Команды Ассемблера

Общие правила:

1. Размеры операндов в командах должны быть одинаковыми (если не указано обратное)!
2. Оба операнда одновременно не могут быть в памяти (переменными)!
3. Нельзя перемещать в (из) сегментный регистр!

Команды передачи данных

Эта группа делится на четыре части: общие, ввода-вывода, адресные и флагов.

Команды передачи данных (Основные команды)

mov назначение, источник	Переслать (скопировать)
pop назначение	Извлечение слова из стека
push источник	Занесение слова в стек
xchg назначение, источник	Перестановка

Основная команда пересылки данных

команда	dst (приемник)	src (источник)
mov dst, src	регистр память	регистр память число*

* Далее "число" =
“непосредственный операнд”

mov назначение <-- источник

Данные из источника передаются в приемник.

```
mov eax, ebx                ; eax <-- ebx

mov ecx, numPages ; ecx <-- numPages
```

Данная команда пересылает значение, содержащееся в *numPages,* в регистр *ecx*.

*Метка numPages определяет адрес в памяти. numPages соответствует данным, записанным по этому адресу.

```
mov level, dl                ; ???
mov [level], dl              ; ???
mov cs, dx                   ; ???
mov ax, edx                  ; ???
mov count, maxCount          ; ???
```

Пересылка значения, записанного в переменной maxCount в переменную count требует выполнения двух шагов и использования промежуточного (свободного) регистра:

```
mov eax, maxCount      ; eax<-- maxCount
mov count, eax         ; count<----- eax
```

Пример

```
a dd 1
b dd ?
...
mov eax, a  ; регистр <- память
mov b,   a  ; ошибка (какая?)
mov b, eax  ; память <- регистр
mov ax, a   ; ошибка (какая?)
mov eax, 2  ; регистр <- число
mov bh, al  ; регистр <- регистр
```

Пересылка с расширением

команда	заполняет старшие разряды dst	dst	src
movzx dst, src	нулем	регистр	регистр память
movsx dst, src	знаком src		

- Размер (dst) > размер (src) !!! - расширение
- **movzx** применимо только для положительных или беззнаковых чисел!!!

Пример

```
a db 1
b dw 1
d db -10 ; 11110110 дк
...
movzx bx, a
movzx ebx, al
movzx bl, ax ; ошибка (какая?)
movzx bx, d
; bx = 0000000011110110 = 246!
movsx bx, d
; bx = 1111111111110110 = -10!
```

Расширение знака

команда	расшифровка	источник	приемник
cbw	convert byte -> word	al	ax
cwd	... word -> double word	ax	dx:ax
cwde	... word -> double word	ax	eax
cdq	... double word -> quad word	eax	edx:eax
cdqe	... double word -> quad word	eax	rax
cqo	quad word -> oct word	rax	rdx:rax

Пример

```
; вычислить x = al + b

b dd 123456
x dd ?
...
mov al, 63 ;
    ; вычислЯЕМ x = al + b
cbw
cwde
add eax, b ; eax = eax + b
mov x, eax
```

Обмен местами

команда	dst	src	
xchg dst, src	регистр память	регистр память	обменивает местами src и dst

команда	dst	
bswap dst	регистр 32	меняет порядок байт в регистре на обратный

xchg меняет местами значения двух регистров либо значение регистра с ячейкой памяти.

```
xchg eax, edx          ;eax <--> edx
xchg ax, things        ; ax <--> things
xchg oldCount, ecx     ; ecx <--> oldCount
```

Пример


```
a db …
b db …
...
xchg a, b ; ошибка (какая?)

xchg a, dl
xchg b, dl
xchg a, dl

xchg al, ah
xchg rbx, rcx

mov eax, 0x12345678
bsawp eax ; eax = 0x78563421
```

Стек

Команда **push** выполняет два действия.

1. Уменьшает на 2 (4) значение (e)sp.
2. Заданное значение записывается в стек по адресу [ss:(e)sp].

```
mov ax,100
push ax      ;sp2
mov bx,100
push bx      ;sp3
```

Каждая команда push должна иметь компенсирующую ее команду **pop**.

Выталкивание нужно проводить ***в обратном порядке***!

*Обычно программы оформляются в виде небольших модулей или подпрограмм. В каждом модуле протолкните в стек все регистры, которые вы планируете в нем использовать. Перед тем как закончить код, вытолкните в обратном порядке из стека эти же регистры.

Для удобства, начиная с i386 введены команды, запоминающие и выталкивающие все РОНы в стек – **pusha** и **popa**.

Для их использования в заголовке программы необходимо подключить инструкции i386 (в Win32 – уже включено!).

Арифметические команды

(Основные команды)

Команды сложения

add назначение, источник	Сложение байт или слов
adc назначение, источник	Сложение с переносом
inc назначение	Инкремент

Команды вычитания

sub назначение, источник	Вычитание
sbb назначение, источник	Вычитание с заемом
cmp назначение, источник	Сравнение
dec назначение	Декремент
neg назначение	Изменение знака числа

Команды умножения

imul источник Умножение величин со знаком

Команды деления

idiv источник. Деление величин со знаком

Сложение и вычитание

команда	семантика	dst	src
add dst, src	dst = dst + src	регистр память	регистр память число
adc dst, src	dst = dst + src + CF		
sub dst, src	dst = dst – src		
sbb dst, src	dst = dst – src – CF		

- Устанавливают 6 флагов:
  - SF, ZF, PF (свойства результата)
  - OF, CF, AF (ход выполнения операции)
- Не различают знаковые и беззнаковые операнды

Команды сложения

add и **adc** суммируют два операнда без и с переносом (флаг **cf**),

inc (инкремент) - команда быстрого добавления 1 к регистру или значению в памяти.

```
add ah, bh          ; ah <-ah+bh
adc ax, bx          ; ax <- ax+bx+cf
```

- Если cf равно 1, то результат получается таким же, как и при добавлении 1 к сумме ax и bx.
- Флаг cf устанавливается в 1, если в предыдущем сложении возникал перенос.

Поэтому adc чаще используется после add, для сложения значений по частям, информируя о переносах, которые возникают при сложении отдельных байтов, слов или двойных слов.

Например, sum := sum + sum:

```
sum dw …
1. mov ax,  sum ;  Переслать в ax  sum
   add sum, ax   ;  Прибавить sum к самому  себе
```

```
2. mov ax, sum   ; Переслать в ax  sum
   add al, byte ptr sum      ; Прибавить младший байт
   adc ah, byte ptr sum+1 ; Прибавить старший байт
   mov sum, ax ; Переслать полученное значение в память
```

Операторы ***word ptr*** и ***byte ptr*** информируют ассемблер о размере данных, адресуемых *sum*. В некоторых случаях ассемблер делает это автоматически, в других эти операторы необходимы (когда из команды *непонятен* размер данных, или размер операнда *отличается* от указанного в DATASEG этой меткой).

Примеры

```
; вычислить z = a + b
a db -1
b dw 2
z dw ?
...
movsx ax, a
add ax, b
mov z, ax ; z = a + b
```

```
; вычислить b = a + b
a db -1
b dw 2
...
movsx ax, a
add b, ax
     ; b = a + b
```

```
; вычислить a = a + 2
a db -1
...
add a, 2  ; a = a + 2
```

Пример использования переноса


```
; сложить 2 длинных числа
; младшие разряды записаны в массивах слева
a dd 0x12345678, 0x12345678, 0x00123456
b dd 0xc6509ffa, 0x00001123
c dd 4 dup (?)
...
mov eax, a      ; eax  = a[0]
add eax, b      ; eax  = a[0]+b[0]
mov c, eax      ; c[0] = a[0]+b[0]

mov eax, a+4    ; eax  = a[1]
adc eax, b+4    ; eax  = a[1]+b[1]+перенос
mov c+4, eax    ; c[1] = a[1]+b[1]+перенос
```

Пример (продолжение)


```
mov eax, a+8    ; eax  = a[2]
adc eax, 0      ; eax  = a[2]+перенос
mov c+8, eax    ; c[2] = a[2]+перенос

xor eax, eax    ; eax  = 0
adc eax, 0      ; eax  = перенос
mov c+12, eax   ; c[3] = перенос
```

Производить ли действия над операндами в памяти или регистрах?

$$a = a + b$$

```
mov ax, a
add ax, b
mov a, ax

; 2 чтения
; 1 запись
```

```
mov ax, b
add a, ax

; 2 чтения (где?)
; 1 запись
```

Производить ли действия над операндами в памяти или регистрах?

$$a = a + b + d + e$$

```
mov ax, a
add ax, b
add ax, d
add ax, e
mov a, ax

; 4 чтения
; 1 запись
```

```
mov ax, b
add a, ax
mov ax, d
add a, ax
mov ax, e
add a, ax

; 6 чтений
; 3 записи
```

Вывод

- **Рационально**:
  - производить действие с операндом в памяти если это единственная операция
  - предварительно загружать операнд в регистр если над ним производится последовательность действий

Сложение с обменом

команда	dst	src
xadd dst, src	регистр память	регистр

- Семантика:

  dst = dst + src

  src = значение dst до сложения

Инкремент и декремент

команда	dst	семантика
inc dst	регистр память	dst = dst + 1
dec dst	регистр память	dst = dst -1

- Не меняет значение флага CF
  - это свойство используется при обработке массивов

```
xadd ebx, ecx   ; ebx1 <- ebx0+ecx0, ecx1 <- ebx0,

    inc ax      ; ax <- ax+1
    inc dh      ; dh<- dh+1
    inc d       ; d<- d+1
```

Команды вычитания

По своей форме вычитание аналогично сложению.

Команда **sub** вычитает значения.

Команда **sbb** делает то же самое, но учитывает возможный заем при предыдущем вычитании многобайтовых значений.

```
sub ax, bx          ; ax <-  ax- bx
sub ecx, 5          ; ecx <-  ecx - 5
sub edx, score      ; edx <-  edx- score
dec ax              ; ax <-  ax-1
dec dl              ; dl <-  dl-1
dec esi             ; esi <-  esi-1
dec balance         ; balance <- balance - 1
```

cmp выполняет вычитание аналогично команде sub, однако сохраняет при этом только значения флагов, которые могут проверяться другими командами – эта команда используется для ***сравнения*** 2-х операндов, после нее должен быть условный переход (анализ флагов).

Изменение знака

команда	dst	
neg dst	регистр память	

- Семантика:

  dst = - dst

neg eax ; eax <= -eax