

# **Отчёт по лабораторной работе 6**

**Архитектура компьютера**

Соловьев Серафим

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Задание для самостоятельной работы . . . . .	19
<b>3</b>	<b>Выводы</b>	<b>22</b>

## Список иллюстраций

2.1	Код программы lab6-1.asm . . . . .	7
2.2	Компиляция и запуск программы lab6-1.asm . . . . .	7
2.3	Код программы lab6-1.asm . . . . .	8
2.4	Компиляция и запуск программы lab6-1.asm . . . . .	9
2.5	Код программы lab6-2.asm . . . . .	10
2.6	Компиляция и запуск программы lab6-2.asm . . . . .	10
2.7	Код программы lab6-2.asm . . . . .	11
2.8	Компиляция и запуск программы lab6-2.asm . . . . .	11
2.9	Код программы lab6-2.asm . . . . .	12
2.10	Компиляция и запуск программы lab6-2.asm . . . . .	13
2.11	Код программы lab6-3.asm . . . . .	14
2.12	Компиляция и запуск программы lab6-3.asm . . . . .	14
2.13	Код программы lab6-3.asm . . . . .	15
2.14	Компиляция и запуск программы lab6-3.asm . . . . .	16
2.15	Код программы variant.asm . . . . .	17
2.16	Компиляция и запуск программы variant.asm . . . . .	18
2.17	Код программы program.asm . . . . .	20
2.18	Компиляция и запуск программы program.asm . . . . .	21

## Список таблиц

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

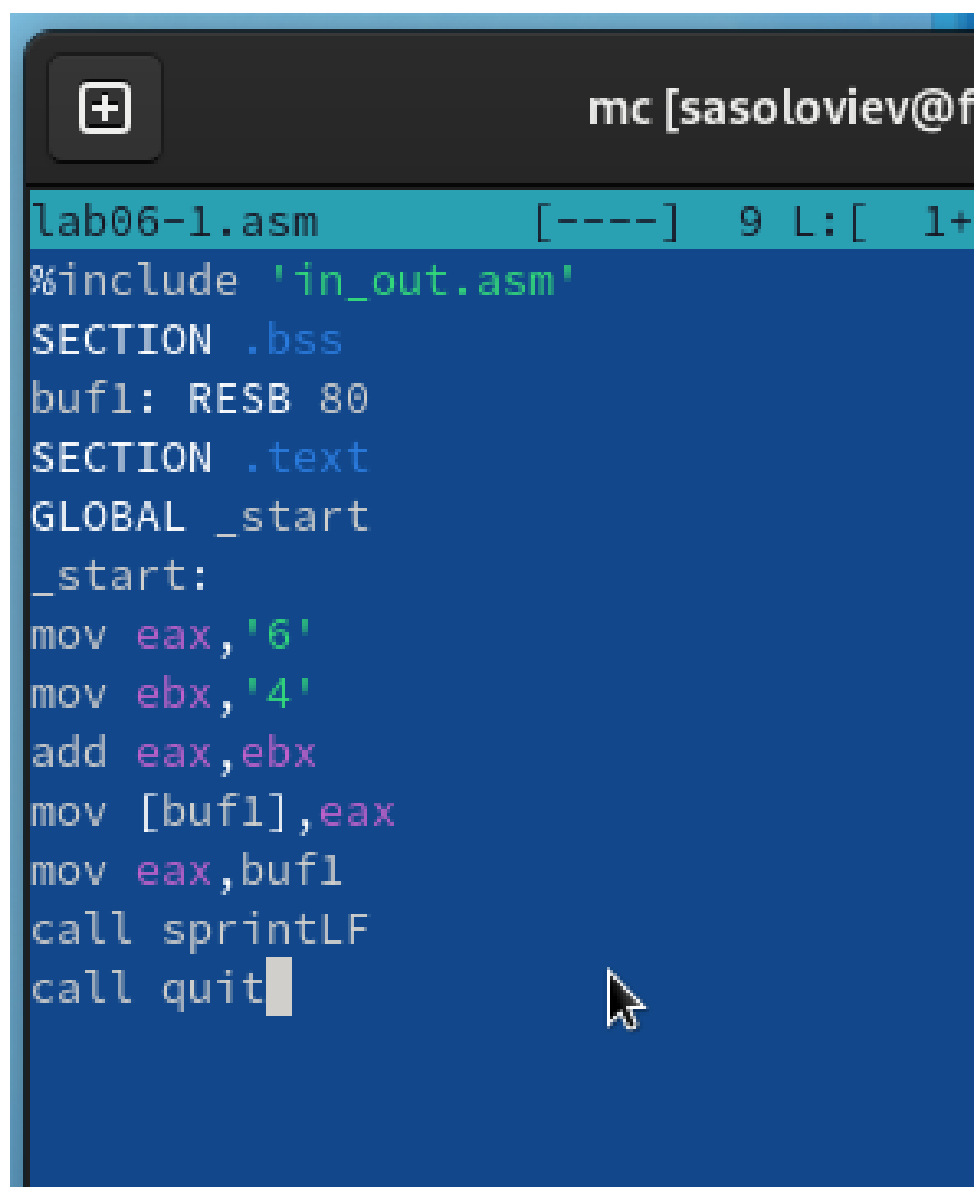
## 2 Выполнение лабораторной работы

Я создал папку для хранения файлов шестой лабораторной работы, перешел в нее и сформировал файл с исходным кодом lab6-1.asm.

В ходе этой лабораторной мы изучим примеры программ, демонстрирующих вывод символов и цифровых данных на экран. Эти программы будут оперировать данными, помещенными в регистр `eax`.

В одной из программ мы помещаем символ '6' в регистр `eax` с помощью инструкции (`mov eax, '6'`) и символ '4' в регистр `ebx` (`mov ebx, '4'`). После этого выполняем сложение значений, находящихся в регистрах `eax` и `ebx` (`add eax, ebx`), и отображаем полученный итог.

Чтобы воспользоваться функцией `sprintf`, которая ожидает адрес в регистре `eax`, мы применяем вспомогательную переменную. Сначала мы копируем содержимое регистра `eax` в переменную `buf1` (`mov [buf1], eax`), затем загружаем адрес `buf1` обратно в регистр `eax` (`mov eax, buf1`) и выполняем вызов функции `sprintf`.



```
lab06-1.asm  [-----]  9  L: [ 1+
#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
```

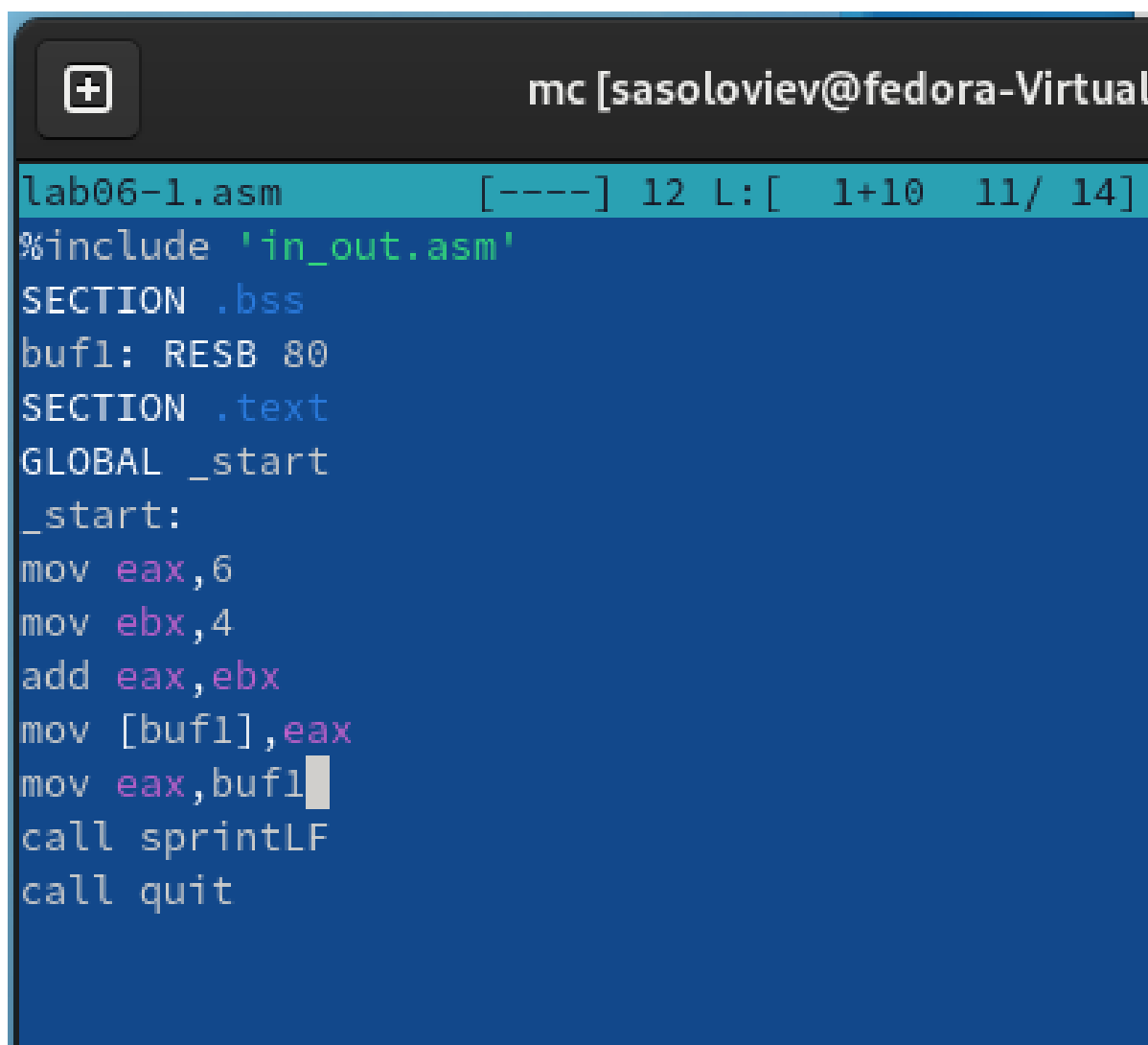
Рис. 2.1: Код программы lab6-1.asm

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

Рис. 2.2: Компиляция и запуск программы lab6-1.asm

В этом примере мы ожидаем на выходе число 10 после сложения значений регистров `eax` и `ebx`. Тем не менее, на экране появится символ 'j', так как двоичный код символа '6' равен 00110110 (что соответствует 54 в десятичной системе), а символа '4' – 00110100 (или 52 в десятичной системе). Инструкция `add eax, ebx` приводит к записи в регистр `eax` суммы этих кодов – 01101010 (или 106 в десятичной системе), что соответствует коду символа 'j'.

После этого я внес изменения в код программы, заменив символы на числовые значения в регистрах.



```
mc [sasoloviev@fedora-Virtual
lab06-1.asm [----] 12 L: [ 1+10 11/ 14]
#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. 2.3: Код программы lab6-1.asm



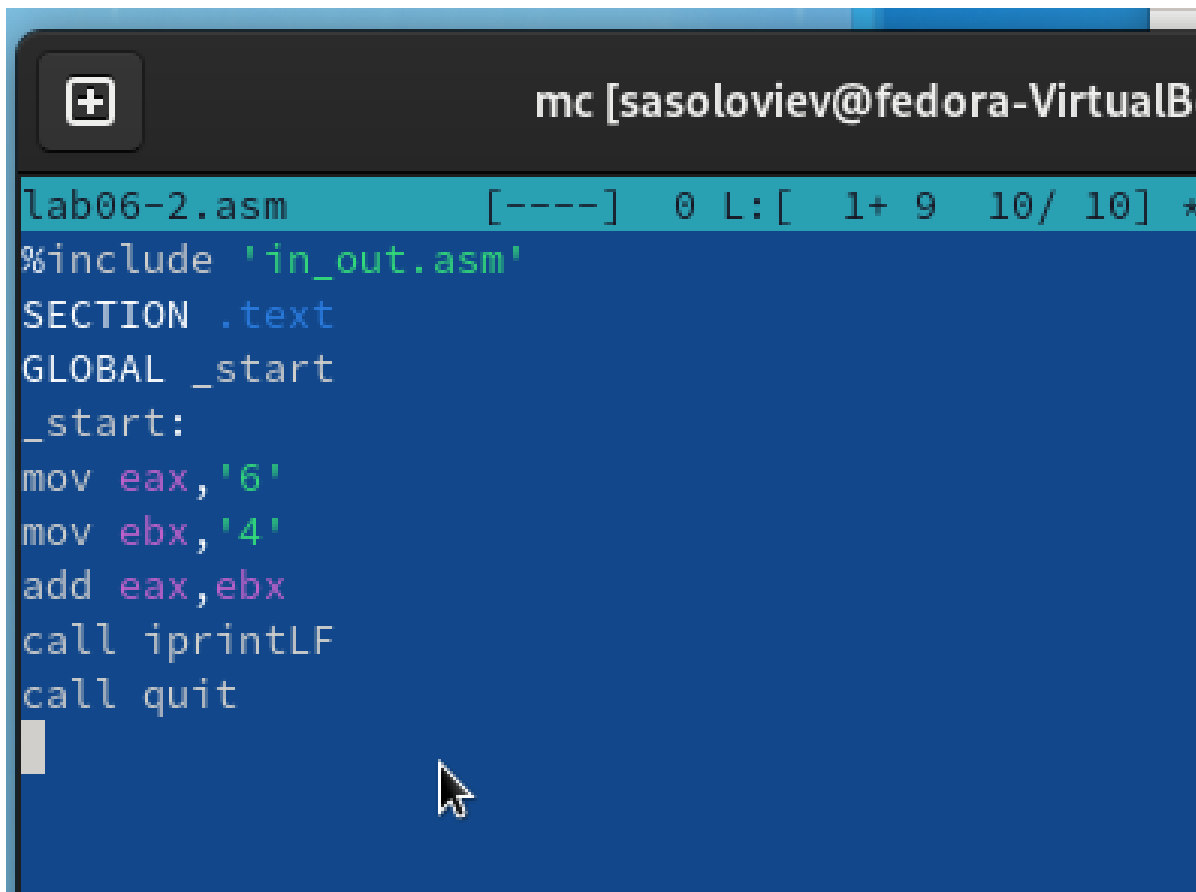
```
[sasoloviev@fedora-VirtualBox lab06]$ nasm -f elf lab06-1.asm
[sasoloviev@fedora-VirtualBox lab06]$ ld -m elf_i386 lab06-1.o -o lab06-1
[sasoloviev@fedora-VirtualBox lab06]$ ./lab06-1

[sasoloviev@fedora-VirtualBox lab06]$
```

Рис. 2.4: Компиляция и запуск программы lab6-1.asm

Так же, как и в прошлый раз, при запуске программы мы не увидим число 10. Вместо этого пройдет вывод символа, который соответствует коду 10. Этот символ представляет собой перевод строки (или возврат каретки). На экране консоли он невидим, но создает пустую строку.

В файле in\_out.asm внедрены вспомогательные процедуры для преобразования ASCII-символов в числовые значения и наоборот. Я внес изменения в код программы, применив эти процедуры.



```
lab06-2.asm [----] 0 L: [ 1+ 9 10/ 10] *
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

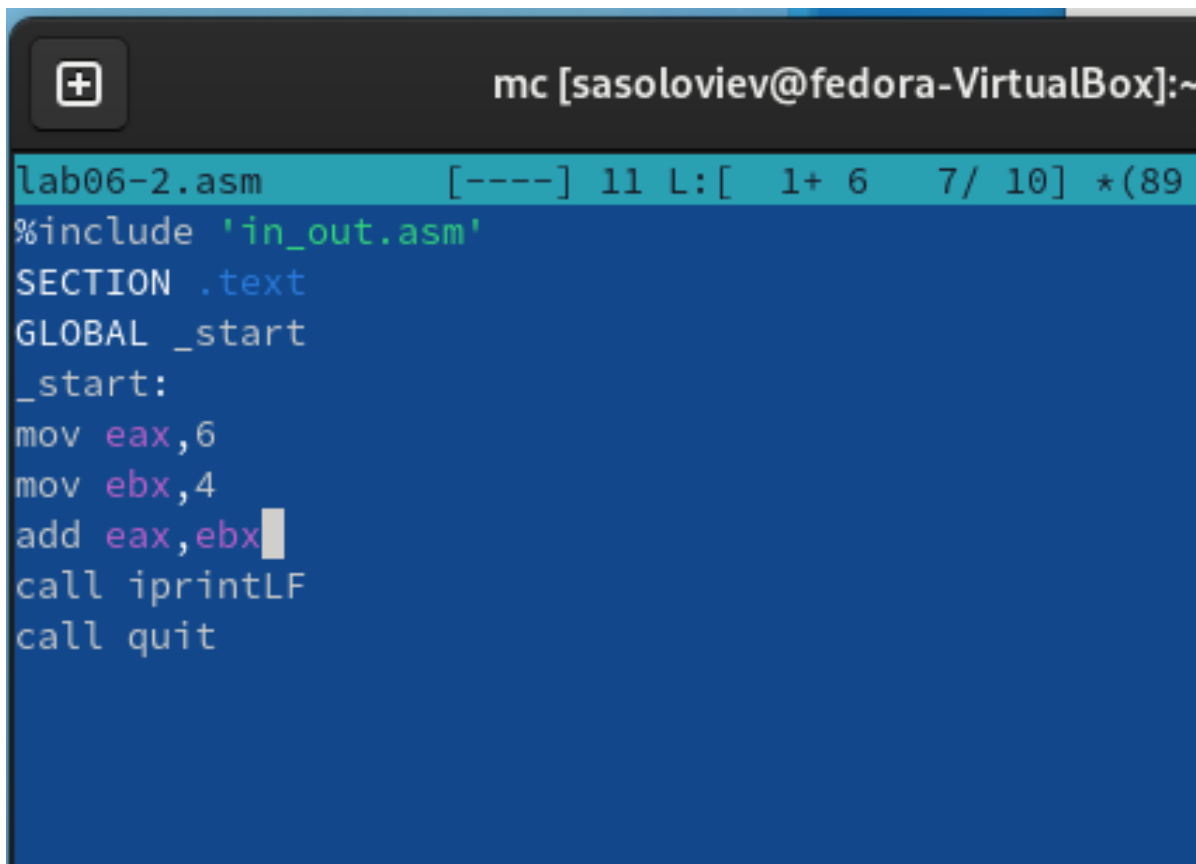
Рис. 2.5: Код программы lab6-2.asm

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

Рис. 2.6: Компиляция и запуск программы lab6-2.asm

После запуска программы на экран будет выведено число 106. В этом случае, подобно первому примеру, команда `add` суммирует численные значения символов '6' и '4' ( $54+52=106$ ). Но в отличие от прошлой версии программы, функция `iprintLF` позволяет отобразить именно число, а не символ с соответствующим числовым кодом.

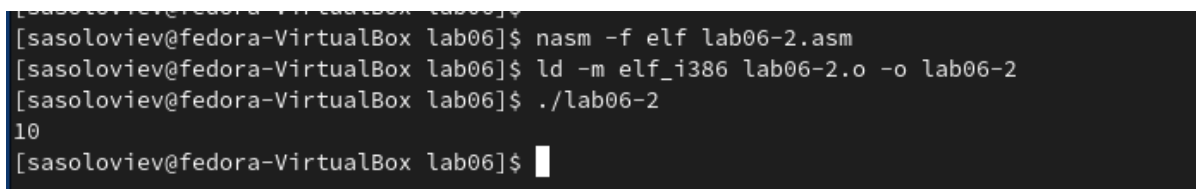
Таким же образом, как и в предыдущем случае, мы провели замену символов на их числовые эквиваленты.



```
mc [sasoloviev@fedora-VirtualBox]:~  
lab06-2.asm [----] 11 L: [ 1+ 6 7/ 10] *(89  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit
```

Рис. 2.7: Код программы lab6-2.asm

Функция iprintLF предназначена для вывода числовых значений, и в данном контексте она работает с числами, а не с их символьными кодами. В результате мы видим на экране число 10.

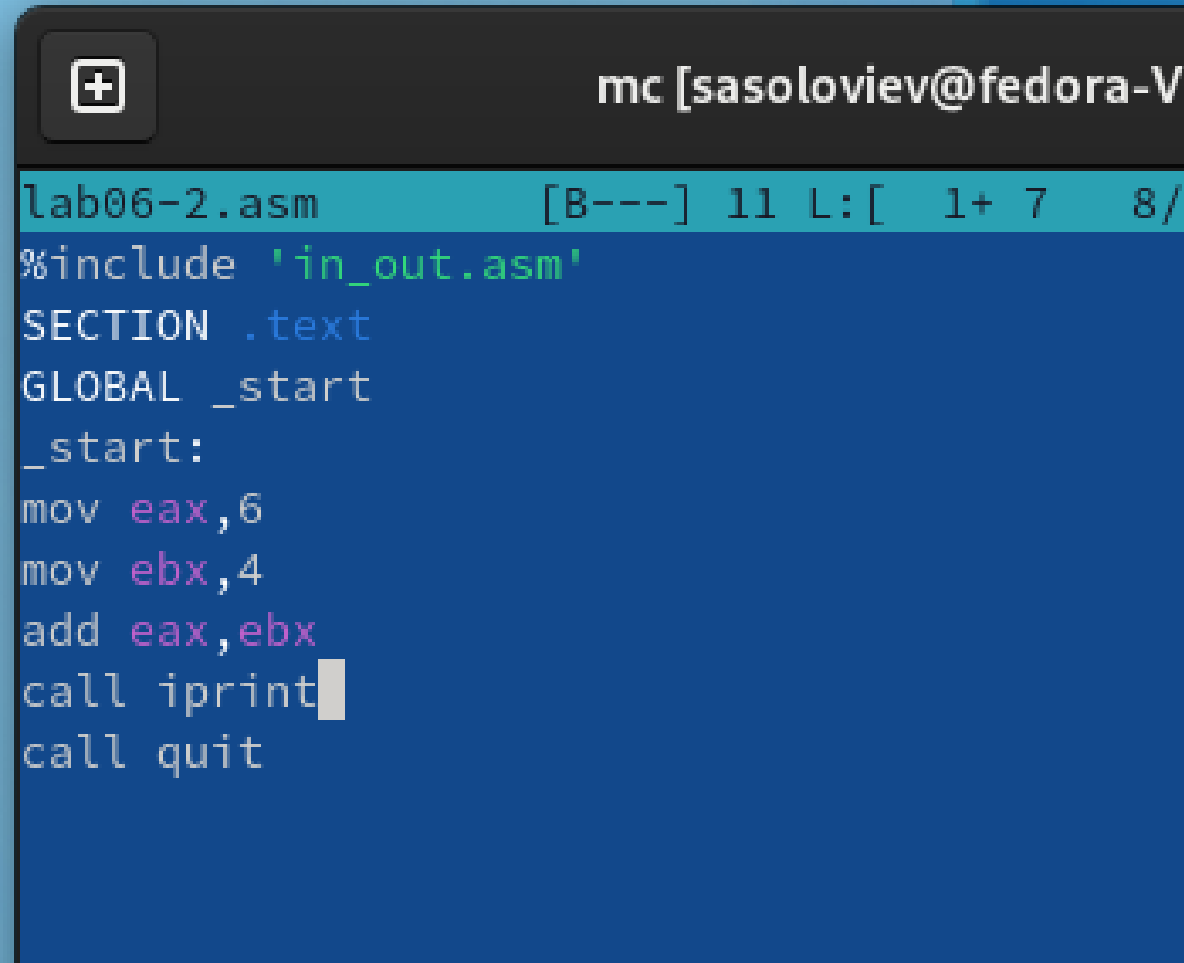


```
[sasoloviev@fedora-VirtualBox lab06]$  
[sasoloviev@fedora-VirtualBox lab06]$ nasm -f elf lab06-2.asm  
[sasoloviev@fedora-VirtualBox lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2  
[sasoloviev@fedora-VirtualBox lab06]$ ./lab06-2  
10  
[sasoloviev@fedora-VirtualBox lab06]$
```

Рис. 2.8: Компиляция и запуск программы lab6-2.asm

Функция `iprintLF` используется для отображения чисел, и в этой ситуации в качестве операндов выступают числа (не их символьные коды), что приводит к выводу числа 10.

Мы произвели замену функции `iprintLF` на `iprint`. Скомпилировали программу, создали исполняемый файл и запустили его. Разница в выводе заключается в отсутствии перевода строки.



```
mc [sasoloviev@fedora-Vi
lab06-2.asm [B---] 11 L: [ 1+ 7 8/
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

Рис. 2.9: Код программы lab6-2.asm

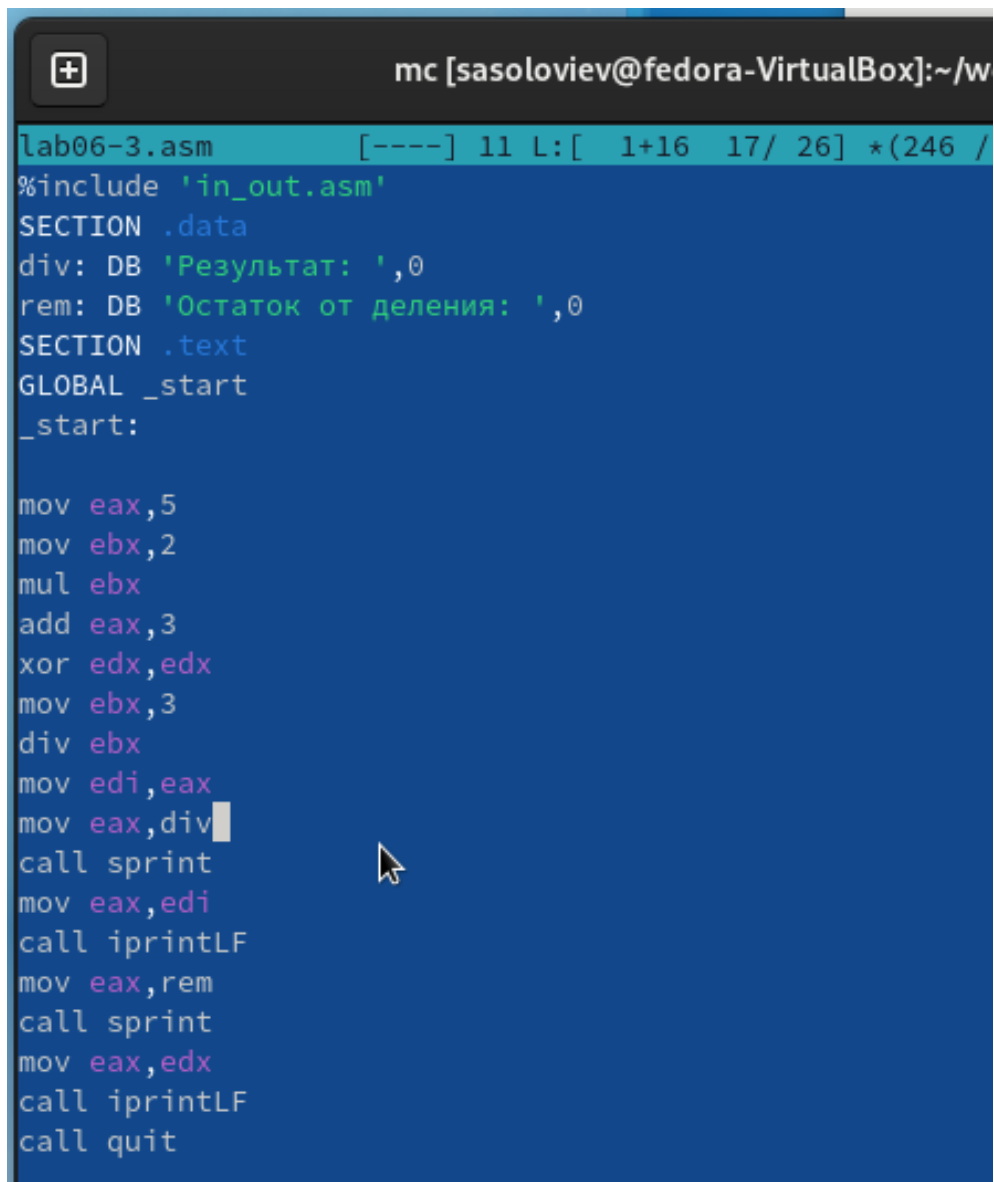
```
[sasoloviev@fedora-VirtualBox lab06]$ nasm -f elf lab06-2.asm
[sasoloviev@fedora-VirtualBox lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[sasoloviev@fedora-VirtualBox lab06]$ ./lab06-2
10[sasoloviev@fedora-VirtualBox lab06]$
```

Рис. 2.10: Компиляция и запуск программы lab6-2.asm

В рамках изучения выполнения арифметических действий в NASM была рассмотрена программа для расчета арифметической функции

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

.



```
mc [sasoloviev@fedora-VirtualBox]:~/w
lab06-3.asm [----] 11 L:[ 1+16 17/ 26] *(246 /
#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
```

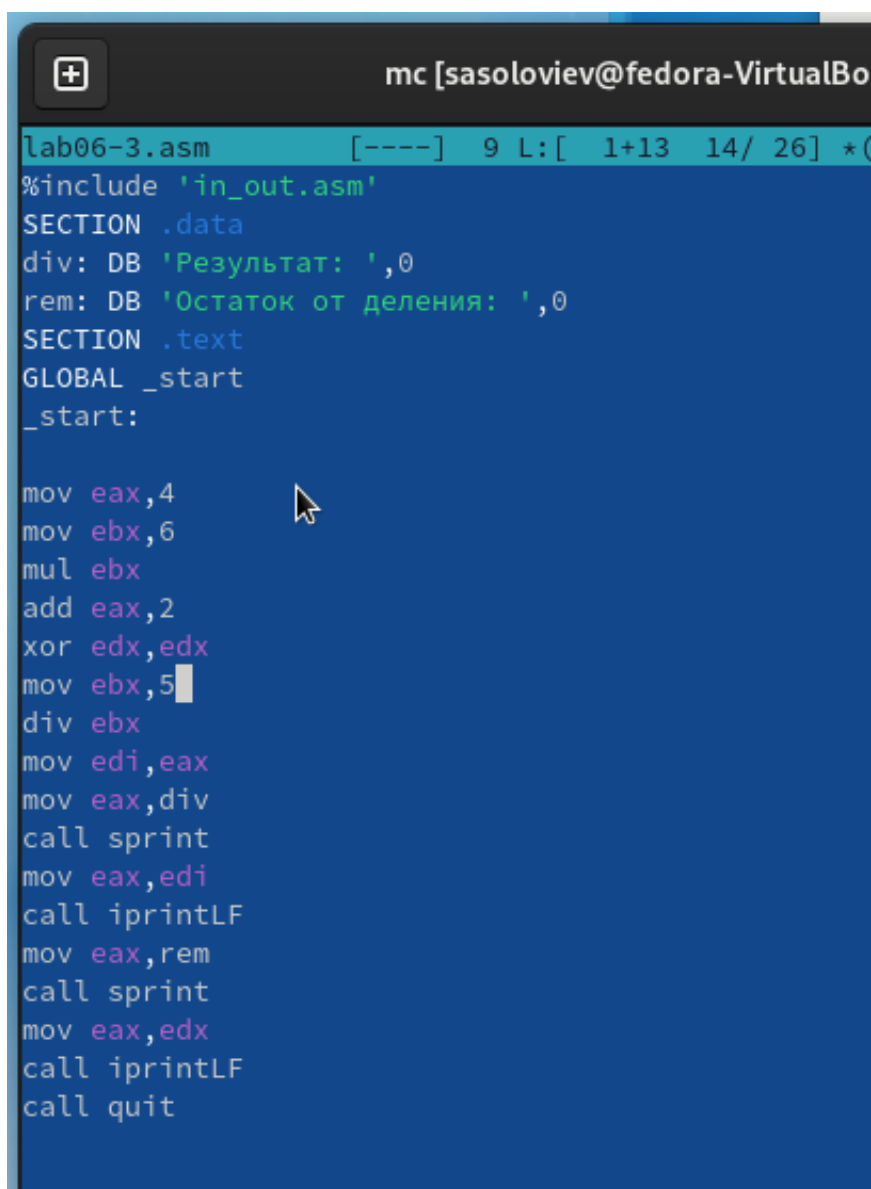
Рис. 2.11: Код программы lab6-3.asm

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

Рис. 2.12: Компиляция и запуск программы lab6-3.asm

Затем модифицировал код программы для того, чтобы она могла вычислять функцию. После создания исполняемого файла он провел его тестирование.

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



```
mc [sasoloviev@fedora-VirtualBo
lab06-3.asm [----] 9 L:[ 1+13 14/ 26] *(
#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. 2.13: Код программы lab6-3.asm

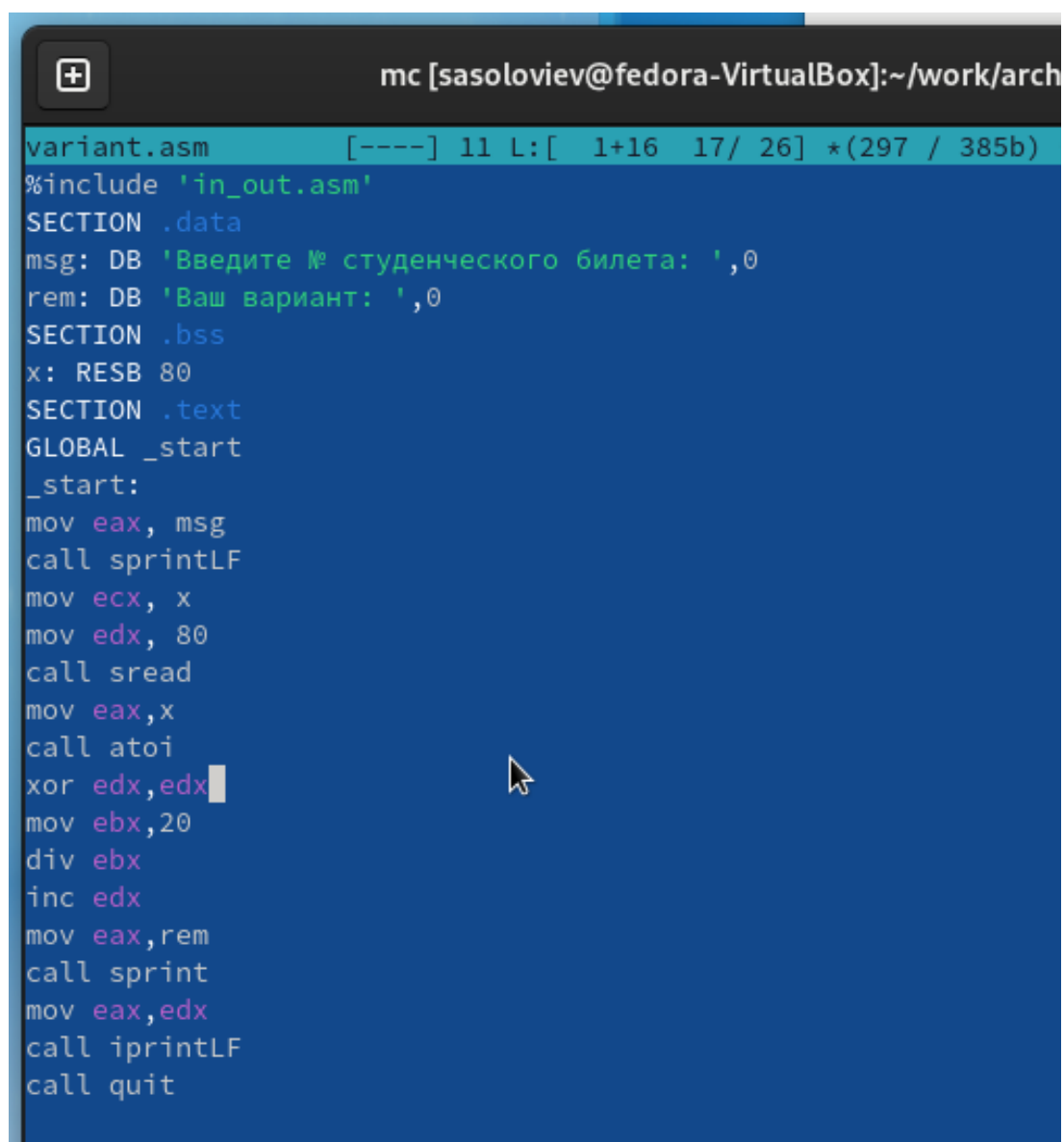
```
[sasoloviev@fedora-VirtualBox lab06]$ nasm -f elf lab06-3.asm
[sasoloviev@fedora-VirtualBox lab06]$ ld -m elf_i386 lab06-3.o -o lab06-3
[sasoloviev@fedora-VirtualBox lab06]$ ./lab06-3
Результат: 5
Остаток от деления: 1
[sasoloviev@fedora-VirtualBox lab06]$
```

Рис. 2.14: Компиляция и запуск программы lab6-3.asm

В качестве другого примера мы рассмотрели программу вычисления варианта задания по номеру студенческого билета.

В этом примере число для выполнения арифметических действий пользователь вводит с клавиатуры. Как было отмечено ранее, ввод осуществляется в виде символов, и для того чтобы арифметические операции в NASM были выполнены правильно, эти символы необходимо преобразовать в числовой формат. Для конвертации можно применить функцию `atoi`, которая содержится в файле `in_out.asm`.





```
variant.asm  [-----] 11 L:[ 1+16 17/ 26] *(297 / 385b)
#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
call atoi
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
```

Рис. 2.15: Код программы variant.asm

```
[sasoloviev@fedora-VirtualBox lab06]$ nasm -f elf variant.asm
[sasoloviev@fedora-VirtualBox lab06]$ ld -m elf_i386 variant.o -o variant
[sasoloviev@fedora-VirtualBox lab06]$ ./variant
Введите № студенческого билета:
1132231837
Ваш вариант: 18
[sasoloviev@fedora-VirtualBox lab06]$
```

Рис. 2.16: Компиляция и запуск программы variant.asm

ответы на вопросы

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

Команда `mov eax, rem` загружает в регистр `eax` строку с текстом “Ваш вариант:”. После этого команда `call sprint` инициирует процедуру, которая выводит эту строку на экран.

2. Для чего используются следующие инструкции?

Команда `mov ecx, x` копирует значение из регистра `ecx` в переменную `x`. Команда `mov edx, 80` присваивает регистру `edx` число 80. Команда `call sread` запускает процедуру чтения данных из стандартного ввода.

3. Для чего используется инструкция “call atoi”?

Команда `call atoi` конвертирует строку символов в целое число.

4. Какие строки листинга отвечают за вычисления варианта?

Команды, выполняющие вычисление варианта, включают: `xor edx, edx` для обнуления регистра `edx`, `mov ebx, 20` для присвоения регистру `ebx` числа 20, `div ebx` для деления значения в аккумуляторе на значение в `ebx`, и `inc edx` для увеличения результата в регистре `edx` на единицу.

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

Остаток от деления после выполнения команды div ebx записывается в регистр edx.

6. Для чего используется инструкция “inc edx”?

Команда inc edx служит для инкремента, то есть увеличения значения в регистре edx на один. Это необходимо для корректного расчета варианта по заданной формуле, где к остатку от деления добавляется единица.

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

Команда mov eax, edx помещает результат вычислений в регистр eax. Затем команда call iprintLF активирует процедуру, которая выводит результат на экран с переводом строки.

## 2.1 Задание для самостоятельной работы

Написать программу вычисления выражения  $y = f(x)$ . Программа должна выводить 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.

Вариант 18 -  $3(x + 10) - 20$  для  $x = 1, x = 5$

```
mc [sasoloviev@fedora-VirtualBox]:~/work/arc
program.asm [----] 0 L: [ 1+18 19/ 29] *(255 / 357b)
#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 sprintLF
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 sprint
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.17: Код программы program.asm

при  $x = 2f(x) = 13$

при  $x = 5f(x) = 25$

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

Рис. 2.18: Компиляция и запуск программы program.asm

Программа считает верно.

## **3 Выводы**

Изучили работу с арифметическими операциями.