

ЭВМ и Периферийные устройства

лекция 7

MOVZX. Расширение без знака.

MOVZX приемник, источник

MOV with zero extend

Перенос в **приемник источника** с заполнением старших битов **нулями всегда**. Команду имеет смысл применять для беззнаковых чисел. Для чисел со знаком вы рискуете потерять знак.

MOVSX. Расширение со знаком.

MOVSX приемник, источник

MOV with sign extend

Перенос в **приемник источника** с заполнением старших битов с **учётом знака**. Команда применяется для знаковых чисел, поскольку так вы не потеряете знак.

LEA.

LEA (Load Effective Address) – команда помещения адреса.

LEA приемник, источник

Команда похожа на mov. Однако она используется исключительно для помещения адреса в указанный регистр.

Приемник – регистр, куда помещается адрес. Именно регистр.

Источник – источник адреса (то есть у чего берем адрес). Под источником подразумевается память.

LEA dx, arr; поместить в dx адрес переменной arr

Можно ещё и так:

MOV dx, offset arr; поместить в dx адрес переменной arr

OFFSET.

OFFSET– команда помещения адреса.

```
MOV dx, offset arr; поместить в dx адрес переменной arr
```

LEA и OFFSET

LEA и OFFSET делают одно и тоже – получают адрес чего-либо. Однако LEA, будучи отдельной командой даёт больше возможностей.

Пусть дан однобайтовый массив arr. Также пусть в bx хранится какой-то номер элемента массива arr. Мы хотим, чтобы в dx оказался адрес номера элемента, хранящегося в bx.

С помощью OFFSET:

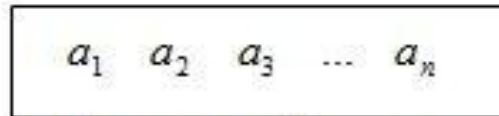
```
mov dx, offset arr  
add dx, bx
```

С помощью LEA:

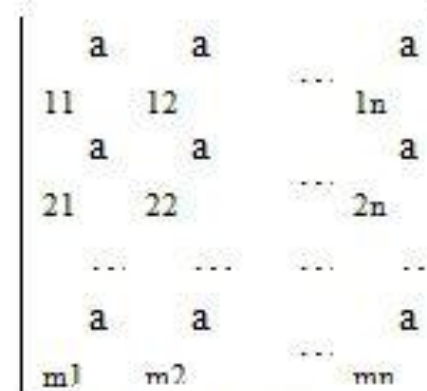
```
lea dx, [arr + bx]
```

Массивы

Массив – куча однотипных чисел в памяти, идущих по порядку.



Одномерный массив



Двухмерный

В Ассемблере специального типа «массив» не существует.

Массивы. Объявление.

1. Перечислением элементов массива в поле операндов одной из директив описания данных. При перечислении элементы разделяются запятыми.

```
;массив из 5 элементов.Размер каждого элемента 4 байта:  
mas dd 1,2,3,4,5
```

```
;массив из 15 элементов. Или массив 3x5. Это как вы решите  
table db 10h, 20h, 30h, 40h, 50h  
      db 60h, 70h, 80h, 90h, 0A0h  
      db 0B0h, 0C0h, 0D0h, 0E0h, 0F0h
```

2. Используя оператор повторения **dup**.

```
;массив из 5 нулевых элементов.  
;Размер каждого элемента 2 байта:  
mas dw 5 dup (0)
```

```
;массив из 32 необъявленных элементов, каждый из которых размером 2  
; байта.  
;Ну или массив 4x8 . Чем его считать - ваше дело  
TwoD dw 4 dup (8 dup (?))
```


Массивы. Объявление.

3. Используя директивы **label** и **rept**. Директива **rept** относится к макросредствам языка ассемблера и вызывает повторение указанное число раз строк, заключенных между директивой и строкой **endm**. К примеру, определим массив байт в области памяти, обозначенной идентификатором *mas_b*. В данном случае директива **label** определяет символическое имя *mas_b*, аналогично тому, как это делают директивы резервирования и инициализации памяти.

```
...  
n=0  
...  
mas_b label byte  
mas_w label word  
rept      4  
    dw 0f1f0h  
endm
```

В результате в памяти будет создана последовательность из четырех слов **f1f0**. Эту последовательность можно трактовать как массив байт или слов в зависимости от того, какое имя области мы будем использовать в программе — *mas_b* или *mas_w*.

Массивы. Объявление.

4. Массив можно задать программно.

```
.data
mas db 10 dup (?) ;исходный массив
i db 0 ;переменная i со значением 0
...
mov ecx,10
go: ;цикл инициализации
    mov bh,i      ;i в bh
    mov mas[si],bh ;запись в массив i
    inc i         ;инкремент i
    inc si        ;продвижение к следующему ;элементу массива
loop go ;повторить цикл
```

Массивы. Базово-индексный режим адресации.

Итоговый адрес при базово индексном режиме адресации складывается из значения двух регистров, один из которых называется **базовым**, другой- **индексным**. Для организации подобного режима адресации может быть использована пара любых 32-разрядных регистров общего назначения.

```
.data
array dw 1000h,2000h,3000h
...
.code
...
mov ebx, OFFSET array
mov esi,2
mov ax,[ebx+esi]          ;AX = 2000h

mov edi, OFFSET array
mov ecx,4
mov ax,[edi+ecx]         ;AX = 3000h

mov ebp, OFFSET array
mov esi,0
mov ax, [ebp+esi]        ;AX = 1000h
```

Массивы. Базово-индексный режим адресации со смещением.

При использовании данного режима адресации для вычисления адреса к содержимому базового и индексного регистров прибавляется дополнительное смещение. Есть 2 варианта записи подобного способа адресации

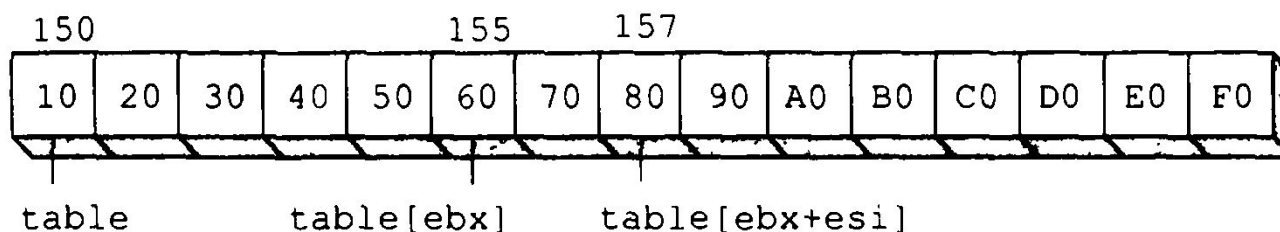
[смещение + база + индекс]
смещение [база + индекс]

Вместо смещения обычно указывается либо имя переменной, либо константное выражение. В качестве базового и индексного регистров может использоваться любой 32-разрядных регистр общего назначения.

Массивы. Базово-индексный режим адресации со смещением.

Возьмём пример с двумерным массивом table. Пусть смещение table равно 150:

```
.data
tabledb 10h, 20h, 30h, 40h, 50h
        db 60h, 70h, 80h, 90h, 0A0h
        db 0B0h, 0C0h, 0D0h, 0E0h, 0F0h
NumCols=5
..
.code
...
mov ebx, NumCols      ;Смещение строки
mov esi,2             ; Номер столбца
mov al, table[ebx+esi] ;[150+5+2]=[157]
                    ;AL=80h
```



Массивы. Масштабирование индексной адресации.

Микропроцессор позволяет *масштабировать* индекс. Это означает, что если указать после имени индексного регистра знак умножения “*” с последующей цифрой 2, 4 или 8, то содержимое индексного регистра будет умножаться на 2, 4 или 8, то есть *масштабироваться*.

Применение масштабирования облегчает работу с массивами, которые имеют размер элементов, равный 2, 4 или 8 байт, так как микропроцессор сам производит коррекцию индекса для получения адреса очередного элемента массива.

.386

```
...  
mas dw 0,1,2,3,4,5  
...  
mov esi,3 ;поместить 3-й элемент массива mas в регистр ax  
mov ax,mas[esi*2]
```

Массивы. Одномерный массив.

Доступ к элементу массива можно получить, зная адрес памяти.

В общем случае для получения адреса элемента в одномерном массиве необходимо начальный (базовый) адрес массива сложить с произведением индекса (i) (номер элемента минус единица) этого элемента на размер элемента массива. Для одномерного массива это можно сделать так:

база + i *размер_элемента_в_байтах

```
mas dw 0,1,2,3,4,5
...
mov esi, offset mas
mov ax,[esi+2*2] ;поместить 2-й элемент (да, 2-й) массива mas в регистр ax
mov ax, mas[2*2] ; можно и так
```

Размер элементов массива вы должны учитывать самостоятельно. В [] вы указываете смещение в **байтах**.

Массивы. Двухмерный массив

Элементы двухмерного массива располагаются в памяти также последовательно.

Например, пусть имеется массив чисел (размером в 1 байт) $mas(i, j)$ с размерностью 4 на 4

($i = 0 \dots 3, j = 0 \dots 3$):

В нашем представлении это выглядит так:

23	04	05	67
05	06	07	99
67	08	09	23
87	09	00	08

В памяти это выглядит так:

23 04 05 67 05 06 07 99 67 08 09 23 87 09 00 08

Массивы. Двухмерный массив.

А что если у нас двухмерный массив? Пусть $i = 0 \dots n-1$ указывает номер строки, а $j = 0 \dots m-1$ указывает номер столбца. n – количество строк в массиве, m – количество столбцов.

Адрес нужного элемента можно вычислить по формуле:

база + (количество_элементов_в_строке * i + j) * размер_элемента

Доступ к двумерному массиву удобно организовывать как к одномерному, но с хитро вычисленным адресом с использованием масштабирования

```
.data
TwoD      dw 4 dup (8 dup (?))
i         integer ?
j         integer ?
```

```
.code
; Мы хотим TwoD[i,j] := 5

mov     eax, 8      ; 8 столбцов в строке
mul     l          ; номер строки
add     ax, j      ; номер столбца
mov     TwoD[eax*2], 5 ; «*2» масштабирование на 2 байта (слово)
```

Получение элемента массива

Двухмерный массив:

$$\text{Element_Address} = \text{Base_Address} + (\text{rowindex} * \text{col_size} + \text{colindex}) * \text{Element_Size}$$

Трехмерный массив

$$\text{Element_Address} = \text{Base} + ((\text{rowindex} * \text{col_size} + \text{colindex}) * \text{depth_size} + \text{depthindex}) * \text{Element_Size}$$

Четырехмерный массив:

$$\text{Element_Address} = \text{Base} + (((\text{rowindex} * \text{col_size} + \text{colindex}) * \text{depth_size} + \text{depthindex}) * \text{Left_size} + \text{Leftindex}) * \text{Element_Size}$$

```
.data
TwoD      dw 4 dup (8 dup (?))
i         integer ?
j         integer ?
```

```
.code
; Мы хотим TwoD[i,j] := 5; :
```

```
mov     eax, 8      ; 8 столбцов в строке
mul     i          ; номер строки
add     ax, j      ; ; номер столбца
mov     TwoD[eax*2], 5 ; «*2» масштабирование на 2 байта (слово)
```

Структуры

Структура это набор переменных (данных). Структура задаётся с помощью директивы `struct` и `ends`. Перед использованием структуры её нужно описать:

```
SOMESTRUCTURE STRUCT
dword1 dd ?
dword2 dd ?
some_word dw ?
abyte db ?
anotherbyte db ?
SOMESTRUCTURE ENDS
```

Уже после можно объявлять её конкретные экземпляры.

Структуры

Структуры можно объявлять как в секции `.data`, так и в секции

```
MYSTRUCT struc  
dword1 dd ?  
dword2 dd ?  
some_word dw ?  
abyte db ?  
anotherbyte db ?  
MYSTRUCT ends
```

```
.data  
msg1 MYSTRUCT <?>  
.data  
msg2 MYSTRUCT <?>
```

Структуры

```
MYSTRUCT struc  
dword1 dd ?  
dword2 dd ?  
some_word dw ?  
abyte db ?  
anotherbyte db ?  
MYSTRUCT ends
```

Для того чтобы получить доступ к записи надо указать метку переменной, которой она обозначена и через точку указать имя поля.

```
mov [msg.dword1], 45h  
xor eax,eax  
mov  eax, [msg.dword1] ; eax = 45
```

при этом запись `msg.dword1` считается обычной меткой данных: берётся смещение метки `msg` плюс смещение поля `dword1` в структуре, размер данных по умолчанию равен размеру директивы указанной после метки поля. Также можно пользоваться обращением к полю при обращении к записи через регистр:

```
mov [msg.dword2], 45h  
xor eax,eax  
lea ebx, msg  
mov  eax, [ebx].dword2 ; eax = 45
```

Структуры

```
MYSTRUCT struc  
dword1 dd ?  
dword2 dd ?  
some_word dw ?  
abyte db ?  
anotherbyte db ?  
MYSTRUCT ends
```

Если имя поля не гарантирует уникальности то лучше использовать такой тип использования записи:

```
mov [msg.dword2], 45h  
xor eax,eax  
lea ebx, msg  
mov eax, [ebx].MYSTRUCT.dword2 ; eax = 45
```

Указанная запись гарантирует, что мы получаем доступ к нужному нам полю, нужной нам структуры.

Структуры

```
MYSTRUCT struc  
dword1 dd ?  
dword2 dd ?  
some_word dw ?  
abyte db ?  
anotherbyte db ?  
MYSTRUCT ends
```

Как и всё остальное, структуры- это всего-лишь данные в памяти. Поэтому вместо обращения по конкретным именам, можно использовать смещение в памяти

```
mov [msg.abyte], 45h  
xor eax,eax  
lea ebx, msg  
mov al, [ebx+10d] ; al = 45
```

к ebx прибавлено 10d, потому что смещение поля abyte в структуре равно 10d.