

# **Отчёт по лабораторной работе 6**

**Архитектура компьютера**

Нилова Кристина Артуровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задания</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Символьные и численные данные в NASM . . . . .	8
4.2	Выполнение арифметических операций в NASM . . . . .	14
4.2.1	Ответы на вопросы . . . . .	19
4.3	Самостоятельное задание . . . . .	20
<b>5</b>	<b>Выводы</b>	<b>23</b>

## Список иллюстраций

4.1	Создание каталога и файла . . . . .	8
4.2	Заполнение файла lab6-1.asm . . . . .	9
4.3	Компиляция текста программы lab6-1.asm . . . . .	10
4.4	Заполнение файла lab6-1.asm . . . . .	11
4.5	Компиляция текста программы lab6-1.asm . . . . .	11
4.6	Заполнение файла lab6-2.asm . . . . .	12
4.7	Компиляция текста программы lab6-2.asm . . . . .	12
4.8	Заполнение файла lab6-2.asm . . . . .	13
4.9	Компиляция текста программы lab6-2.asm . . . . .	14
4.10	Компиляция текста программы lab6-2.asm . . . . .	14
4.11	Заполнение файла lab6-3.asm . . . . .	15
4.12	Компиляция текста программы lab6-3.asm . . . . .	15
4.13	Заполнение файла lab6-3.asm . . . . .	16
4.14	Компиляция текста программы lab6-3.asm . . . . .	17
4.15	Заполнение файла variant.asm . . . . .	18
4.16	Компиляция текста программы variant.asm . . . . .	19
4.17	Заполнение файла prog.asm . . . . .	21
4.18	Компиляция текста программы prog.asm . . . . .	22

## Список таблиц

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

## 2 Задания

1. Изучение арифметических действий
2. Примеры программ с вычислениями
3. Выполнение заданий для самостоятельной работы.

### 3 Теоретическое введение

Схема команды целочисленного сложения `add` (от англ. `addition` - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда.

Команда целочисленного вычитания `sub` (от англ. `subtraction` – вычитание) работает аналогично команде `add`.

Существуют специальные команды: `inc` (от англ. `increment`) и `dec` (от англ. `decrement`), которые увеличивают и уменьшают на 1 свой операнд.

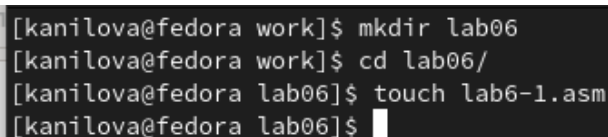
Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производятся по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда `mul` (от англ. `multiply` – умножение), для знакового умножения используется команда `imul`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. `divide` - деление) и `idiv`

## 4 Выполнение лабораторной работы

### 4.1 Символьные и численные данные в NASM

Я создала каталог для хранения программ к лабораторной работе номер шесть, после чего перешла в этот каталог и создала файл с исходным кодом программы под названием lab6-1.asm, как показано на рисунке [4.1].

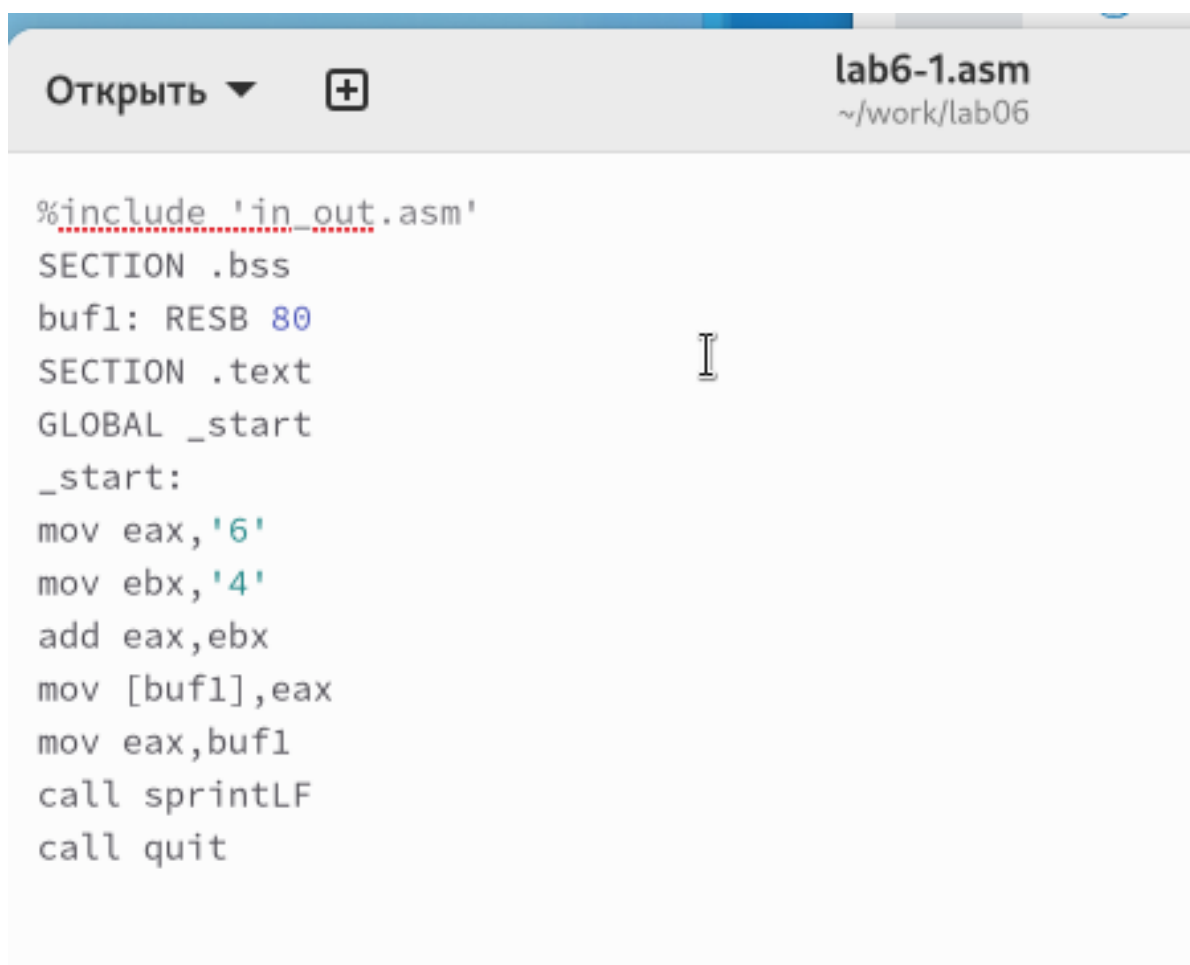


```
[kanilova@fedora work]$ mkdir lab06  
[kanilova@fedora work]$ cd lab06/  
[kanilova@fedora lab06]$ touch lab6-1.asm  
[kanilova@fedora lab06]$
```

Рис. 4.1: Создание каталога и файла

Теперь давайте взглянем на примеры программ, которые отображают символы и числовые значения на экран. Эти программы будут использовать регистр `eax` для вывода значений.






```
Открыть ▼  lab6-1.asm  
~/work/lab06  
  
%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
```

Рис. 4.2: Заполнение файла lab6-1.asm

В одном из примеров, который вы можете увидеть на рисунке [4.2], программа записывает символ '6' в регистр `eax` с помощью инструкции `mov eax, '6'`, а символ '4' помещается в регистр `ebx` с помощью инструкции `mov ebx, '4'`. Затем я выполняю сложение значений, находящихся в регистрах `ebx` и `eax`, и результат записывается обратно в `eax` с помощью инструкции `add eax, ebx`. После этого нужно вывести результат на экран.

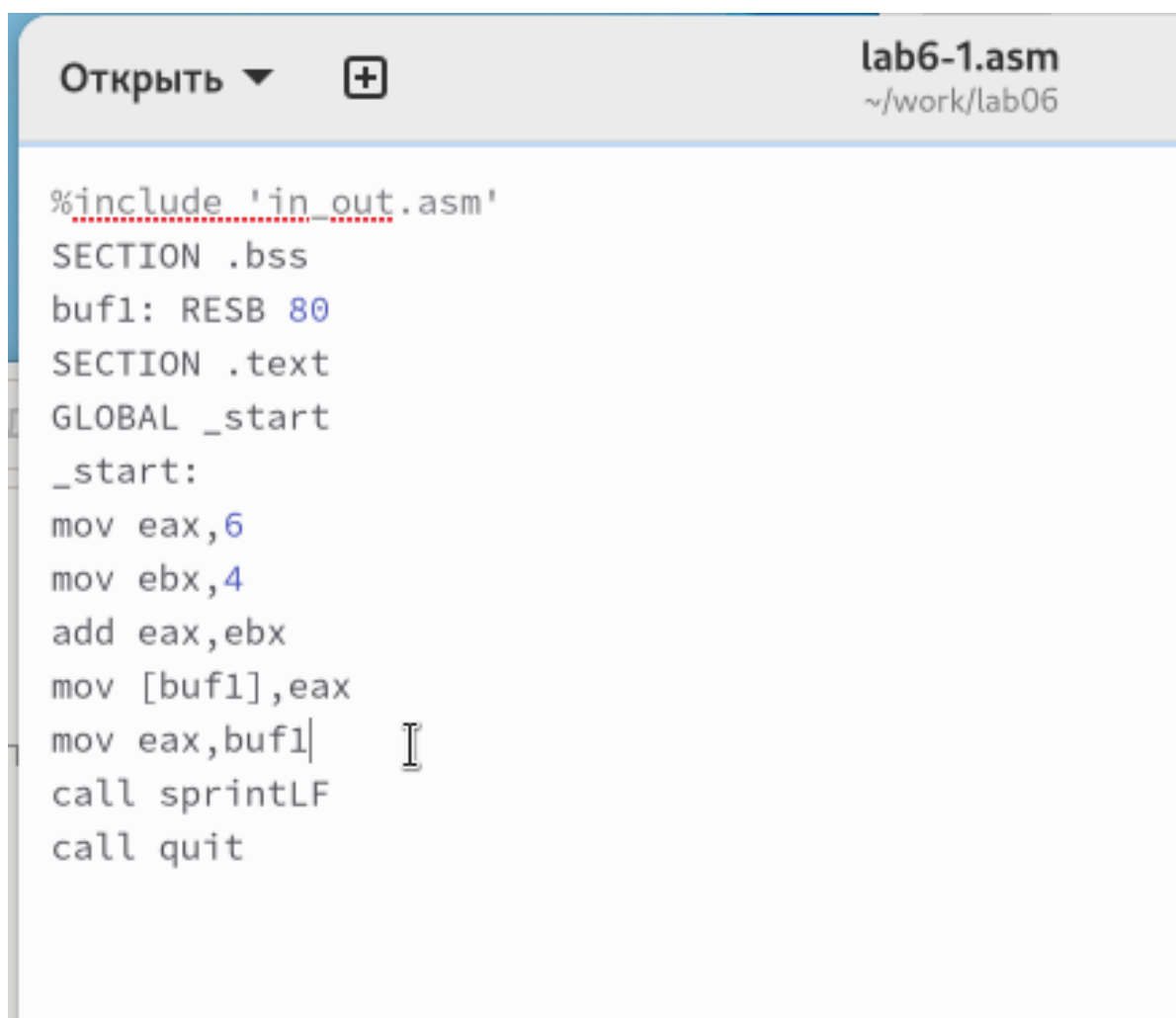
```
[kanilova@fedora lab06]$  
[kanilova@fedora lab06]$ nasm -f elf lab6-1.asm  
[kanilova@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1  
[kanilova@fedora lab06]$ ./lab6-1  
j  
[kanilova@fedora lab06]$
```

Рис. 4.3: Компиляция текста программы lab6-1.asm

Однако, чтобы функция `sprintLF` работала правильно, необходимо передать в `eax` адрес данных. Поэтому я использую вспомогательную переменную `buf1`. Сначала я перемещаю значение из `eax` в `buf1` с помощью инструкции `mov [buf1], eax`, а потом загружаю адрес переменной `buf1` обратно в регистр `eax` (`mov eax, buf1`), прежде чем вызвать `sprintLF`.

В итоге, вместо того чтобы увидеть на экране число 10, мы видим символ 'j'. Это происходит из-за особенностей кодировки символов: символ '6' имеет двоичный код 00110110 (или 54 в десятичном виде), а '4' — 00110100 (или 52 в десятичном виде). Сложив эти значения, мы получаем двоичный код 01101010, что в десятичной системе равно 106, и это соответствует символу 'j', как показано на рисунке [4.3].

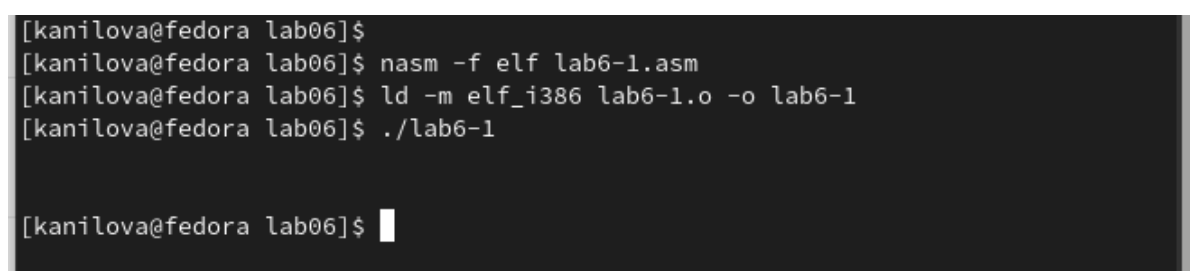
Теперь я внесу изменения в исходный код программы, чтобы в регистрах хранились числовые значения, а не символы, как это показано на иллюстрации [4.4].



```
Открыть ▼ + lab6-1.asm
~/work/lab06

%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
```

Рис. 4.4: Заполнение файла lab6-1.asm



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

[kanilova@fedora lab06]$
```

Рис. 4.5: Компиляция текста программы lab6-1.asm

Когда я запускаю программу, она не выдаёт число 10. На экран выводится символ с ASCII-кодом 10, который означает новую строку, как видно на рисунке

[4.5]. Этот символ невидим в терминале, но он создаёт пустую строку в моём выводе.

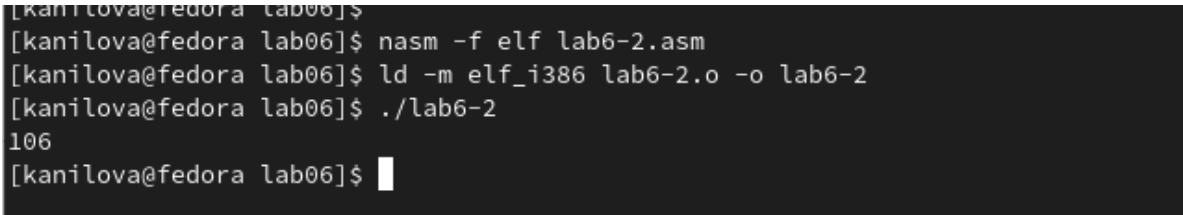
Я помню, что ранее в файле `in_out.asm` были созданы специальные функции, чтобы преобразовывать символы ASCII в числа и обратно. Используя эти функции, я изменила код программы, что можно увидеть на иллюстрации [4.6].



```
Открыть + lab6-2.asm
~/work/lab06

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

Рис. 4.6: Заполнение файла `lab6-2.asm`



```
[kanilova@fedora lab06]$
[kanilova@fedora lab06]$ nasm -f elf lab6-2.asm
[kanilova@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[kanilova@fedora lab06]$ ./lab6-2
106
[kanilova@fedora lab06]$
```

Рис. 4.7: Компиляция текста программы `lab6-2.asm`

Когда я запустила модифицированную программу, на экране появилось число 106, как это показано на рисунке [4.7]. В этом случае, так же как и в первом

примере, команда `add` сложила ASCII-коды цифр '6' и '4' ( $54+52=106$ ). Но благодаря функции `iprintLF`, на этот раз на экран вывелось именно число, а не символ, соответствующий его ASCII-коду.

Таким же образом, как и в предыдущем примере, я заменила символы на их числовые эквиваленты, что изображено на рисунке [4.8].



```
Открыть ▾ + lab6-2.asm
~/work/lab06

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

Рис. 4.8: Заполнение файла lab6-2.asm

В этой ситуации функция `iprintLF` использовалась для отображения числового значения, так как операнды были числами, а не символьными кодами. Это привело к результату, который равен числу 10, как можно увидеть на рисунке [4.9].

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

Рис. 4.9: Компиляция текста программы lab6-2.asm

Затем я заменила функцию `iprintLF` на функцию `iprint`. После того, как я скомпилировала и запустила программу, результат изменился: теперь вывод не содержал переноса строки, что иллюстрируется на рисунке [4.10].

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

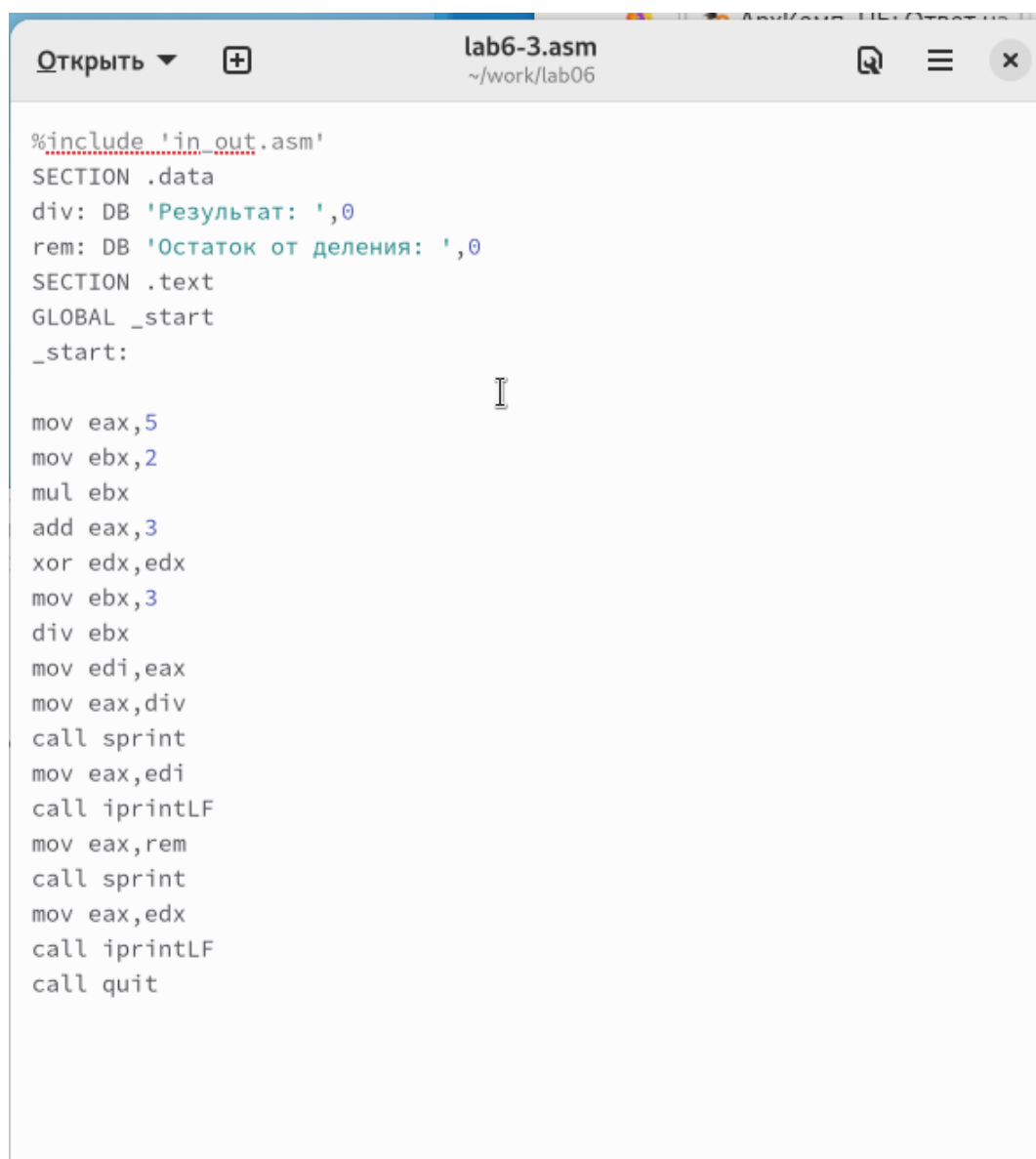
Рис. 4.10: Компиляция текста программы lab6-2.asm

## 4.2 Выполнение арифметических операций в NASM

Возьмем, к примеру, код, который я написала для вычисления функции

$$f(x) = (5 * 2 + 3) / 3$$

. Вы можете посмотреть, как он работает, на иллюстрациях [4.11] и [4.12].

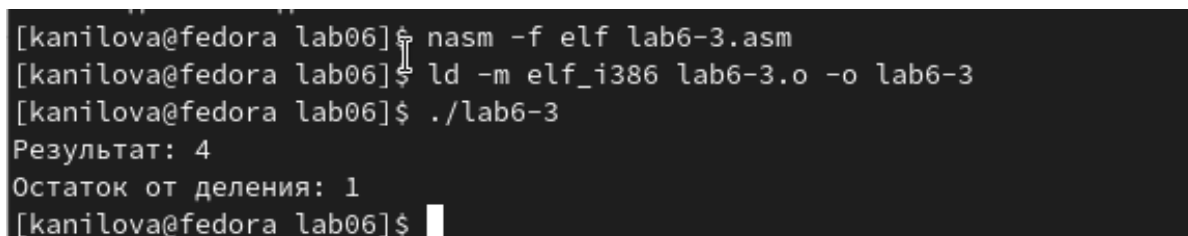


```
Открыть + lab6-3.asm ~/work/lab06

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,5
mov ebx,2
mul ebx
add eax,3
xor edx,edx
mov ebx,3
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 4.11: Заполнение файла lab6-3.asm



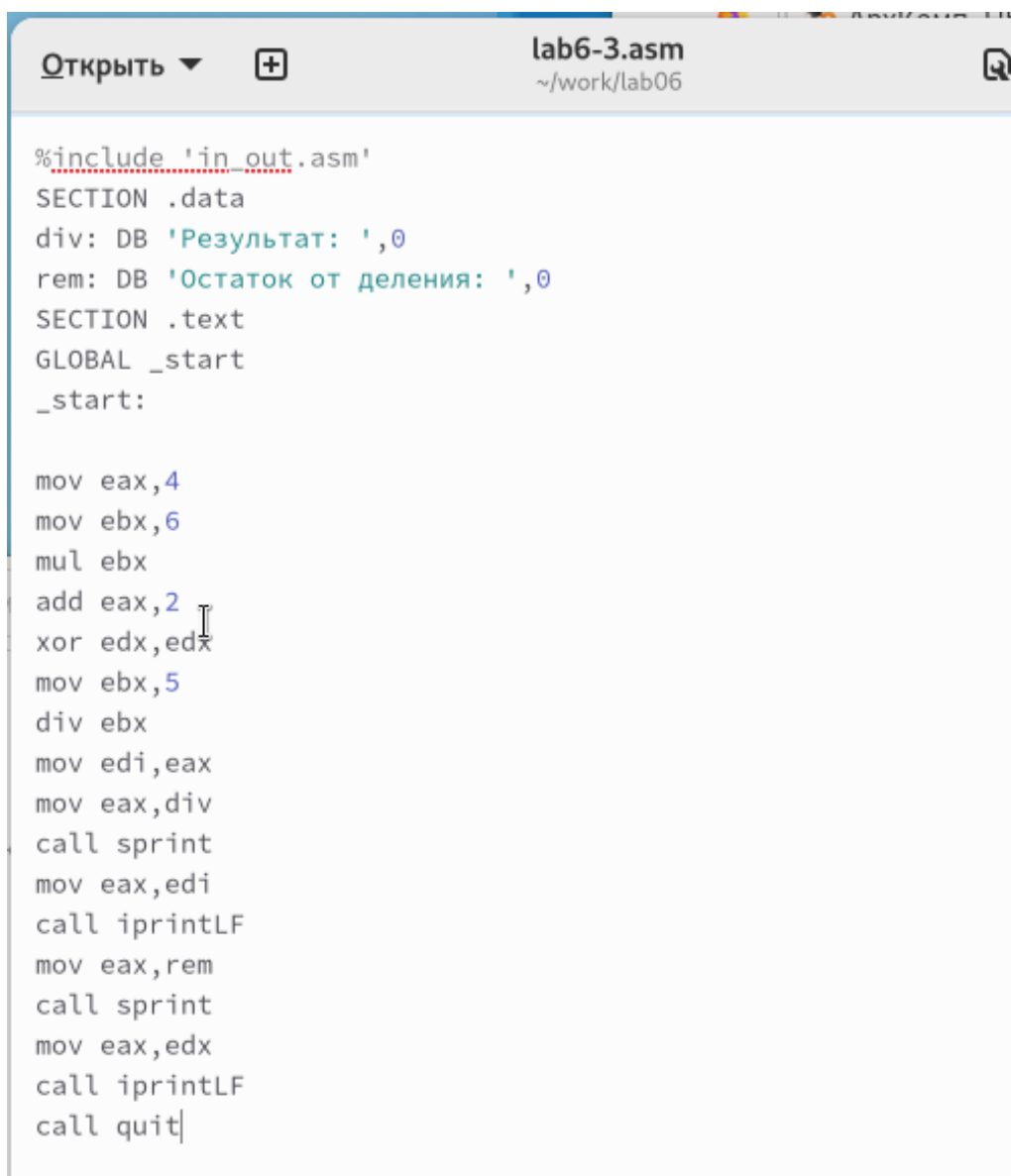
```
[kanilova@fedora lab06]$ nasm -f elf lab6-3.asm
[kanilova@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3
[kanilova@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[kanilova@fedora lab06]$
```

Рис. 4.12: Компиляция текста программы lab6-3.asm

После этого я немного изменила исходный код, чтобы он мог рассчитать другую функцию:

$$f(x) = (4 * 6 + 2) / 5$$

. После того как я скомпилировала измененный код и получила исполняемый файл, я провела его тестирование. Показано на рисунках [4.13] и [4.14].



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

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,4
mov ebx,6
mul ebx
add eax,2
xor edx,edx
mov ebx,5
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

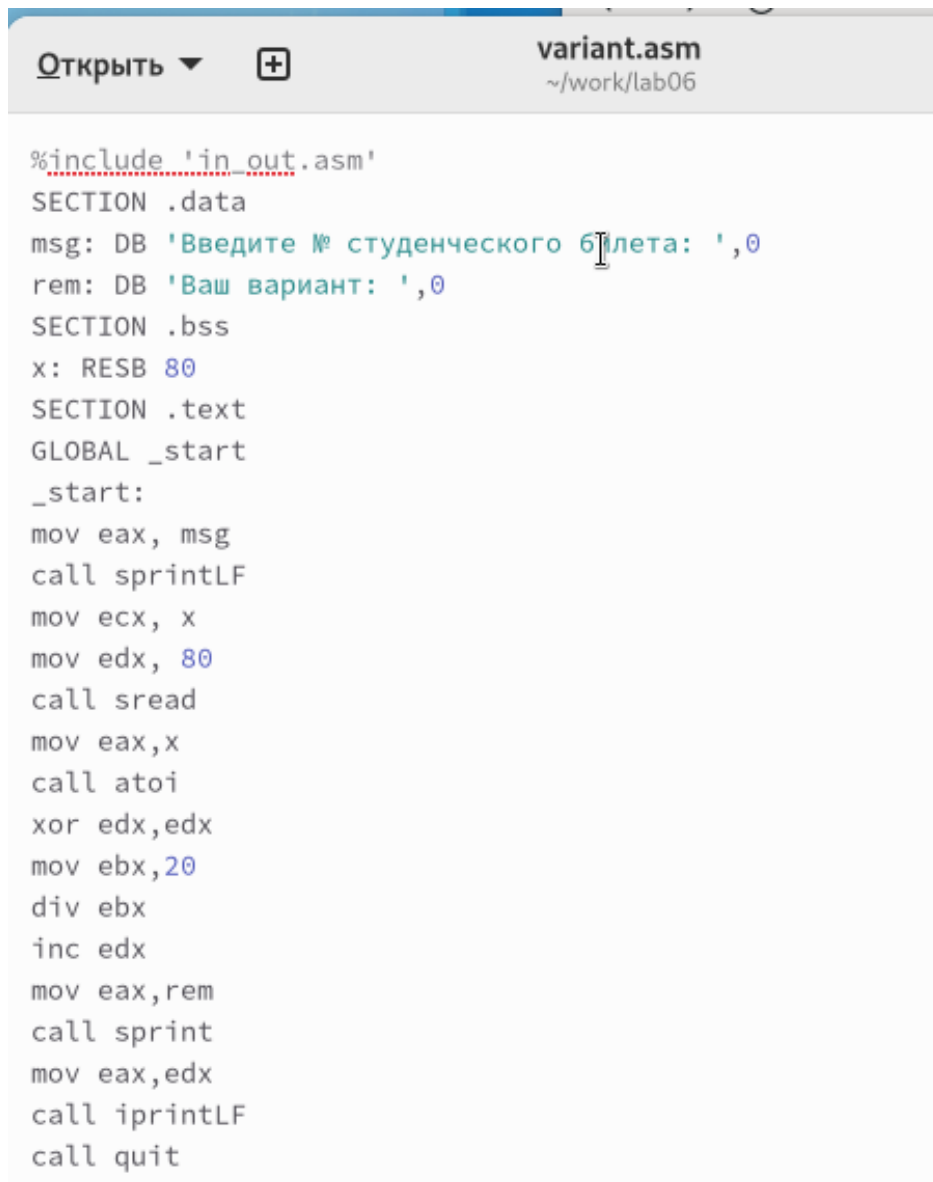
Рис. 4.13: Заполнение файла lab6-3.asm



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

Рис. 4.14: Компиляция текста программы lab6-3.asm

Давайте взглянем на программу, которая вычисляет результаты, исходя из номера студенческого билета. Показано на рисунках [4.15] и [4.16].



```
%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
call atoi
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprint
mov eax, edx
call iprintLF
call quit
```

Рис. 4.15: Заполнение файла variant.asm

```
[kanilova@fedora lab06]$  
[kanilova@fedora lab06]$ nasm -f elf variant.asm  
[kanilova@fedora lab06]$ ld -m elf_i386 variant.o -o variant  
[kanilova@fedora lab06]$ ./variant  
Введите № студенческого билета:  
1132230797  
Ваш вариант: 18  
[kanilova@fedora lab06]$
```

Рис. 4.16: Компиляция текста программы variant.asm

### 4.2.1 Ответы на вопросы

1. Какие строки листинга отвечают за вывод на экран сообщения ‘Ваш вариант:’?

Строки кода, отвечающие за показ сообщения “Ваш вариант:”, включают загрузку этого сообщения в регистр с помощью команды “mov eax, rem”, а также вызов функции вывода на экран с помощью “call sprint”.

2. Для чего используются следующие инструкции?

Инструкции “mov ecx, x” и “mov edx, 80” используются для передачи значения из переменной x в регистр ecx и установки числа 80 в регистр edx соответственно. Для считывания номера студенческого билета применяется команда “call sread”.

3. Для чего используется инструкция “call atoi”?

Инструкция “call atoi” служит для конвертации строки символов в числовое значение.

4. Какие строки листинга отвечают за вычисления варианта?

Вычисление варианта производится с помощью обнуления регистра edx командой “xor edx, edx”, загрузки числа 20 в регистр ebx с помощью “mov ebx, 20”, выполнения деления на 20 через “div ebx” и увеличения результата на единицу с помощью “inc edx”.

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

При выполнении операции деления “div ebx” остаток помещается в регистр edx.

6. Для чего используется инструкция “inc edx”?

Команда “inc edx” используется для инкрементирования содержимого регистра edx на единицу, что необходимо при вычислении варианта.

7. Какие строки листинга отвечают за вывод на экран результата вычислений?

Вывод результатов вычислений на экран осуществляется путем помещения результата в регистр eax с помощью “mov eax, edx” и последующего вызова функции “call iprintLF” для вывода на экран.

### 4.3 Самостоятельное задание

Написать программу вычисления выражения  $y = f(x)$ . Программа должна вывести выражение для вычисления, выводить запрос на ввод значения  $x$ , вычислять заданное выражение в зависимости от введенного  $x$ , выводить результат вычислений. Вид функции  $f(x)$  выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений  $x_1$  и  $x_2$  из 6.3.

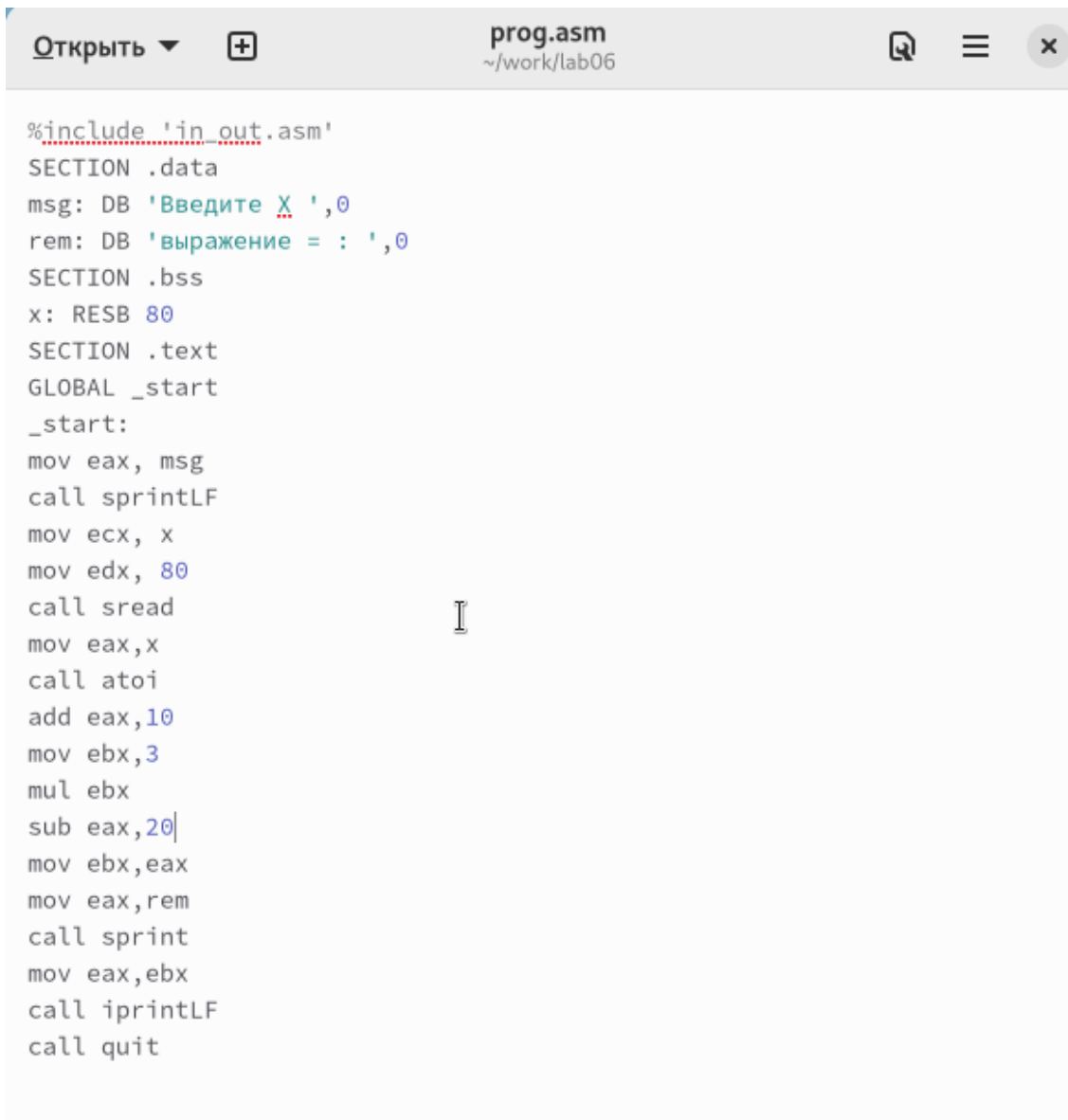
Получили вариант 18 -

$$3(x + 10) - 20$$

для

$$x_1 = 1, x_2 = 5$$

(рис. [4.17] и [4.18])



```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите X ',0
rem: DB 'выражение = : ',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
call atoi
add eax, 10
mov ebx, 3
mul ebx
sub eax, 20
mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 4.17: Заполнение файла prog.asm

```
[kanilova@fedora lab06]$ nasm -f elf prog.asm
[kanilova@fedora lab06]$ ld -m elf_i386 prog.o -o prog
[kanilova@fedora lab06]$ ./prog
Введите X
1
выражение = : 13
[kanilova@fedora lab06]$ ./prog
Введите X
5
выражение = : 25
[kanilova@fedora lab06]$
```

Рис. 4.18: Компиляция текста программы prog.asm

## **5 Выводы**

Изучили работу с арифметическими операциями.