

# 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: Трофимов Владислав Алексеевич

Группа: НКАбд-06-25

МОСКВА

2025 г.

# Содержание

Список иллюстраций .....	3
1 Цель работы .....	4
2 Задание .....	5
3 Теоретическое введение .....	6
4 Выполнение лабораторной работы .....	8
4.1 Символьные и численные данные в NASM .....	8
4.2 Выполнение арифметических операций в NASM .....	12
4.3 Ответы на контрольные вопросы .....	15
4.4 Задание для самостоятельной работы .....	16
5 Выводы .....	17
6 Список литературы .....	18

## **Список иллюстраций**

Рис. 4.1: Создание нового каталога

Рис. 4.2: Сохранение новой программы

Рис. 4.3: Запуск изначальной программы

Рис. 4.4: Измененная программа

Рис. 4.5: Запуск измененной программы

Рис. 4.6: Вторая программа

Рис. 4.7: Вывод второй программы

Рис. 4.8: Вывод измененной второй программы

Рис. 4.9: Замена функции вывода во второй программе

Рис. 4.10: Третья программа

Рис. 4.11: Запуск третьей программы

Рис. 4.12: Изменение третьей программы

Рис. 4.13: Запуск измененной третьей программы

Рис. 4.14: Программа для подсчета варианта

Рис. 4.15: Запуск программы для подсчета варианта

Рис. 4.16: Запуск и проверка программы

# 1 Цель работы

Цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

## **2 Задание**

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий для самостоятельной работы

### 3 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. - 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. 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. Среди инструкций NASM нет такой, которая выводит числа (не в символьном 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.

## 4 Выполнение лабораторной работы

### 4.1 Символьные и численные данные в NASM

Создаю каталог для программ лабораторной работы №6 и перехожу в него, создаю там файл (рис. 4.1).

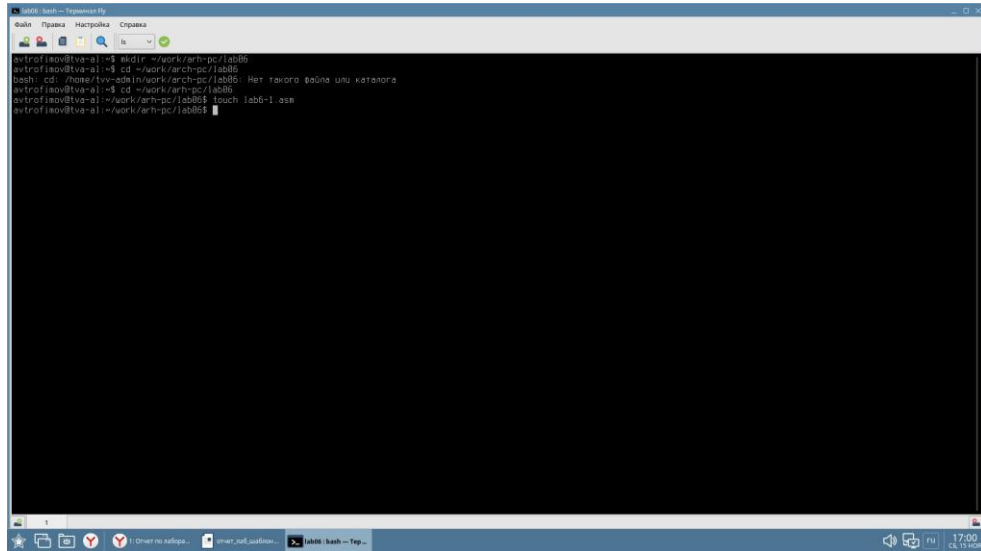


Рис. 4.1: Создание нового каталога

В созданном файле ввожу программу из листинга (рис. 4.2).

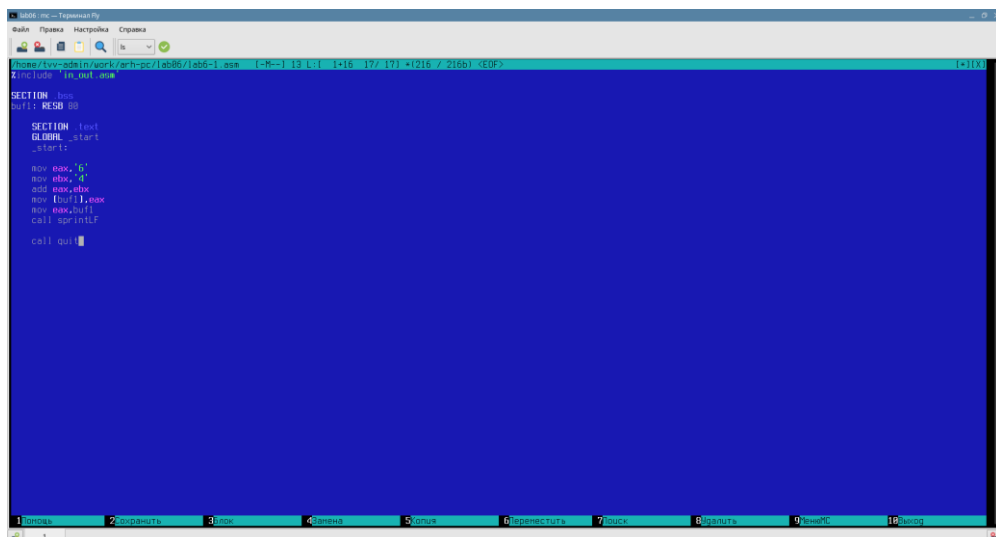


Рис. 4.2: Сохранение новой программы



Создаю исполняемый файл и запускаю его, вывод программы отличается от предполагаемого изначально, ибо коды символов в сумме дают символ j по таблице ASCII. (рис. 4.3)

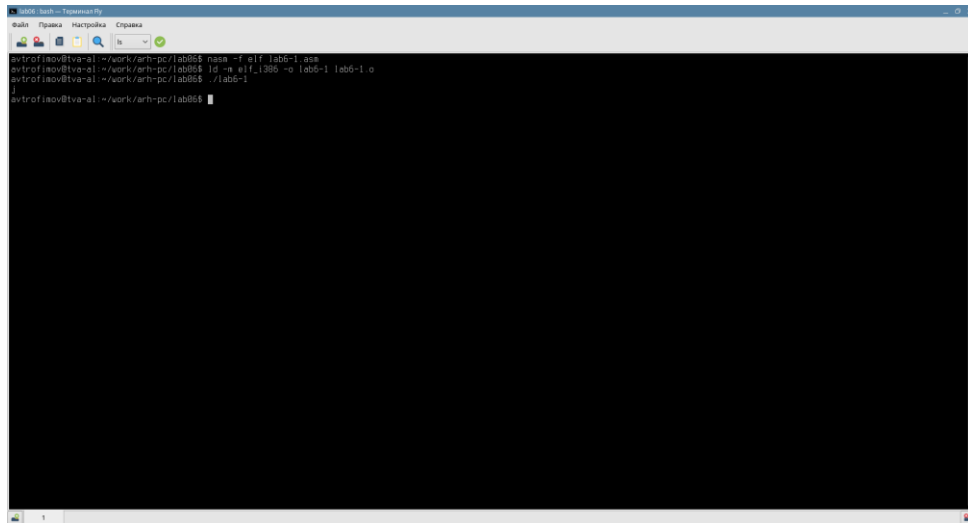


Рис. 4.3: Запуск inicialьной программы

Изменяю текст inicialьной программы, убрав кавычки (рис. 4.4).

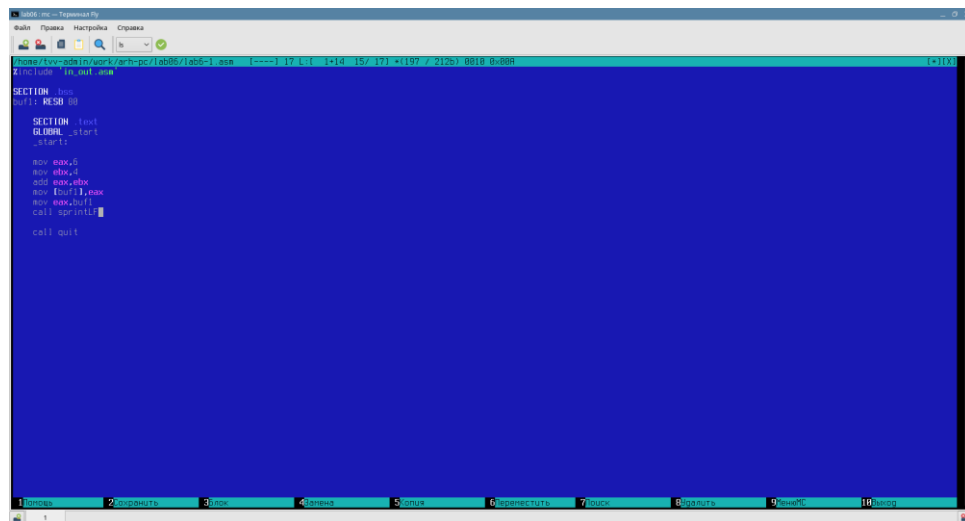


Рис. 4.4: Измененная программа

На этот раз программа выдала пустую строку, это связано с тем, что символ 10 означает переход на новую строку (рис. 4.5).

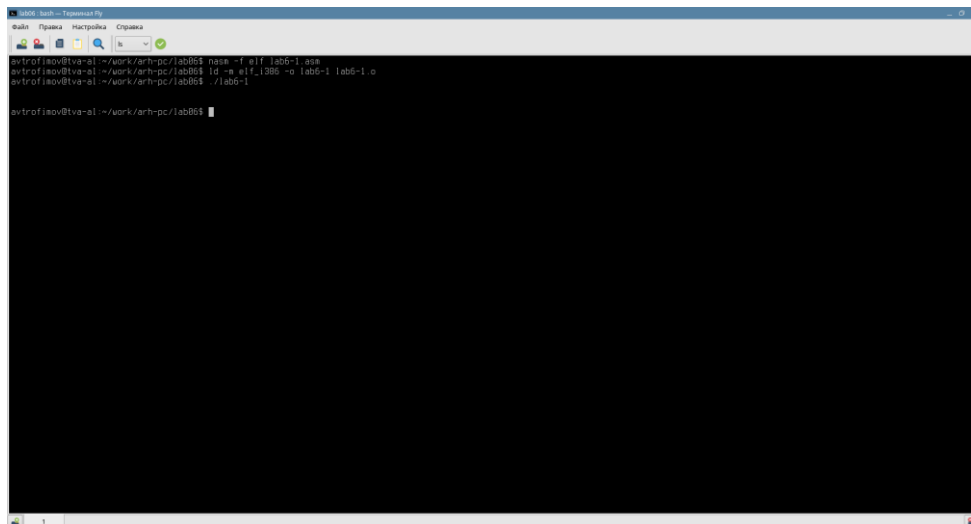


Рис. 4.5: Запуск измененной программы

Создаю новый файл для будущей программы и записываю в нее код из листинга (рис. 4.6).

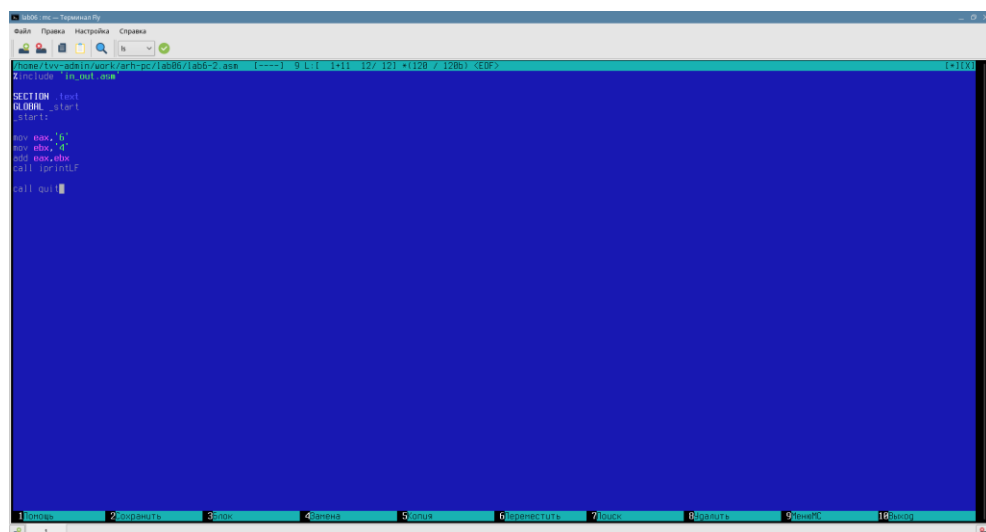
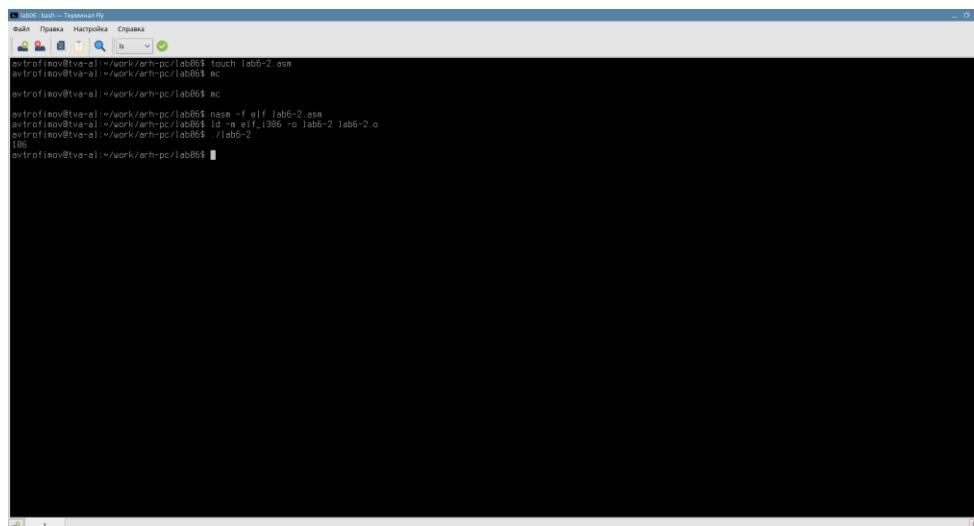


Рис. 4.6: Вторая программа

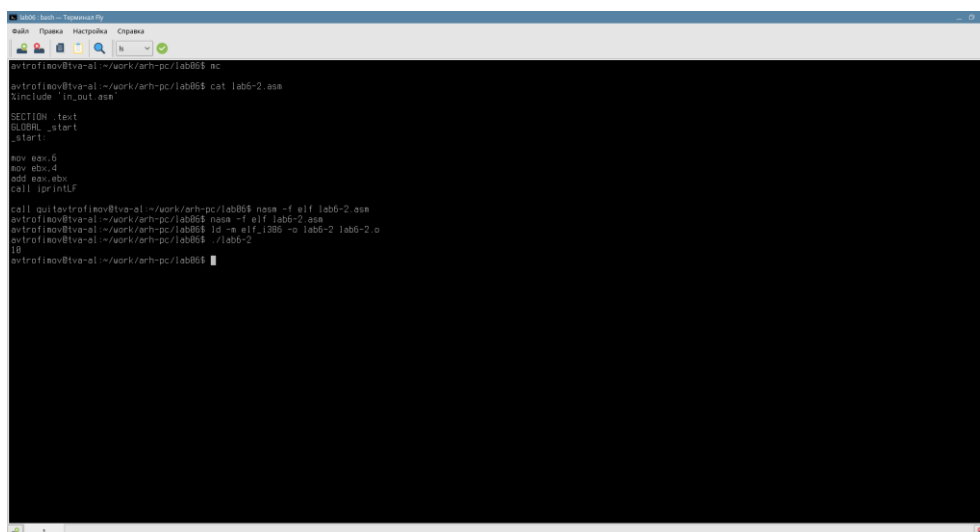
Создаю исполняемый файл и запускаю его, теперь отображается результат 106, программа, как и в первый раз, сложила коды символов, но вывела само число, а не его символ, благодаря замене функции вывода на `iprintLF` (рис. 4.7).



```
avtrofiyov@tva-al:~/work/erh-pc/lab86$ touch lab6-2.asm
avtrofiyov@tva-al:~/work/erh-pc/lab86$ nc
avtrofiyov@tva-al:~/work/erh-pc/lab86$ nc
avtrofiyov@tva-al:~/work/erh-pc/lab86$ nasm -f elf lab6-2.asm
avtrofiyov@tva-al:~/work/erh-pc/lab86$ ld -o elf_1386 -o lab6-2 lab6-2.o
avtrofiyov@tva-al:~/work/erh-pc/lab86$ ./lab6-2
186
avtrofiyov@tva-al:~/work/erh-pc/lab86$
```

Рис. 4.7: Вывод второй программы

Убрав кавычки в программе, я снова ее запускаю и получаю предполагаемый изначально результат. (рис. 4.8).



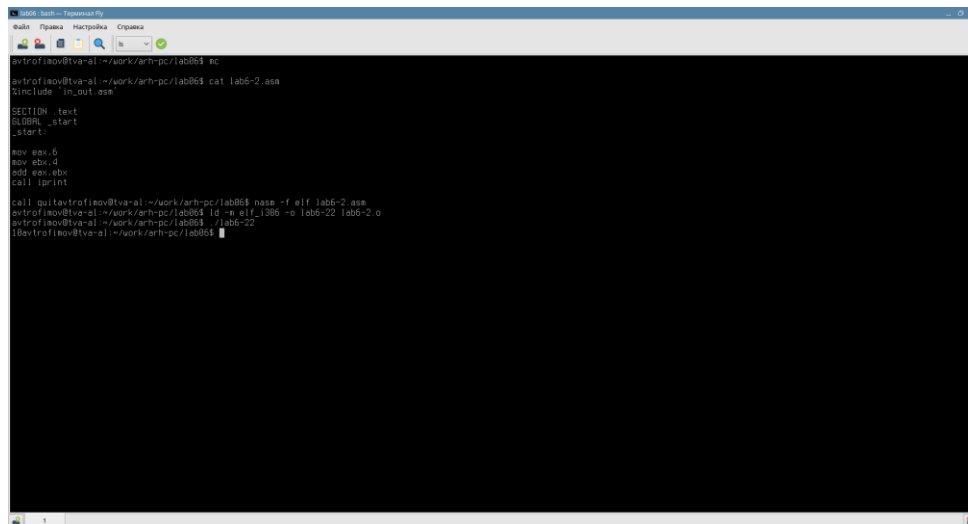
```
avtrofiyov@tva-al:~/work/erh-pc/lab86$ nc
avtrofiyov@tva-al:~/work/erh-pc/lab86$ cat lab6-2.asm
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:
mov ebx, 0
mov ebx, 4
add eax, ebx
call _printf

call quitavtrofiyov@tva-al:~/work/erh-pc/lab86$ nasm -f elf lab6-2.asm
avtrofiyov@tva-al:~/work/erh-pc/lab86$ nasm -f elf lab6-2.asm
avtrofiyov@tva-al:~/work/erh-pc/lab86$ ld -o elf_1386 -o lab6-2 lab6-2.o
avtrofiyov@tva-al:~/work/erh-pc/lab86$ ./lab6-2
186
avtrofiyov@tva-al:~/work/erh-pc/lab86$
```

Рис. 4.8: Вывод измененной второй программы

Заменив функцию вывода на `iprintf`, я получаю тот же результат, но без переноса строки (рис. 4.9).



```
avtrofiyov@tve-al:~/work/arm-pc/lab06$ nc
avtrofiyov@tve-al:~/work/arm-pc/lab06$ cat lab6-2.asm
include "in_out.asm"

SECTION .text
GLOBAL _start
_start:

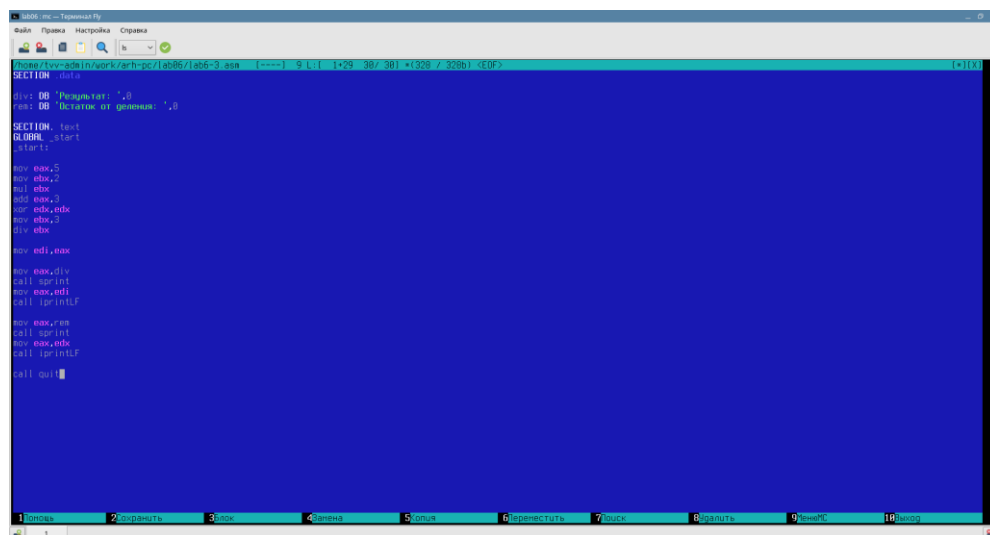
mov eax,5
mov ebx,4
add ebx,ebx
call iprint

call quitavtrofiyov@tve-al:~/work/arm-pc/lab06$ nasm -f elf lab6-2.asm
avtrofiyov@tve-al:~/work/arm-pc/lab06$ ld -r elf_1386 -o lab6-22 lab6-2.o
avtrofiyov@tve-al:~/work/arm-pc/lab06$ ./lab6-22
18avtrofiyov@tve-al:~/work/arm-pc/lab06$
```

Рис. 4.9: Замена функции вывода во второй программе

## 4.2 Выполнение арифметических операций в NASM

Создаю новый файл и копирую в него содержимое листинга (рис. 4.10).



```
avtrofiyov@tve-al:~/work/arm-pc/lab06$ cat lab6-3.asm
[----] 9 L 1 1+29 36/ 301 +1328 / 328b) <EDF>
SECTION .data
div: DB "Результат: ",0
rest: DB "Остаток от деления: ",0

SECTION .text
GLOBAL _start
_start:

mov eax,5
mov ebx,2
mul ebx
mov ecx,3
div ebx,ecx
mov ebx,2
div ebx

mov edi,eax
mov ecx,div
call printf
mov eax,edi
call printf

mov ecx,rest
call printf
mov eax,ecx
call printf

call quit
```

Рис. 4.10: Третья программа

Программа выполняет арифметические вычисления, на вывод идет результирующее выражения и его остаток от деления (рис. 4.11).

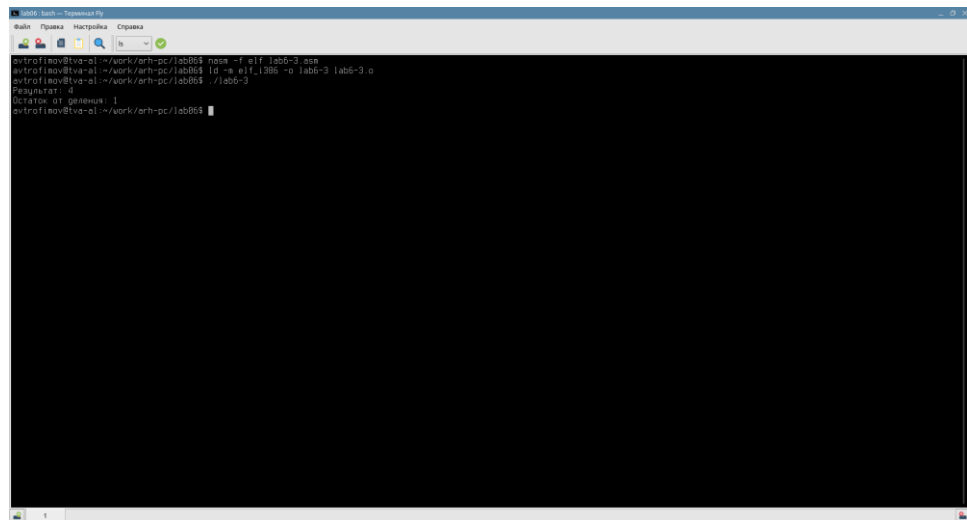


Рис. 4.11: Запуск третьей программы

Заменяя переменные в программе для выражения  $f(x) = (4*6+2)/5$  (рис. 4.12).

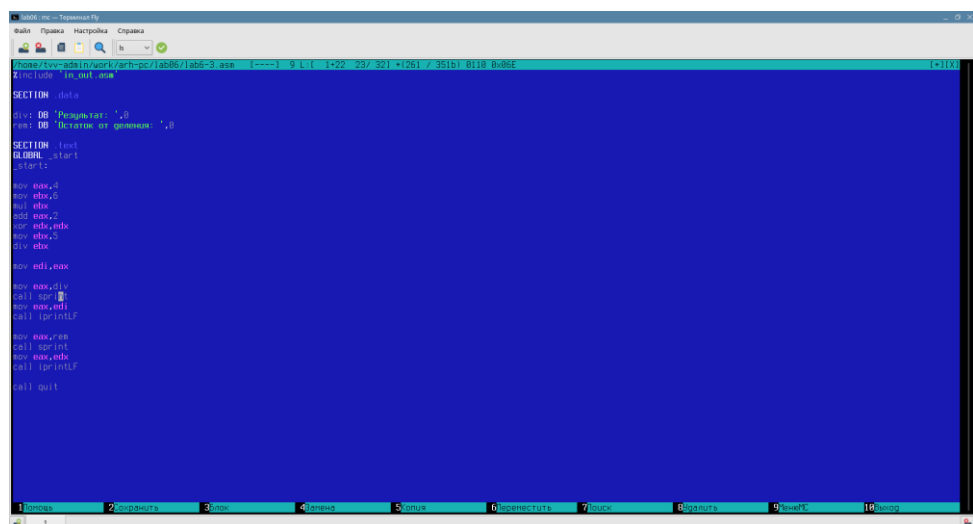
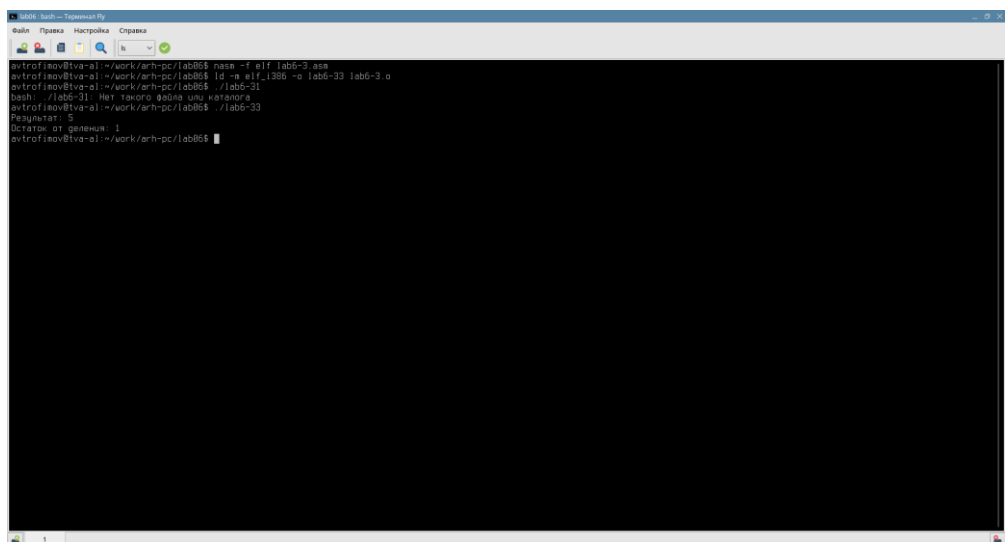


Рис. 4.12: Изменение третьей программы

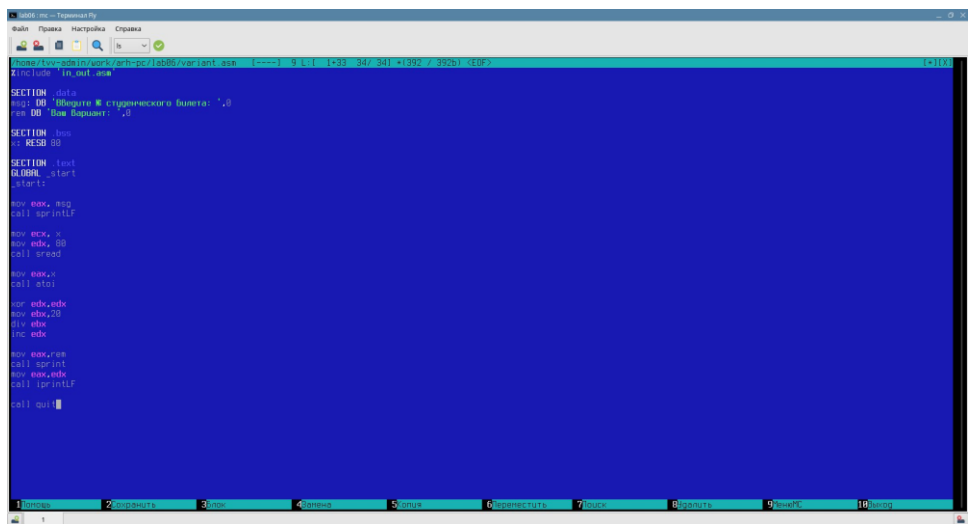
Запуск программы дает корректный результат (рис. 4.13).



```
avtrofiov@iva-al:~/work/ark-pc/lab86$ nasa -f elf lab6-3.asm
avtrofiov@iva-al:~/work/ark-pc/lab86$ ld -m elf_i386 -o lab6-33 lab6-3.o
avtrofiov@iva-al:~/work/ark-pc/lab86$ ./lab6-33
bash: ./lab6-33: Нет такого файла или каталога
avtrofiov@iva-al:~/work/ark-pc/lab86$ ./lab6-33
Результат: 5
Остаток от деления: 1
avtrofiov@iva-al:~/work/ark-pc/lab86$
```

Рис. 4.13: Запуск измененной третьей программы

Создаю новый файл и помещаю текст из листинга (рис. 4.14).



```
SECTION .data
msg DD "Введите № студенческого билета: ",0
len DD "Ваш вариант: ",0

SECTION .bss
r: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprintf

mov ecx, x
mov edx, 80
call read

mov ecx, r
call atoi

mov ebx, edx
mov ebx, 20
div ebx
inc ebx

mov ecx, ror
call sprintf
call sprintf
call printf

call quit
```

Рис. 4.14: Программа для подсчета варианта

Запустив программу и указав свой номер студенческого билета, я получил свой вариант для дальнейшей работы. (рис. 4.15).

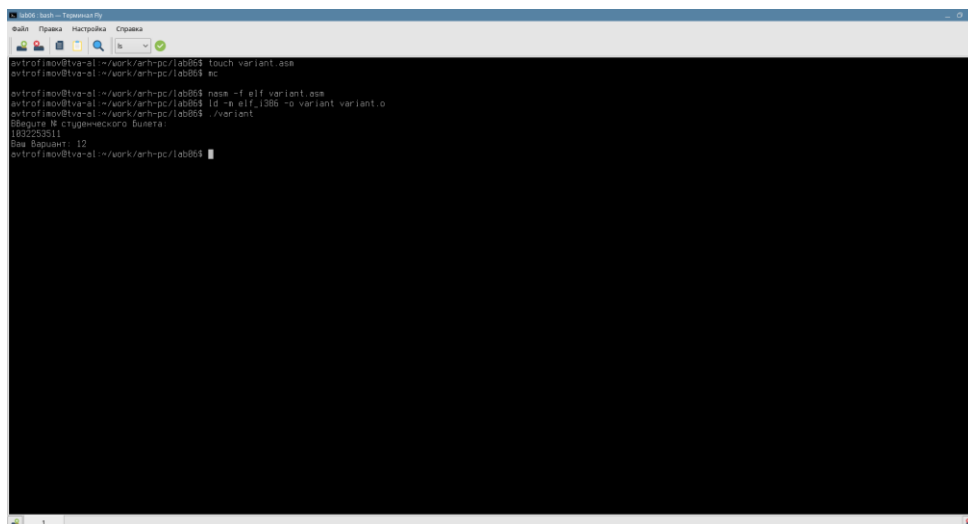


Рис. 4.15: Запуск программы для подсчета варианта

### 4.3 Ответы на контрольные вопросы

1. За вывод сообщения “Ваш вариант” отвечают строки кода:

```
mov eax,rem
```

```
call sprint
```

2. Инструкция `mov esx, x` используется, чтобы положить адрес вводимой строки `x` в регистр `esx` `mov edx, 80` - запись в регистр `edx` длины вводимой строки `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры.

3. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.

4. За вычисления варианта отвечают строки:

```
xor edx,edx
```

```
mov ebx,20
```

```
div ebx
```

```
inc edx
```

5. При выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx`.

6. Инструкция `inc edx` увеличивает значение регистра `edx` на 1.

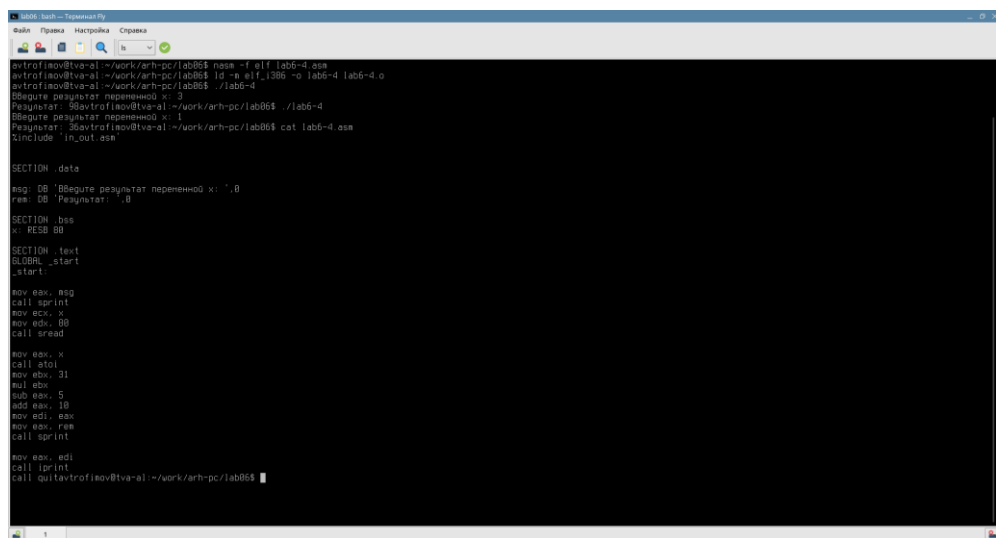
7. За вывод на экран результатов вычислений отвечают строки:

```
mov eax,edx
```

```
call iprintLF
```

## 4.4 Задание для самостоятельной работы

В соответствии с выбранным вариантом, я реализую программу для подсчета функции  $f(x) = 10 + (31x - 5)$ , проверка на нескольких переменных показывает корректное выполнение программы (рис. 4.16).



```
avtrofi@avtrofi:~$ nasm -f elf lab6-4.asm
avtrofi@avtrofi:~$ ld -m elf_i386 -o lab6-4 lab6-4.o
avtrofi@avtrofi:~$ ./lab6-4
Введите результат переменной x: 3
Результат: 96avtrofi@avtrofi:~$ ./lab6-4
Введите результат переменной x: 1
Результат: 36avtrofi@avtrofi:~$ cat lab6-4.asm
#include "in_out.asm"

SECTION .data
msg: db "Введите результат переменной x: ",0
res: db "Результат: ",0

SECTION .bss
x: resb 80

SECTION .text
GLOBAL _start

_start:

mov eax, msg
call sprint
mov ecx, x
mov edx, 08
call sread

mov eax, x
call stol
mov ebx, 31
mul ebx
add eax, 5
add eax, 19
mov edi, eax
mov ebx, res
call sprint

mov ebx, edi
call iprint
call quitavtrofi@avtrofi:~$
```

Рис. 4.16: Запуск и проверка программы



## **5 Выводы**

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

## 6 Список литературы

[Курс ТУИС](#)

[Лабораторная работа №6](#)

[Программирование на языке ассемблера NASM Столяров А. В.](#)