

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

### ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

дисциплина:     *Архитектура компьютера*

Студент: Ниemek Яи Жак

Группа: НММБд-04-24

МОСКВА

2025\_\_ г.

## Цель работы

Освоить арифметические инструкции языка ассемблера NASM.

## Задание

1. Создайте каталог для программам лабораторной работы № 6, перейдите в него и создайте файл lab6-1.asm
2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр `eax`
3. Далее изменим текст программы и вместо символов, запишем в регистры числа.
4. Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно.
5. Аналогично предыдущему примеру изменим символы на числа.
6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения

## Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации. Существует три основных способа адресации: • Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`. • Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,2`. • Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию. Например, определим переменную `intg DD 3` – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой `intg`. В таком случае, команда `mov eax,[intg]` копирует из памяти по адресу `intg` данные в регистр `eax`. В свою очередь команда `mov [intg],eax` запишет в память по адресу `intg` данные из регистра `eax`. Также рассмотрим команду `mov eax,intg`. В этом случае в регистр `eax` запишется адрес `intg`. Допустим, для `intg` выделена память начиная с ячейки с адресом `0x600144`, тогда команда `mov eax,intg` аналогична команде `mov eax,0x600144` – т.е. эта команда запишет в регистр `eax` число `0x600144`.

### 6.2.2. Арифметические операции в NASM

#### 6.2.2.1. Целочисленное сложение `add`. Схема команды целочисленного сложения `add` (от англ. *addition* - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда `add` работает как с числами со знаком, так и без знака и выглядит следующим образом: `add ,`. Допустимые сочетания операндов для команды `add` аналогичны сочетаниям операндов для команды `mov`. Так, например, команда `add eax,ebx` прибавит значение из регистра `eax` к значению из регистра `ebx` и запишет результат в регистр `eax`. Примеры: `add ax,5` ; `AX = AX + 5` `add dx,cx` ; `DX = DX + CX` `add dx,cl` ; Ошибка: разный размер операндов.

#### 6.2.2.2. Целочисленное вычитание `sub`. Команда целочисленного вычитания `sub` (от англ. *subtraction* – вычитание) работает аналогично команде `add` и выглядит

следующим образом: `sub` , Так, например, команда `sub ebx,5` уменьшает значение регистра `ebx` на 5 и записывает результат в регистр `ebx`.

### 6.2.2.3. Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: `inc` (от англ. increment) и `dec` (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд. Эти команды содержат один операнд и имеют следующий вид: `inc` `dec` Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания. Так, например, команда `inc ebx` увеличивает значение регистра `ebx` на 1, а команда `dec ax` уменьшает значение регистра `ax` на 1.

### 6.2.2.4. Команда изменения знака операнда `neg`.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака `neg`: `neg` Команда `neg` рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера. `mov ax,1 ; AX = 1 neg ax ; AX = -1`

### 6.2.2.5. Команды умножения `mul` и `imul`.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производятся по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда `mul` (от англ. multiply – умножение): `mul` Для знакового умножения используется команда `imul`: `imul` Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре `EAX`, `AX` или `AL`, а результат помещается в регистры `EDX:EAX`, `DX:AX` или `AX`, в зависимости от размера операнда

#### 6.1. Таблица 6.1. Регистры используемые командами умножения в Nasm

Размер операнда	Неявный множитель	Результат умножения
1 байт	<code>AL</code>	<code>AX</code>
2 байта	<code>AX</code>	<code>DX:AX</code>
4 байта	<code>EAX</code>	<code>EDX:EAX</code>

Пример использования инструкции `mul`: `a dw 270 mov ax, 100 ; AX = 100 mul a ; AX = AXa, mul bl ; AX = ALBL mul ax ; DX:AX = AX*AX`

### 6.2.2.6. Команды деления `div` и `idiv`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. divide - деление) и `idiv`: `div` ; Беззнаковое деление `idiv` ; Знаковое деление В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры

#### 6.2. Таблица 6.2. Регистры используемые командами деления в Nasm

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	<code>AX</code>	<code>AL</code>	<code>AH</code>
2 байта	<code>DX:AX</code>	<code>AX</code>	<code>DX</code>
4 байта	<code>EDX:EAX</code>	<code>EAX</code>	<code>EDX</code>

Например, после выполнения инструкций `mov ax,31 mov dl,15 div dl` результат 2 (31/15) будет записан в регистр `al`, а остаток 1 (остаток от деления 31/15) — в регистр `ah`. Если делитель — это слово (16-бит), то делимое должно записываться в регистрах `dx:ax`. Так в результате выполнения инструкций `mov ax,2 ; загрузить в регистровую mov dx,1 ; пару dx:ax значение 10002h mov bx,10h div bx` в регистр `ax` запишется частное 1000h (результат деления 10002h на 10h), а в регистр `dx` — 2 (остаток от деления).

### 6.2.3. Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ

кодируется одним байтом. Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться. Среди инструкций NASM нет такой, которая выводит числа (не в символическом виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно. Для выполнения лабораторных работ в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- `iprint` – вывод на экран чисел в формате ASCII, перед вызовом `iprint` в регистр `eax` необходимо записать выводимое число (`mov eax, ...`).
- `iprintLF` – работает аналогично `iprint`, но при выводе на экран после числа добавляет к символ перевода строки.
- `atoi` – функция преобразует ASCII-код символа в целое число и запишет результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax, ...`).

## Выполнение лабораторной работы

- 1) Создаю каталог для программ лабораторной работы № 6, перехожу в него и создаю файл `lab6-1.asm`

```
nyemeckyai@fedora:~/work/arch-pc/lab05$ cd
nyemeckyai@fedora:~$ cd work
nyemeckyai@fedora:~/work$ cd arch-pc
nyemeckyai@fedora:~/work/arch-pc$ mkdir lab06
nyemeckyai@fedora:~/work/arch-pc$ cd lab06
nyemeckyai@fedora:~/work/arch-pc/lab06$ touch lab6-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
lab6-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06 — nano touch lab6-1.asm
~/.work/arch-pc/lab06
[1/2] touch
#include 'in_out.asm'
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx
mov [buf1], eax
mov eax, buf1
call sprintLF
call quit
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.asm touch
nyemeckyai@fedora:~/work/arch-pc/lab06$
nyemeckyai@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-1 alb6-1.o
```

```
mc [nyemeckyai@fedora]:~/work/arch-pc/lab06 — /usr/bin/mc -P /var/tmp/mc-ny...
~/work/arch-pc/lab06

lab6-1.asm [----] 0 L: [ 1+ 0 1/ 14] *(0 / 173b) 0037 0x025 [*] [X]
#include 'in_out.asm'
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintfLF
call quit
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ touch lab6-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.asm lab6-1.o lab6-2.asm touch
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
~/work/arch-pc/lab06

lab6-2.asm [----] 0 L: [ 1+ 0
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2 lab6-2.o
ld: cannot find lab6-2: No such file or directory
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
nyemeckyai@fedora:~/work/arch-pc/lab06$ ./lab6-2
106
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeck

GNU nano 8.1
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ nano lab6-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ nano lab6-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
nyemeckyai@fedora:~/work/arch-pc/lab06$ ./lab6-2
10
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
ld: cannot find variant.o: No such file or directory
nyemeckyai@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
nyemeckyai@fedora:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1032245942
Ваш вариант: 3
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ touch lab6-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3.asm touch
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ touch lab6-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3.asm touch
nyemeckyai@fedora:~/work/arch-pc/lab06$ nano lab6-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
nyemeckyai@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab06$ touch variant.asm
nyemeckyai@fedora:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.o lab6-2.asm lab6-3 lab6-3.o variant.asm
lab6-1.asm lab6-2 lab6-2.o lab6-3.asm touch
nyemeckyai@fedora:~/work/arch-pc/lab06$
```

```
mc [nyemeckyai@fedora]:~/work/arch-pc/lab06 — /usr/bin/mc -P /var/tmp/mc-ny...
~ /work/arch-pc/lab06

variant.asm [----] 0 L: [ 1+ 0 1/ 26] *(0 / 492b) 0037 0x025
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите № студенческого билета: ',0
rem: DB 'Ваш вариант: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintf
mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
```

## Ответы на вопросы для самопроверки

### 1. Какой синтаксис команды сложения чисел?

Сложение выполняется с помощью команды ADD:

ADD <операнд1>, <операнд2>

Где <операнд1> – регистр или ячейка памяти, в которую записывается результат, а <операнд2> – регистр, ячейка памяти или непосредственное значение.

Пример:

```
mov eax, 5      ; Загружаем 5 в EAX
add eax, 3      ; Прибавляем 3, теперь EAX = 8
```

### 2. Какая команда выполняет умножение без знака?

Для умножения без знака используется команда MUL:

MUL <операнд>

- <операнд> – множитель (может быть регистром или ячейкой памяти).

- Умножает значение регистра AX, EAX или RAX на <операнд>.

Пример (8-битное умножение):

```
mov al, 5      ; Загружаем 5 в AL
mov bl, 3      ; Загружаем 3 в BL
mul bl         ; AX = AL * BL (результат в AX)
```

Для 16-битных значений:

```
mov ax, 200
mov bx, 3
mul bx        ; DX:AX = AX * BX (результат в DX:AX)
```

---

### 3. Какой синтаксис команды деления чисел без знака?

Для деления без знака используется команда DIV:

DIV <операнд>

- Делит AX, DX:AX или EDX:EAX на <операнд>.
- Частное сохраняется в AL (8-бит), AX (16-бит), EAX (32-бит).
- Остаток сохраняется в AH, DX, EDX соответственно.

Пример (8-битное деление):

```
mov ax, 10
mov bl, 3
div bl      ; AL = частное, AH = остаток
```

Пример (16-битное деление):

```
mov dx, 0
mov ax, 100
mov bx, 7
div bx      ; AX = частное, DX = остаток
```

---

### 4. Куда помещается результат при умножении двухбайтовых операндов?

При умножении двухбайтовых (16-битных) чисел:

- **Результат занимает 32 бита.**
- **Старшая часть** (старшие 16 бит) сохраняется в DX.
- **Младшая часть** (младшие 16 бит) сохраняется в AX.

Пример:

```
mov ax, 200
mov bx, 300
mul bx      ; Результат: DX:AX = AX * BX
```

Если результат больше 65535 (FFFFh), старшие биты попадут в DX.



---

## 5. Перечислите арифметические команды с целочисленными операндами и дайте их назначение.

Команда	Описание
ADD	Сложение двух операндов (ADD <i>eax</i> , <i>ebx</i> )
SUB	Вычитание второго операнда из первого (SUB <i>eax</i> , <i>ebx</i> )
MUL	Умножение без знака (MUL <i>ebx</i> )
IMUL	Умножение со знаком (IMUL <i>ebx</i> )
DIV	Деление без знака (DIV <i>ebx</i> )
IDIV	Деление со знаком (IDIV <i>ebx</i> )
INC	Увеличение операнда на 1 (INC <i>eax</i> )
DEC	Уменьшение операнда на 1 (DEC <i>eax</i> )
NEG	Изменение знака числа (NEG <i>eax</i> , если <i>eax</i> = 5, станет <i>eax</i> = -5)

---

## 6. Где находится делимое при целочисленном делении операндов?

Делимое находится в следующих регистрах:

- **8-битное деление:** делимое в AX (AH:AL).
- **16-битное деление:** делимое в DX:AX (старшая часть в DX, младшая в AX).
- **32-битное деление:** делимое в EDX:EAX (старшая часть в EDX, младшая в EAX).

Пример для 16-битного деления:

```
mov dx, 0
mov ax, 1000    ; Делимое (DX:AX = 0000:03E8h = 1000)
mov bx, 25      ; Делитель
div bx          ; AX = частное, DX = остаток
```

---

## 7. Куда помещаются неполное частное и остаток при делении целочисленных операндов?

После команды DIV:

- **Частное** (результат деления) помещается в AL, AX или EAX.
- **Остаток** от деления сохраняется в AH, DX или EDX.

Разрядность	Делимое	Делитель	Частное	Остаток
8 бит	AX (AH:AL)	BL	AL	AH
16 бит	DX:AX	BX	AX	DX
32 бит	EDX:EAX	EBX	EAX	EDX

Пример 32-битного деления:

```
mov edx, 0
```

```
mov eax, 100000 ; Делимое (EDX:EAX = 0000:0186A0h)
mov ebx, 250    ; Делитель
div ebx         ; EAX = частное, EDX = остаток
```

Теперь:

- EAX = 400 (100000 / 250),
  - EDX = 0 (остаток 0).
- 

## Вывод

У меня получилось освоить арифметические инструкции языка ассемблера NASM