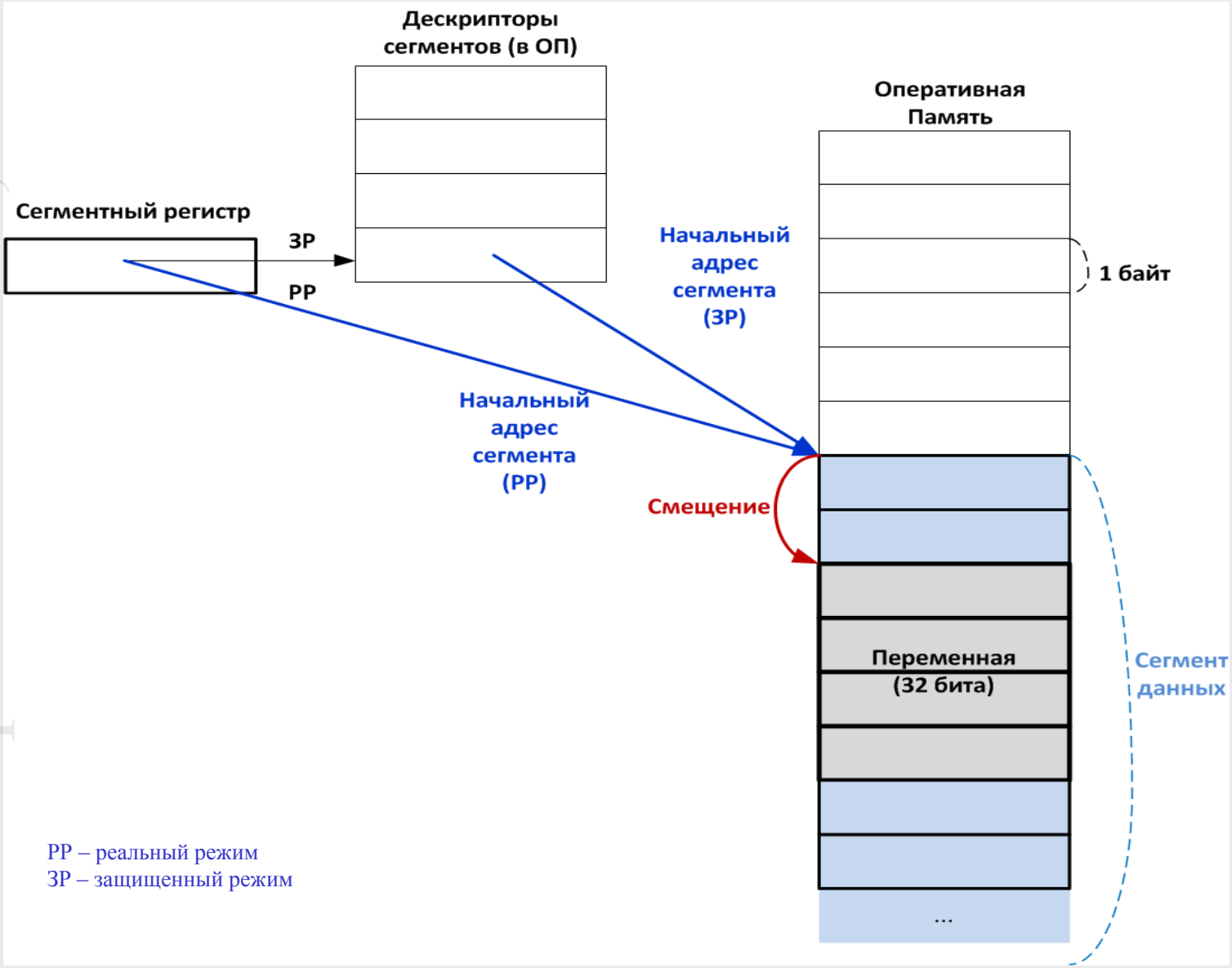
Способы адресации данных и обработка массивов

Общая схема адресации данных в памяти



РР – реальный режим
ЗР – защищенный режим

Общая схема адресации данных в памяти

в 32-х разрядном защищенном режиме (protected mode, далее - ЗР)

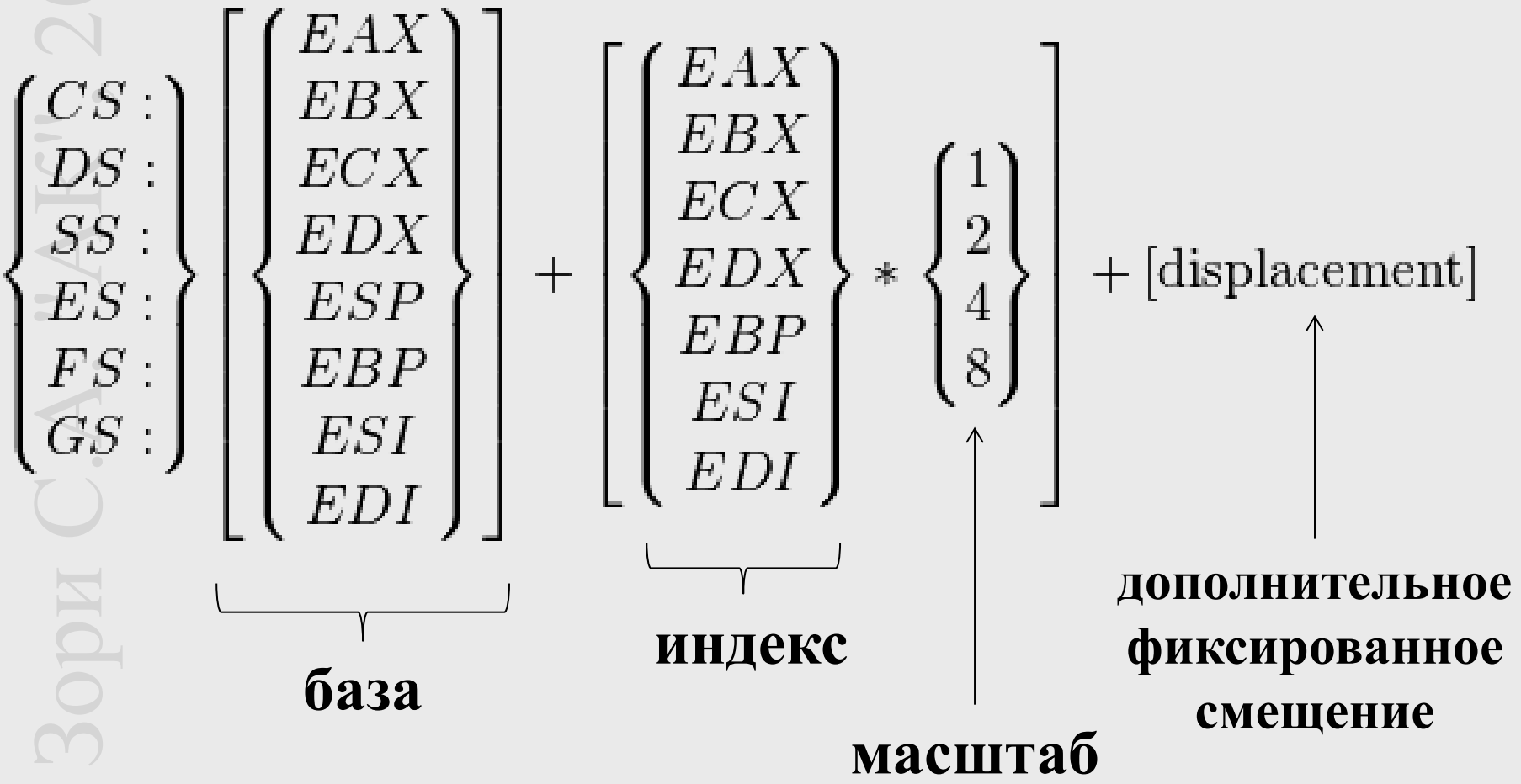
$$\begin{Bmatrix} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{Bmatrix} \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} \right] + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} * \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + [\text{displacement}]$$

16 или 32 бит

или $[метка]$

смещение от начала сегмента

Общая схема адресации данных в памяти



Типы адресации в 32-х битном режиме

тип	формула вычисления адреса	примеры
регистровая (register)	**-**	mov **ax**, **bx**
непосредственная (immediate)	-	mov ax, **2**
прямая (direct)	смещение в метке	mov ax, **variable** jmp **label**
косвенная (register indirect)	смещение в регистре	mov ax, **[esi]** imul **word ptr [esi]**
по базе со смещением (based)	смещение в регистре + const	mov ax, **array[esi]** mov ax, **[esi+2]**

Типы адресации в 32-х битном режиме

тип	формула вычисления адреса	примеры
индексированная (indexed)	смещение = регистр*масштаб + const	mov ax, **array + esi*2** mov ax, **[esi*2] + 2**
по базе с индексированием (base-indexed)	смещение = регистр + регистр + const	mov bx, **arr[eax][esi]** mov bx, **[eax+esi+2]**
по базе с индексированием и масштабированием (base-indexed with scale)	смещение = регистр(база) + регистр(индекс)* масштаб + const	mov bx, **arr[eax+esi*4]** mov bx, **[eax+esi*8+2]**

Сложные способы адресации

- Для любого операнда, находящегося в памяти, выражение для вычисления адреса задается непосредственно в команде
- Упрощает доступ к элементам массивов, матриц и структур при написании формальных процедур обработки данных

Непосредственная

Операнд задан в самой команде

mov ax, 12

Прямая адресация

Прямой адрес - это смещение переменной в памяти относительно любого базового сегмента.

Т.е. ***физический адрес определяется*** как сумма содержимого **сегментного регистра,** используемого в команде (в Реальном режиме, в ЗР – сегментный регистр – указатель на Дескриптор Сегмента, где хранится адрес базы (начала) сегмента), и смещения, **задаваемого меткой** (имя переменной).

Например, для обращения к переменной в сегменте данных:

```
inc MyMoney      ; Прибавить 1 к значению MyMoney
```

* Метка *MyMoney* ассемблируется в адрес смещения, по которому записана переменная (данные). При прямой адресации фиксируется только смещение.

Обычно обращения по прямому адресу выполняются относительно сегмента, адресуемого регистром **ds**. Чтобы это изменить, можно определить **сегментное переназначение** следующим образом:

mov cx, es:0verWord

Эта команда загружает слово с меткой 0verWord, записанный в сегменте, адресуемом es.

```
mov dh, cs:CodeByte      ; dh <- байт из кодового
                                 сегмента
mov dh, ss:StackByte     ; dh<-байт из стекового
                         сегмента
mov dh, ds:DataByte      ; dh<-байт из сегмента
                         данных
```

- Переназначение действует только в пределах одной команды!

Косвенная адресация

Вместо обращения к переменной в памяти по имени можно использовать один из РОН (в базовой системе команд - одного из трех регистров – **ebx**, **esi, edi**) в качестве указателя на данные в памяти.

Т.е. ***физический адрес определяется*** как сумма базового адреса сегмента, адресуемого **сегментным регистром** (используемого в команде) и смещения, задаваемого содержимым регистров **ebx, esi** или **edi***.

```
mov cx, [ebx]          ; Скопировать слово из памяти
                       по адресу [ebx] в cx
dec byte ptr [esi]     ; Уменьшить значение байта в
                       памяти по адресу [esi]
```

Операторы **DWORD PTR**, **WORD PTR** и **BYTE PTR** необходимы, если Assembler не может определить, *что* адресует косвенно регистр в памяти (размер) – данные DD, DW или DB.

В первой строке данные, адресуемые ebx, пересылаются в 16-битовый регистр cx, следовательно, в операторе WORD PTR нет необходимости, так как ассемблер знает размер данных из контекста команды.

Во второй строке необходимо использовать оператор BYTE PTR, так как у ассемблера нет другого способа узнать – уменьшать значение байта, слова или двойного слова.

При косвенной адресации по умолчанию используется сегмент, адресуемый **ds**.

Вы можете использовать переназначения для изменения этого сегмента на любой из других:

```
add WORD PTR es:[ebx], 3  ; Прибавить 3 к слову в es:ebx
dec BYTE PTR ss:[esi]     ; Уменьшить байт по адресу ss:esi
mov cx, cs:[edi]          ; Загрузить слово из сегмента кода
mov ax, es:[dx]
```

Для загрузки <u>смещения</u> переменной (данных) в один из РОН процессора используется команда **LEA**:

```
lea ebx, Person
```

Загрузка адреса

команда	dst	адрес
lea dst, адрес	регистр	любое доступное выражение для вычисления адреса

- Вычисляет эффективный адрес и помещает в регистр dst
- Используется для:
  - загрузки адреса переменной или метки
  - быстрых арифметических вычислений*

Использование адресации для быстрых произвольных вычислений*

- Пример: вычислить AX = AX*2 + CX + 51

```
shl ax, 1
add ax, cx
add ax, 51
```

эквивалентно

```
lea ax, [ax*2+cx+51]
```

* *не использовать в лабораторных работах!*

Косвенная адресация

- Часто используется:
  - для обращению к переменной по ее адресу;
  - для быстрого прохода по элементам любого размера в одномерном массиве

- *В современных процессорах система команд позволяет использовать **любые** **РОН** в качестве указателей!

Пример

записать в i-й элемент массива «13», а следующие 10 обнулить

```
arr  dw 100 dup(?)
i    dd …
...
     mov eax, i
     lea edi, arr[eax*2]
          ; edi – адрес i-го элемента
     mov word ptr [edi], 13
     mov ecx, 10
for:
     add edi, 2
     mov word ptr [edi], 0
     loop for
```

Адресация по базе со смещением

- Часто используется:
  - для обращению к элементу массива *байт* (смещение <- начало массива);
  - для обращению к полю структуры по ее адресу;

- Например: считать Y из структуры

```
struct {
        int X, Y, Z;
        },
```

адрес которой записан в EAX:

```
mov ebx, [eax+4]
```

Адресация по базе со смещением

- Например: обнулить i-й элемент массива байт:

```
arr db 100 dup(?)
i   dd ?
...
    mov eax, i
    mov arr[eax], 0
```

Индексированная адресация

- Используются **индексные РОН**ы как указатель и **масштаб** (смещение масштабируется размером элемента данных)
- Часто используется для обращения к элементам массива размером 2, 4, 8 байт;
- Например: обнулить i-й элемент массива:

```
arr dd 100 dup(?)
i   dd ?
...
    mov esi, i
    mov arr[esi*4], 0
```

Адресация по базе с индексированием и масштабированием (базово-индексная)

- *Некоторые* случаи применения:
  - для обращения к элементу массива или его части, заданной адресом начала;
  - для обработки многомерных массивов;

```
EA = [base_reg + index_reg*scale + displ]
```

Базово-индексная адресация является ***самым мощным методом обращения к памяти***!

Изменяя значения **базового** и **индексного** регистров, и используя **масштабирование** и **дополнительное цифровое смещение**, программа может **адресовать в памяти сложные структуры данных** и создавать **формальные процедуры обработки данных**!

Задание дополнительного смещения в команде

- Компилятор вычисляет константу-смещение в формуле адресации как сумму всех смещений

- Например:

```
mov eax, [ebx + esi*4 + 12]
```

Адресация по базе с индексированием и масштабированием

- Например, обработка массива из N элементов:

```
size      dd …(N)
array     dw …(N) dup(?)

    lea ebx, array
    mov ecx, size
    mov esi, 0     ; i=0
fori:
    mov ax, [ebx + esi*2]
    …              ; обработка i- того элемента
   inc esi         ; i++
    loop fori
```

Примеры

Пример – Сортировка массива слов методом пузырька

- Вариант 1. *Адреса* элементов в ESI и EDI

```
size      dd 100
ar        dw 100 dup(?)

          lea esi, ar
          mov ecx, size
fori:
          push ecx

          mov ax, [esi]
          mov edi, esi
```

Пример – сортировка массива слов


```
forj:
    cmp ax, [edi]
    jle skipxchg
    xchg ax, [edi]
skipxchg:
    add edi, 2
    loop forj

    pop ecx
    mov [esi], ax

    add esi, 2
    loop fori
```

Пример – сортировка массива слов

- Вариант 2. *Индекс* элемента в ECX

```
size    dd 100
ar      dw 100 dup(?)

    mov ecx, size
fori:
    push ecx

    mov ax, ar[ecx*2-2]
```

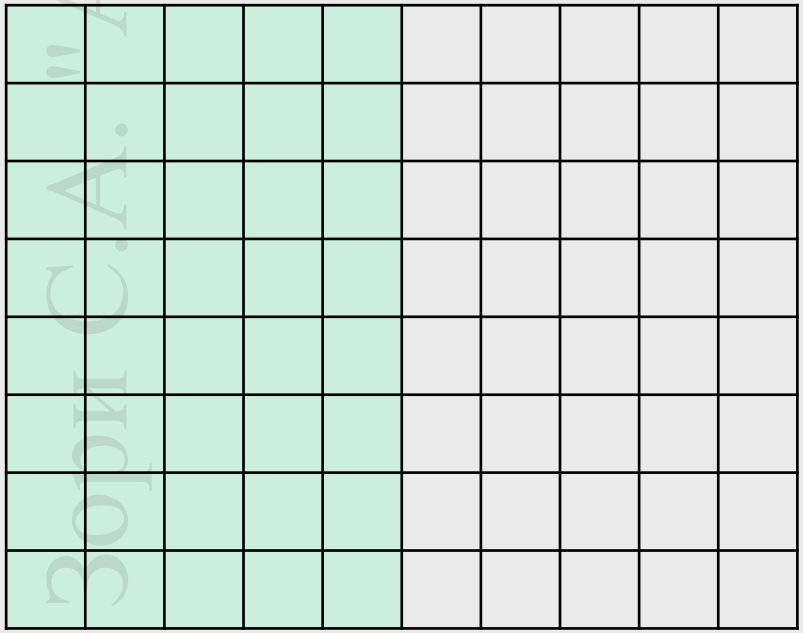
Пример – сортировка массива слов


```
forj:
        cmp ax, ar[ecx*2-2]
        jge skipxchg
        xchg ax, ar[ecx*2-2]
skipxchg:

        loop forj

        pop ecx
        mov ar[ecx*2-2],ax


        loop fori
```

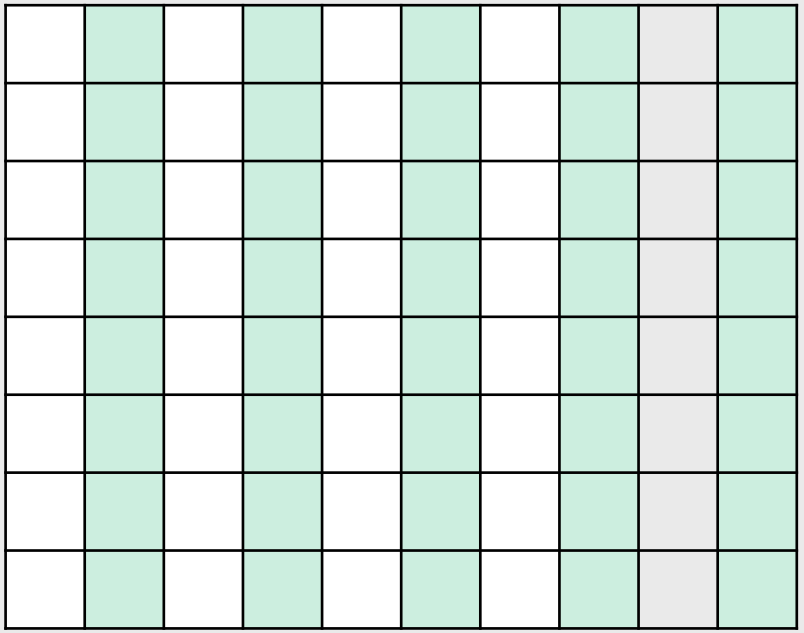
Пример – обработка матриц

- Скопировать нечетные столбцы матрицы слов B в левую половину матрицы A

A



B



Пример – обработка матриц

```
; размер – 8*10
a    dw 80 dup(?)
b    dw 80 dup(?)

     xor esi, esi ; смещение начала
                  ; строки в массиве

     ; цикл по строкам
     mov ecx, 8
for_row:
     push ecx
```

Пример – обработка матриц


```
    ; цикл по столбцам
    mov ecx, 5    ; номер столбца в A
                  ; начиная с 1
for_col:
    mov ax, b[esi][ecx*4-2]
    mov a[esi][ecx*2-2], ax
    loop for_col

    add esi, 20 ; размер строки в байтах
    loop fori
```