

# **Отчёт по лабораторной работе 6**

**дисциплина: Архитектура компьютера**

Быкасов Владислав Дмитриевич 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
2.3	Ответы на вопросы . . . . .	16
2.4	Задание для самостоятельной работы . . . . .	17
<b>3</b>	<b>Выводы</b>	<b>20</b>

## Список иллюстраций

2.1	Программа lab6-1.asm . . . . .	7
2.2	Запуск программы lab6-1.asm . . . . .	7
2.3	Программа lab6-1.asm с числами . . . . .	8
2.4	Запуск программы lab6-1.asm с числами . . . . .	9
2.5	Программа lab6-2.asm . . . . .	9
2.6	Запуск программы lab6-2.asm . . . . .	10
2.7	Программа 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	Программа lab6-3.asm . . . . .	13
2.11	Запуск программы lab6-3.asm . . . . .	13
2.12	Программа lab6-3.asm с другим выражением . . . . .	14
2.13	Запуск программы lab6-3.asm с другим выражением . . . . .	14
2.14	Программа variant.asm . . . . .	15
2.15	Запуск программы variant.asm . . . . .	16
2.16	Программа work.asm . . . . .	18
2.17	Запуск программы work.asm . . . . .	19

## Список таблиц

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

## 2 Выполнение лабораторной работы

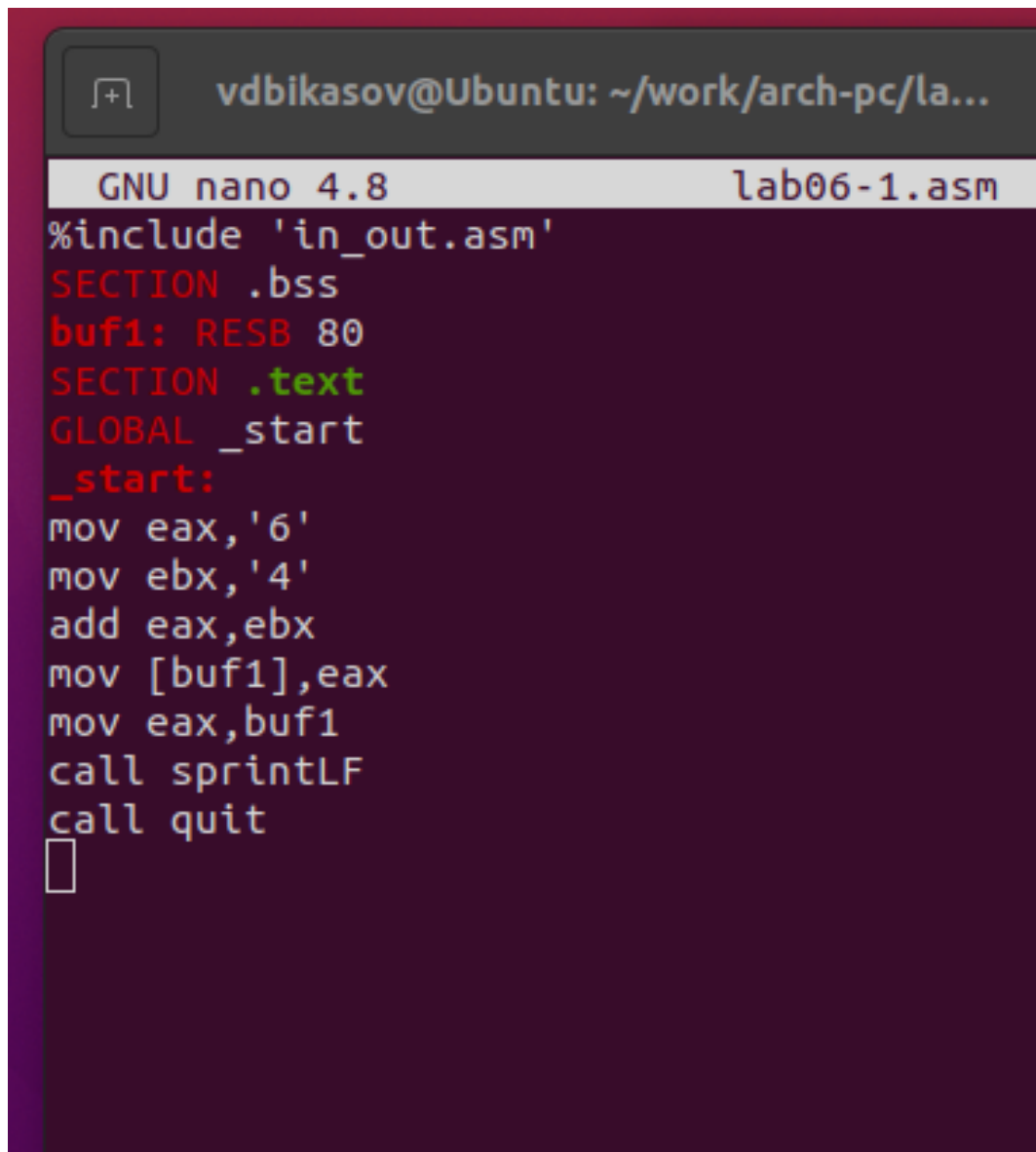
### 2.1 Символьные и численные данные в NASM

Я создал каталог для программ лабораторной работы № 6, перешел в него и создал файл lab6-1.asm.

В этом разделе рассмотрим примеры программ, которые выводят символьные и численные значения. Программы будут выводить данные, записанные в регистр `eax`.

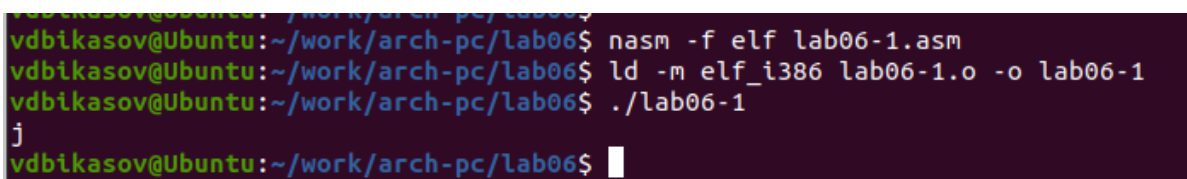
В данной программе в регистр `eax` записывается символ '6' (с помощью команды `mov eax, '6'`), в регистр `ebx` записывается символ '4' (с помощью команды `mov ebx, '4'`). Затем к значению в регистре `eax` прибавляется значение из регистра `ebx` (командой `add eax, ebx`), и результат сохраняется в `eax`. После этого выводим результат.

Так как для работы функции `sprintf` в регистр `eax` должен быть записан адрес, создаем дополнительную переменную. Сначала записываем значение из регистра `eax` в переменную `buf1` (команда `mov [buf1], eax`), затем записываем адрес этой переменной в регистр `eax` (команда `mov eax, buf1`) и вызываем функцию `sprintf`.



```
vdbikasov@Ubuntu: ~/work/arch-pc/la...
GNU nano 4.8 lab06-1.asm
#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: Программа lab6-1.asm



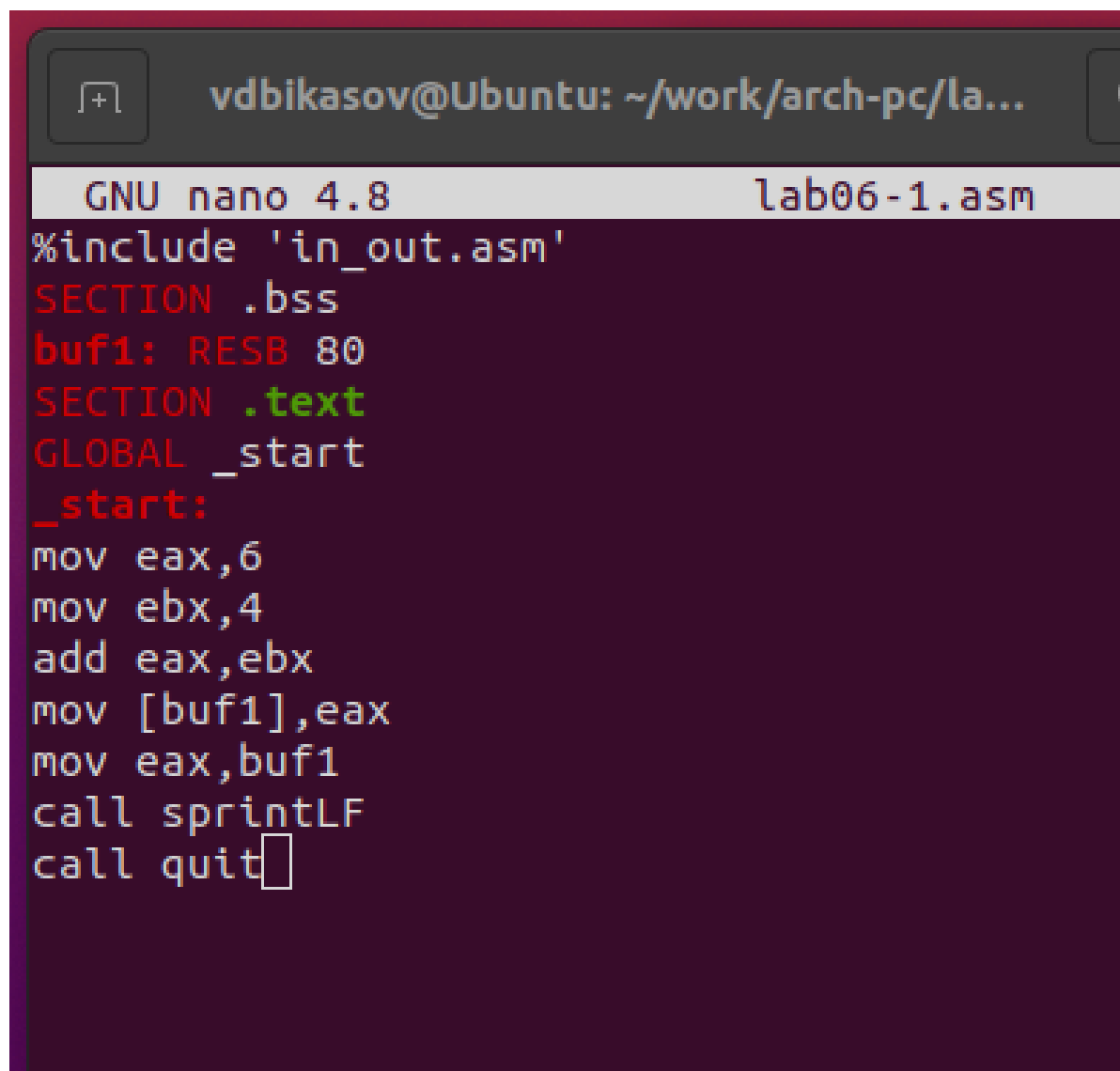
```
vdbikasov@Ubuntu: ~/work/arch-pc/la...
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./lab06-1
j
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ █
```

Рис. 2.2: Запуск программы lab6-1.asm

При выводе значения из регистра `eax` мы ожидали увидеть число 10, но на самом деле вывелся символ 'j'. Это связано с тем, что код символа '6' равен

00110110 (54 в десятичной системе), а код символа '4' — 00110100 (52). Когда эти значения сложились (с помощью команды `add eax, ebx`), результатом стало значение 106, что в свою очередь соответствует символу 'j' в таблице ASCII.

Затем я изменил программу, заменив символы на числа.



```
vdbikasov@Ubuntu: ~/work/arch-pc/la...
GNU nano 4.8 lab06-1.asm
#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: Программа lab6-1.asm с числами

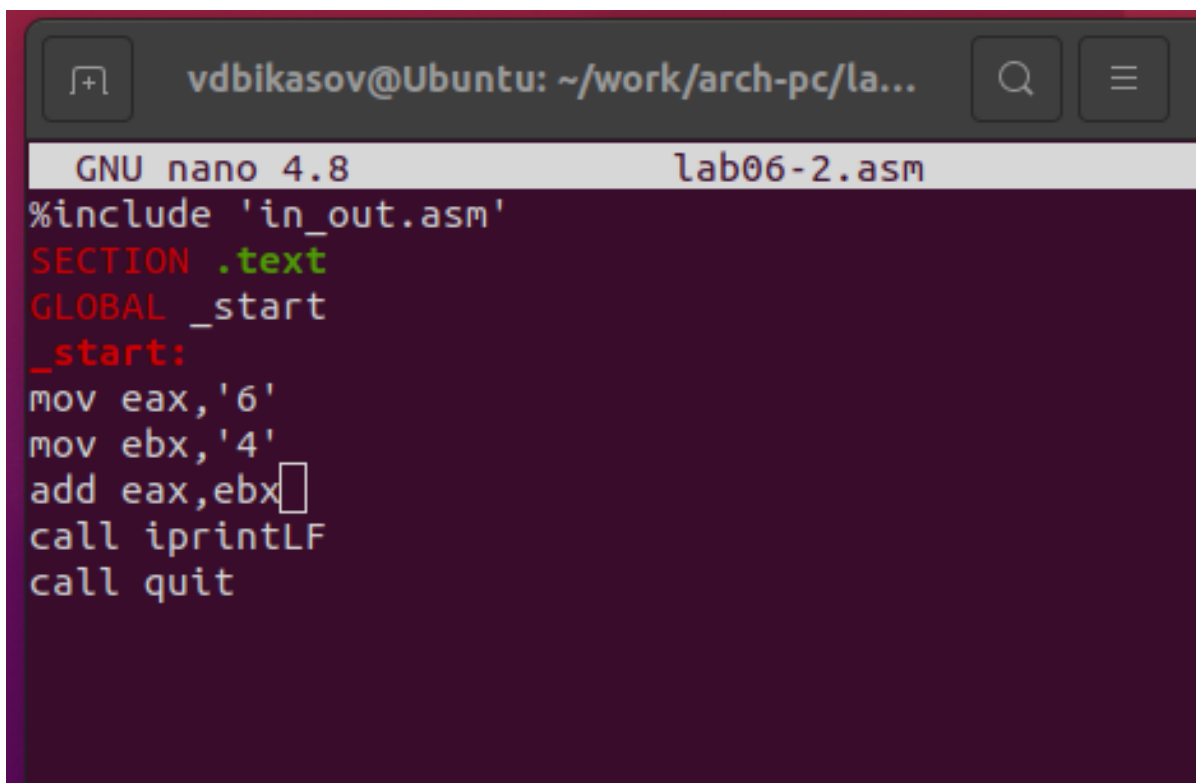


```
vdbikasov@ubuntu:~/work/arch-pc/lab06$  
vdbikasov@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm  
vdbikasov@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1  
vdbikasov@ubuntu:~/work/arch-pc/lab06$ ./lab06-1  
  
vdbikasov@ubuntu:~/work/arch-pc/lab06$
```

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

Как и в предыдущем примере, при исполнении программы мы не получим число 10, а на экране появится символ с кодом 10, который представляет собой символ конца строки (возврат каретки). В консоли он не отображается, но добавляет новую строку.

Для работы с числами в файле `in_out.asm` предусмотрены подпрограммы для преобразования символов в числа и наоборот. Я преобразовал программу, используя эти функции.



```
GNU nano 4.8 lab06-2.asm  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,'6'  
mov ebx,'4'  
add eax,ebx  
call iprintLF  
call quit
```

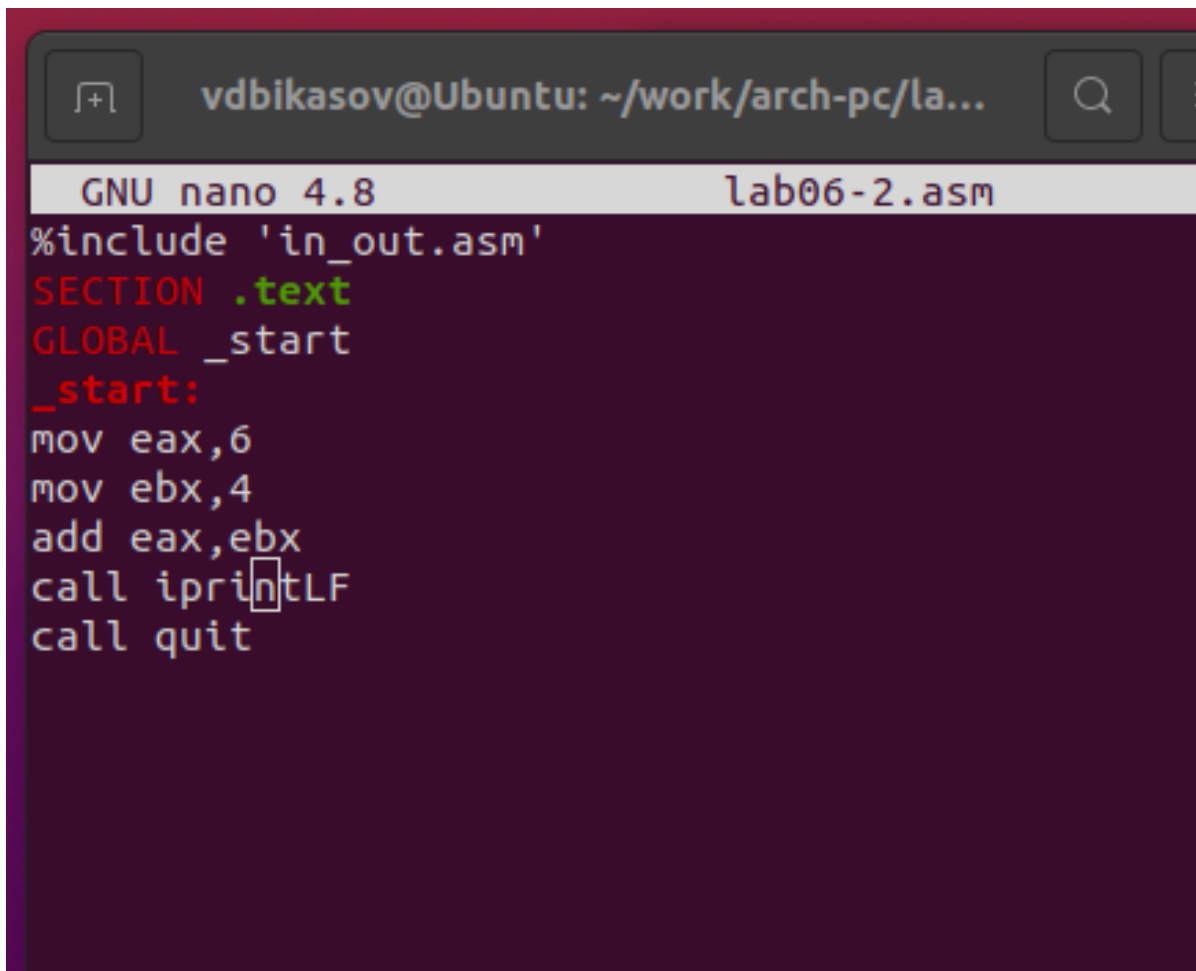
Рис. 2.5: Программа lab6-2.asm

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

Рис. 2.6: Запуск программы lab6-2.asm

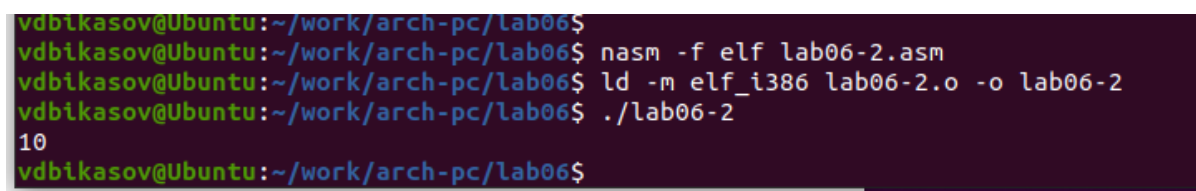
Результатом выполнения программы стало число 106. Здесь, как и в предыдущем примере, команда `add` складывает коды символов '6' и '4' ( $54 + 52 = 106$ ). Однако теперь функция `iprintLF` позволяет вывести это число, а не символ, код которого равен 106.

Заменил символы на числа, и результат вывода — число 10, так как функции выводят именно числовые значения.



```
vdbikasov@Ubuntu: ~/work/arch-pc/la...
GNU nano 4.8 lab06-2.asm
#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: Программа lab6-2.asm с числами



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

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

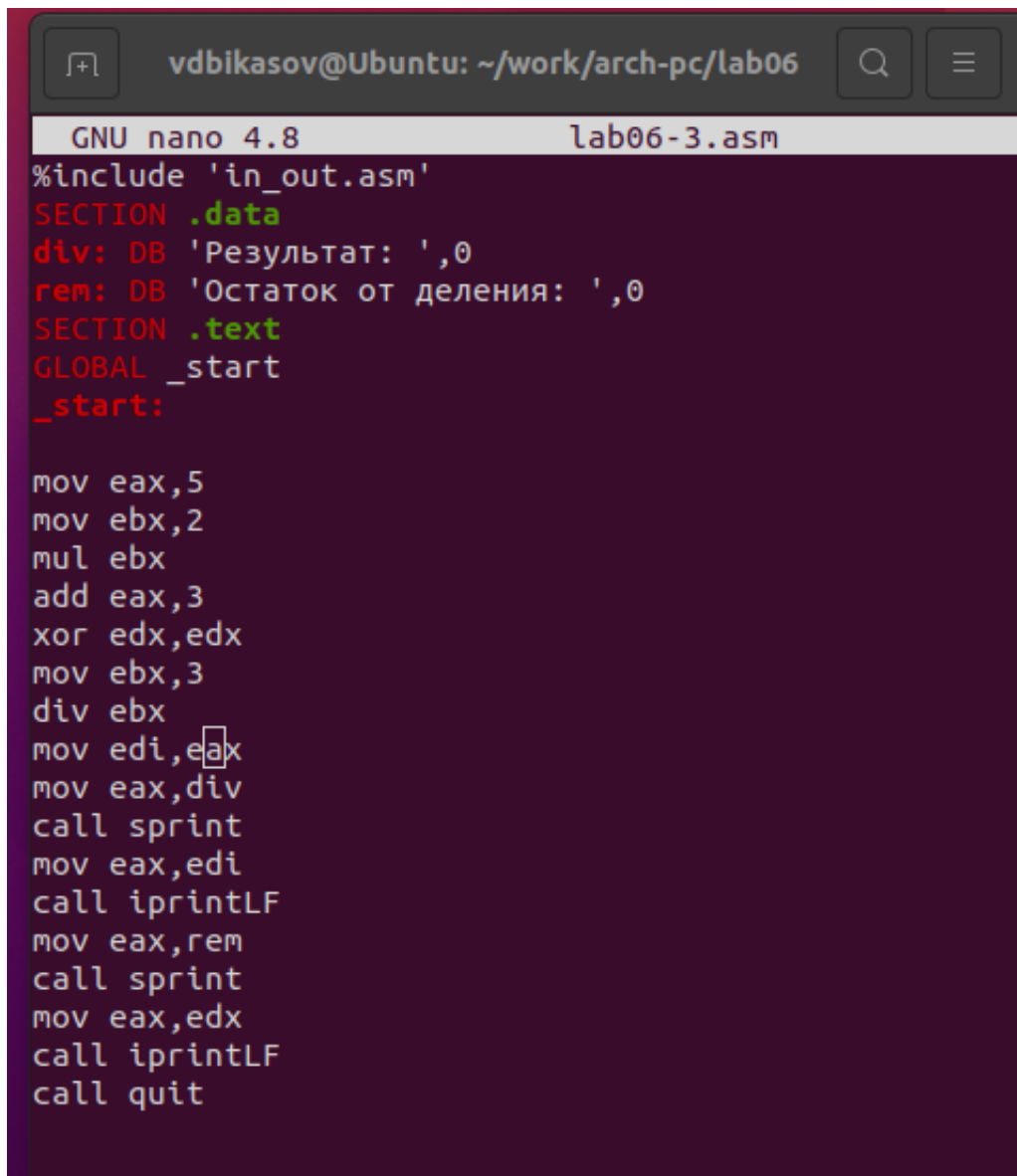
Позже заменил функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Результат отличается тем, что теперь выводится значение без переноса строки.

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

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

## 2.2 Выполнение арифметических операций в NASM

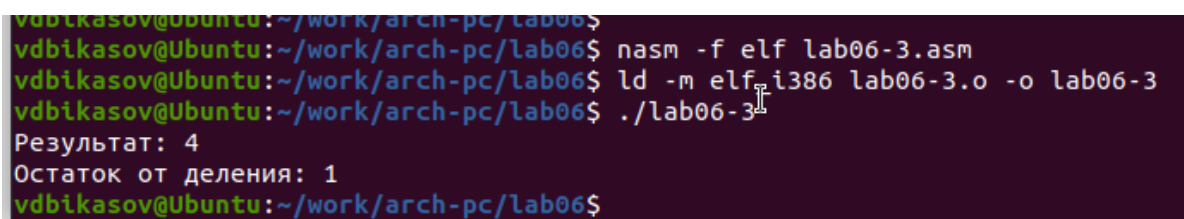
Теперь рассмотрим пример программы, которая выполняет арифметические операции, используя выражение  $f(x) = (5 * 2 + 3) / 3$ .



```
GNU nano 4.8 lab06-3.asm
#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: Программа lab6-3.asm

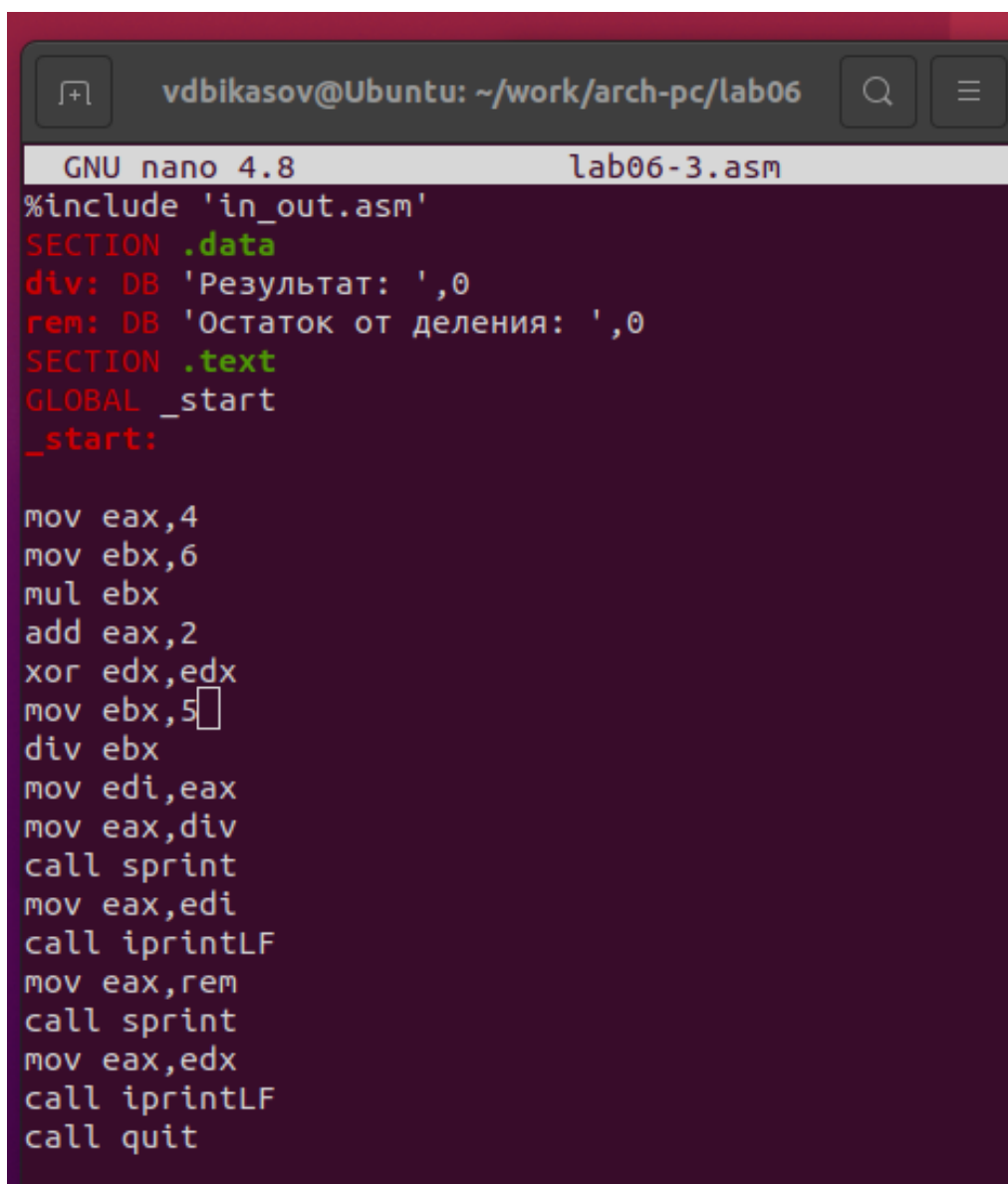


```
vdbikasov@Ubuntu:~/work/arch-pc/lab06$
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
vdbikasov@Ubuntu:~/work/arch-pc/lab06$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

Рис. 2.11: Запуск программы lab6-3.asm

Я изменил программу для вычисления выражения  $f(x) = (4 * 6 + 2)/5$ , создал

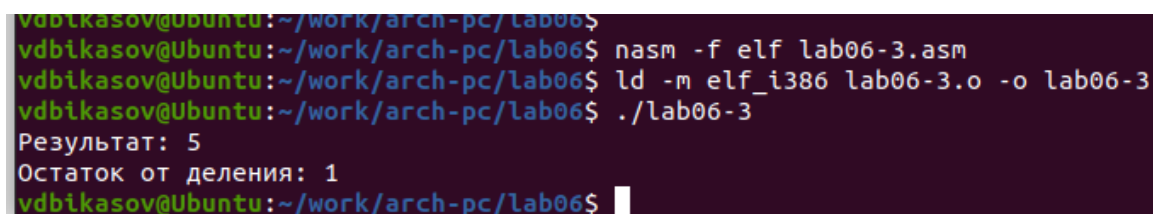
исполняемый файл и проверил его работу.



```
vdbikasov@Ubuntu: ~/work/arch-pc/lab06
GNU nano 4.8 lab06-3.asm
%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: Программа lab06-3.asm с другим выражением

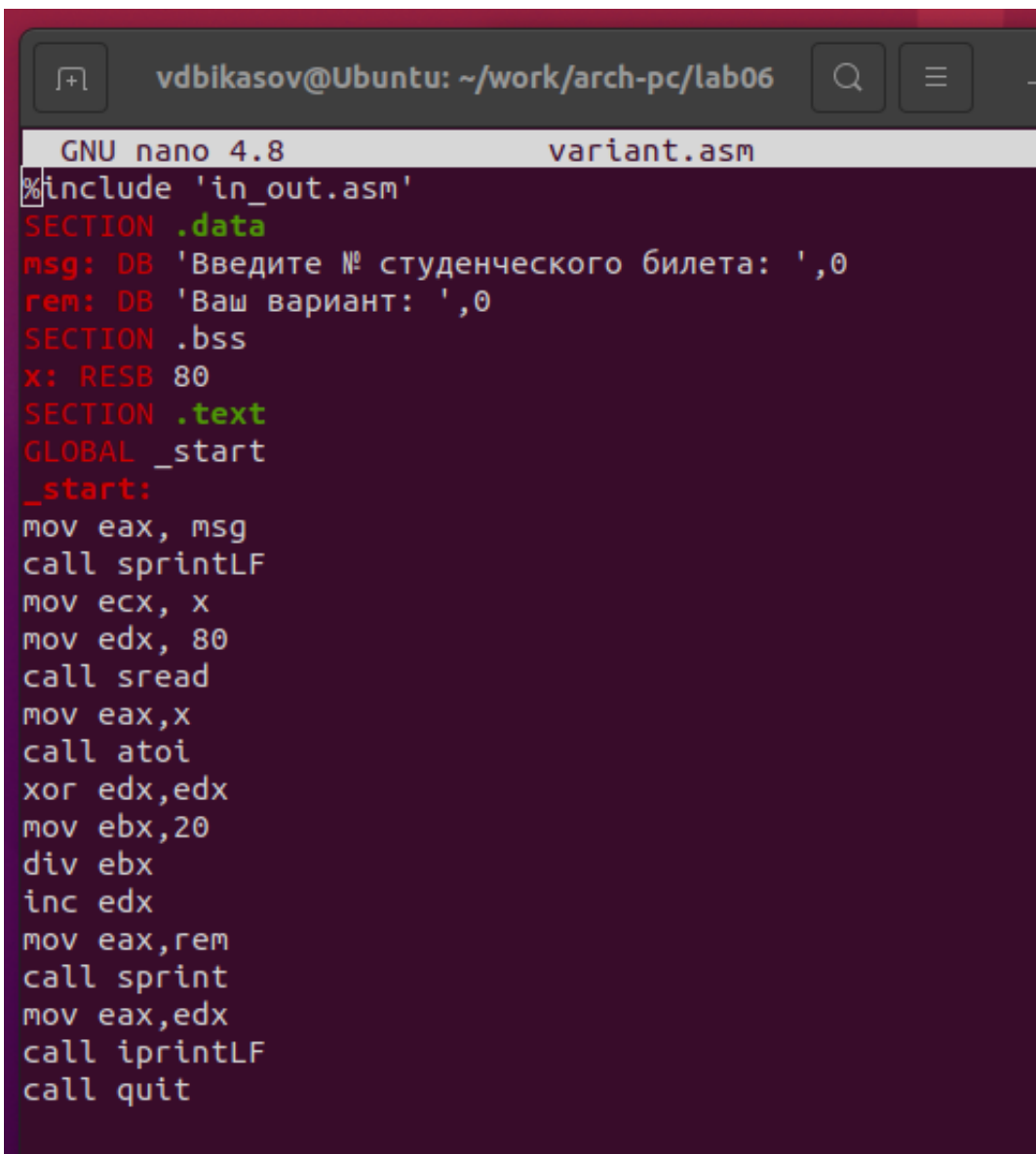


```
vdbikasov@Ubuntu:~/work/arch-pc/lab06$
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./lab06-3
Результат: 5
Остаток от деления: 1
vdbikasov@Ubuntu:~/work/arch-pc/lab06$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

Рис. 2.13: Запуск программы lab06-3.asm с другим выражением

Еще один пример — программа для вычисления варианта задания, используя номер студенческого билета.

В этом случае значение для вычислений вводится с клавиатуры. Как я уже упоминал, ввод данных осуществляется в символьной форме, и для правильной работы арифметических операций необходимо преобразовать символы в числа. Для этого используется функция `atoi` из файла `in_out.asm`.

A screenshot of a terminal window on Ubuntu. The window title is 'vdbikasov@Ubuntu: ~/work/arch-pc/lab06'. The editor is GNU nano 4.8, editing the file 'variant.asm'. The code is as follows:

```
%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: Программа `variant.asm`

```
vdbikasov@Ubuntu:~/work/arch-pc/lab06$  
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant.asm  
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant  
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./variant  
Введите № студенческого билета:  
1132242462  
Ваш вариант: 3  
vdbikasov@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.15: Запуск программы variant.asm

## 2.3 Ответы на вопросы

1. Какие строки листинга отвечают за вывод на экран сообщения ‘Ваш вариант:’?
  - Инструкция `mov eax, get` переносит значение переменной с фразой ‘Ваш вариант:’ в регистр `eax`.
  - Инструкция `call sprint` вызывает подпрограмму для вывода строки.
2. Для чего используются следующие инструкции?
  - Инструкция `mov ecx, x` — для помещения значения переменной `x` в регистр `ecx`.
  - Инструкция `mov edx, 80` — для помещения значения `80` в регистр `edx`.
  - Инструкция `call sread` — для вызова подпрограммы для считывания значения студенческого билета.
3. Для чего используется инструкция “call atoi”?
  - Инструкция “call atoi” используется для преобразования введенных символов в числовой формат.
4. Какие строки листинга отвечают за вычисления варианта?
  - Инструкция `xor edx, edx` обнуляет регистр `edx`.
  - Инструкция `mov ebx, 20` записывает значение `20` в регистр `ebx`.



- Инструкция `div ebx` выполняет деление номера студенческого билета на 20.
- Инструкция `inc edx` увеличивает значение регистра `edx` на 1.

В данном случае происходит деление номера студенческого билета на 20. Остаток от деления сохраняется в регистре `edx`, и к нему прибавляется 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`”?

- Инструкция “`inc edx`” увеличивает значение в регистре `edx` на 1, что необходимо для вычисления варианта.

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

- Инструкция `mov eax, edx` записывает результат вычислений в регистр `eax`.
- Инструкция `call iprintLF` вызывает подпрограмму для вывода результата на экран.

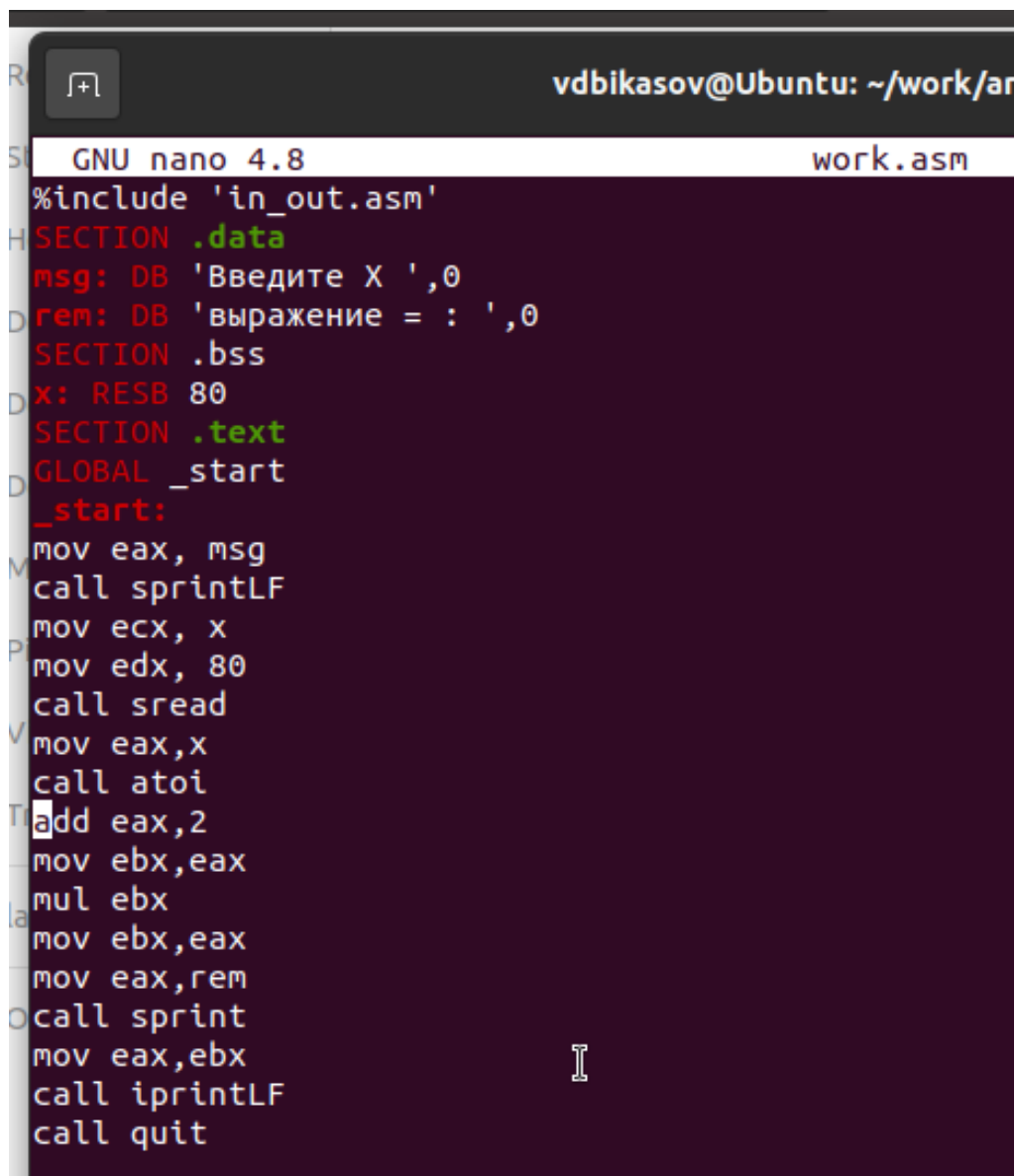
## 2.4 Задание для самостоятельной работы

Написал программу для вычисления выражения  $y = f(x)$ . Программа должна выводить выражение для вычисления, запросить ввод значения  $x$ , вычислить выражение в зависимости от введенного  $x$  и вывести результат. В функцию  $f(x)$  выбрал вариант из таблицы 6.3 в соответствии с номером, полученным в лабораторной работе. Создал исполняемый файл и проверил его работу для значений  $x_1$  и  $x_2$ .

Получил вариант 3:  $f(x) = (2 + x)^2$  для  $x=2$  и  $x=8$ .

При  $x=2$  результат — 16.

При \$ x=8 \$ результат — 100.



```
GNU nano 4.8 work.asm
#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, 2
mov ebx, eax
mul ebx
mov ebx, eax
mov eax, rem
call sprint
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.16: Программа work.asm

```
vdbikasov@Ubuntu: ~/work/arch-pc/lab06$ nasm -f elf work.asm
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 work.o -o work
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./work
Введите X
2
выражение = : 16
vdbikasov@Ubuntu:~/work/arch-pc/lab06$ ./work
Введите X
8
выражение = : 100
vdbikasov@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.17: Запуск программы work.asm

Программа работает корректно.

## **3 Выводы**

Изучили работу с арифметическими операциями.