=5
```

Описание стека

Работа со стеком

Стек – область памяти, организованная для хранения и извлечения данных по принципу «первым зашёл, последним вышел»

- Назначение
  - Временное хранение данных
  - Хранение адреса возврата из вызванной функции
  - Передача параметров между функциями
- Сохранение состояния регистров
- Пересылка “без регистров”
- Единица данных – байт
- Вершина стека – **`esp`**
- Заполнение от старших адресов к младшим
- Настройка стека
  - явной загрузкой регистров
  - операционной системой
- Минимальная глубина
  - если не используем, то 128 байт
  - если используем, то своё+128 байт
- Доступ к элементам – **`ebp`**

Команды работы со стеком

Работа со стеком

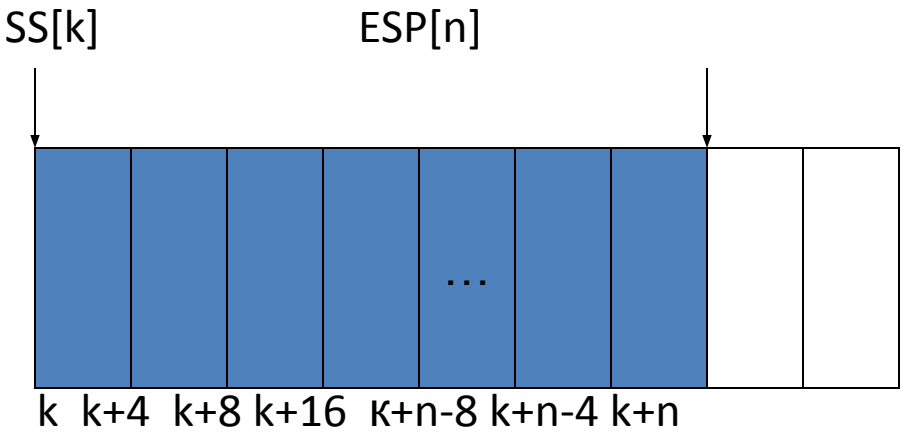
- **push**
- **pop**

- **pusha**
- **popa**

- **pushf**
- **popf**

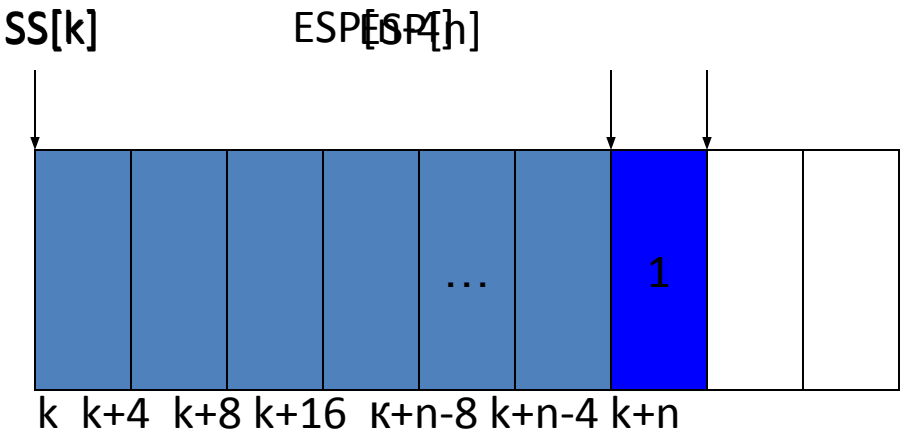
- **pushad**
- **popad**

Пример использования стека



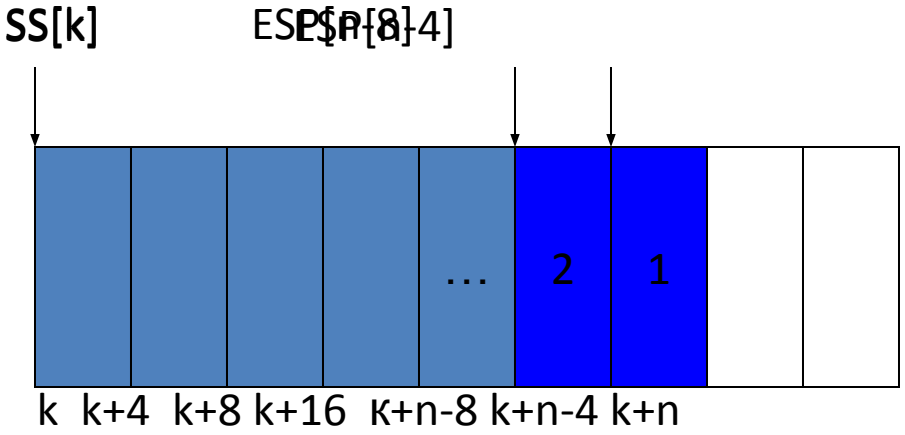

```
PUSH    1
PUSH    2
POP ebx
POP eax
NOP
```

Пример использования стека



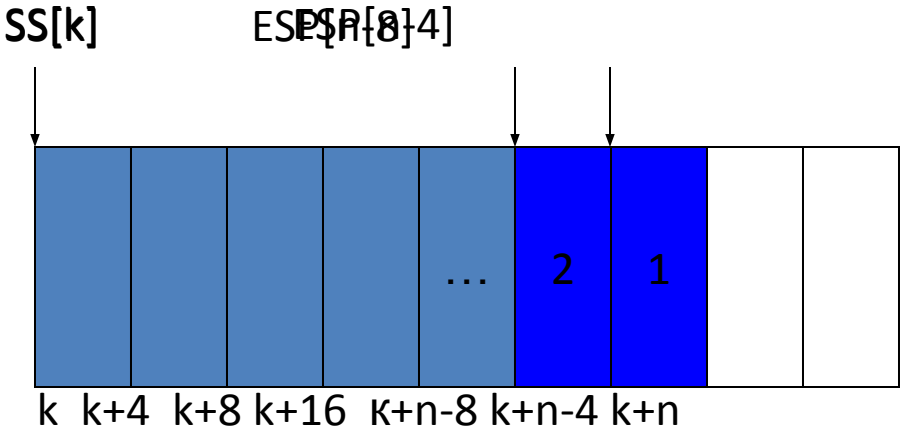
PUSH 1
PUSH 2
POP ebx
POP eax
NOP

Пример использования стека



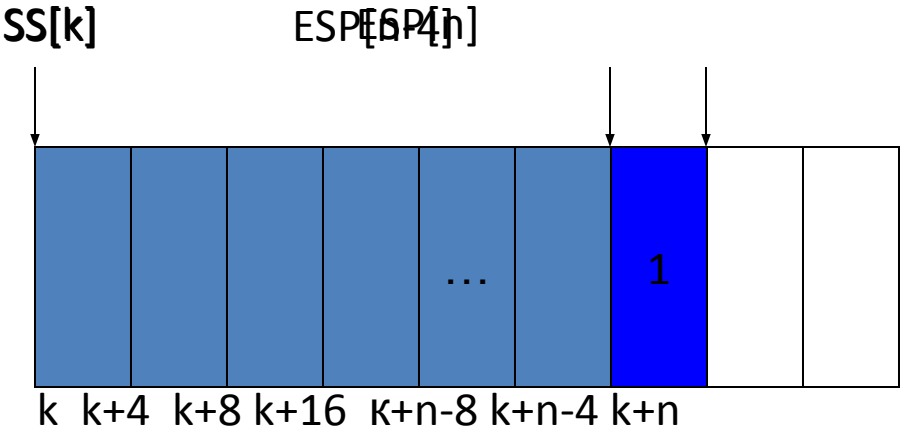

```
        PUSH    1
   →    PUSH    2
   →    POP ebx
        POP eax
        NOP
```

Пример использования стека



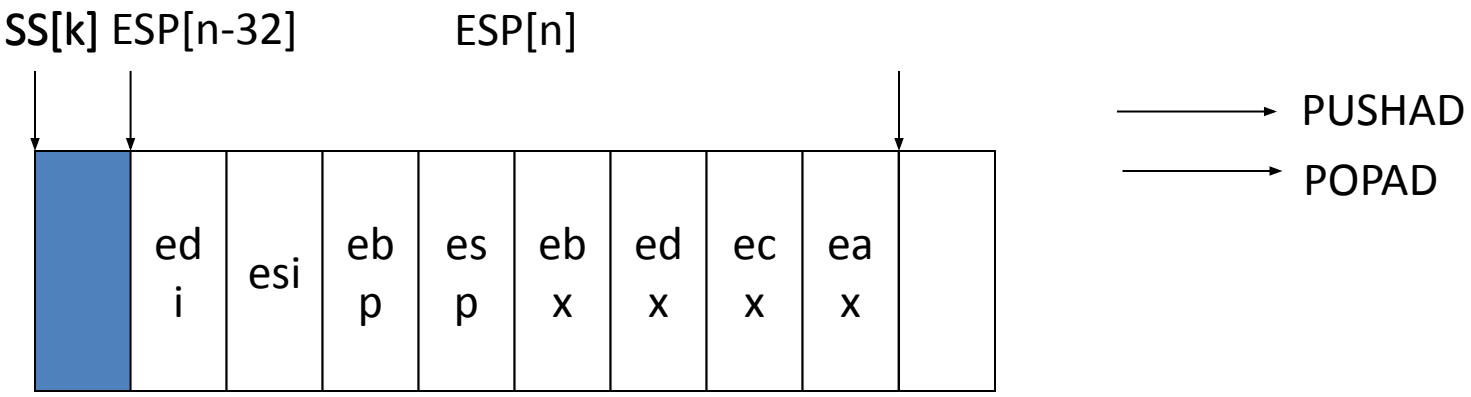

```
        PUSH    1
        PUSH    2
  ----> POP ebx       ; =2
  ----> POP eax
        NOP
```

Пример использования стека




```
        PUSH    1
        PUSH    2
        POP ebx         ; =2
  ----> POP eax         ; =1
  ----> NOP
```

Пример использования стека



Арифметические команд

- **neg**
- **inc**
- **dec**

- **add**
- **adc**
- **sub**
- **sbb**
- **cmp**
- **mul**
- **imul**

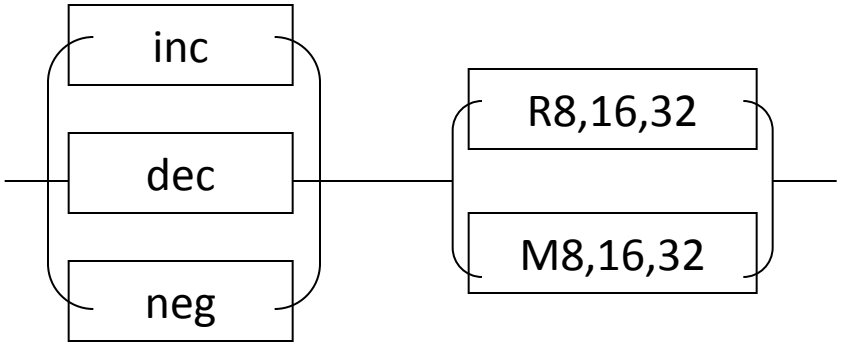
Двоичная арифметика

Арифметические команды

`inc` – увеличение на 1

`dec` – уменьшение на 1

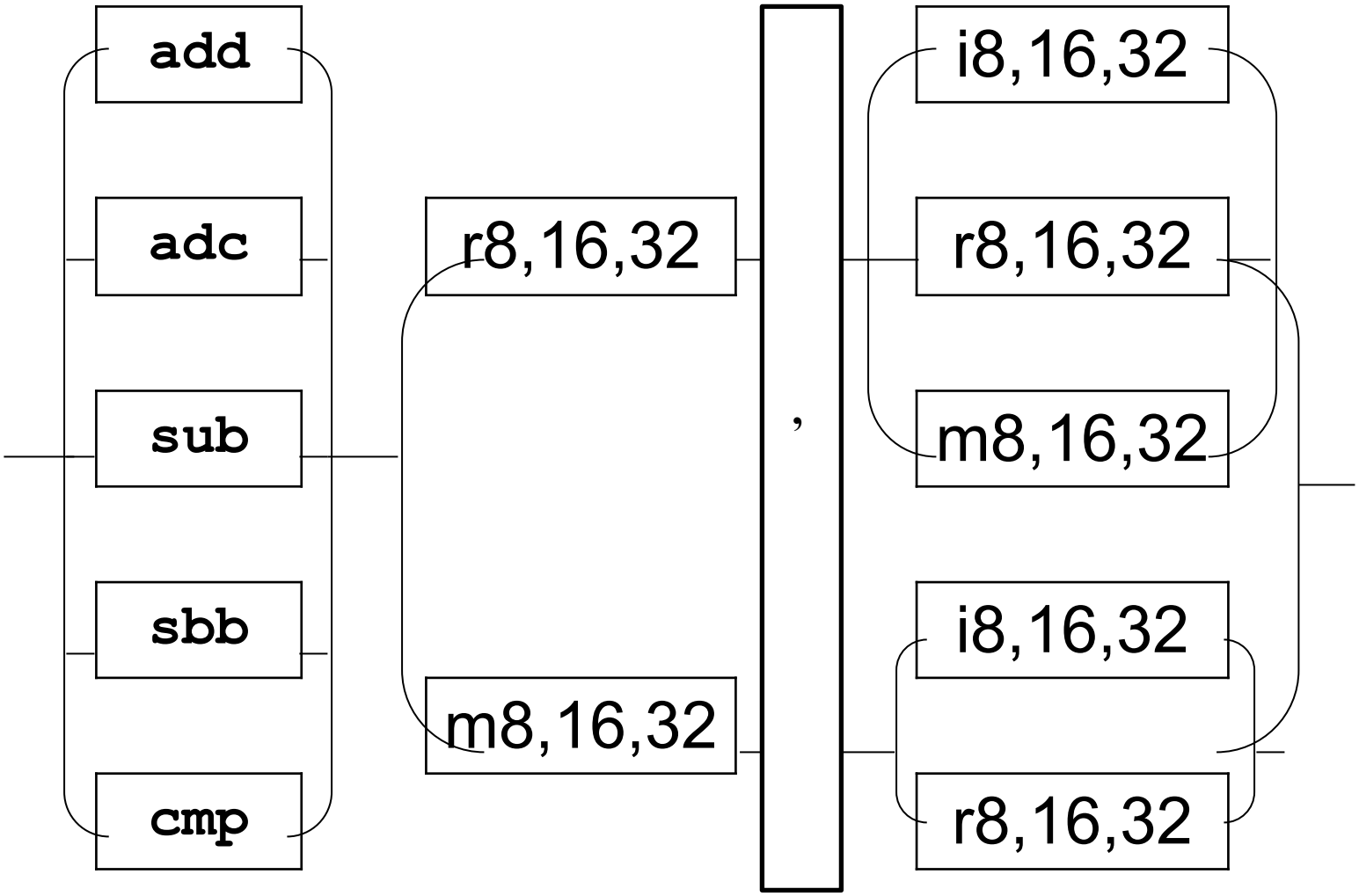
`neg` – смена знака



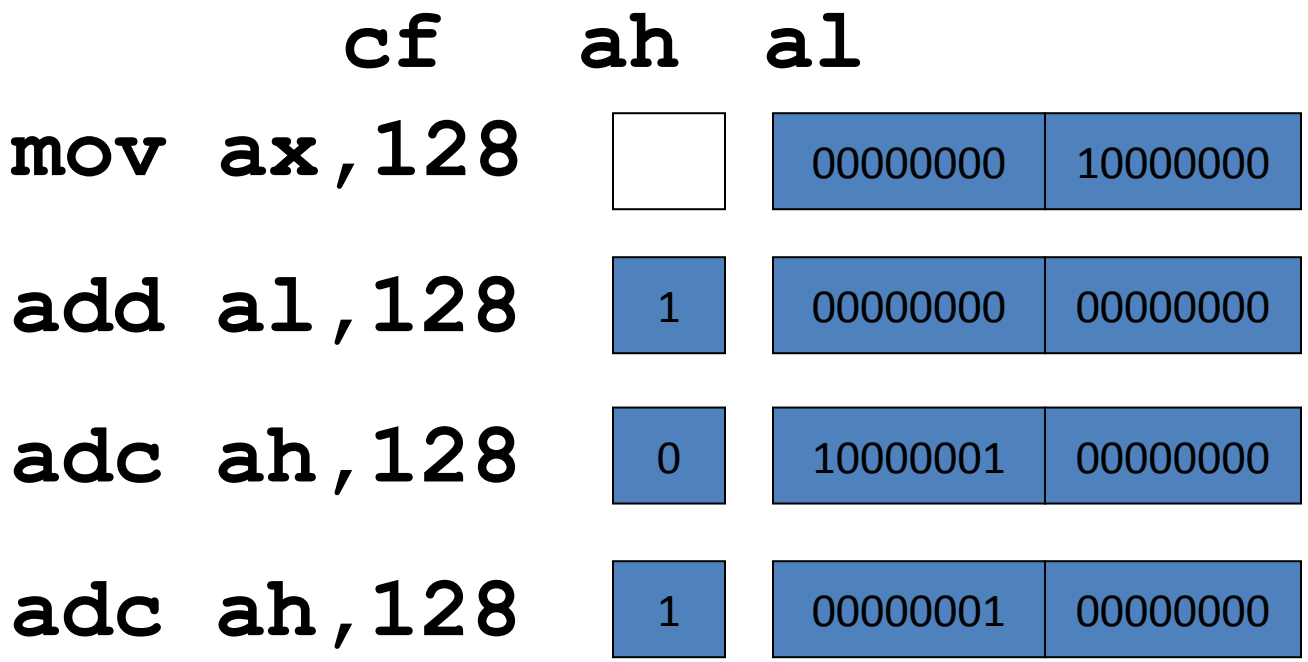
`inc` и **`dec`** не изменяют флаг **`cf`**.

Сложение и вычитание

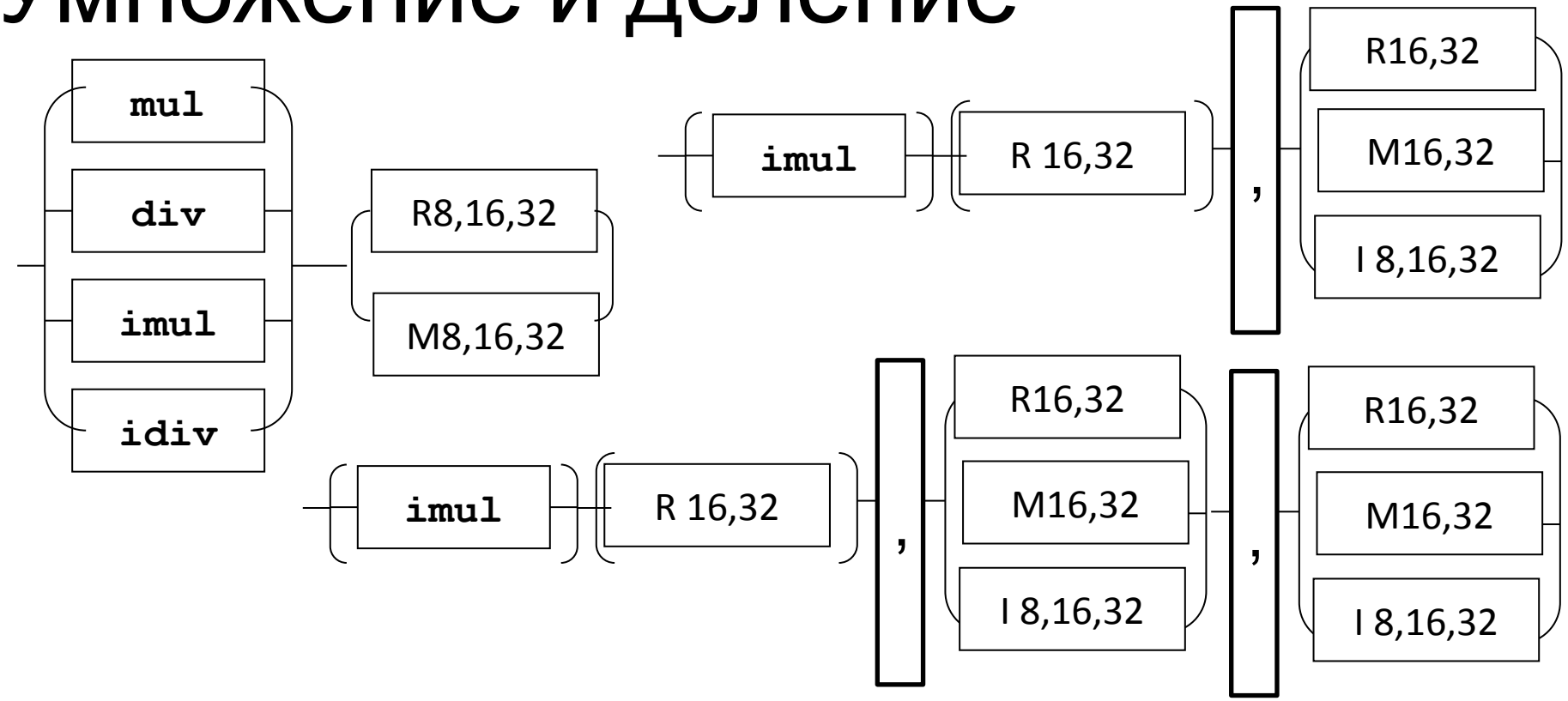
Арифметические команды



Пример сложения



Умножение и деление



Тип операнда	Первый сомножитель	Результат	Делимое	Результат	
				Частное	Остаток
8	**al**	**ax**	**ax**	**al**	**ah**
16	**ax**	**dx:ax**	**dx:ax**	**ax**	**dx**
32	**eax**	**edx:eax**	**edx:eax**	**eax**	**edx**

Исключительные ситуации команда **`div`**

- делитель равен нулю;
- частное велико – не входит в отведённую под него разрядную сетку, что может случиться в следующих случаях:
  - при делении делимого величиной в слово на делитель величиной в байт, причём значение делимого в более чем 256 раз больше значения делителя;
  - при делении делимого величиной в двойное слово на делитель величиной в слово, причём значение делимого в более чем 65 536 раз больше значения делителя;
  - при делении делимого величиной в учетверённое слово на делитель величиной в двойное слово, причём значение делимого в более чем 4 294 967 296 раз больше значения делителя.

Исключительные ситуации команда **idiv**

- делитель равен нулю;
- частное велико – не входит в отведённую под него разрядную сетку, что может случиться в следующих случаях:
  - при делении делимого величиной в слово со знаком на делитель величиной в байт со знаком, причём значение делимого в более чем 128 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –128 до +127);
  - при делении делимого величиной в двойное слово со знаком на делитель величиной в слово со знаком, причём значение делимого в более чем 32 768 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –32 768 до +32 768);
  - при делении делимого величиной в учетверённое слово со знаком на делитель величиной в двойное слово со знаком, причём значение делимого в более чем 2 147 483 648 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –2 147 483 648 до +2 147

Инструкции преобразования данных

Арифметические команды

- Байт в слово

  cbw ; al ax

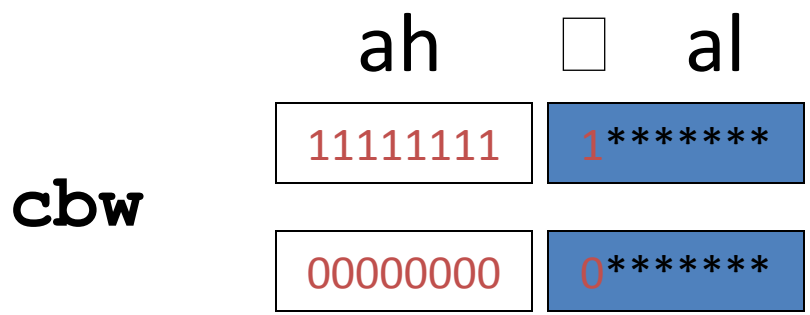
- Слово в двойное слово

  cwd ; ax dx

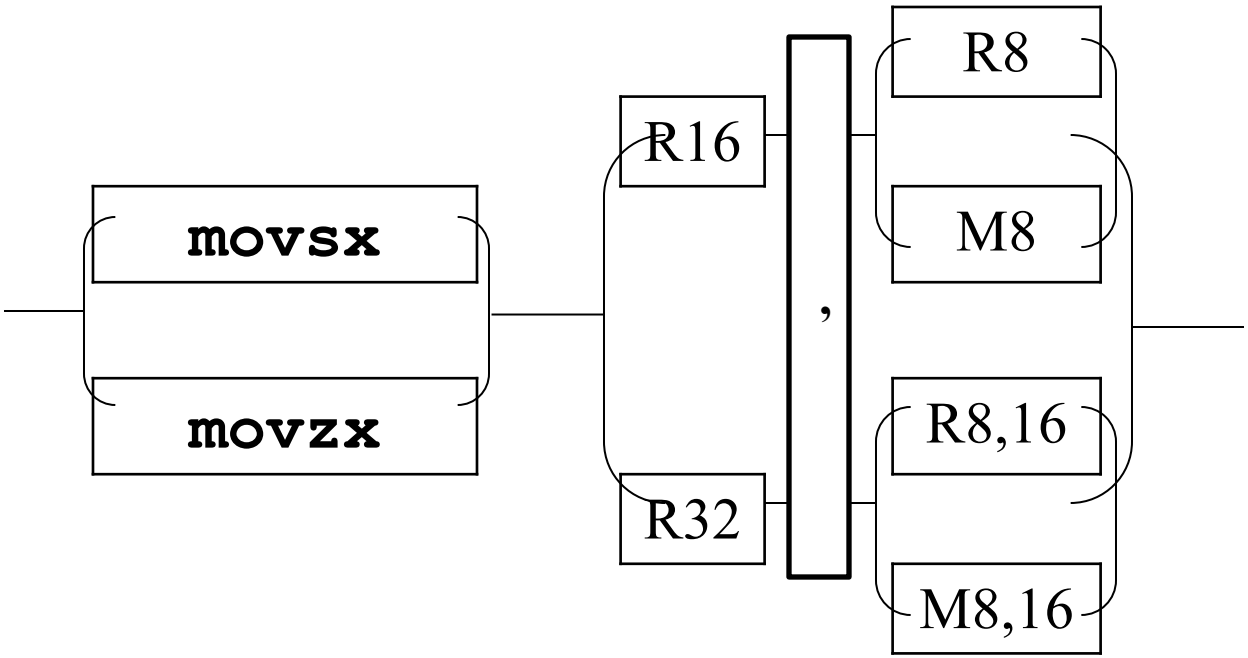
  cwde ; ax eax

- Двойное слово в учетверенное

  cdq ; eax edx



Пересылка данных с расширением



movsx – с учетом знака

movzx – без учета знака