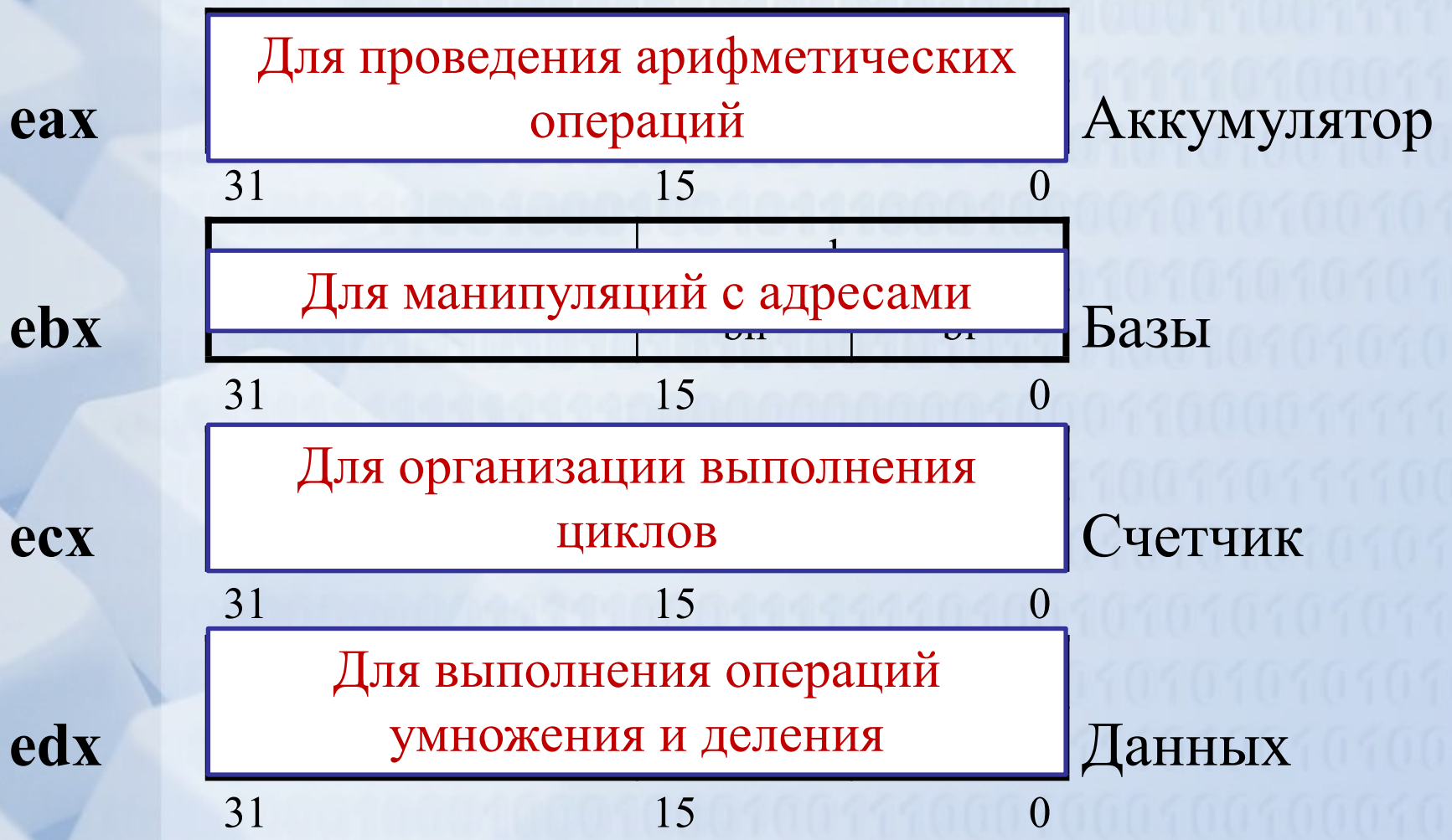
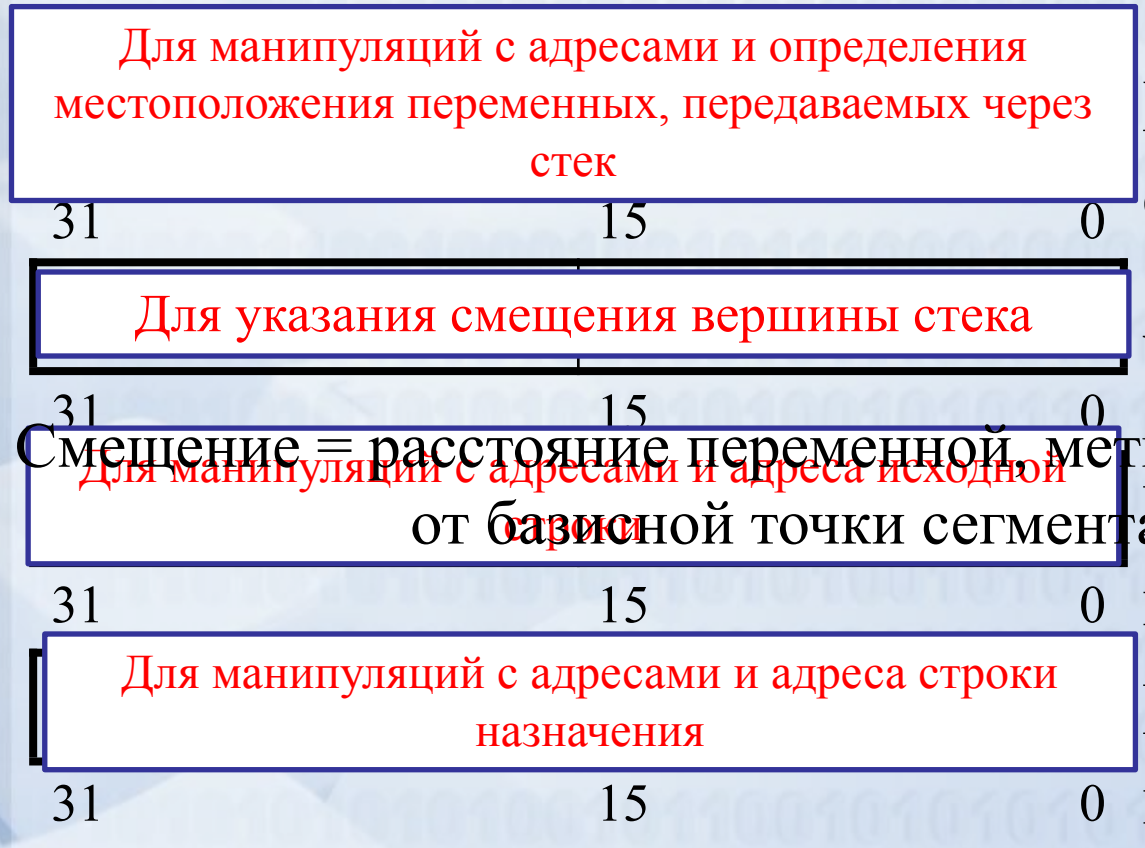
Основные компоненты языка Ассемблер

Регистры общего назначения



Регистры данных

Регистры общего назначения



База кадра стека

Указатель стека

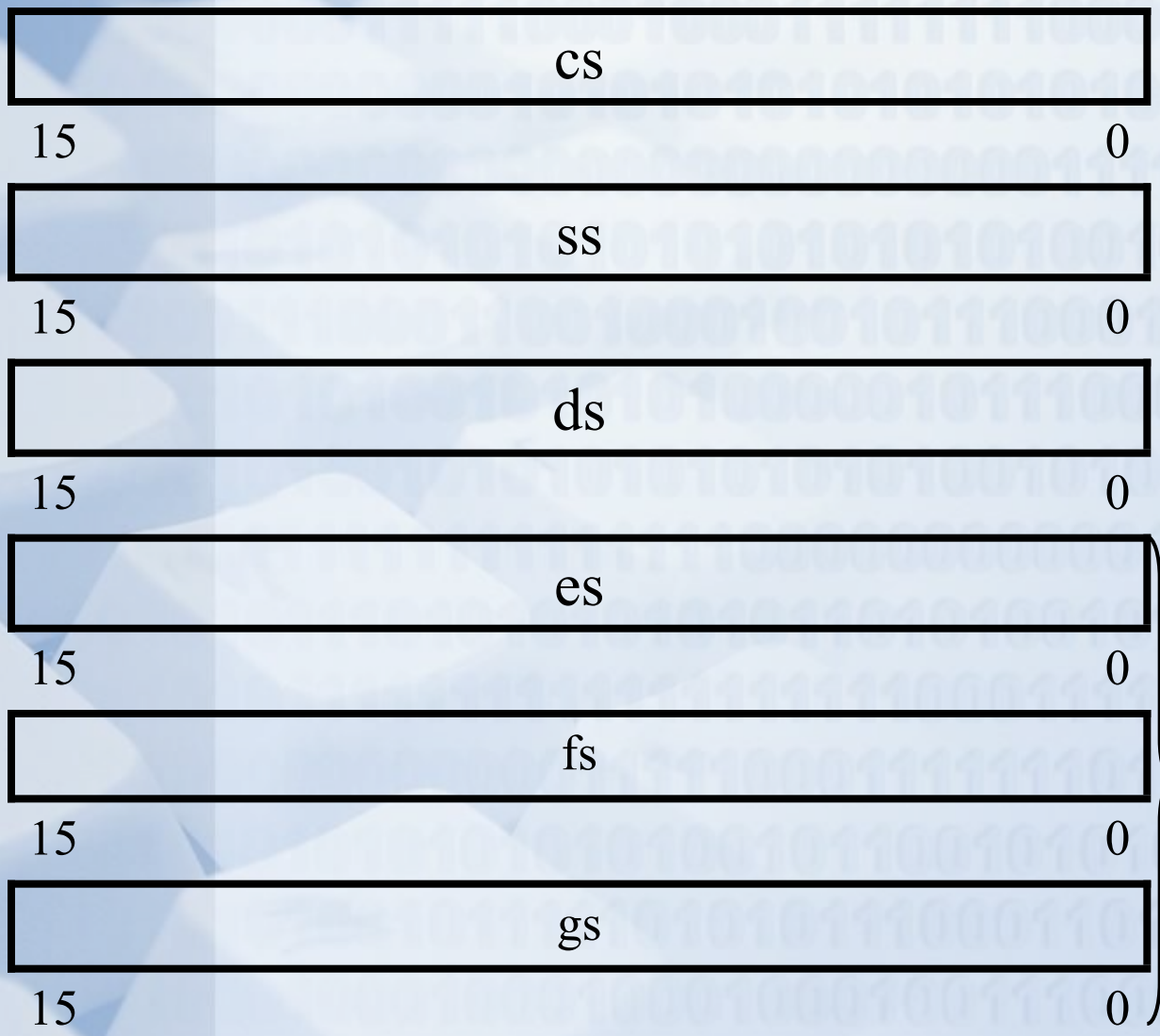
Индекс источника

Индекс приемника

Смещение = расстояние переменной, метки или команды от базисной точки сегмента.

Индекс-регистры: содержат смещение данных и команд

Сегментные регистры



Сегмент команд

Сегмент стека

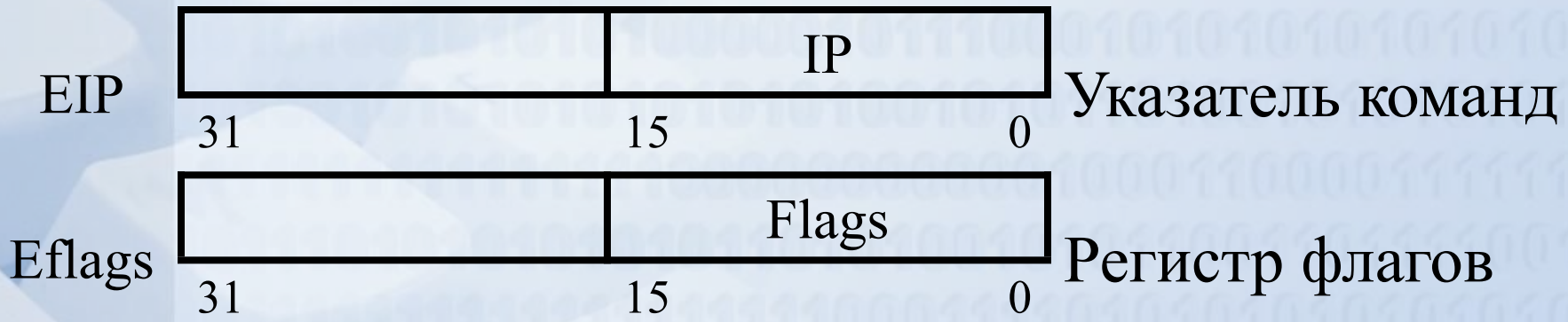
Сегмент данных

Дополнительные сегменты данных

Содержат адрес «базисной» точки для каждого из сегментов

Регистры управления

Содержит смещение следующей команды относительно базисной точки сегмента команд



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

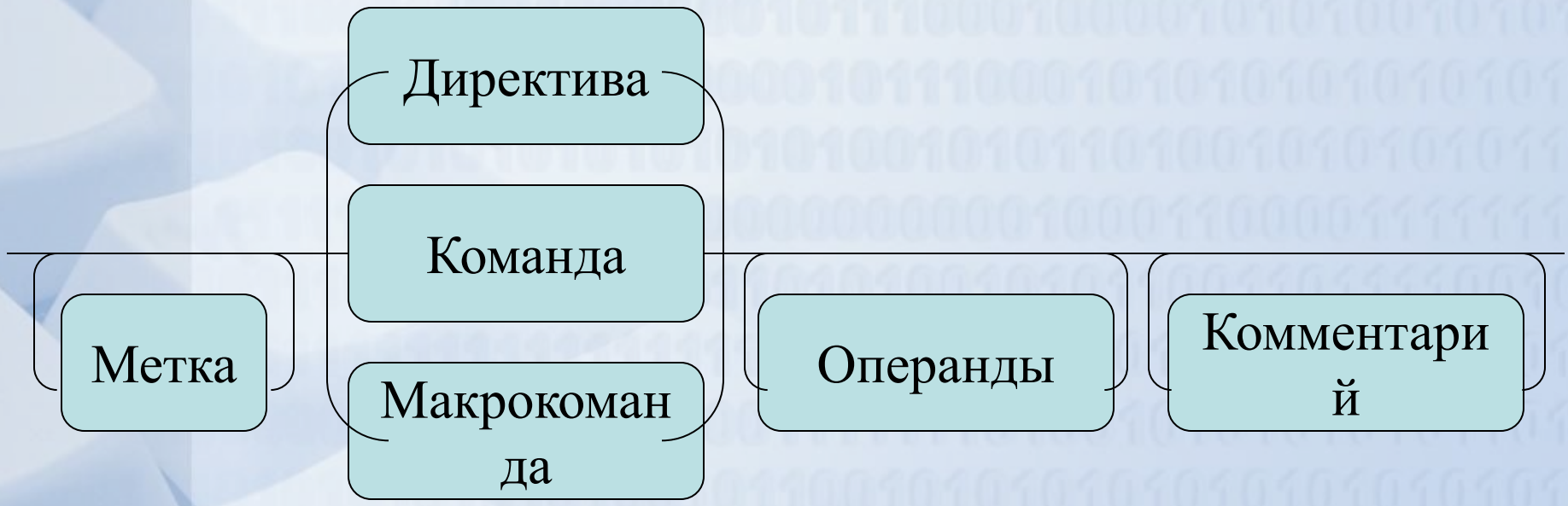
Флаги состояния

№ бита	Мнемоника	Флаг	Содержание и назначение
0	cf	Carry Flag	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.
2	pf	Parity Flag	Этот флаг - только для 8 младших разрядов операнда любого размера. 1, когда 8 младших разрядов результата содержат четное число единиц; 0, когда 8 младших разрядов результата содержат нечетное число единиц.
4	af	Auxiliary carry Flag	Только для команд, работающих с BCD-числами: 1- в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде; 0-переносов не было.
6	zf	Zero Flag	1 - результат нулевой; 0 - результат ненулевой.
7	sf	Sign Flag	Отражает состояние старшего бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно).
11	of	Overflow Flag	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.
12	iopl	Input/Output Privilege Level	Используется в защищенном режиме работы микропроцессора для контроля доступа к командам ввода-вывода в зависимости от уровня привилегий задачи.
13			
14	nt	Nested Task	Используется в защищенном режиме работы микропроцессора для фиксации того факта, что одна задача вложена в другую.

Системные флаги

№ бита	Мнемоника	Флаг	Содержание и назначение
8	tf	Trace Flag	Предназначен для организации пошаговой работы микропроцессора: 1- микропроцессор генерирует прерывание с номером 1 после выполнения каждой машинной команды. Может использоваться при отладке программ, в частности отладчиками; 0 - обычная работа
9	if	Interrupt enable Flag	Предназначен для маскирования аппаратных прерываний (прерываний по входу INTR): 1- аппаратные прерывания разрешены; 0- аппаратные прерывания запрещены
16	rf	Resume Flag	Используется при обработке прерываний от регистров отладки
17	vm	Virtual 8086 Mode	Признак работы микропроцессора в режиме виртуального 8086: 1 - процессор работает в режиме виртуального 8086; 0 - процессор работает в реальном или защищенном режиме.
18	ac	Alignment Check	Предназначен для разрешения контроля выравнивания при обращениях к памяти.
Флаг управления			
10	df	Directory Flag	Определяет направление поэлементной обработки в цепочечных командах: 0 - от начала строки к концу; 1 - от конца строки к ее началу.

Формат инструкции на языке ассемблера



Директива описания сегмента

Имя SEGMENT $[_1]$ $[_2]$ $[_3]$ $[_4]$

<инструкции языка>

Имя ENDS

Здесь $[_1]$ - тип выравнивания

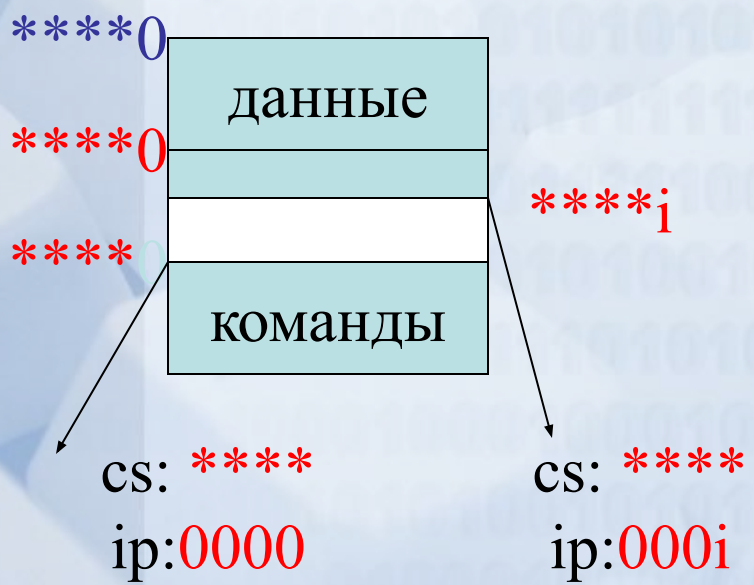
$[_2]$ - тип объединения

$[_3]$ - класс

$[_4]$ - размер адреса (для i386 и выше)

Имя – константа, содержащая номер параграфа начала сегмента

- Тип выравнивания
  - ✔ BYTE x 1
  - ✔ WORD x 2
  - ✔ DWORD x 4
  - ✔ PARA x 16
  - ✔ PAGE x 256
  - ✔ MEMPAGE x 1024



- Тип объединения
  - ✔ PRIVATE
  - ✔ PUBLIC (MEMORY)
  - ✔ COMMON
  - ✔ STACK
  - ✔ AT ****
- Размер адреса
  - ✔ USE16
  - ✔ USE32
- Класс

  ‘имя_класса’

Упрощенные директивы определения сегмента

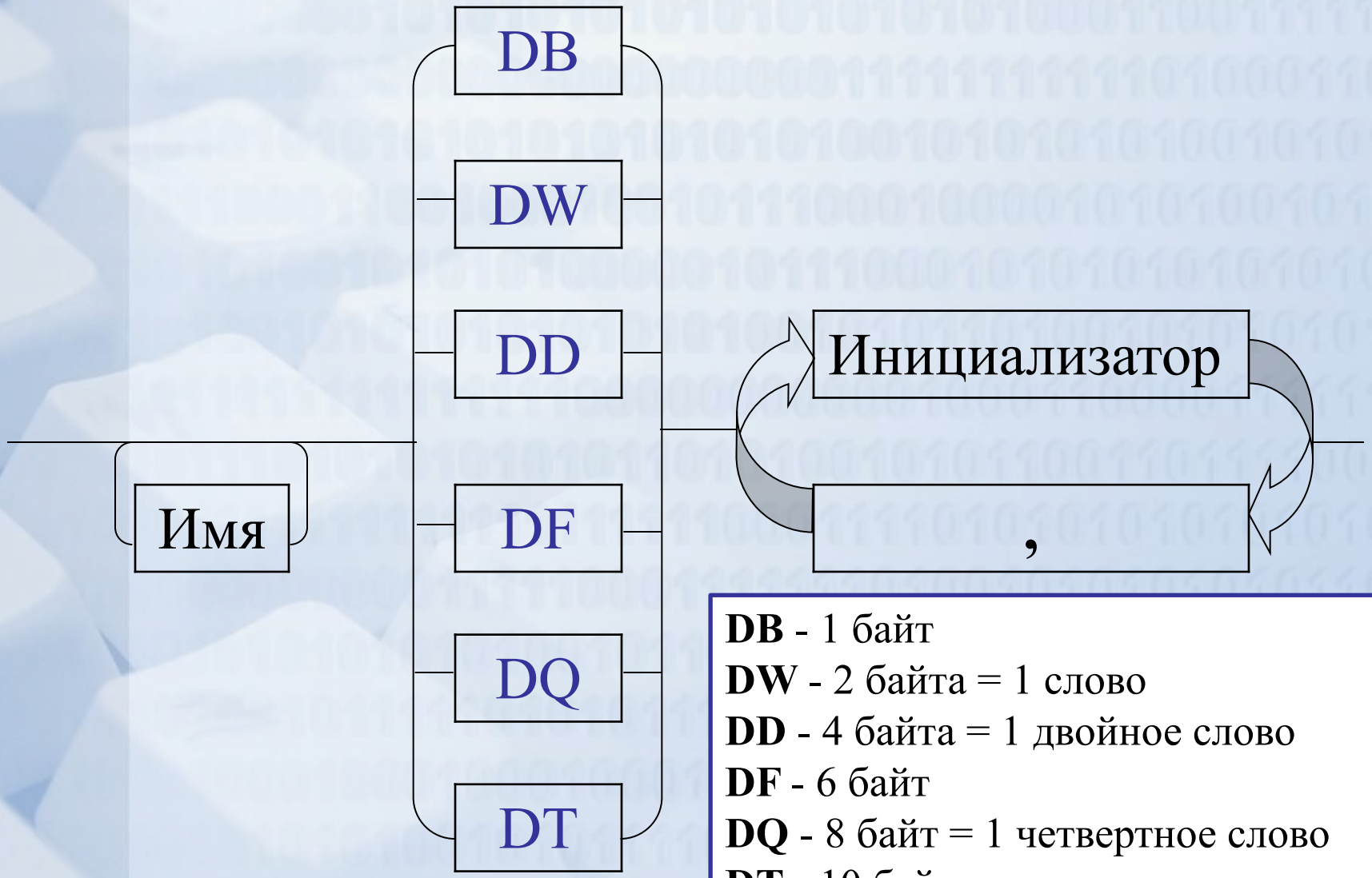
Формат директивы	Назначение
.CODE [имя]	Начало или продолжение сегмента кода
.DATA	Начало или продолжение сегмента инициализированных данных.
.STACK [размер]	Начало или продолжение сегмента стека модуля. Пара-метр [размер] задает размер стека

```
data segment
    ; add your data here!
ends

stack segment
    dw   128  dup(0)
ends
```

```
code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax
    ; add your code here
ends
end start
```

Директивы описания данных



DB - 1 байт
DW - 2 байта = 1 слово
DD - 4 байта = 1 двойное слово
DF - 6 байт
DQ - 8 байт = 1 четвертное слово
DT - 10 байт

Пример

```
Data segment
    A   db  ?
    B   db  ‘abcd’,4 dup(‘*’)                ; ”abcd****”
    C   dw -1,0,1
    D   dw C
    E   dd  D
    F   db  0fh,15,17q,1111b                 ;4 dup(15)
    G   dd  -1.5
    H   dq  ”hgfedcba$”
Data  ends
```

Символ «?» означает, что значение ячеек не будет определено

Символ «$» означает, что значение ячеек не будет

Основные команды ассемблера

Пересылки данных	Арифметические	Логические
Передачи управления	Обработки цепочек	Управления работой ЦП

Обозначения

В документации по Ассемблеру различные форматы операндов представлены следующими аббревиатурами:

- SR – сегментный регистр
- r8, r16, r32 – регистр общего назначения
- m8, m16,m32 – адрес области памяти
- i8,i16,i32 – непосредственное значение (константа)

Команды пересылки данных

Общего назначения	Работы с адресами	Работы со стеком	Преобразования данных
Mov	Lea	Puch	Xlat
Xchg	Lss	Pop	
	Lds	Pucha	
	Les	Popa	
	Lfs	Puchf	
	Lgs	Popf	

Инструкция MOV

Команда **MOV,** хоть название ее и происходит от слова «move» (перемещать), на самом деле не перемещает, а копирует значение из источника в приемник:

MOV *приемник*, *источник*

mov ax,[number]	**заносим значение переменной number в регистр AX**
mov [number],bx	**загрузить значение регистра BX в переменную number**
mov bx,cx	**занести в регистр BX значение регистра CX**
mov al ,1	**занести в регистр AL значение 1**
mov dh,cl	**занести в регистр DH значение регистра CL**
mov esi,edi	**копировать значение регистра EDI в регистр ESI**
mov word [number],1	**сохранить 16-битное значение 1 в переменную "number"**

Инструкция MOV

Процессоры семейства x86 позволяют использовать в командах только один косвенный аргумент. Следующая команда копирования значения, находящегося по адресу **number_one**, в область памяти с адресом **number_two**, **недопустима**:

```
mov [number_two], [number_one]    ;НЕПРАВИЛЬНО!!!
```

Чтобы скопировать значение из одной области памяти в другую, нужно использовать промежуточный регистр:

```
mov ax, [number_one]      ;загружаем в AX 16-битное значение
                          ;"number_one"
mov [number_two], ax      ;а затем копируем его в переменную
                          ;"number_two"
```

Оба операнда команды MOV должны быть одного размера:

```
mov ax, bl          ;НЕПРАВИЛЬНО! - Операнды разных размеров.
```

Инструкция MOV

Для копирования значения BL в регистр AX мы должны «расширить диапазон», то есть скопировать весь BX в AX, а затем загрузить 0 в AX:

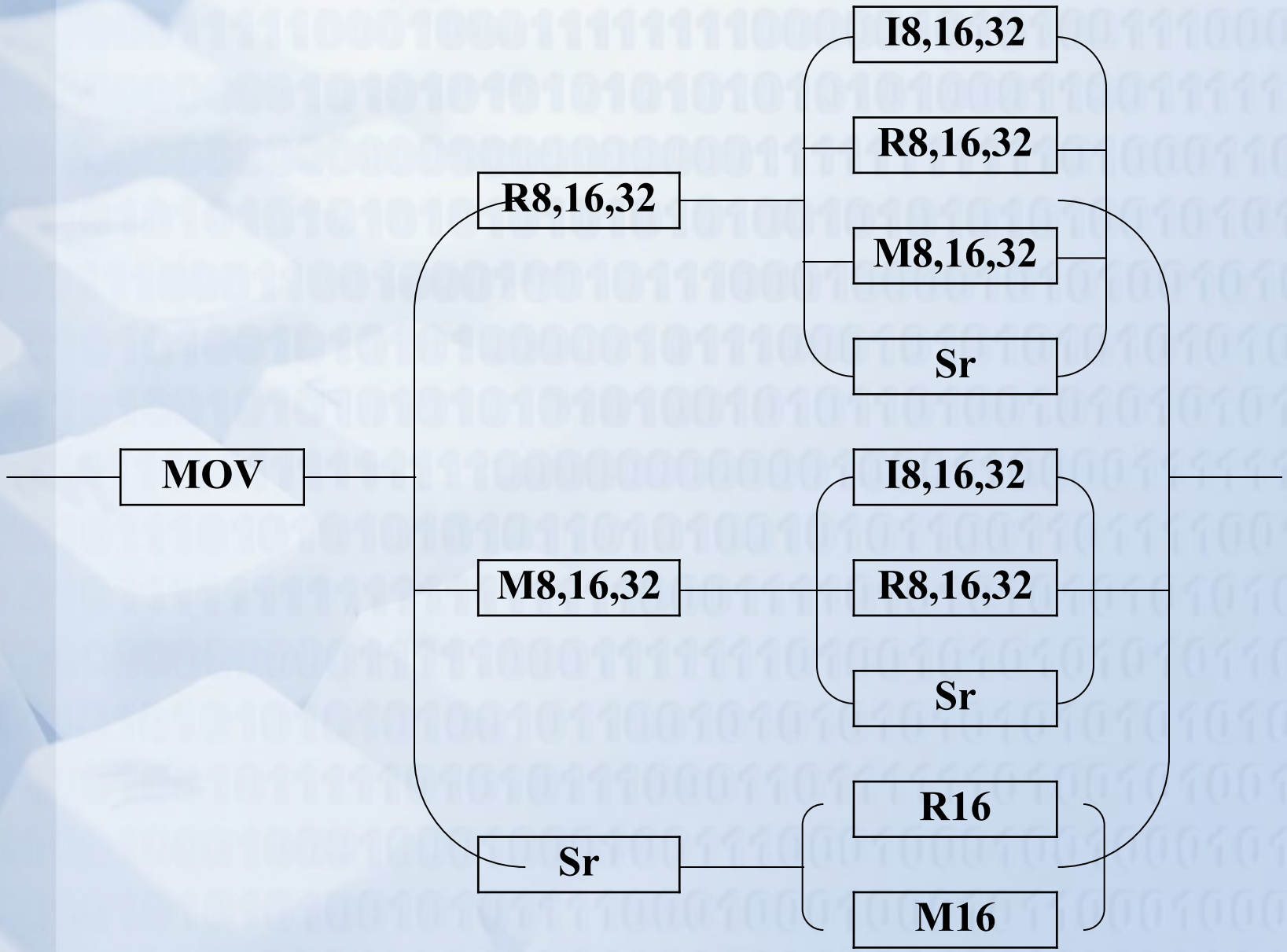
```
mov ax, bx    ;загружаем BX в AX
mov ah, 0     ;''сбрасываем" верхнюю часть AX — записываем в нее O
```

Можно поступить и наоборот: сначала сбросить весь AX, а затем загрузить BL в младшую часть AX (AL):

```
mov ax, O      ;AH = O, AL = O
mov al, bl     ;заносим в AL значение BL
```

Точно так же можно скопировать 16-битное значение в 32-битный регистр.

Инструкция MOV



Организация ввода вывода

Для вывода на экран сообщения, возможно, использовать прерывание 21h.

Вывод строки на экран:

```
mov  ah,  09h  ; поместить в регистр ah номер функции прерывания 21h
mov  dx,  offset str1  ; в регистр dx помещается указатель на строку
int  21h  ;          вызов прерывания 21h
```

Вывод символа на экран (выводимый символ находится в регистре dl):

```
mov  ah,  02h ;  поместить в регистр ah номер функции прерывания 21h
int  21h ;          вызов прерывания 21h
```

Ввод символа с клавиатуры (введенный символ находится в регистре al):