Арифметические операции

Инструкции сложения **ADD** и вычитания **SUB**

- Команда ADD требует двух операндов, как и команда MOV:
- `ADD o1 , o2`
- Команда ADD складывает оба операнда и записывает результат в o1, предыдущее значение которого теряется.

Инструкции сложения **ADD** и вычитания **SUB**

- Точно так же работает команда вычитания — SUB:
- `SUB o1 , o2`
- Результат, o1-o2, будет сохранен в o1, исходное значение o1 будет потеряно.

Например…

```
mov  ax,  8          ; заносим в AX число 8
mov  cx,  6          ;заносим в CX число 6
mov  dx,  cx         ;копируем CX в DX, DX = 6
add  dx,  ax         ;DX = DX + AX
```

Размеры переменных

- *Бит* (*bit*) - двоичный разряд.
  Байт (*byte*) - последовательность из 8 бит.
  ***Слово* (*word*)** - последовательность из двух байт (16 бит).
  ***Двойное слово* (*double word*)** - последовательность из четырех байт (32 бита).

Проверяем, понимаете ли вы, что происходит

Что получится в результате?

```
add eax,8
sub  ecx,ebp
add  byte  [number]
```

```
sub word  [number],  4
```

```
add dword  [number],  4
```

```
sub byte  [number],  al
```

```
sub  ah,al
```

Что получится в результате?

```
add eax,8                 EAX = EAX + 8
sub  ecx,ebp              ECX = ECX - EBP
add byte  [number]        добавляем значение 4
                          к переменной number
                          размером в 1 байт
                          (диапазон значений 0-255)
sub word  [number], 4     number = number — 4
                          размером в 2 байта
                          (диапазон значений 0-65535)
add dword [number], 4     добавляем значение 00000004
                          к "number"
sub byte  [number], al    вычитаем значение регистра AL
                          из "number"
sub  ah,al                вычитаем AL из AH, результат
                          помещаем в AH
```

Команды инкрементирования **INC** и декрементирования **DEC**

- Команда INC добавляет, а DEC вычитает единицу из единственного операнда. Допустимые типы операнда — такие же, как у команд ADD и SUB, а формат команд таков:

```
INC O1          ;o1 = o1 + 1
DEC O1          ;o1 = o1 - 1
```

Например…

Увеличение на единицу значения регистра AL выглядит следующим образом:

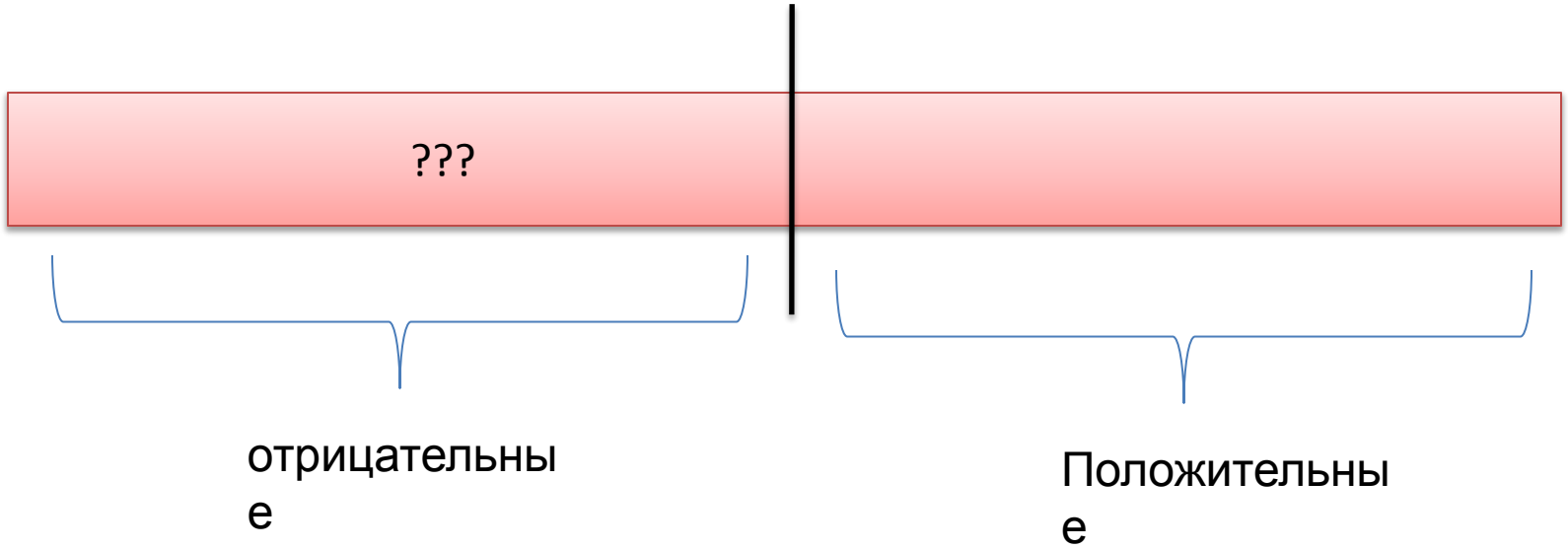
```
add al, 1          ;AL = AL + 1
inc al             ;AL = AL + 1
```

Увеличение на единицу значения 16-битной переменной number:

```
inc word [number]  ;мы должны указать размер
                   ;переменной — word
```

Отрицательные числа — целые числа со знаком

- Один байт может содержать числа в диапазоне от 0 до 255.
- Код дополнения заменяет этот диапазон другим — от -128 до 127.



Решение – маппинг!

- Диапазон от 0 до 127 отображается сам на себя,
- отрицательным числам сопоставляется диапазон от 128 до 255:
- числу -1 соответствует число 255,
- числу -2 — 254 и т.д.
- Процесс отображения отрицательных чисел в дополнительный код иногда называют **маппингом**

Например…

```
mov  ax,  -6          ;AX = -6
mov     dx,-     6    ;DX = - 6
add  ax,dx            ;AX = AX + DX
```

Команды для работы с отрицательными числами. Команда **NEG**

- Используя NEG, вы можете преобразовывать положительное целое число в отрицательное и наоборот.
- Инструкция NEG имеет только один операнд, который может быть регистром или адресом памяти.
- Размер операнда — лю бой: 8, 16 или 32 бита.
- `NEG eax`
- `NEG byte [number]`

Целочисленное умножение и деление (команды **MUL**, **DIV**).

- Операции умножения и деления имеют свою специфику. В результате умножения двух чисел мы можем получить число, диапазон которого будет в два раза превышать диапазон операндов.
- Деление целых чисел — это операция целочисленная, поэтому в результате образуются два значения: частное и остаток.
- С целью упрощения реализации команд умножения и деления эти команды спроектированы так, что один из операндов и результат находятся в фиксированном регистре, а второй операнд указывается программистом.

Что куда заносим?

- В 8-разрядной форме операнд может быть любым 8-битным регистром или адресом памяти. Второй операнд всегда хранится в AL. Результат (произведение) будет записан в регистр AX

(r/m8) * AL -> AX

Что куда заносим?

- В 16-разрядной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда хранится в AX. Результат сохраняется в паре DX·AX

(r/m16) * AX -> DX:AX

Что куда заносим?

- В 32-разрядной форме второй операнд находится в регистре EAX, а результат записывается в пару EDX.EAX.

(r/m32) * EAX -> EDX:EAX

Например…

Пример 1: умножить значения, сохраненные в регистрах BH и CL, результат сохранить в регистр AX:

```
mov al, bh    ;AL = BH — сначала заносим в AL второй операнд
mul cl        ;AX = AL * CL — умножаем его на CL
```

Результат будет сохранен в регистре AX.

Пример: вычислить 486^2, результат сохранить в DX:AX:

```
mov ax, 486   ; AX = 486
mul ax        ; AX * AX -> DX:AX
```

Пример 2: вычислить диаметр по радиусу, сохраненному в 8-битной переменной `radius1`, результат записать в 16-битную переменную `diameter1`:

```
mov al, 2                ; AL = 2
mul byte [radius1]       ; AX = radius * 2
mov [diameter1],ax       ; diameter <- AX
```

Команда **IMUL**

- Команда IMUL умножает целые числа со знаком и может использовать один, два или три операнда.
- Когда указан один операнд, то поведение IMUL будет таким же, как и команды MUL, просто она будет работать с числами со знаком.
- Если указано два операнда, то инструкция IMUL умножит первый операнд на второй и сохранит результат в первом операнде, поэтому первый операнд всегда должен быть регистром. Второй операнд может быть регистром, непосредственным значением или адресом памяти.

Например…

```
imul edx,ecx          ;EDX = EDX * ECX
imul ebx,[sthing]     ;умножает 32-разрядную переменную
                      ;"sthing" на EBX, результат будет
                      ;сохранен в EBX
```

```
imul ecx,6            ; ECX = ECX * 6
```

А что, если три?

- Если указано три операнда, то команда IMUL перемножит второй и третий операнды, а результат сохранит в первый операнд.
- Первый операнд — только регистр, второй может быть любого типа, а третий должен быть только непосредственным значением:

Например…

```
imul edx,ecx,7          EDX = ECX * 7
imul ebx,[sthing],9     умножаем переменную "sthing" на 9,
                        результат будет сохранен EBX

imul ecx,edx,11         ECX = EDX * 11
```

Команды **DIV** и **IDIV**

- В 8-битной форме переменный операнд (делитель) может быть любым 8-битным регистром или адресом памяти. Делимое содержится в AX. Результат сохраняется так: частное — в AL, остаток — в AH.

AX / (r/m8) -> AL, остаток -> AH

Команды **DIV** и **IDIV**

- В 16-битной форме операнд может быть любым 16-битным регистром или адресом памяти. Второй операнд всегда находится в паре DX:AX. Результат сохраняется в паре DX:AX (DX — остаток, AX — частное).

DX:AX / (r/ml6) -> AX, остаток -> DX

Команды **DIV** и **IDIV**

- В 32-разрядной форме делимое находится в паре EDX:EAX, а результат записывается в пару EDX:EAX (частное в EAX, остаток в EDX).

EDX:EAX / (r/m32) -> EAX, остаток -> EDX

- Команда IDIV используется для деления чисел со знаком, синтаксис ее такой же, как у команды DIV.

Например…

Пример 1: разделить 13 на 2, частное сохранить в BL, а остаток в — BH:

```
mov  ax,13      AX = 13
mov  cl,2       CL = 2
div  cl         делим на CL
mov bx,ax       ожидаемый результат находится в AX,
                копируем в BX
```

Например..

Пример 2: -вычислить радиус по диаметру, значение которого сохранено в 16-битной переменной diameterl, результат записать в radiusl, а остаток проигнорировать.

```
mov ax,[diameterlj      AX = diameterl
mov Ы, 2                загружаем делитель 2
div Ы                   делим
mov [radiusl],al        сохраняем результат
```

На этом с математическими функциями все ☺️