

# **Отчёт по лабораторной работе 6**

**Арифметические операции в NASM.**

Ермолаев Денис Николаевич НБИбд-01-24

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Символьные и численные данные в NASM . . . . .	6
2.2	Выполнение арифметических операций в NASM . . . . .	12
2.2.1	Ответы на вопросы . . . . .	16
2.3	Задание для самостоятельной работы . . . . .	17
<b>3</b>	<b>Выводы</b>	<b>20</b>

## Список иллюстраций

2.1	Код программы lab6-1.asm . . . . .	7
2.2	Запуск программы lab6-1.asm . . . . .	7
2.3	Код программы lab6-1.asm с числами . . . . .	8
2.4	Запуск программы lab6-1.asm с числами . . . . .	9
2.5	Код программы lab6-2.asm . . . . .	10
2.6	Запуск программы lab6-2.asm . . . . .	10
2.7	Код программы lab6-2.asm с числами . . . . .	11
2.8	Запуск программы lab6-2.asm с числами . . . . .	11
2.9	Запуск программы lab6-2.asm без переноса строки . . . . .	12
2.10	Код программы lab6-3.asm . . . . .	13
2.11	Запуск программы lab6-3.asm . . . . .	13
2.12	Код программы lab6-3.asm с новым выражением . . . . .	14
2.13	Запуск программы lab6-3.asm с новым выражением . . . . .	14
2.14	Код программы variant.asm . . . . .	15
2.15	Запуск программы variant.asm . . . . .	16
2.16	Код программы task.asm . . . . .	18
2.17	Запуск программы task.asm . . . . .	19

## **Список таблиц**

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

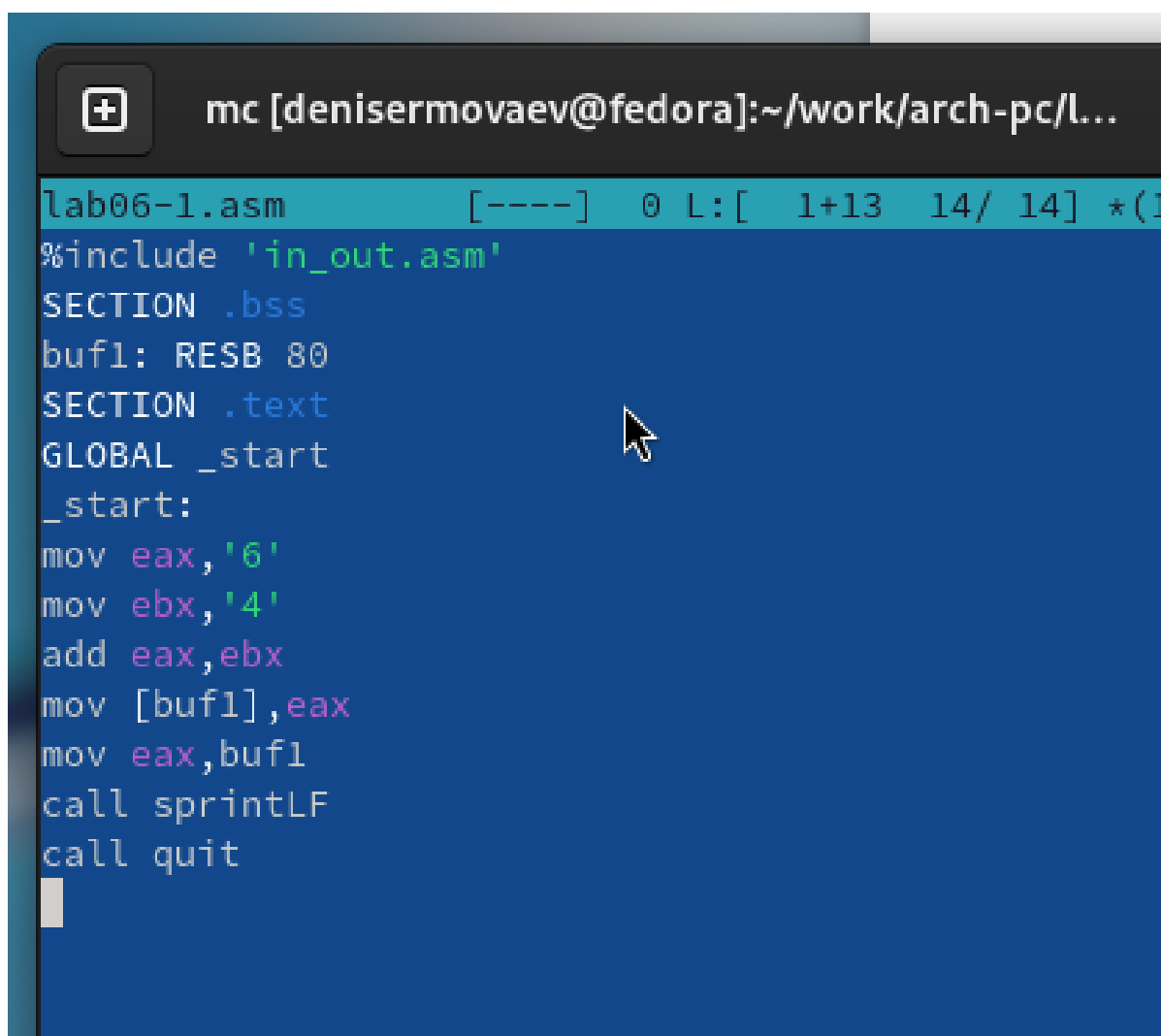
## 2 Выполнение лабораторной работы

### 2.1 Символьные и численные данные в NASM

Создаю папку для программ лабораторной работы № 6, перехожу в неё и создаю файл lab6-1.asm.

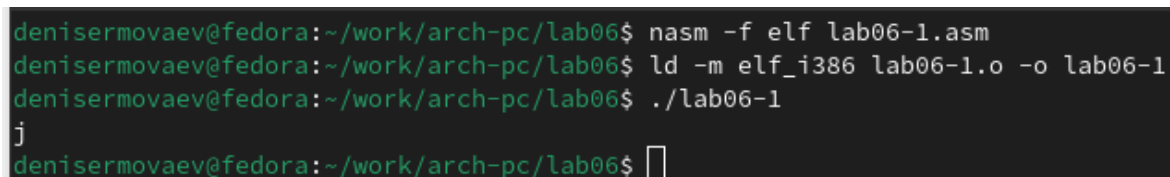
Рассмотрим примеры программ, которые выводят символы и числа. Программы будут выводить значения из регистра еах.

В этой программе в регистр еах записывается символ '6' (инструкция `mov еах,'6'`), а в регистр ебх — символ '4' (инструкция `mov ебх,'4'`). Затем к значению в еах добавляется значение из ебх (инструкция `add еах,ебх`), и результат записывается обратно в еах. После этого выводим результат.



```
lab06-1.asm [-----] 0 L:[ 1+13 14/ 14] *(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



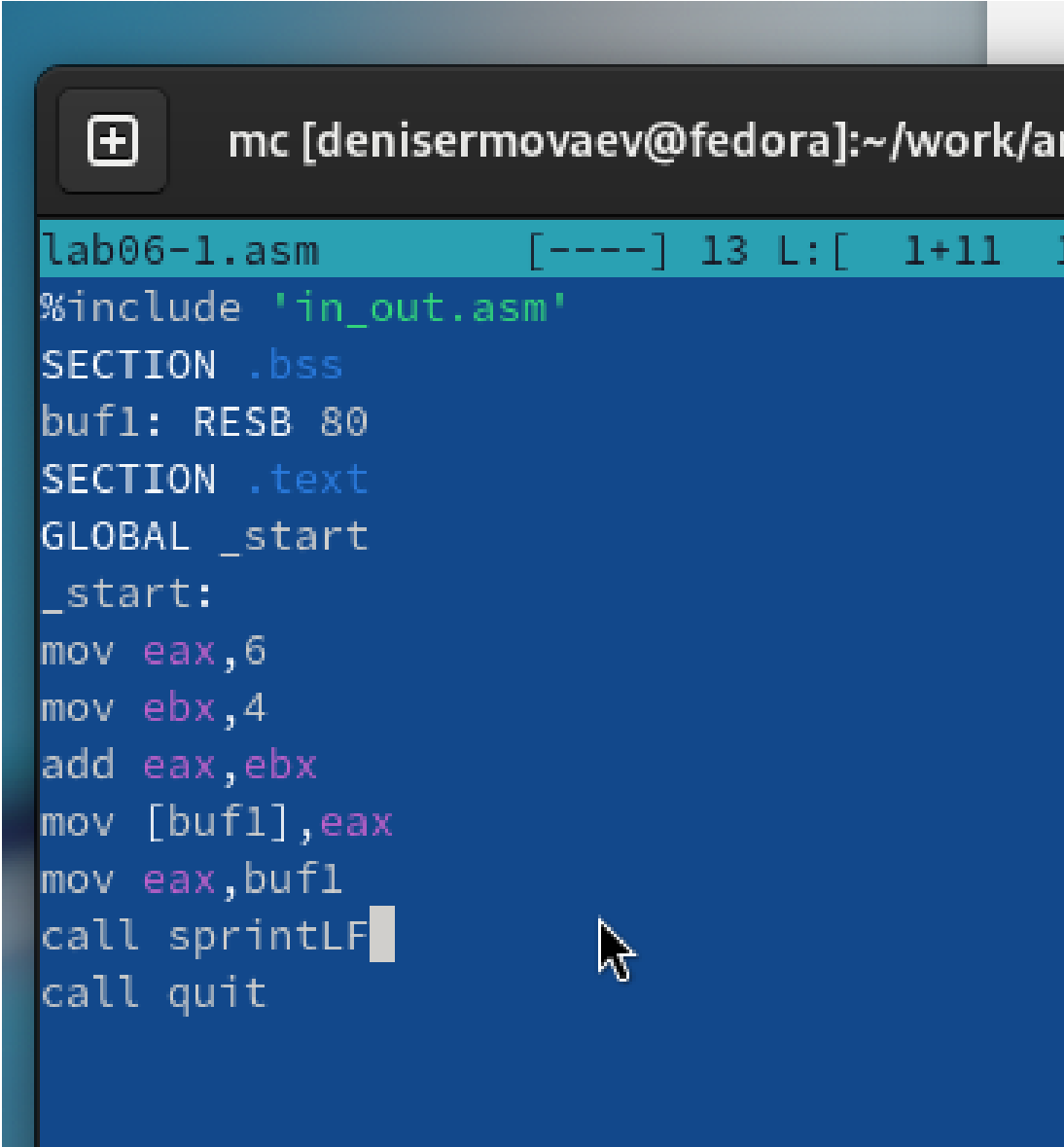
```
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
denisermovaev@fedora:~/work/arch-pc/lab06$ ./lab06-1
j
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.2: Запуск программы lab6-1.asm

При выводе значения из `eax` ожидаем увидеть число 10. Однако вместо этого выводится символ 'j'. Это связано с тем, что код символа '6' в двоичном формате — 00110110 (54 в десятичной системе), а код символа '4' — 00110100 (52). После

сложения в еах получаем 01101010 (106), что соответствует символу 'j'.

Теперь изменим программу и вместо символов запишем в регистры числа.



```
mc [denisermovaev@fedora]:~/work/ar  
lab06-1.asm [-----] 13 L: [ 1+11 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 sprintfLF  
call quit
```

Рис. 2.3: Код программы lab6-1.asm с числами

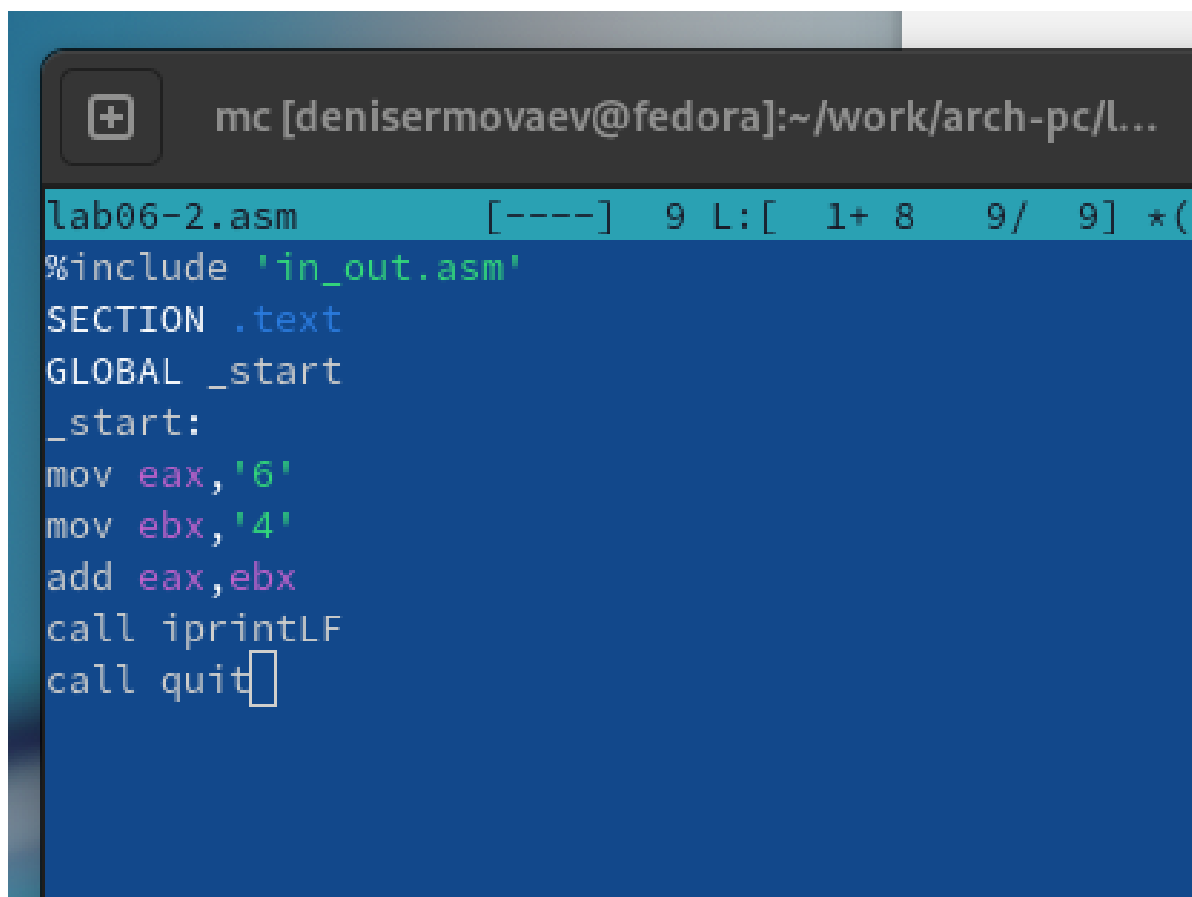


```
denisermovaev@fedora:~/work/arch-pc/lab06$  
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm  
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1  
denisermovaev@fedora:~/work/arch-pc/lab06$ ./lab06-1  
  
denisermovaev@fedora:~/work/arch-pc/lab06$  
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
```

Рис. 2.4: Запуск программы lab6-1.asm с числами

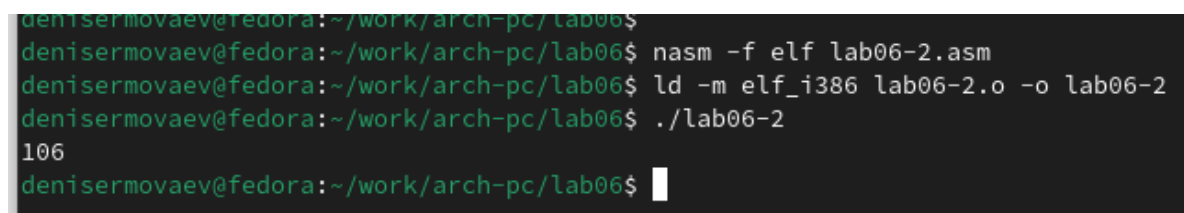
Как и в предыдущем случае, при выполнении программы не получаем число 10. В этот раз выводится символ с кодом 10, что означает конец строки (возврат каретки). В консоли он не отображается, но добавляет пустую строку.

Для работы с числами в файле `in_out.asm` есть подпрограммы, которые преобразуют символы ASCII в числа и обратно. Изменяем программу, используя эти функции.



```
lab06-2.asm [----] 9 L:[ 1+ 8 9/ 9] *(
#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

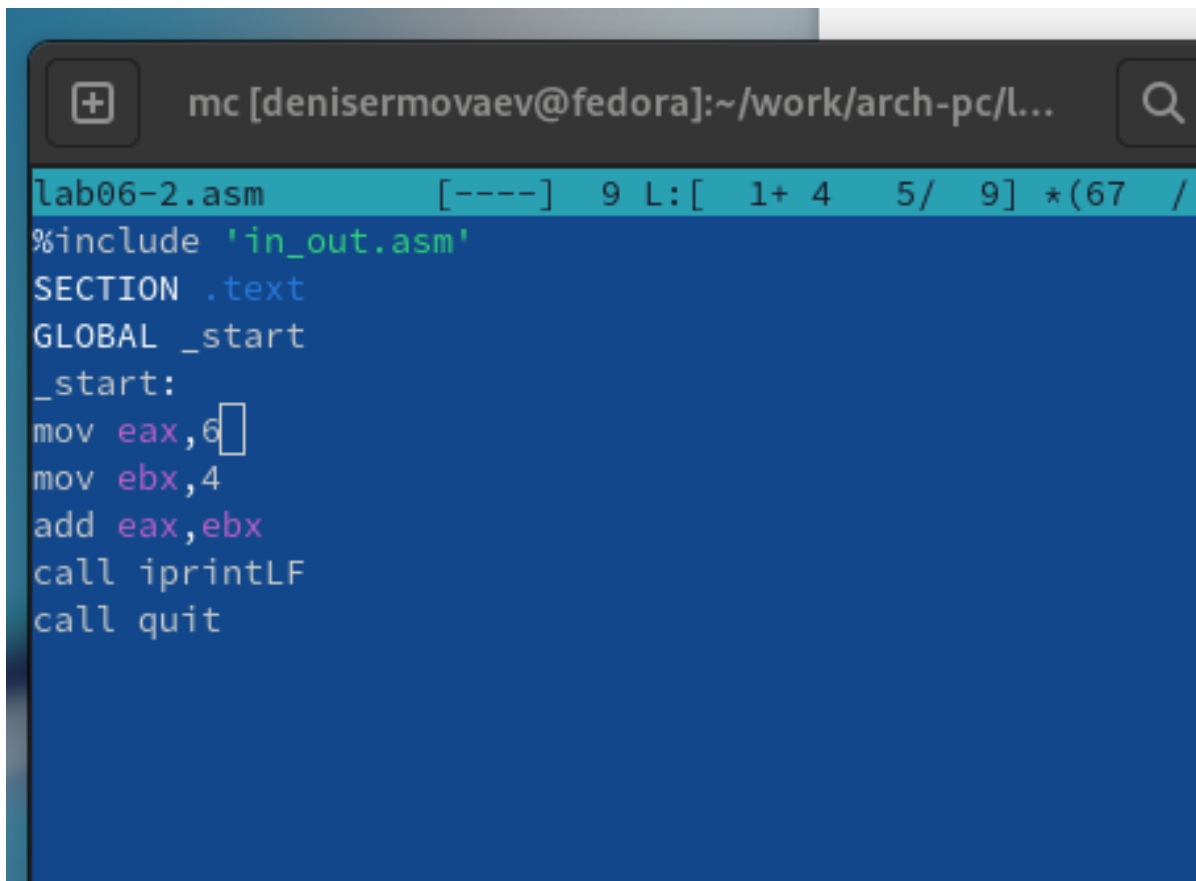


```
denisermovaev@fedora:~/work/arch-pc/lab06$
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
denisermovaev@fedora:~/work/arch-pc/lab06$ ./lab06-2
106
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.6: Запуск программы lab6-2.asm

В результате выполнения программы мы получим число 106. Здесь команда `add` складывает коды символов '6' и '4' ( $54 + 52 = 106$ ). Но, в отличие от предыдущей программы, функция `iprintLF` выводит число, а не соответствующий ему символ.

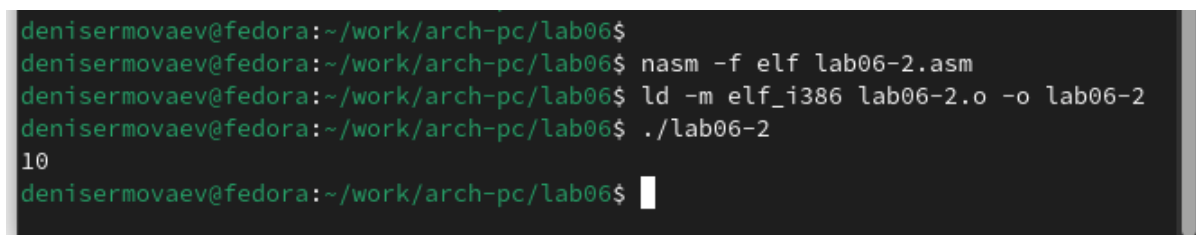
Теперь снова изменим символы на числа.



```
lab06-2.asm [----] 9 L:[ 1+ 4 5/ 9] *(67 /
#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 с числами

Функция iprintLF позволяет вывести число, так как операндами являются числа. Поэтому получаем число 10.



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

Рис. 2.8: Запуск программы lab6-2.asm с числами

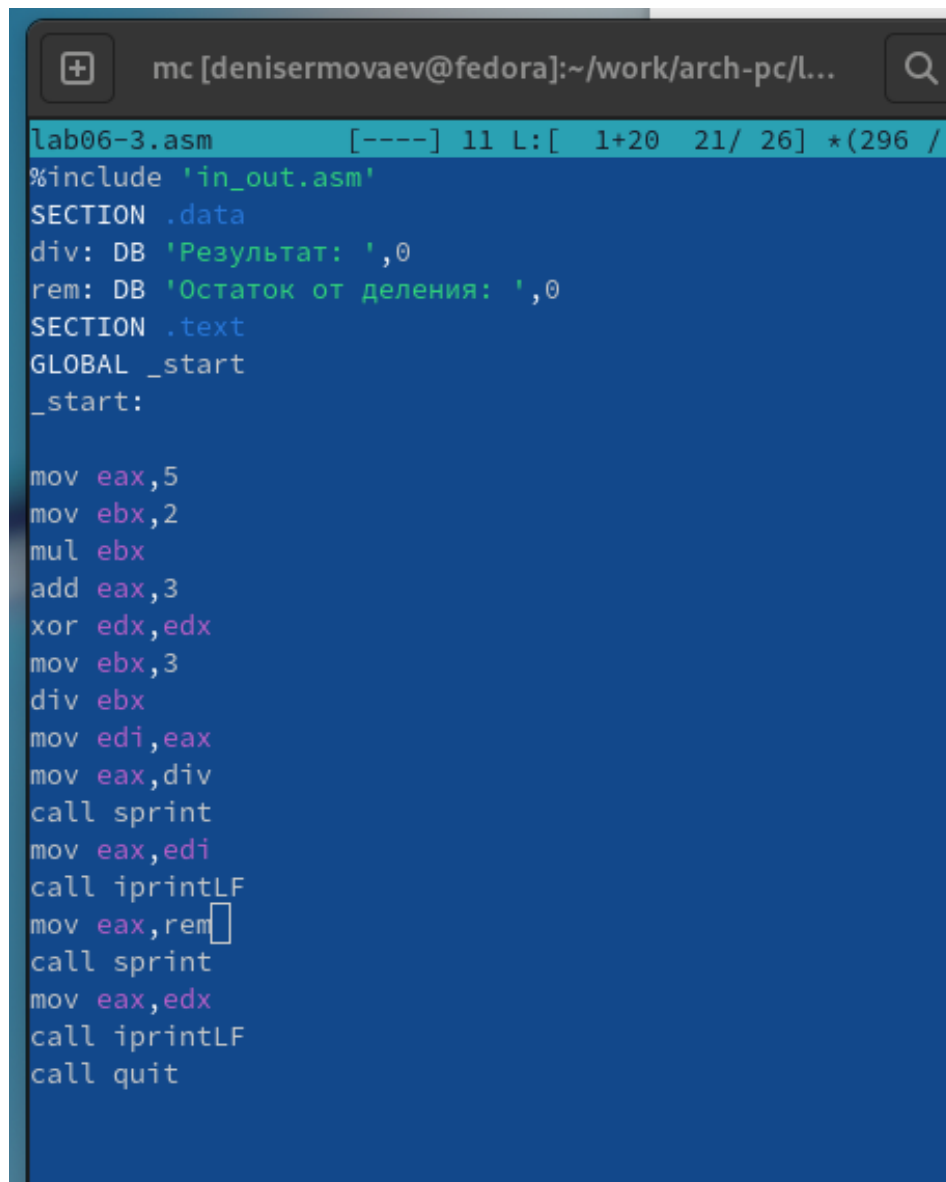
Заменяю функцию iprintLF на iprint. Создал исполняемый файл и запустил его. Вывод отличается тем, что нет переноса строки.

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

Рис. 2.9: Запуск программы lab6-2.asm без переноса строки

## 2.2 Выполнение арифметических операций в NASM

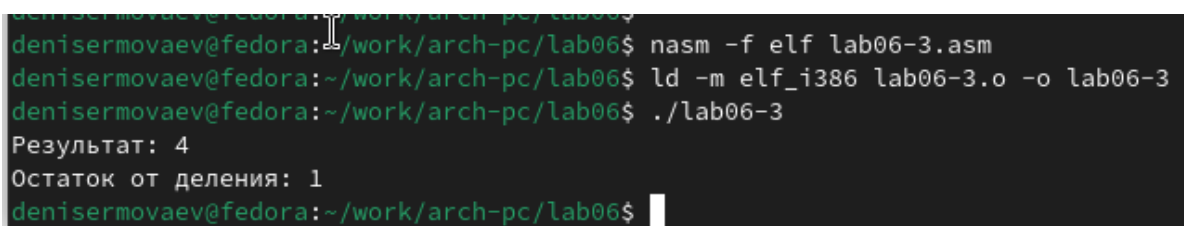
Для примера выполнения арифметических операций в NASM рассмотрим программу, вычисляющую выражение  $f(x) = (5 * 2 + 3)/3$ .



```
lab06-3.asm [-----] 11 L: [ 1+20 21/ 26] *(296 /
#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. 2.10: Код программы lab6-3.asm

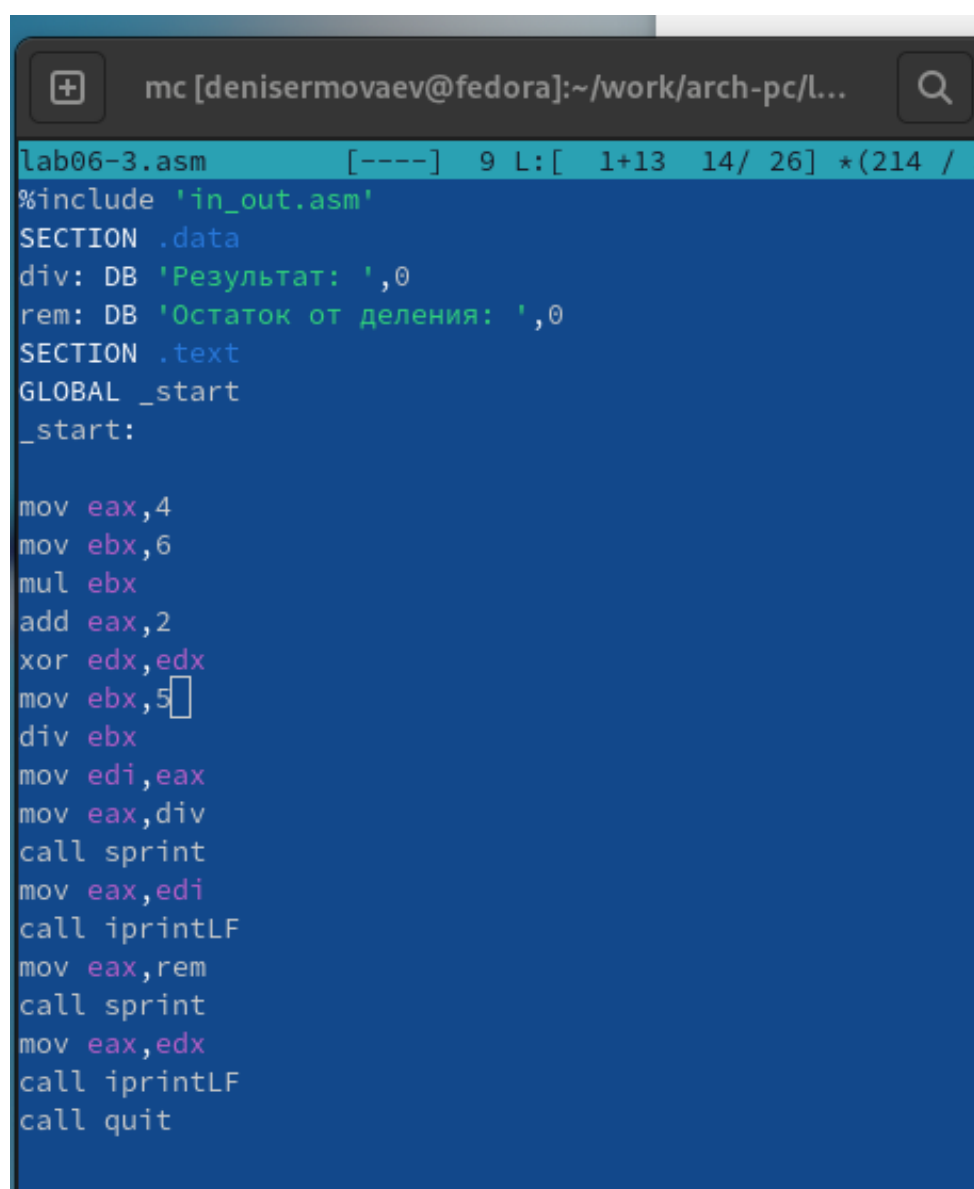


```
denisermovaev@fedora:~/work/arch-pc/lab06$
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
denisermovaev@fedora:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.11: Запуск программы lab6-3.asm

Изменяю программу для вычисления выражения  $f(x) = (4 * 6 + 2) / 5$ . Создаю

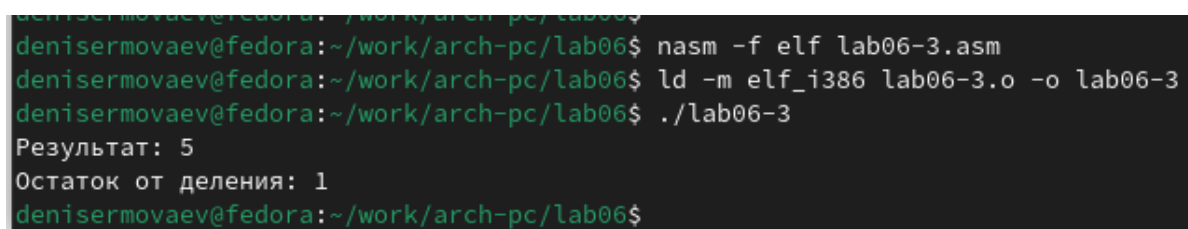
исполняемый файл и проверяю его работу.



```
lab06-3.asm [-----] 9 L: [ 1+13 14/ 26] *(214 / 3
#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
```

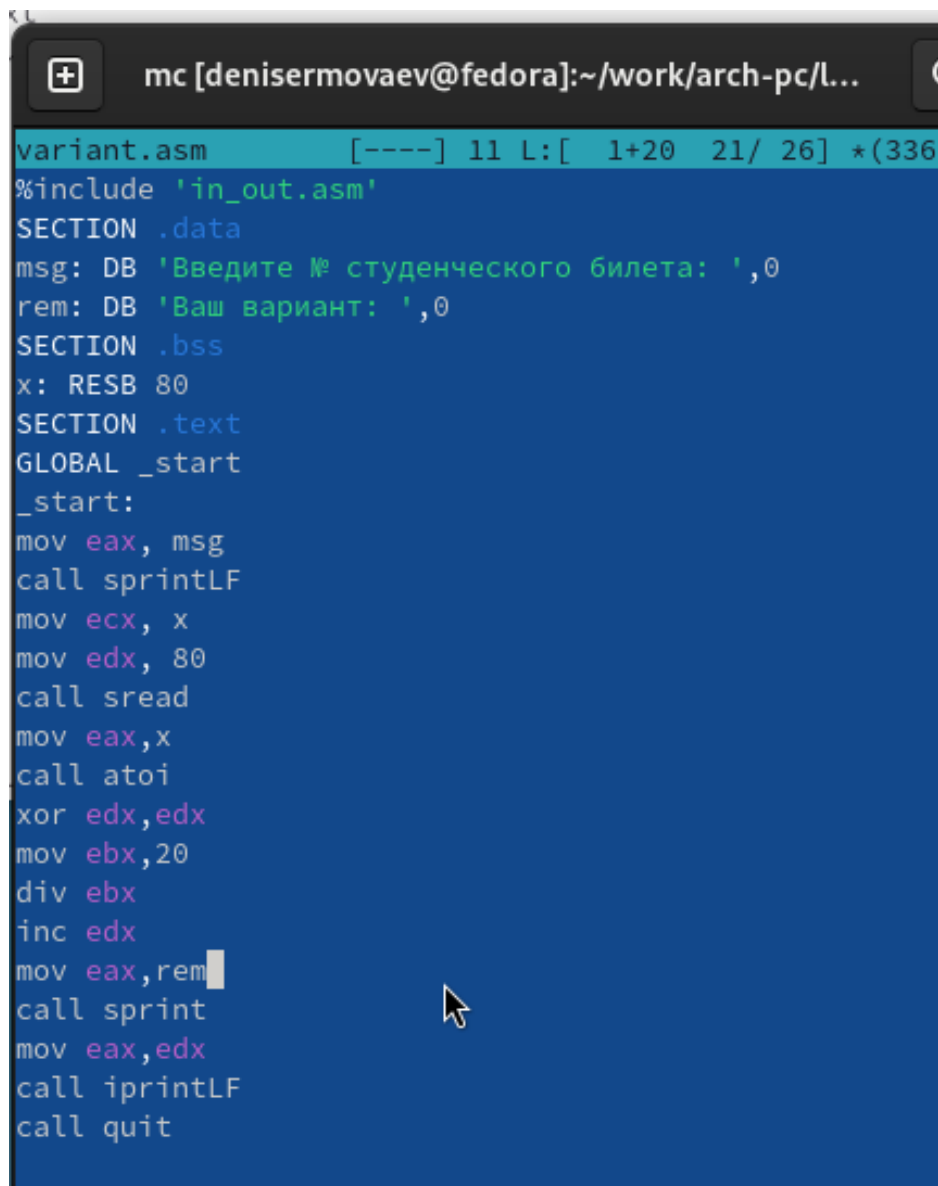
Рис. 2.12: Код программы lab6-3.asm с новым выражением



```
denisermovaev@fedora: ~/work/arch-pc/lab06
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
denisermovaev@fedora:~/work/arch-pc/lab06$ ./lab06-3
Результат: 5
Остаток от деления: 1
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.13: Запуск программы lab6-3.asm с новым выражением

Рассмотрим ещё одну программу, вычисляющую вариант задания по номеру студенческого билета.



```
variant.asm [----] 11 L: [ 1+20 21/ 26] *(336
#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. 2.14: Код программы variant.asm

```
denisermovaev@fedora:~/work/arch-pc/lab06$  
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm  
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant  
denisermovaev@fedora:~/work/arch-pc/lab06$ ./variant 1132243803  
Введите № студенческого билета:  
1132243803  
Ваш вариант: 4  
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.15: Запуск программы variant.asm

Здесь число, над которым нужно выполнять арифметические операции, вводится с клавиатуры. Поскольку ввод осуществляется в символьном виде, символы нужно преобразовать в числа. Для этого можно использовать функцию `atoi` из файла `in_out.asm`.

### 2.2.1 Ответы на вопросы

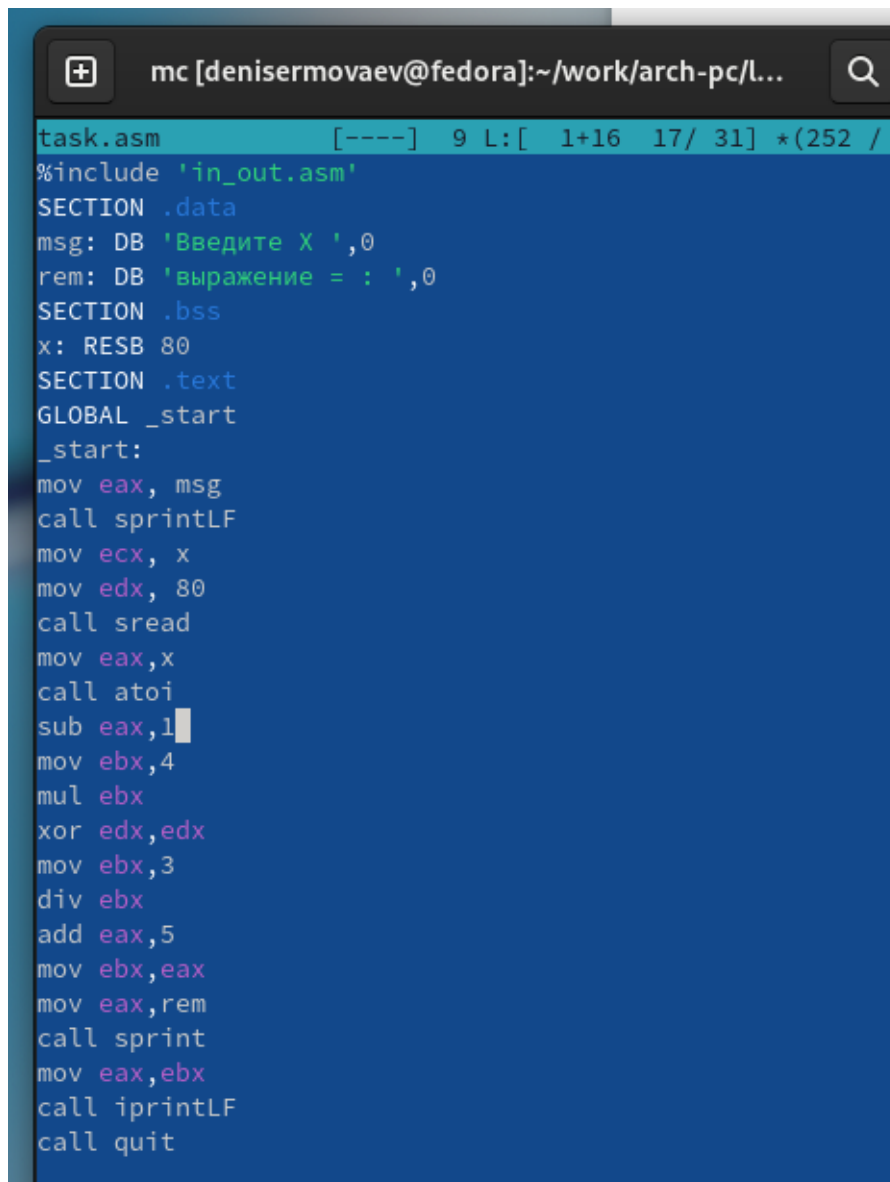
1. Какие строки отвечают за вывод сообщения 'Ваш вариант:'?
  - Инструкция `mov eax, ptr` загружает значение переменной с фразой 'Ваш вариант:' в регистр `eax`.
  - Инструкция `call sprint` вызывает подпрограмму для вывода строки.
2. Для чего нужны следующие инструкции?
  - Инструкция `mov ecx, x` перемещает значение переменной `x` в регистр `ecx`.
  - Инструкция `mov edx, 80` перемещает значение 80 в регистр `edx`.
  - Инструкция `call sread` вызывает подпрограмму для считывания номера студенческого билета из консоли.
3. Для чего нужна инструкция `call atoi`?
  - Инструкция `call atoi` используется для преобразования введенных символов в числовой формат.
4. Какие строки отвечают за вычисления варианта?



- Инструкция `xor edx, edx` обнуляет регистр `edx`.
  - Инструкция `mov ebx, 20` загружает значение 20 в регистр `ebx`.
  - Инструкция `div ebx` делит номер студенческого билета на 20.
  - Инструкция `inc edx` увеличивает значение регистра `edx` на 1. Здесь происходит деление номера студенческого билета на 20, а в регистре `edx` хранится остаток, к которому прибавляется 1.
5. В какой регистр записывается остаток от деления при выполнении `div ebx`?
- Остаток от деления записывается в регистр `edx`.
6. Для чего нужна инструкция `inc edx`?
- Инструкция `inc edx` увеличивает значение в регистре `edx` на 1, как это предусмотрено формулой для вычисления варианта.
7. Какие строки отвечают за вывод результата вычислений на экран?
- Инструкция `mov eax, edx` помещает результат вычислений в регистр `eax`.
  - Инструкция `call iprintLF` вызывает подпрограмму для вывода значения на экран.

## 2.3 Задание для самостоятельной работы

Напишите программу для вычисления выражения  $y = f(x)$ . Программа должна выводить формулу, запрашивать ввод значения  $x$ , вычислять выражение в зависимости от введенного  $x$  и выводить результат. Форму функции  $f(x)$  выберите из таблицы 6.3 вариантов заданий в соответствии с номером, полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его для значений  $x_1$  и  $x_2$  из 6.3. Получили вариант  $4 - \frac{4}{3}(x - 1) + 5$  для  $x = 4, x = 10$ .



```
task.asm [----] 9 L: [ 1+16 17/ 31] *(252 /
#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
sub eax, 1
mov ebx, 4
mul ebx
xor edx, edx
mov ebx, 3
div ebx
add eax, 5
mov ebx, eax
mov eax, rem
call sprint
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.16: Код программы task.asm

При  $x = 4$  результат — 9.

При  $x = 10$  результат — 17.

```
denisermovaev@fedora:~/work/arch-pc/lab06$  
denisermovaev@fedora:~/work/arch-pc/lab06$ nasm -f elf task.asm  
denisermovaev@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task  
denisermovaev@fedora:~/work/arch-pc/lab06$ ./task  
Введите X  
4  
выражение = : 9  
denisermovaev@fedora:~/work/arch-pc/lab06$ ./task  
Введите X  
10  
выражение = : 17  
denisermovaev@fedora:~/work/arch-pc/lab06$
```

Рис. 2.17: Запуск программы task.asm

Программа работает корректно.

## **3 Выводы**

Изучили работу с арифметическими операциями.