# БГТУ, ФИТ, ПОИТ, 3 семестр, Языки программирования Введение в язык Ассемблер

## 1. Косвеная адресация (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP).

Чаще всего используются регистры:

**ESI** (индекс источника)

**EDI** (индекс получателя).

## 1.1 Прямая адресация

В ассемблере прямая адресация возможна в том случае, если переменной присвоена метка.

Пример прямой адресации:

MAS DB 'HELLO'

MOV AL, MAS ;содержимое байта с именем MAS загружается в AL -> AL='H'

**Имя** переменной (метка MAS) — значение, соответствующее смещению данной переменной относительно начала сегмента, в котором она размещена. Прямую адресацию неудобно применять при обработке массивов, т.к. каждому элементу массива невозможно присвоить собственную метку.

# <u> 1.2 Косвенная адресация</u>

Адресуемая память:

необходимо **заранее** загрузить относительный адрес с помощью оператора **offset** (смещение) обрабатываемой области памяти в РОН.

При косвенной адресации в качестве *указателя* на текущий элемент массива используется один из 32-разрядных регистров общего назначения (РОН):

EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

Синаксис:

[<имя регистра>]

Для перехода с следующему элементу массива достаточно увеличить значение указателя на *долину* элемента массива.

Такая адресация называется **ковенной**, а регистр, в котором хранится адрес эдемента массива, называется **косвенным операндом** (indirect operand).

### Пример косвенной адресации:

```
.DATA
                                 ; сегмент данных
ddMS
             1,2,3,4,5,6,7
        dd
ddMD
        dd
             7 dup(?)
.CODE
                                 ; сегмент кода
main PROC
                                 ; точка входа main
                                 ; смещение ddMS -> esi (косвенный операнд)
   mov esi, offset ddMS
    mov edi, offset ddMD
                                 ; смещение ddMD -> edi (косвенный операнд)
    mov eax, [esi]
                                 ; 4 байта по адресу из esi (косвенная адресация) -> eax
    mov [edi], eax
                                 ; значение из eax -> по адресу в edi (косвенная адресация)
    add esi,4
                                 ; настраиваем указатель на следующий элемент массива ddMS
    add edi,4
    mov eax, [esi]
    mov [edi], eax
                                             Контрольные значения 1
                                             Поиск (Ctrl+E)
    add esi,4
                                                             Значение
    add edi,4
                                                *(&ddMD+0)
    mov eax, [esi]
                                                  *(&ddMD+1)
    mov [edi], eax
                                                *(&ddMD+2)
    push 0
                                 ; код возврата процесса Windows(параметр ExitProcess)
    call ExitProcess
                                 ; так завершается любой процесс Windows
```

В регистр ESI загружается смещение массива из 7 элементов ddMS (каждый элемент типа двойное слово = 4 байта; инициализирован целочисленными значениями 1, 2, 3, 4, 5, 6, 7, 8; длина массива = 7\*4 байтов).

В регистр ЕВІ загружается смещение массива из 7 элементов ddMD (4 байта).

Команда MOV загружает 4 байта в регистр EAX (приемник). Второй операнд (источник) — косвенный операнд, в котором хранится смещение первого элемента массива ddMS.

Добавив (ADD) к указателю (ESI) длину элемента массива (4 байта) получим доступ к следующему элементу массива.

Пример перемещает значения типа WORD из массива dwMS в массив dwMD. Значение указателей ESI и EDI в этом случае увеличивается на 2 (длина элемента массива):

```
.const
                            ; сегмент констант
.data
                             ; сегмент данных
dwMS
       dw 1,2,3,4,5,6,7
dwMD dw 7 dup(?)
.code
                             ; сегмент кода
main PROC
                             ; начало процедуры
  mov esi, offset dwMS ; смещение ddMS -> esi
mov edi, offset dwMD ; смещение ddMD -> edi
  mov ax, [esi]
                            ; 2 байта по адресу в esi -> ax
  mov [edi], ax
                             ; ax-> по адресу в edi
  add esi, 2
  add edi, 2
  mov eax, [esi]
                            ; 2 байта по адресу в esi -> ax
  mov [edi], eax
                             ; ex-> по адресу в edi
   add esi, 4
   add edi, 4
   mov eax, [esi]
                             ; 2 байта по адресу в esi -> ax
  mov [edi], eax
                             ; ax-> по адресу в edi
              Имя
                                  Значение
                *(&dwMD+0)
                                  1
                                            процесса (параметр ExitProcess )
  push 0
                *(&dwMD+1)
                                  2
   call ExitP
                                           заканчиваться любой процесс Windows
                *(&dwMD+2)
                                  3
main ENDP
                                           дуры
                             ; конец модуля, main - точка входа
end main
```

### Пример для однобайтовых массивов:

```
byte 1,2,3,4,5,6,7
bMD
      byte 7 dup(?)
.code
                            ; сегмент кода
main PROC
                            ; начало процедуры
  mov esi, offset bMS
                           ; смещение ddMS -> esi
  mov edi, offset bMD
                           ; смещение ddMD -> edi
  mov al, [esi]
                            ; 1 байт по адресу в esi -> al
  mov [edi], al
                            ; al-> по адресу в edi
  inc esi
                            ; ++esi
  inc edi
                           ; ++edi
  mov al, [esi]
                            ; 1 байта по адресу в esi -> al
  mov [edi], al
                            ; al-> по адресу в edi
  inc esi
                            ; ++esi
  inc edi
                            ; ++edi
  mov al, [esi]
                            ; 1 байт по адресу в esi -> al
  mov [edi], al
                            · al-> no annecy в edi
                  Имя
                                     Значени
                                      1 '\x1'
                    2 '\x2'
                     *(&bMD+1)
                                    3'\x3' __ecca (параметр ExitProcess )
  push 0
                    *(&bMD+2)
                            ; так должен заканчиваться любой процесс Windows
   call ExitProcess
main ENDP
                            ; конец процедуры
end main
                            ; конец модуля, main - точка входа
```

Пример. Использование косвенной адресации для нахождения суммы первых 3-х элементов массива ddMS:

```
.model flat,stdcall
                           ; модель памяти, соглашение о вызовах
includelib kernel32.lib
                          ; компановщику: компоновать с kernel32.lib
ExitProcess PROTO :DWORD ; прототип функции
.stack 4096
                           ; сегмент стека объемом 4096
.const
                           ; сегмент констант
.data
                           ; сегмент данных
ddMS
     dd 1,2,3,4,5,6,7
ddMD byte 7 dup(?)
.code
                            ; сегмент кода
main PROC
                            ; начало процедуры
  mov esi, offset ddMS
                         ; смещение ddMS -> esi
  mov eax, [esi]
  add esi,4
                         Имя
                                             Значение
  add eax, [esi]
                           eax
                                             6
  add esi,4
  add eax, [esi]
                           ; код возрата процесса (параметр ExitProcess )
  push 0
  call ExitProcess
                            ; так должен заканчиваться любой процесс Windows
main ENDP
                            ; конец процедуры
end main
                            ; конец модуля, main - точка входа
```

# 1.3 Косвенная адресация. Операнды с индексом.

Синтаксис первой формы представления:

имя\_переменной[индексный\_регистр]

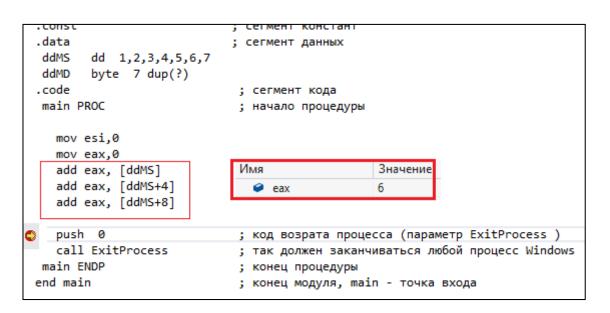
```
ExitProcess PROTO
                   :DWORD ; прототип функции
                           ; сегмент стека объемом 4096
.stack 4096
.const
                           ; сегмент констант
.data
                           ; сегмент данных
ddMS
      dd 1,2,3,4,5,6,7
ddMD byte 7 dup(?)
.code
                            ; сегмент кода
main PROC
                            ; начало процедуры
  mov esi,0
  mov eax,0
  add eax, ddMS[esi]
  add esi,4
  add eax, ddMS[esi]
  add esi,4
  add eax, ddMS[esi]
                                   Значение оцесса (параметр ExitProcess )
  push 0
               Имя
  call ExitProc
                                             нчиваться любой процесс Windows
                  eax
main ENDP
end main
                            ; конец модуля, main - точка входа
```

## 1.4 Косвенная адресация. Операнды с индексом.

Синтаксис второй формы представления:

[имя\_переменной+индексный\_регистр]

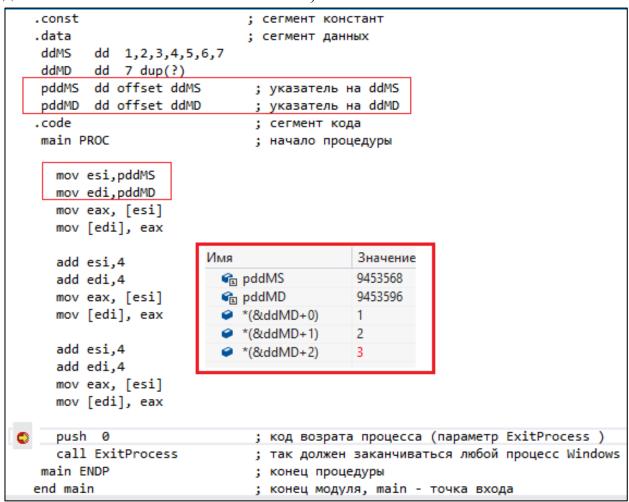
```
.stack 4096
                            ; сегмент стека объемом 4096
.const
                            ; сегмент констант
.data
                            ; сегмент данных
ddMS
       dd 1,2,3,4,5,6,7
ddMD
       byte 7 dup(?)
.code
                             ; сегмент кода
main PROC
                             ; начало процедуры
  mov esi,0
   mov eax,0
   add eax, [ddMS]
   add esi,4
  add eax, [ddMS+esi]
   add esi,4
  add eax, [ddMS+esi]
                      eax
   push 0
                                            роцесса (параметр ExitProcess )
   call ExitProcess
                             ; так должен заканчиваться любой процесс Windows
main ENDP
                             ; конец процедуры
end main
                             ; конец модуля, main - точка входа
```



#### 2. Указатели

Указателем называется переменная, содержащая адрес другой переменной.

Запись указателя с оператором **OFFSET** (возвращает смещение метки данных относительно начала сегмента):



### 3. Команды переходов

После загрузки программы в память процессор начинает автоматически выполнять последовательность ее команд. При этом счетчик команд (EIP) автоматически изменяется на длину выполненной команды и всегда указывает на адрес следующей команды. Изменить порядок следования команд можно с помощью команд передачи управления.

3.1 Команда **JMP** – команда безусловной передачи управления на другой участок кода программы по метке.

Синтаксис:

**ЈМР** метка\_перехода

```
.data
                            ; сегмент данных
 ddMS
       dd 1,2,3,4,5,6,7
       dd 7 dup(?)
ddMD
 pddMS dd offset ddMS
                            ; указатель на ddMS
pddMD dd offset ddMD
                            ; указатель на ddMD
.code
                            ; сегмент кода
main PROC
                             ; начало процедуры
  mov esi,pddMS
   mov edi,pddMD
   mov eax, [esi]
   mov [edi], eax
   jmp
         L1
                             ; переход по адресу L1
   add esi,4
   add edi,4
   mov eax, [esi]
   mov [edi], eax
                             ; метка
L1:
   add esi,4
   add edi,4
   mov eax, [esi]
   mov [edi], eax
                             ; код возрата процесса (параметр ExitProcess )
   push 0
                             ; так должен заканчиваться любой процесс Windows
   call ExitProcess
 main ENDP
                             ; конец процедуры
```

3.2 Команда **LOOP** выполняет блок команд заданное число раз.

В качестве счетчика используется регистр ЕСХ.

Предварительно в **регистр ECX** загружается количество повторений цикла. Выполнение:

- На каждом шаге выполнения цикла значение **ECX** автоматически уменьшается на 1 и сравнивается с 0.
- Если результат не ноль переход по метке.
- В противном случае выпоняется следующая по порядку команда.

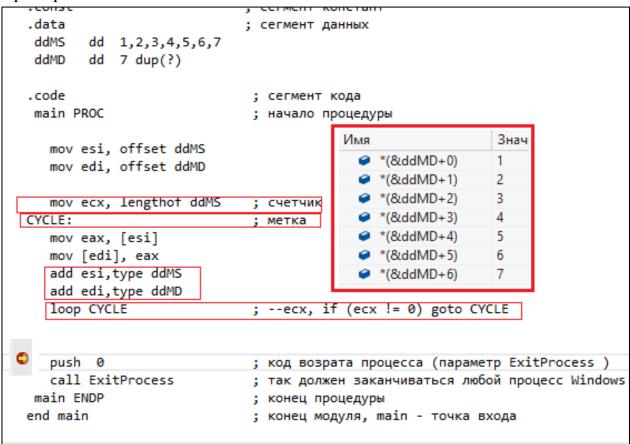
### Синтаксис:



# Пример 1:

```
.data
                          ; сегмент данных
ddMS
       dd 1,2,3,4,5,6,7
ddMD
       dd 7 dup(?)
.code
                           ; сегмент кода
main PROC
                           ; начало процедуры
                                           Имя
                                                              Знач
  mov esi, offset ddMS
                                             1
  mov edi, offset ddMD
                                             *(&ddMD+1)
                                                              2
                                                              3
                                             *(&ddMD+2)
  mov ecx, 7
                           ; счетчик
                                                              4
                                             *(&ddMD+3)
CYCLE:
                            метка
  mov eax, [esi]
                                             *(&ddMD+4)
                                                              5
  mov [edi], eax
                                             6
  add esi,4
                                                              7
                                             add edi,4
  loop CYCLE
                           ; --ecx, if (ecx != 0) goto CYCLE
                           ; код возрата процесса (параметр ExitProce
  push 0
  call ExitProcess
                           ; так должен заканчиваться любой процесс и
main ENDP
                           ; конец процедуры
end main
                           ; конец модуля, main - точка входа
```

### Пример 2:



Оператор ТҮРЕ возвращает размер элемента массива в байтах.

Пример 3. пересылка элементов одного массива в другой оформлена в виде процедуры proc1:

```
.data
                           ; сегмент данных
 ddMS
       dd 1,2,3,4,5,6,7
 ddMD
       dd 7 dup(?)
.code
                            ; сегмент кода
 main PROC
                            ; начало процедуры
   call proc1
                            ; поместить в стек адрес следующей
                            ; команды и јтр proc1
                                                     Имя
                                                                        3н
                                                       *(&ddMD+0)
                                                                        1
  push 0
                            ; код возрата процесса
                                                       *(&ddMD+1)
                                                                        2
  call ExitProcess
                            ; так должен заканчивать
                                                                        3
                                                       *(&ddMD+2)
 main ENDP
                            ; конец процедуры
                                                                        4
                                                       *(&ddMD+3)
                                                                        5
 proc1 PROC
                                                       *(&ddMD+4)
                          ; начало процедуры
  mov esi, offset ddMS
                                                       6
  mov edi, offset ddMD
                                                       *(&ddMD+6)
   mov ecx, lengthof ddMS
                            ; счетчик
CYCLE:
                            ; метка
  mov eax, [esi]
   mov [edi], eax
   add esi, type ddMS
   add edi, type ddMD
   loop CYCLE
                            ; --ecx, if (ecx != 0) goto CYCLE
  ret
                            ; рор адрес возврата и јтр
 proc1 ENDP
                            ; конец процедуры
                            ; конец модуля, main - точка входа
end main
```

Оператор lengthof возвращает количество элементов в массиве

# 4. Операции со стеком: PUSH, POP, PUSHAD, POPAD, CALL, RET, регистр ESP

В регистре **ESP** хранится 32-разрядное смещение (адрес) вершины стека.

Содержимое ESP изменяется автоматически следующими командами:

CALL, RET, PUSH и POP

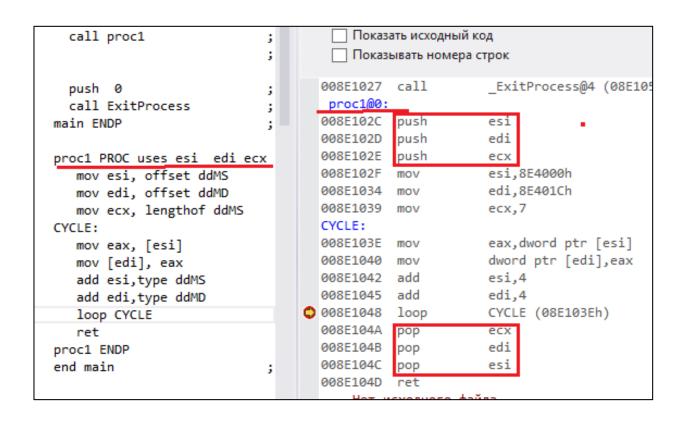
# 4.1 Команды работы со стеком:

**PUSH** – помещает 32-разрядное число в стек и вычитает 4 байта из значения, хранящегося в ESP.

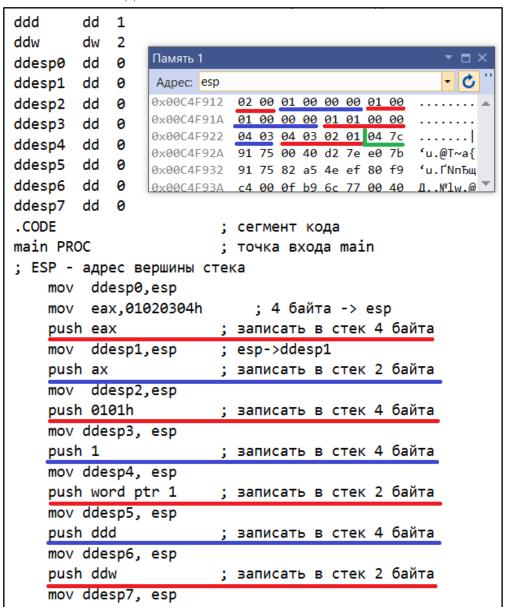
**POP** — извлекает 32-разрядное число из стека и прибавляет 4 байта к значению, хранящемуся в ESP.

Сохранить несколько используемых в процедуре регистров можно опреатором **USES.** Это необходимо, чтобы процедура не «испортила» их значение.

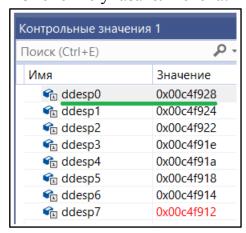
По команде **USES** сохраняются перечисленные регистры при входе в процедуру и они восстанавливаются непосредственно перед выходом из процедуры:



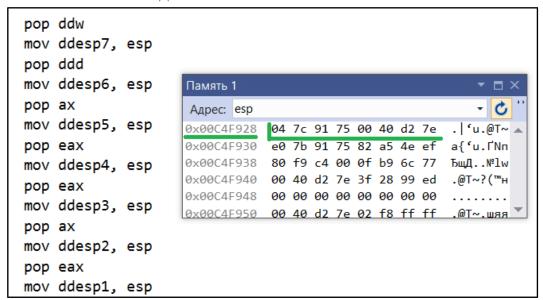
### Вставка слов и двойных слов в стек:



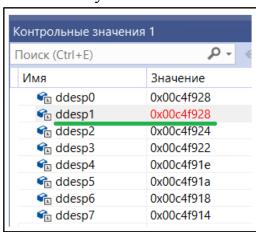
#### Изменение указателя стека:



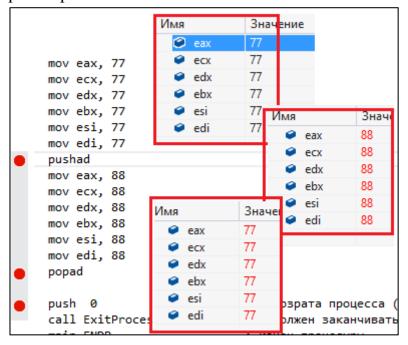
### Извлечение слов и двойных слов из стека:



# Изменение указателя стека:



**4.2** Команды **PUSHAD** и **POPAD** – сохраняют 32-разрядные значения всех регистров и восстанавливают их соответственно:



# 5. Логические команды AND, OR, XOR, NOT

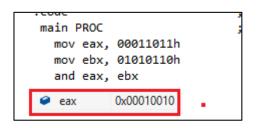
Команда **AND** выполняет операцию логического И (&) с соответствующими парами битов операндов команды и помещает результат в операнд-получатель.

### Синтаксис:



Таблица истинности для операции логичекого И:

| X | Y | X AND Y |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 0       |
| 1 | 0 | 0       |
| 1 | 1 | 1       |



```
main PROC
                            ; начало процедуры
  mov eax, 11111011h
  mov ebx, 01010110h
  and ax, bx
           0x11110010
  eax
  coue
                              ; сегмент кода
  main PROC
                               ; начало процедуры
    mov eax, 11111101h
    mov ebx, 00000010h
    and al, bl
  eax
             0x11111100
```

```
.data
                           ; сегмент данных
ddMS
       dd 1,2,3,4,5,6,7
ddMD
       dd 7 dup(?)
ddAND dd 11111111h
dwAND dw 1111h
bAND
       byte 11111111b
.code
                             ; сегмент кода
main PROC
                             ; начало процедуры
  mov eax, 10101001h
  and ddAND, eax
  and eax, ddAND
  and dwAND, ax
  and ax, dwAND
  and al, bAND
  and bAND, ah
```

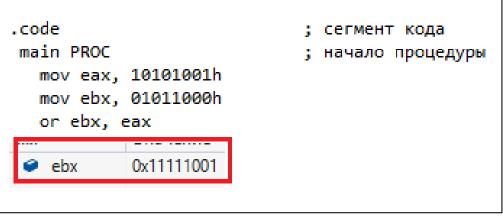
Команда  $\mathbf{OR}$  выполняет операцию логического ИЛИ (|) с соответствующими парами битов операндов команды и помещает результат в операнд-получатель.

# Синтаксис:

| OR | получатель | источник |
|----|------------|----------|
|----|------------|----------|

Таблица истинности для операции логичекого ИЛИ:

| X | Y | X OR Y |
|---|---|--------|
| 0 | 0 | 0      |
| 0 | 1 | 1      |
| 1 | 0 | 1      |
| 1 | 1 | 1      |



```
ddAND dd 11111111h
dwAND dw 1111h
bAND
       byte 11111111b
.code
                             ; сегмент кода
main PROC
                             ; начало процедуры
  mov eax, 10101001h
  or ddAND, eax
  or eax, ddAND
  or dwAND, ax
  or ax, dwAND
  or al, bAND
  or bAND, ah
  or eax, 2
  or ddAND, 2
  or dwAND, 2
  or al, 5
```

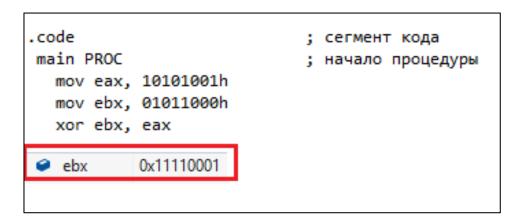
Команда **XOR** выполняет операцию исключающего ИЛИ с соответствующими парами битов операндов команды и помещает результат в операнд-получатель.

# Синтаксис:



Таблица истинности для операции исключающего ИЛИ:

| X | Y | X XOR Y |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 1       |
| 1 | 0 | 1       |
| 1 | 1 | 0       |



Команда **NOT** выполняет инверсию всех битов операнда (в результате получается обратный код числа).

# Синтаксис:



Таблица истинности для операции отрицания:

| X | NOT X |
|---|-------|
| 0 | 1     |
| 0 | 1     |
| 1 | 0     |
| 1 | 0     |

