Практика 04: арифметические команды.

Представление знаковых чисел.

Рассматривать двоичную систему счисления не будем, считаем что она известна. Рассмотрим представление на уровне аппаратуры знаковых чисел. Для представления чисел со знаком используется специальное кодирование. Старший бит в этом случае обозначает знак числа. Если знаковый бит равен нулю, то число положительное, иначе — отрицательное. Понятно, что положительное число со знаком будет выглядеть точно так же, как и число без знака.

С отрицательными числами чуть сложнее. Исторически для представления отрицательных чисел в компьютерах использовались разные виды кодирования: прямой, обратный и дополнительный код. В настоящее время наиболее часто используется дополнительный код, в том числе и в процессорах x86.

Чтобы сделать из положительного числа отрицательное, необходимо проинвертировать все его биты (0 заменяем на 1, а 1 заменяем на 0) и затем к младшему разряду прибавить единицу. Например, представим -5 в дополнительном коде:

В обратную сторону переводится точно также. Со знаковыми и беззнаковыми числами нужно быть внимательным, потому что только вы знаете, какие числа используются в вашей программе! Процессору абсолютно по барабану, какие данные он обрабатывает, поэтому невнимательность может привести к ошибке. Один и тот же байт может интерпретироваться по-разному, в зависимости от того со знаком число или без. Например, числу со знаком -5 соответствует число без знака 251:

При программировании на ассемблере (как, впрочем, и на многих других языках) необходимо учитывать ещё один важный момент. А именно — ограничение диапазона представления чисел. Например, если размер беззнаковой переменной равен 1 байт, то она может принимать всего 256 различных значений.

Это означает, что мы не сможем представить с её помощью число, больше 255 (111111112). Для такой же переменной со знаком максимальным значением будет 127 (011111112), а минимальным -128 (100000002). Аналогично определяется диапазон для 2- и 4-байтных переменных.
Кстати, так как процессор Intel 8086 был 16-битным и обрабатывал за одну команду 16-бит, то 16-битная переменная называется слово (word), а 32-битная — двойное слово (double word, dword). Эти названия сохранились в ассемблере даже для 32-битных процессоров (и в WIN32 API, например). И от них же происходят названия директив dw (Define Word) и dd (Define Dword). Ну а db — это Define Byte.
Диапазон значений зависит от типа переменной:

Byte (8bit) - 0 - 255 (без знака) -128 - 127 (со знаком)
Word (16 bit) - 0 - 65535 (без знака) -32768 - 32767 (со знаком)
DoubleWord (32 bit) - 0 - 4294967295 (без знака) -2147483648 - 2147483647 (со знаком)

Если результат какой-то операции выйдет за пределы диапазона представления чисел, то случится переполнение и результат будет некорректным. (Например, при сложении двух положительных чисел, можно получить отрицательное число!) Поэтому нужно быть внимательным при программировании и предусмотреть обработку таких ситуаций, если они могут возникнуть.

Преобразования числа в строку.

Для начала рассмотрим алгоритм преобразования числа в строку. Алгоритм нужен, чтобы иметь возможность выводить

числа на экран. Оформим ввиде макроса.

```
macro numtostr num,str
{
mov eax,[num]
xor ecx,ecx
mov ebx,10
numtostrlopp1:
xor edx,edx
div ebx
add edx,'0'
push edx
inc ecx
test eax,eax
jnz numtostrloop1
mov edi,str
numtostrloop2:
pop edx
mov byte[edi],dl
inc edi
dec ecx
test ecx,ecx
jnz numtostrloop2
}
```

Макрос преобразует число из переменной Num в строку в переменную Str. Вызвать макрос можно так

```
mov [inputnumber],12345
numtostr inputnumber,outputstr
```

Команды сложения и вычитания.

Перед описанием арифметических команд упомянем, что в случае двух операндов действуют те же самые ограничения на операнды, что и для команды MOV.

ADD
сложение беззнаковых и знаковых чисел. Требуется два операнда и результат помещается в первый операнд. Пример:

add eax,5 ; EAX = EAX + 5
add edx,ecx ; EDX = EDX + ECX

Недопустимы операнды разного размера. Пример:

add eax,dl ; ошибка - операнды разного размера

SUB
вычитание беззнаковых и знаковых чисел. Результата аналогично сложению помещается в первый операнд. Так же требуется, чтобы операнды были одинакового размера. Примеры:

sub eax,ebx ; EAX = EAX - EBX
sub ebx,5 ; EBX = EBX - 5
sub ecx,bl ; недопустимо - разный размер операндов

Вообще вычитание выполняется при помощи сложения. процессор меняет знак второго операнда на противоположный и выполняет сложение.

NEG

команда с одним операндом - меняет знак на противоположный. Пример:

neg ebx ; ebx = -ebx

С командами **INC** и **DEC** – инкрементом и декрементом мы уже встречались. Они имеют по одному параметру.

Пример программы:
вычислим 2 - (3 + 5 - 1) + (- (-8)). Используем байтовые переменные.

```
mov al, 2
mov ah, 3
add ah, 5    ; 3+5
dec ah       ;  3+5-1
sub al,ah    ;  2-(3+5-1)
mov cl,-8
neg cl       ;  -(-8)
add al,cl    ;  2-(3+5-1)+(-(-8))
```

Переполнение и перенос.

Вычислим 250+25, считая что операнды байтовые и беззнаковые. В результате вместо 275 получим 19. В чем дело? Результат сложения оказался больше байта и произошло переполнение. Для записи результата потребовался отсутствующий, 9ый бит. А младшие 8 бит как раз и дали 21. 9ый бит не потерялся, а занесен в флаг CF, находящийся в слове состояния процессора. Для обработки таких ситуаций есть дополнительные команды сложения и вычитания.

ADC

сложение с учетом переполнения. ADC обычно используется в многобайтных или многословных (multi-word) операциях сложения. В таком случае она идет вслед за командой ADD, которая возвращает сумму младших разрядов многобайтных (многословных) операндов, позволяя при сложении старших разрядов учитывать перенос. Пример в 32 бит:

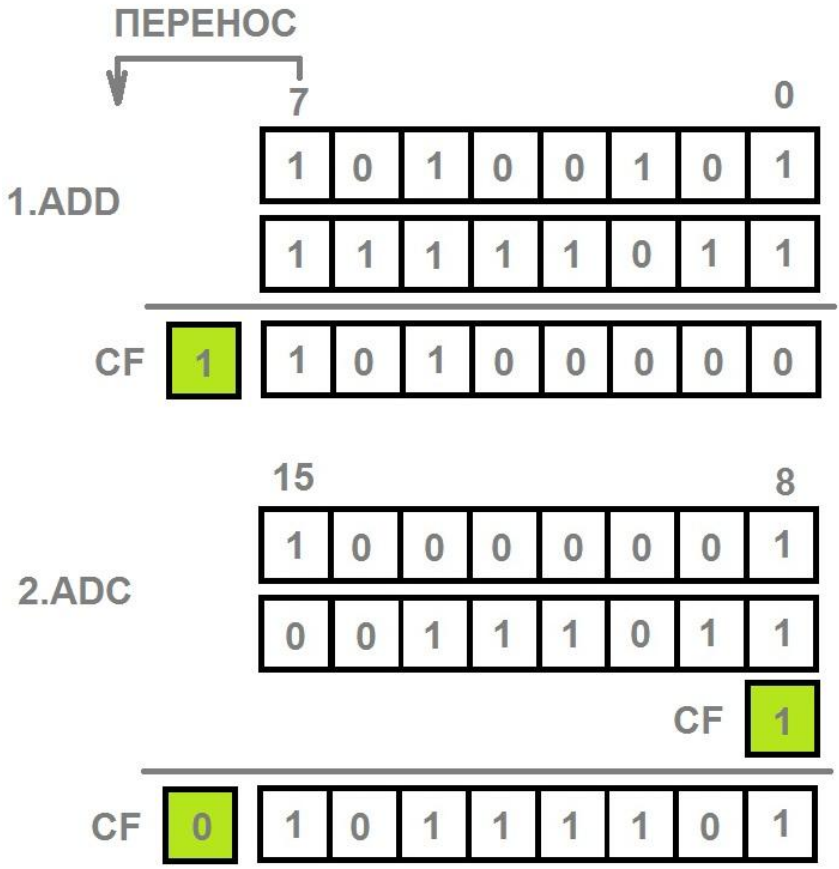
```
XOR EDX,EDX
ADD EAX,EBX    ;  EAX =EAX + EBX + CF
ADC EDX,0      ;  EDX= EDX + CF
```

Итоговый результат в паре регистров EDX:EAX.
Команда ADC позволяет манипулировать целочисленными операндами как в беззнаковом формате, так и в формате со знаком. При сложении данных со знаком флаг знака SF будет отражать знак полученного результата. Флаг переполнения OF установится в 1, если при сложении целочисленных значений со знаком, представленных в обратном коде или в дополнительном коде, произошло переполнение (перенос из старшего значащего разряда, которому соответствует бит, предшествующий разряду знака), то есть полученный результат превышает доступный размер операнда-назначения. По сути, это аналогично тому, как флаг CF отражает переполнение (перенос) при сложении беззнаковых операндов. Например, при сложении двух 32-битных значений, представленных в обратном коде, это может выглядеть следующим образом:

```
ADD EAX,EBX     ;  EAX = EAX + EBX
INTO            ; переход к обработчику прерывания
                ; в случае переполнения, т.е. OF=1
```

```
ADC EAX,0   ; EAX = EAX + CF, учитываем перенос
            ;(необходимо для сложения в обратном коде)
JNS M1      ; переход, если результат положительный
XOR EAX, 7FFFFFFFh    ; преобразование отрицательного
                      ;значения в EAX к прямому коду
M1:
```

По сути схема выглядит так :



SBB

Аналогично вычитание с учетом переполнения, точнее это называется вычитание с учетом заёма. Пример пары чисел в регистрах EDX:EAX и EBX:ECX:

```
SUB EAX,ECX ; EAX=EAX-ECX
            ; CF=1 если EAX<ECX
SBB EDX,EBX  ;  EDX=EDX-EBX-CF
```

И результат оказывается в паре EDX:EAX.
В случае чисел со знаком код аналогичен сложению.
Стоит помнить, что при работе с такими большими числами нельзя использовать команды INC и DEC, так как они не меняют флаг CF и переполнение теряется.

Умножение и деление.

Для начала рассмотрим умножение без знака.

MUL

один параметр, а именно второй множитель. Первый множитель и результат хранятся в заранее определенном регистре и зависят от разрядности второго множителя. Пример:

```
MUL DL      ;  AX = AL * DL
MUL DX      ;  DX:AX = AX * DX
MUL EDX     ;  EDX:EAX = EAX * EDX
```

где EDX:EAX опять же означает что младшее слово результата хранится в EAX. а старшее в EDX. Если EDX=0, то и CF=OF=0.

Для умножения со знаком есть отдельная команда:

IMUL

где может быть три формы записи,в зависимости от количества операндов. Пример с одним операндом:

IMUL BL ; AX = AL*BL
IMUL BX ; DX:AX = AX*BX
IMUL EBX ; EDX:EAX = EAX*EBX

Пример с двумя операндами:

IMUL AX,BX ; AX=AX*BX
IMUL AX,10 ; AX=AX*10
IMUL EAX,10 ; EAX=EAX*10

в этом случае отбрасывается старшая часть произведения, о чем сигнализирует CF=OF=1. Эта форма записи не работает с 8 битными регистрами и переменными.
Пример с тремя операндами, где третий операнд всегда непосредственная константа:

IMUL AX,BX,10 ; AX=BX*10
IMUL EAX,EBX,10 ; EAX=EBX*10

Здесь тоже отбрасываются старшие разряды произведения (CF=OF=1) и так же нельзя использовать 8 битные регистры и переменные.
Деление является целочисленным, то есть при делении всегда формируется частное и остаток от деления. Для использования дробей нужно воспользоваться либо командами математического сопроцессора,или программной имитацией.

Деление беззнаковых чисел выполняется командой

DIV

обладающей одним единственным операндом - делителем. Положение делимого, частного и остатка от деления определяется разрядностью операнда. Пример:

DIV BL ; AL = AX/BL и остаток AH
DIV BX ; AX = DX:AX / BX и остаток в DX
DIV EBX ; EAX = EDX:EAX / EBX и остаток в EDX

В случаях когда

- делимое равно 0
- частное не помещается в отведенную ячейку памяти

возникает прерывание. Его нужно обрабатывать отдельно, ну или стараться избегать.
Для операций со знаковыми числами используется команда

IDIV

обладающая точно такими же параметрами, как и DIV. Сказанное о DIV, про место размещения результата и возможность возникновения прерывания относится также и к IDIV.
Обычно в программировании на ASM стараются избегать использования команд умножении/деления и стараются из заменять сложениями и битовыми сдвигами.

Пример программы:
Вычислим 11+(21*4+24)/4. Используем байтовые переменные.

mov al, 21

```
mov dl,4
mul dl ; 21*4
add al,24  ; 21*4+24
mov dl, 4
div dl ; (21*4+24)/4
add al,11      ; 11+(21*4+24)/4
```

Задание 1:
вычислите 11+(32*9+6)/2

SHL EAX,2 ; сдвиг содержимого регистра EAX на 2 бита влево
SHR EAX,2 ; сдвиг содержимого регистра EAX на 2 бита вправо

Например, SHR AX,2 преобразует значение 0001100010101000 в 0110001010100000. Фактически это умножение на 4, но выполняется значительно быстрее.

Задание:
Заменить умножение EAX на 9 на другие команды.
Решение:

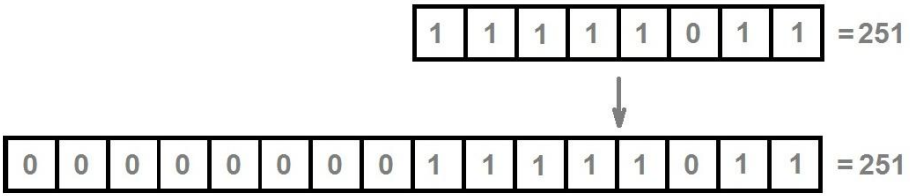
```
MOV EBX,EAX
SHR EAX,3
ADD EAX,EBX
```

Преобразование типов.

При арифметических операциях часто приходится оперировать значениями разных размеров, что недопустимо по формату команд. Для корректного выполнения команд приходится преобразовывать типы данных.

Из большего типа сделать меньший очень просто - надо использовать младшую часть от данных. Однако стоит помнить, что если в старших битах были значения, то они будут потеряны и результат окажется некорректным.
Преобразование чисел без знака к большему размеру выполняется просто: достаточно заполнить старшие биты нулями
Это легко делается при помощи MOV. Однако есть и отдельная команда

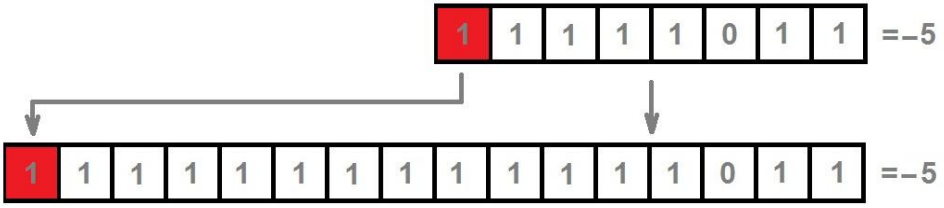
MOVZX



которая заполнение нулями проводит автоматически, например

MOVZX BX,CL
MOVZX EBX,CL
MOVZX EBX,CX

Для чисел со знаком все сложнее:надо анализировать знак и выбирать для заполнения либо 0 либо 1.



Для этого есть команда

MOVSX

Так же есть команды

- **CBW** - преобразовывает байт к слову
- **CWD** - преобразовывает слово к двойному слову
- **CWDE** - преобразовывает слово к двойному слову

Они не имеют параметров. Пример:

CBW ; AX = AH
CWD ; DX:AX = AX
CWDE ; EAX = AX

Стоит учитывать, что это расширение со знаком.

Задание 2:
Написать «обратный» макрос – из строки в число.