

# Отчёт по лабораторной работе №6

## Арифметические операции в NASM

Соловьев Богдан Михайлович

### Содержание

#### 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

#### 2 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`.

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 символьном виде). По- этому, 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 записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,`).

### 3 Выполнение лабораторной работы

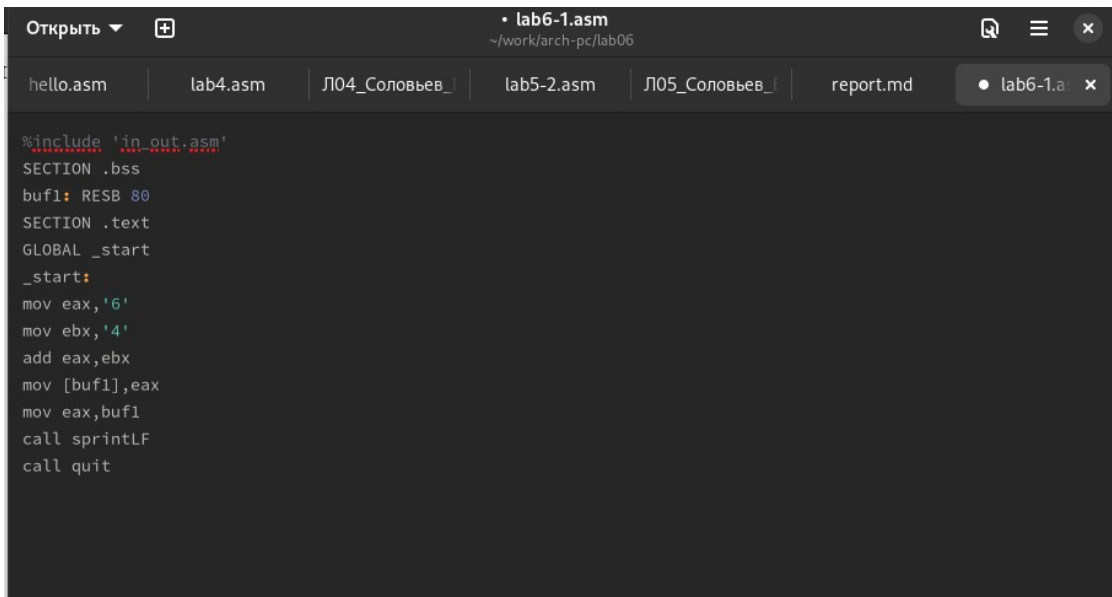
#### 3.1 Выполнение заданий лабораторной работы

Создаю новую директорию и новый файл `lab6-1.asm` (рис. 1).

```
[bmsolovjev@fedora ~]$ mkdir ~/work/arch-pc/lab06
[bmsolovjev@fedora ~]$ cd ~/work/arch-pc/lab06
[bmsolovjev@fedora lab06]$ touch lab6-1.asm
[bmsolovjev@fedora lab06]$
```

*Figure 1: Создание файла и директории*

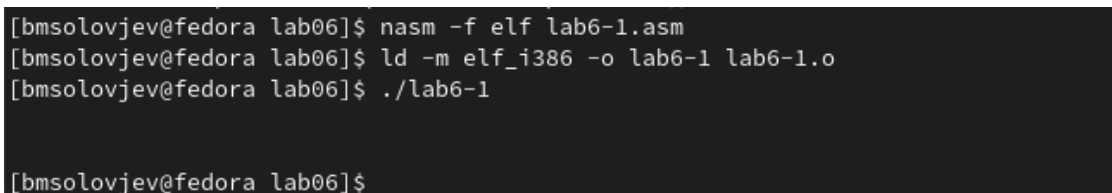
Записываю в файл `lab6-1.asm` код программы (рис. 2).



```
%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
```

Figure 2: Код в файле lab6-1.asm

Меняю в коде “4” и “6” на 4 и 6. Создаю и запускаю исполняемый файл (рис. 3).




```
[bmsolovjev@fedora lab06]$ nasm -f elf lab6-1.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[bmsolovjev@fedora lab06]$ ./lab6-1

[bmsolovjev@fedora lab06]$
```

Figure 3: Программа lab6-1.asm

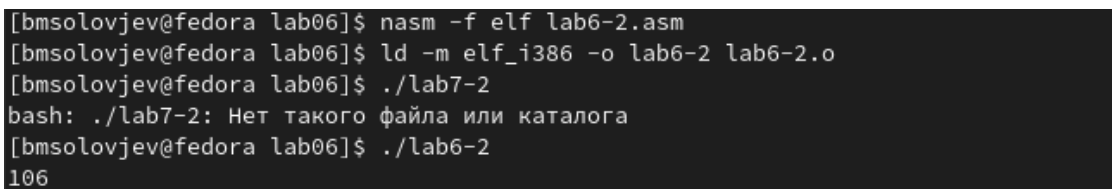
Создаю новый файл (рис. 4).



```
[bmsolovjev@fedora lab06]$ touch lab6-2.asm
```

Figure 4: Создание нового файла

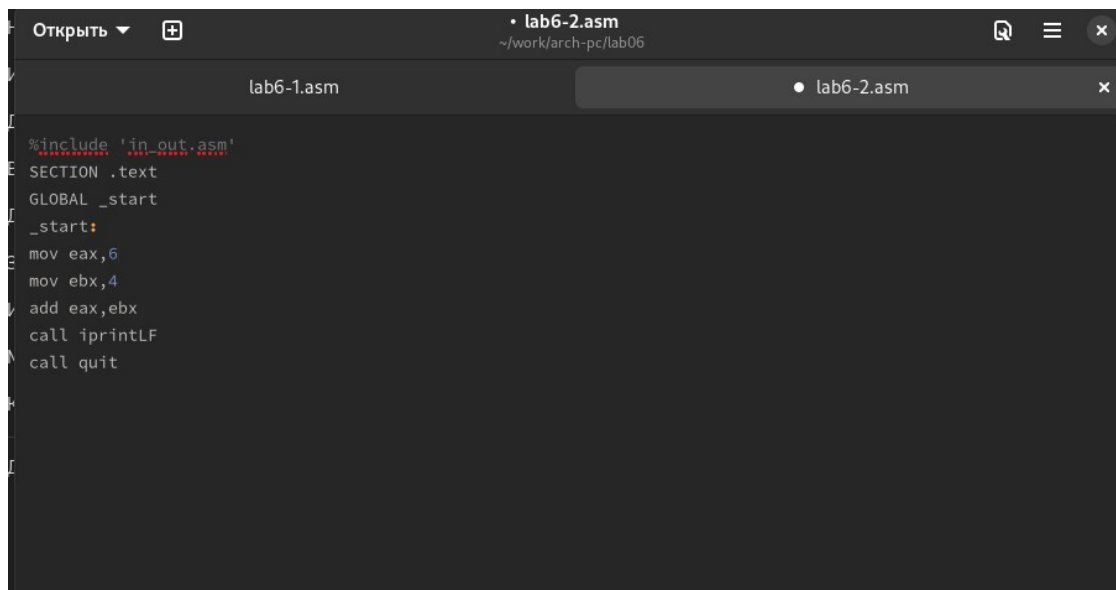
В файл lab6-2.asm копирую другой текст программы и запускаю исполняемый файл (рис. 5).



```
[bmsolovjev@fedora lab06]$ nasm -f elf lab6-2.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[bmsolovjev@fedora lab06]$ ./lab7-2
bash: ./lab7-2: Нет такого файла или каталога
[bmsolovjev@fedora lab06]$ ./lab6-2
106
```

Figure 5: Программа lab6-2.asm с кавычками

Меняю в коде программы lab6-2.asm “6” и “4” на 6 и 4 (рис. 6).



```
lab6-2.asm
~/.work/arch-pc/lab06

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

Figure 6:

Запускаю изменённый код программы (рис. 7).

```
[bmsolovjev@fedora lab06]$ nasm -f elf lab6-2.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[bmsolovjev@fedora lab06]$ ./lab6-2
10
```

Figure 7: Программа lab6-2.asm без кавычек

Создаю новый файл lab6-3.asm (рис. 8).

```
[bmsolovjev@fedora lab06]$ touch lab6-3.asm
[bmsolovjev@fedora lab06]$
```

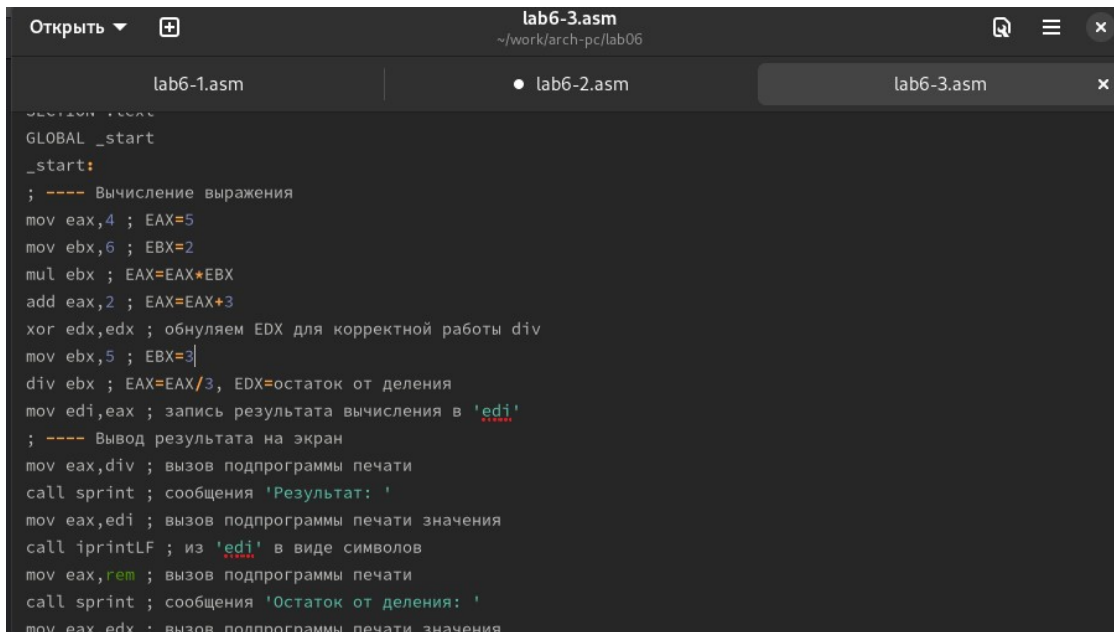
Figure 8: Создание файла lab6-3.asm

В этот файл я вставляю текст другой программы и запускаю программу(рис. 9).

```
[bmsolovjev@fedora lab06]$ touch lab6-3.asm
[bmsolovjev@fedora lab06]$ nasm -f elf lab6-3.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[bmsolovjev@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[bmsolovjev@fedora lab06]$
```

Figure 9: Программа lab6-3

Меняю код программы так, чтобы она вычисляла значение функции  $f(x) = (4 * 6 + 2) / 5$  (рис. 10).



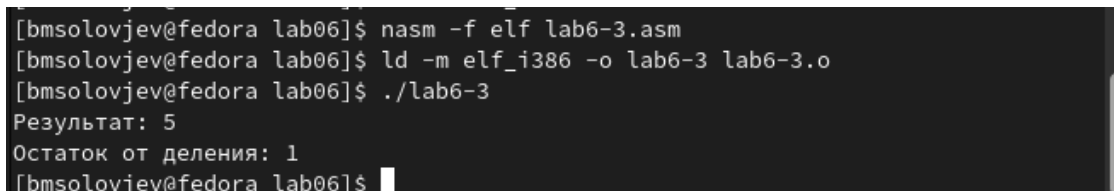
```
lab6-3.asm
~/.work/arch-pc/lab06

lab6-1.asm | lab6-2.asm | lab6-3.asm x

GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения
```

Figure 10: Новый код программы lab6-3

Запускаю исполняемый файл программы lab6-3 (рис. 11).



```
[bmsolovjev@fedora lab06]$ nasm -f elf lab6-3.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[bmsolovjev@fedora lab06]$ ./lab6-3
Результат: 5
Остаток от деления: 1
[bmsolovjev@fedora lab06]$
```

Figure 11: Изменённая программа lab6-3

### 3.2 Ответы на дополнительные вопросы

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения 'Ваш вариант: '? За вывод этих строк отвечают строки кода:

```
mov eax,rem
call sprint
```

2. Для чего используются следующие инструкции?

```
mov ecx, x ; перемещаем адрес строки x в регистр ecx
mov edx, 80 ; запись в регистр edx длины вводимой строки
call sread ; вызов подпрограммы, которая считывает сооб
```

3. Для чего используется инструкция "call atoi"? Эта инструкция преобразует ASCII код строки в целое число и записывает результат в регистр eax
4. Какие строки листинга 6.4 отвечают за вычисления варианта? За листинг отвечают следующие строки:

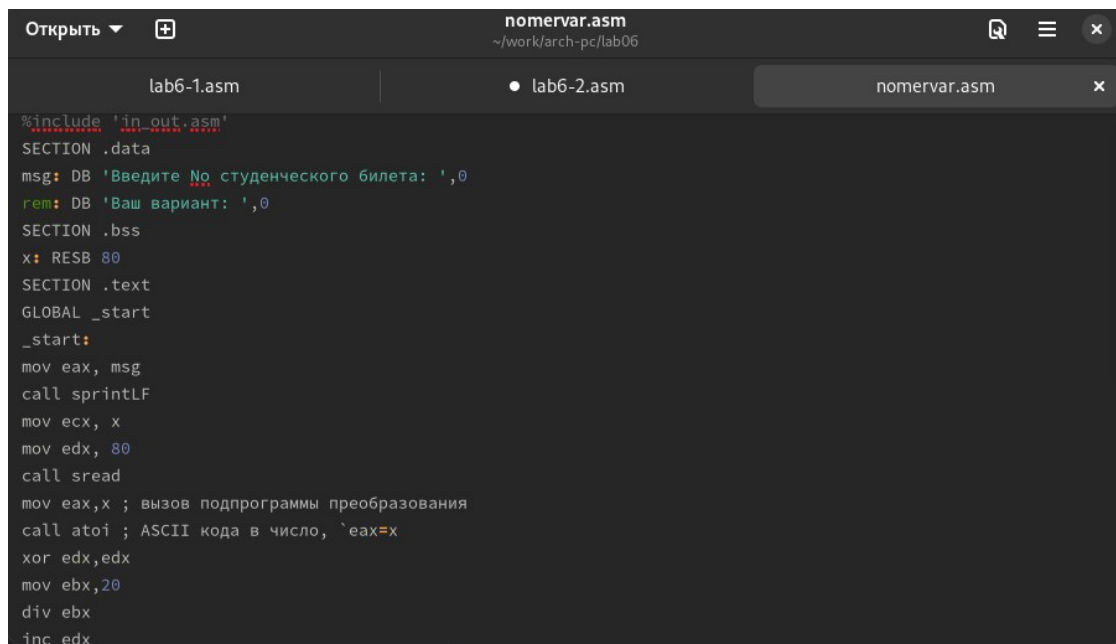
```
xor edx,edx ; обнуление edx для корректной работы div
mov ebx,20 ; ebx = 20
div ebx ; eax = eax/20, edx - остаток от деления
inc edx ; edx = edx + 1
```

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? Остаток от деления записывается в регистр edx
6. Для чего используется инструкция “inc edx”? Эта инструкция увеличивает значение регистра edx на 1
7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений? За вывод на экран результатов вычислений отвечают строки:

```
mov eax,edx
call iprintLF
```

### 3.3 Задачи для самостоятельного решения

Создаю файл nomervar.asm, где ввожу код для программы, которая будет выводить номер варианта (рис. 12).



```
Открыть ▾ + nomervar.asm
~/work/arch-pc/lab06

lab6-1.asm | lab6-2.asm | nomervar.asm x
%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 sprintLF
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
```

Figure 12: Программа, считающая номер варианта

Запускаю программу (рис. 13).

```
[bmsolovjev@fedora lab06]$ nasm -f elf nomervar.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o nomervar nomervar.o
[bmsolovjev@fedora lab06]$ ./nomervar
Введите No студенческого билета:
1132236042
Ваш вариант: 3
[bmsolovjev@fedora lab06]$
```

Figure 13: Мой вариант

Код моей программы (рис. 14).

```
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
add eax, 2
mov ebx, eax
mul ebx
mov edi, eax
; ---- Вывод результата на экран
mov eax, rem ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax, edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
```

Figure 14: Код моей программы

Проверка моей программы (рис. 15).

```
[bmsolovjev@fedora lab06]$ nasm -f elf zadacha.asm
[bmsolovjev@fedora lab06]$ ld -m elf_i386 -o zadacha zadacha.o
[bmsolovjev@fedora lab06]$ ./zadacha
(2+x)^2 Введите x: 7
Результат y = 81
[bmsolovjev@fedora lab06]$ 98
bash: 98: команда не найдена...
^[A[bmsolovjev@fedora lab06]$ ./zadacha
(2+x)^2 Введите x: 98
Результат y = 10000
[bmsolovjev@fedora lab06]$ ./zadacha
(2+x)^2 Введите x: 12
Результат y = 196
[bmsolovjev@fedora lab06]$
```

Figure 15: Проверка моей программы

## 4 Выводы

При выполнении данной лабораторной работы я освоил арифметические инструкции языка ассемблера NASM.

## Список литературы