

Отчёт по лабораторной работе 6

Архитектура компьютеров и операционные системы

Абдулфазова Лейла Али гызы

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Символьные и численные данные в NASM	9
4.2	Выполнение арифметических операций в NASM	14
4.2.1	Ответы на вопросы по программе variant.asm	18
4.3	Выполнение заданий для самостоятельной работы	20
5	Выводы	23

Список иллюстраций

4.1	Редактирование файла lab6-1.asm	10
4.2	Проверка кода lab6-1.asm	10
4.3	Редактирование файла lab6-1.asm	11
4.4	Проверка кода lab6-1.asm	11
4.5	Редактирование файла lab6-2.asm	12
4.6	Проверка кода lab6-2.asm	12
4.7	Редактирование файла lab6-2.asm	13
4.8	Проверка кода lab6-2.asm	13
4.9	Редактирование файла lab6-2.asm	14
4.10	Проверка кода lab6-2.asm	14
4.11	Редактирование файла lab6-3.asm	15
4.12	Проверка кода lab6-3.asm	15
4.13	Редактирование файла lab6-3.asm	16
4.14	Проверка кода lab6-3.asm	16
4.15	Редактирование файла variant.asm	17
4.16	Проверка кода variant.asm	18
4.17	Редактирование файла calc.asm	21
4.18	Проверка кода calc.asm	22

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Изучение арифметических операций в ассемблере
2. Изучение типов данных в ассемблере
3. Выполнение заданий, рассмотрение примеров
4. 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.

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`.

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

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре `EAX, AX` или `AL`, а результат помещается в регистры `EDX:EAX, DX:AX` или `AX`, в зависимости от размера операнда.

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

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера

делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 6.2.

4 Выполнение лабораторной работы

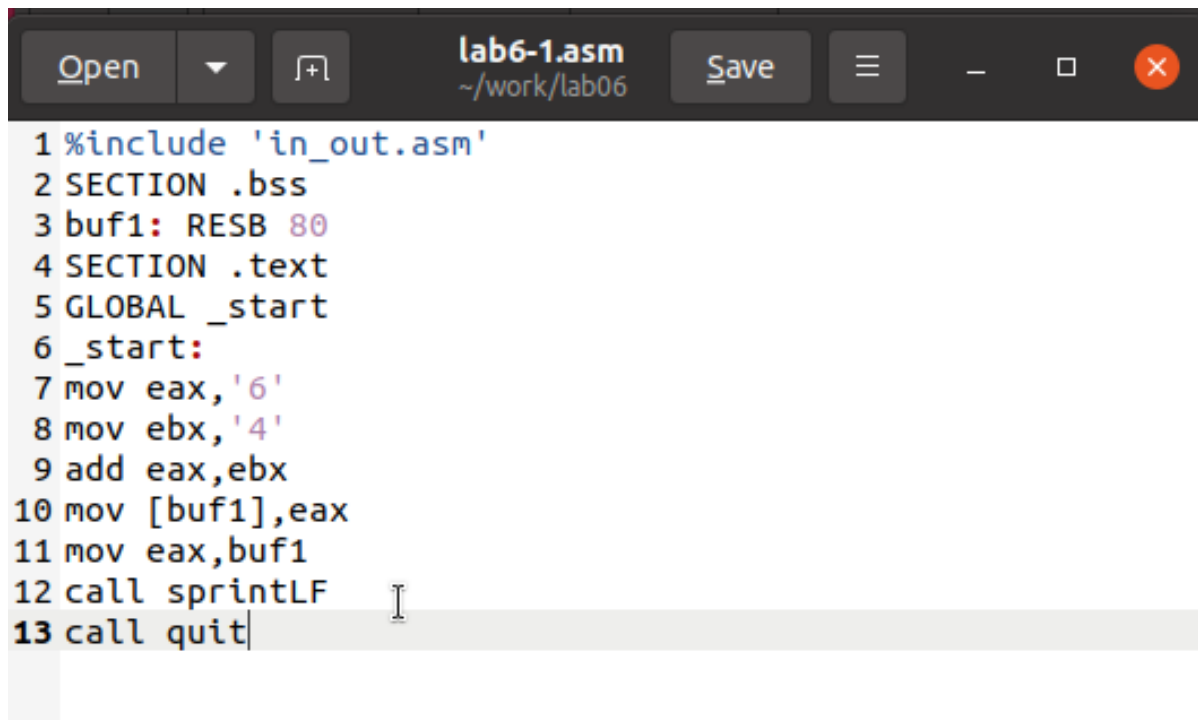
4.1 Символьные и численные данные в NASM

Создала каталог для программ лабораторной работы №6, перешла в него и создала файл с названием “lab6-1.asm”.

Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр еах.

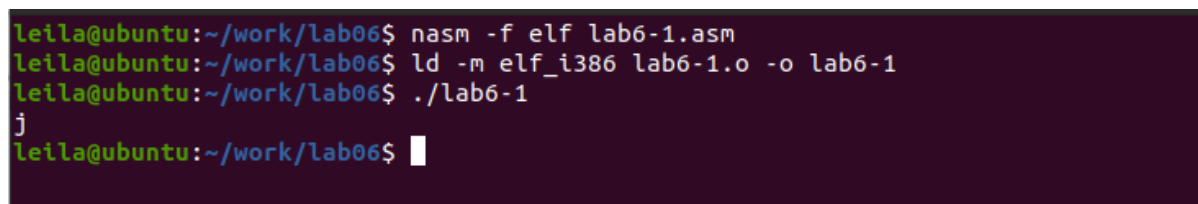
В данной программе, в регистр еах записан символ ‘6’, а в регистр ебх символ ‘4’. Затем мы прибавляем значение регистра ебх к значению в регистре еах (результат сложения будет записан в регистр еах). После этого мы выводим результат.

Так как для работы функции sprintLF в регистр еах должен быть записан адрес, мы используем дополнительную переменную. Мы записали значение регистра еах в переменную с именем “buf1”, а затем записали адрес переменной buf1 в регистр еах и вызвали функцию sprintLF.



```
1 %include 'in_out.asm'
2 SECTION .bss
3 buf1: RESB 80
4 SECTION .text
5 GLOBAL _start
6 _start:
7 mov eax, '6'
8 mov ebx, '4'
9 add eax, ebx
10 mov [buf1], eax
11 mov eax, buf1
12 call printf
13 call quit
```

Рис. 4.1: Редактирование файла lab6-1.asm

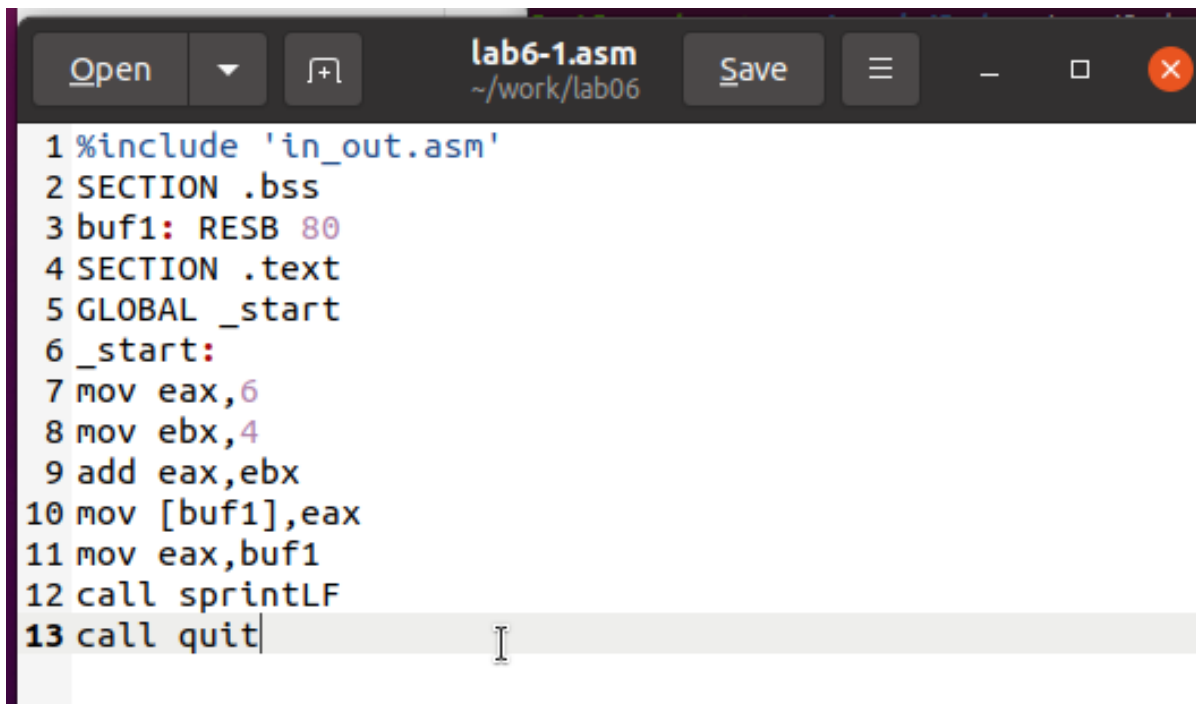


```
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-1.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-1.o -o lab6-1
leila@ubuntu:~/work/lab06$ ./lab6-1
j
leila@ubuntu:~/work/lab06$
```

Рис. 4.2: Проверка кода lab6-1.asm

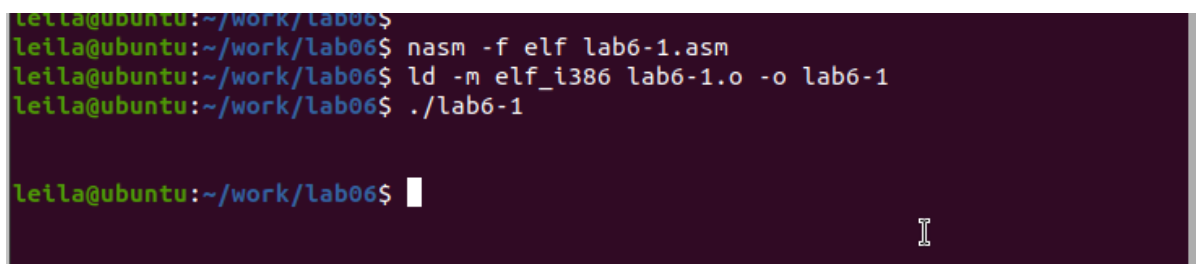
В данном случае, при выводе значения регистра `eax`, ожидалось увидеть число 10. Однако, результатом был символ 'j'. Это произошло потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда `add eax, ebx` записала в регистр `eax` сумму кодов – 01101010 (106), что в свою очередь является кодом символа 'j'.

Далее был изменен текст программы и вместо символов записаны числа.



```
1 %include 'in_out.asm'
2 SECTION .bss
3 buf1: RESB 80
4 SECTION .text
5 GLOBAL _start
6 _start:
7 mov eax,6
8 mov ebx,4
9 add eax,ebx
10 mov [buf1],eax
11 mov eax,buf1
12 call sprintLF
13 call quit
```

Рис. 4.3: Редактирование файла lab6-1.asm



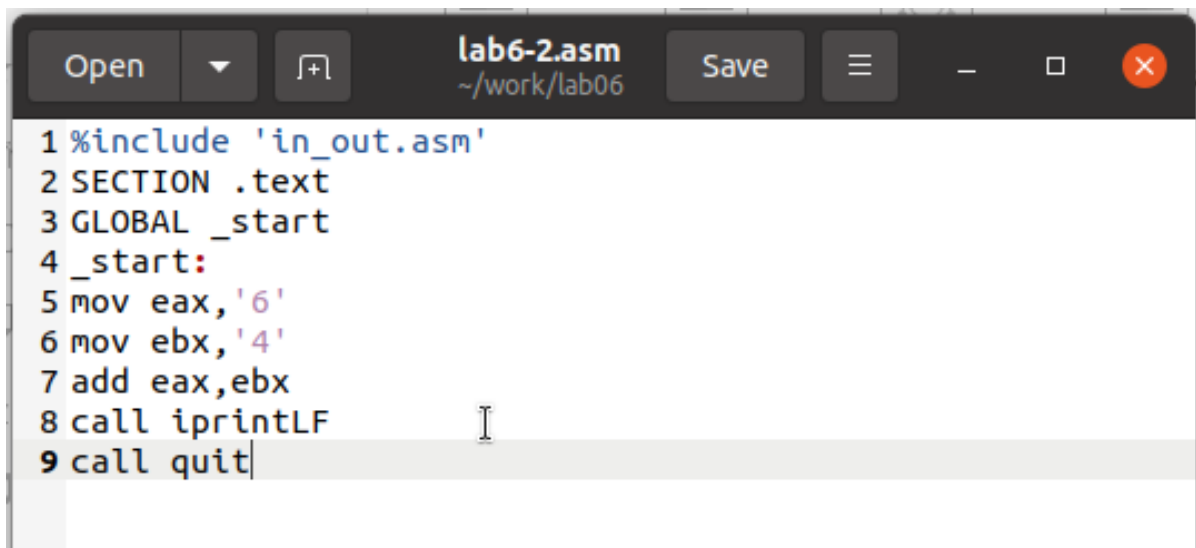
```
leila@ubuntu:~/work/lab06$
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-1.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-1.o -o lab6-1
leila@ubuntu:~/work/lab06$ ./lab6-1

leila@ubuntu:~/work/lab06$
```

Рис. 4.4: Проверка кода lab6-1.asm

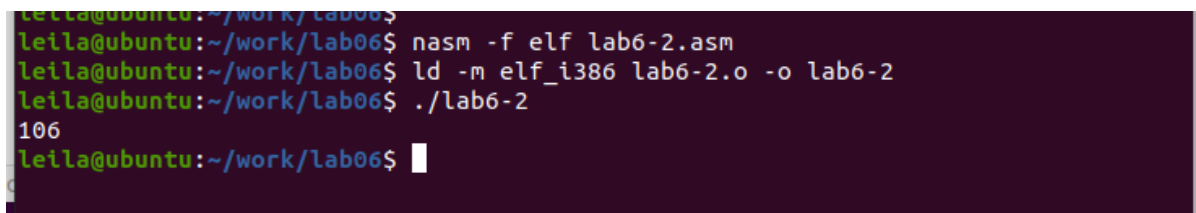
В процессе выполнения программы не получили ожидаемое число 10. Вместо этого был выведен символ с кодом 10. Это символ конца строки (возврат каретки), который в консоли не отображается, но добавляет пустую строку.

В файле “in_out.asm” реализованы подпрограммы для работы с числами и преобразования символов ASCII. Был модифицирован текст программы с использованием этих функций.

A screenshot of a text editor window titled "lab6-2.asm" with a dark theme. The window has a menu bar with "Open", "Save", and window control buttons. The code is as follows:

```
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax, '6'
6 mov ebx, '4'
7 add eax, ebx
8 call iprintLF
9 call quit
```

Рис. 4.5: Редактирование файла lab6-2.asm

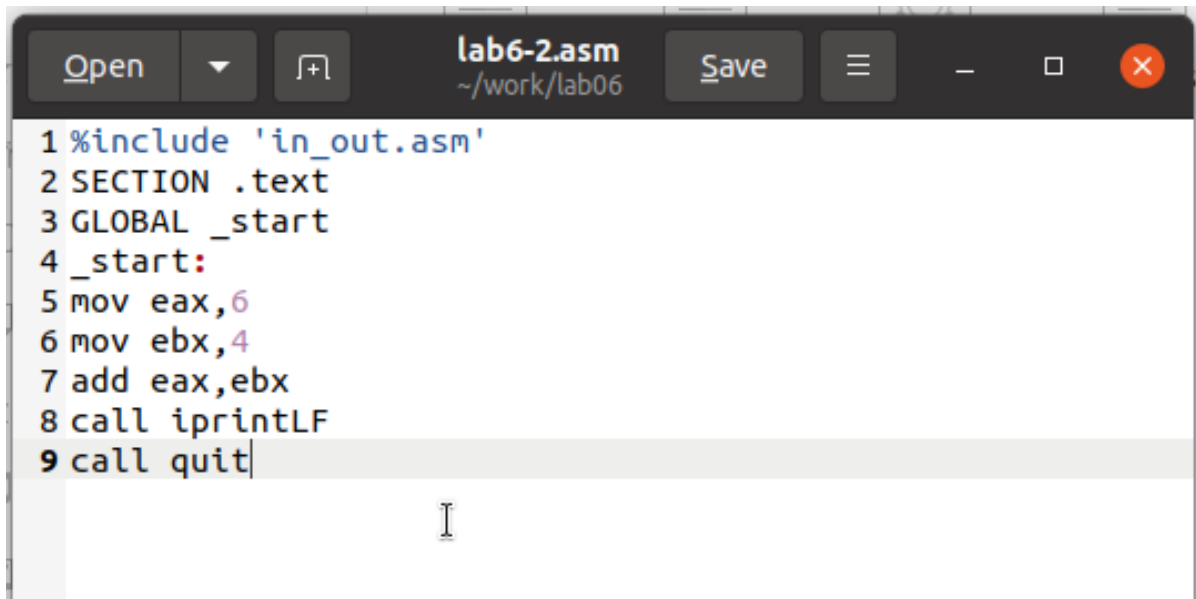
A screenshot of a terminal window with a dark background. The user "leila" is at the prompt "leila@ubuntu:~/work/lab06\$". The commands and output are:

```
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-2.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
leila@ubuntu:~/work/lab06$ ./lab6-2
106
leila@ubuntu:~/work/lab06$
```

Рис. 4.6: Проверка кода lab6-2.asm

В результате выполнения обновленной программы было выведено число 106. Здесь, как и в первом случае, команда `add` складывает коды символов '6' и '4' ($54 + 52 = 106$). Но в отличие от предыдущей версии, функция `iprintLF` позволяет напечатать само число, а не символ с соответствующим кодом.

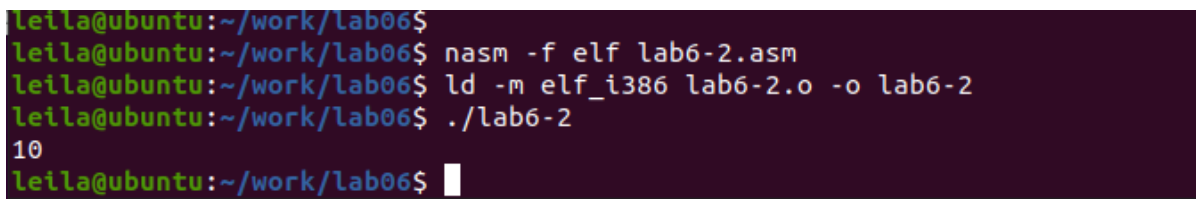
По аналогии с предыдущим примером, были заменены символы на числа.



```
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax,6
6 mov ebx,4
7 add eax,ebx
8 call iprintLF
9 call quit
```

Рис. 4.7: Редактирование файла lab6-2.asm

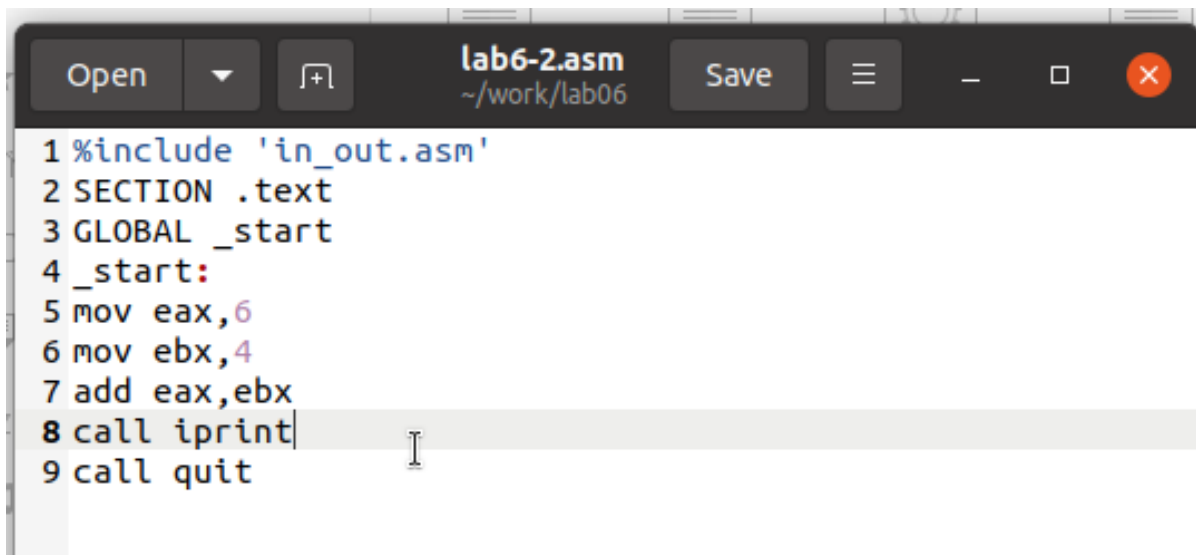
Функция `iprintLF` позволяет выводить числа, и на этот раз в качестве операндов использовались именно числа, а не коды символов. В результате мы получили число 10.



```
leila@ubuntu:~/work/lab06$
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-2.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
leila@ubuntu:~/work/lab06$ ./lab6-2
10
leila@ubuntu:~/work/lab06$
```

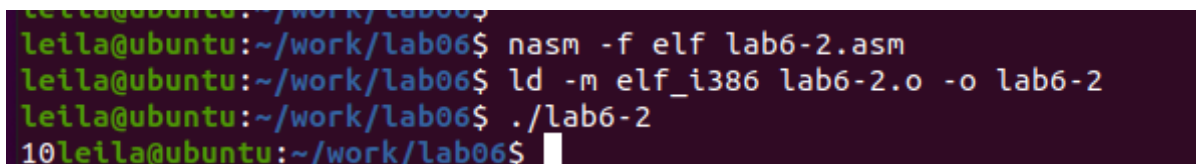
Рис. 4.8: Проверка кода lab6-2.asm

Далее была заменена функция `iprintLF` на `iprint`, создан исполняемый файл и запущен. Вывод теперь отличается отсутствием перехода на новую строку.



```
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax,6
6 mov ebx,4
7 add eax,ebx
8 call iprint
9 call quit
```

Рис. 4.9: Редактирование файла lab6-2.asm



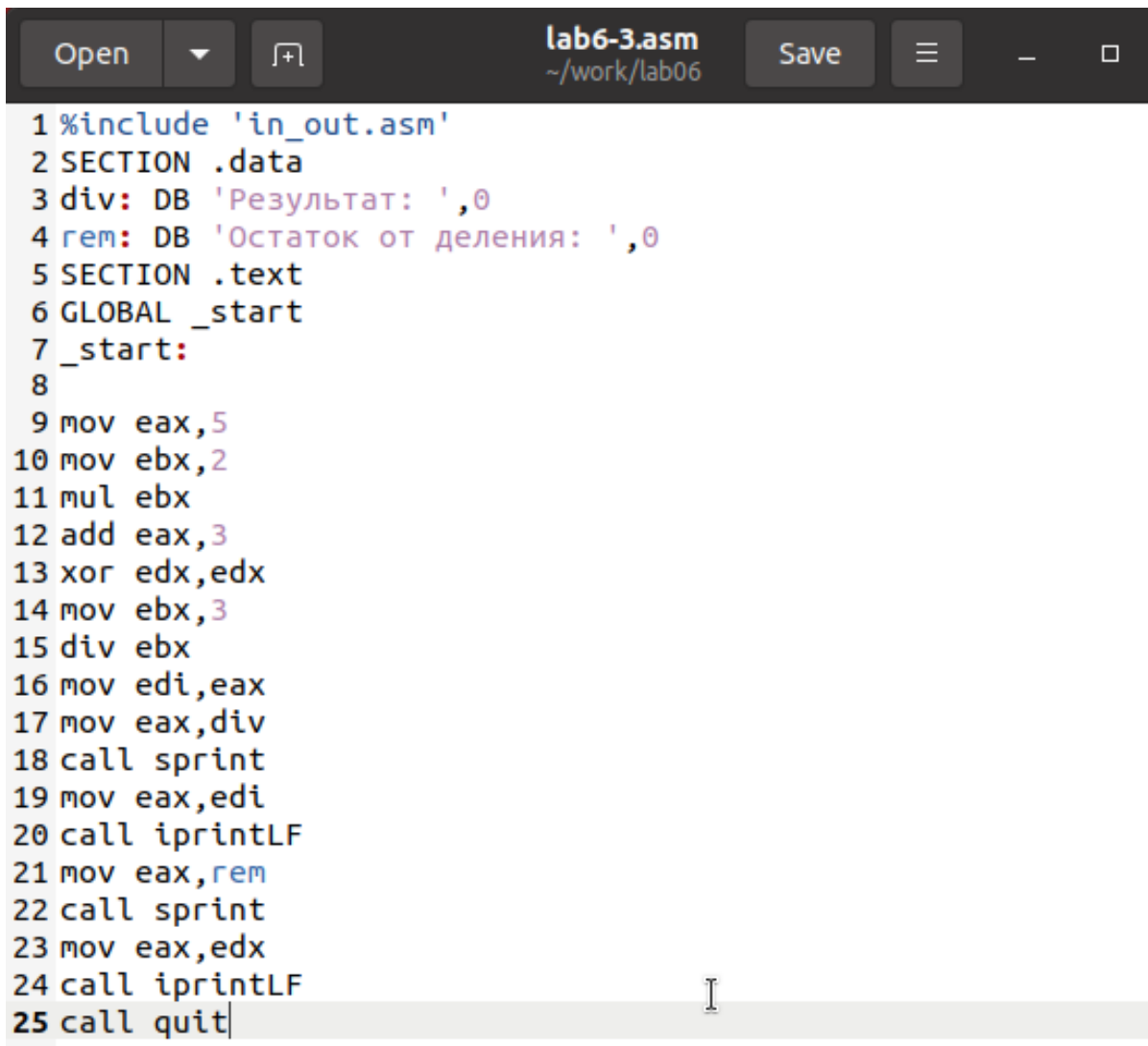
```
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-2.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
leila@ubuntu:~/work/lab06$ ./lab6-2
10leila@ubuntu:~/work/lab06$
```

Рис. 4.10: Проверка кода lab6-2.asm

4.2 Выполнение арифметических операций в NASM

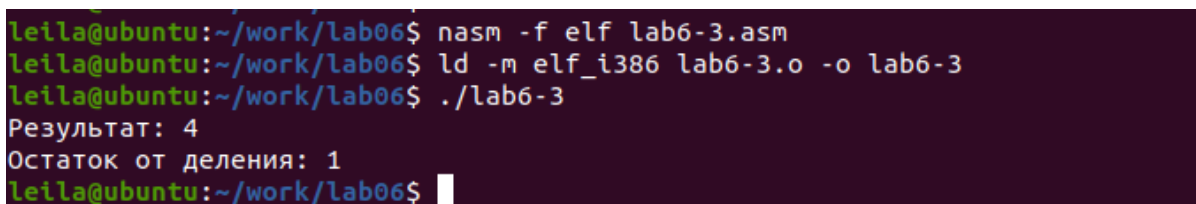
В качестве примера выполнения арифметических операций в NASM рассмотрим программу для вычисления арифметического выражения

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



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5 SECTION .text
6 GLOBAL _start
7 _start:
8
9 mov eax,5
10 mov ebx,2
11 mul ebx
12 add eax,3
13 xor edx,edx
14 mov ebx,3
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 mov eax,rem
22 call sprint
23 mov eax,edx
24 call iprintLF
25 call quit
```

Рис. 4.11: Редактирование файла lab6-3.asm



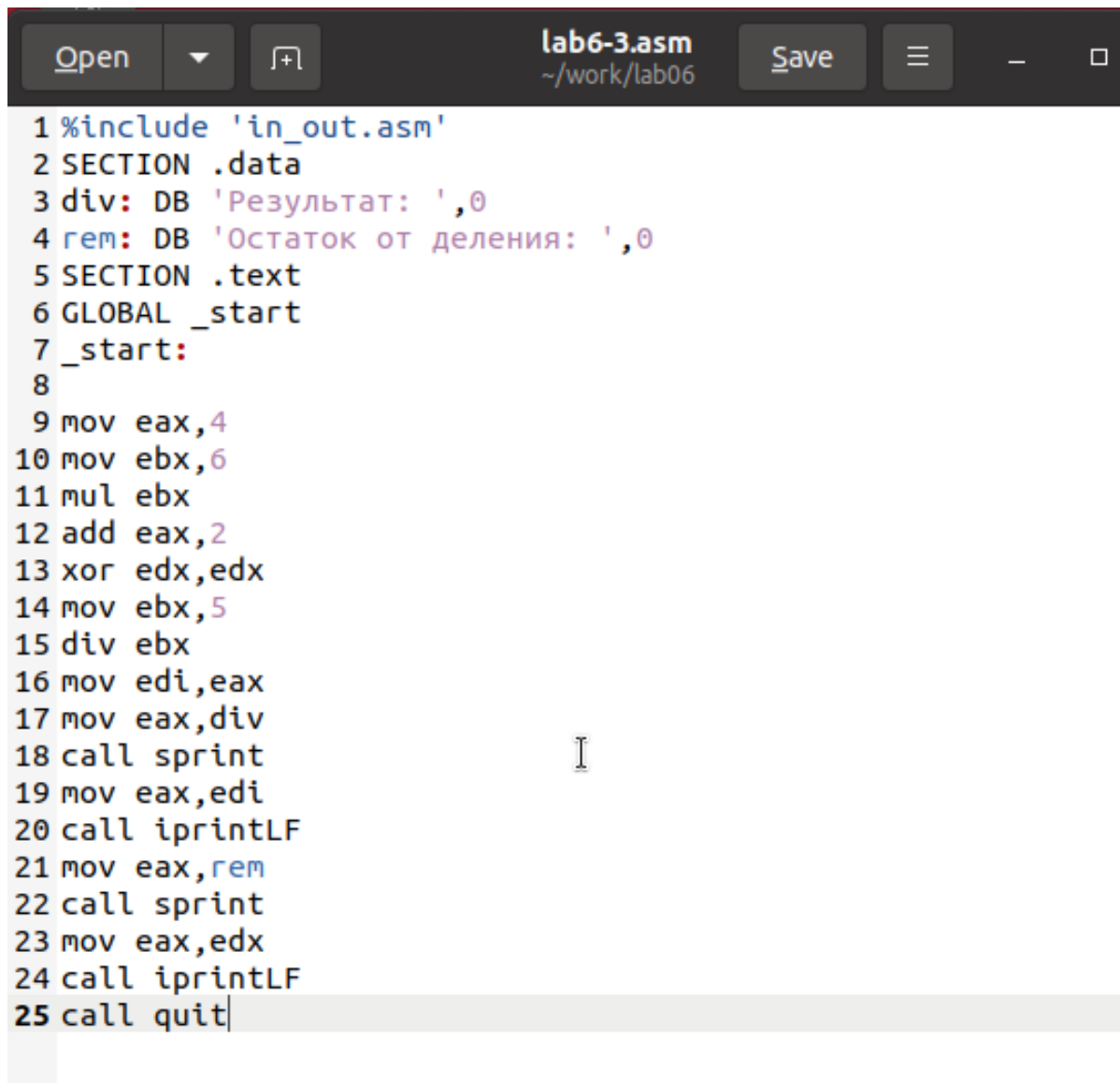
```
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-3.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
leila@ubuntu:~/work/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
leila@ubuntu:~/work/lab06$
```

Рис. 4.12: Проверка кода lab6-3.asm

Изменила текст программы для вычисления выражения

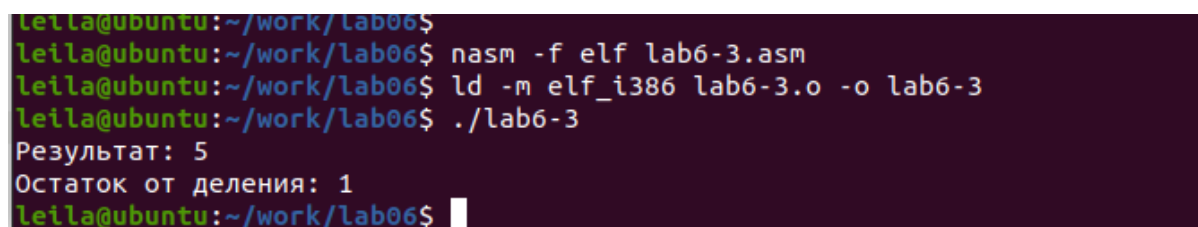
$$f(x) = (4 * 6 + 2) / 5$$

. Создала исполняемый файл и проверила его работу.



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5 SECTION .text
6 GLOBAL _start
7 _start:
8
9 mov eax,4
10 mov ebx,6
11 mul ebx
12 add eax,2
13 xor edx,edx
14 mov ebx,5
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 mov eax,rem
22 call sprint
23 mov eax,edx
24 call iprintLF
25 call quit
```

Рис. 4.13: Редактирование файла lab6-3.asm

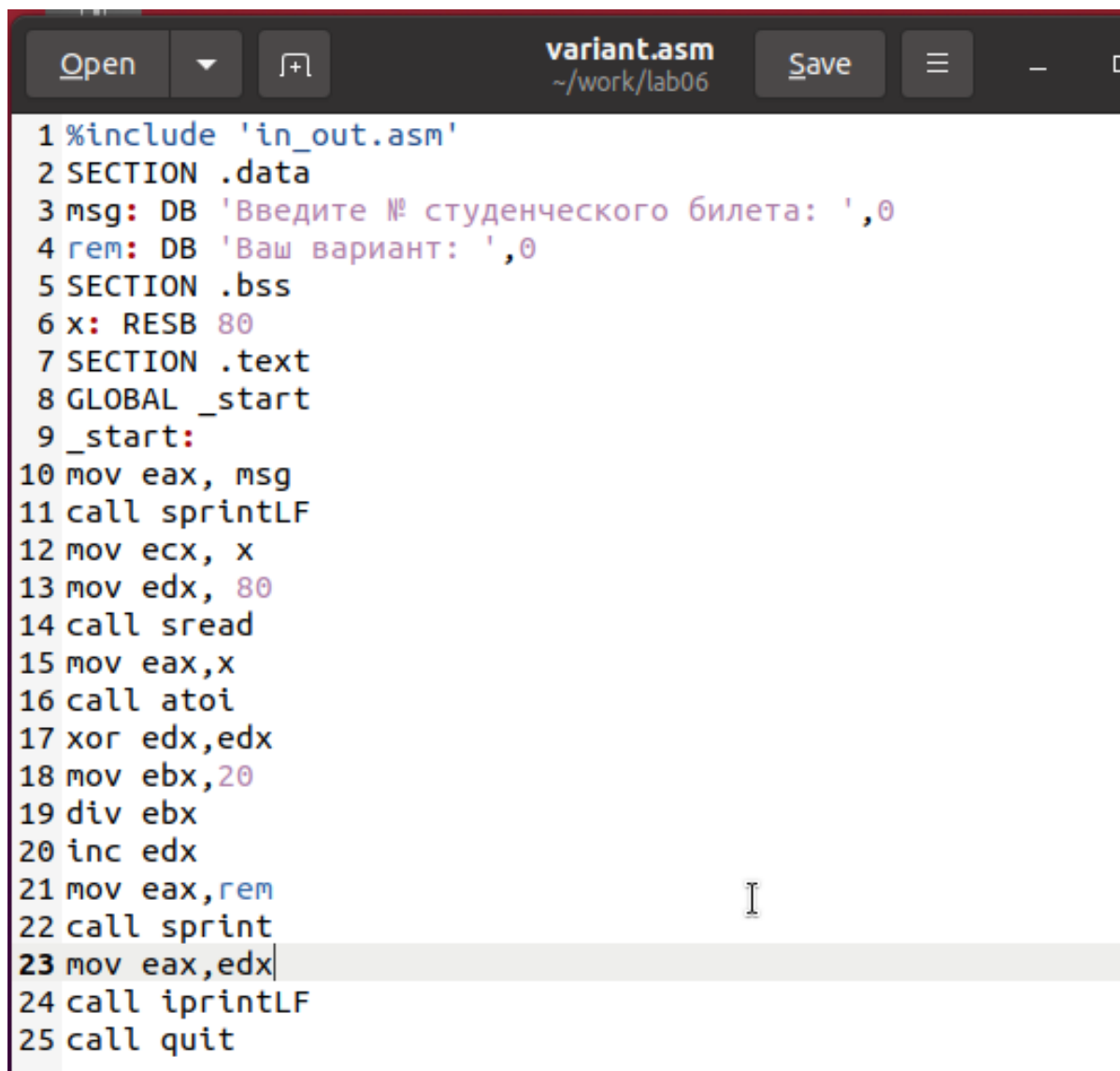


```
leila@ubuntu:~/work/lab06$
leila@ubuntu:~/work/lab06$ nasm -f elf lab6-3.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
leila@ubuntu:~/work/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
leila@ubuntu:~/work/lab06$
```

Рис. 4.14: Проверка кода lab6-3.asm

В качестве еще одного примера рассмотрим программу для вычисления варианта задания на основе номера студенческого билета.

В этом случае число, над которым нужно выполнять арифметические операции, вводится с клавиатуры. Как уже отмечалось ранее, ввод с клавиатуры осуществляется в символьном виде. Для корректной работы арифметических операций в NASM эти символы необходимо преобразовать в числовой формат. С этой целью можно использовать функцию `atoi` из файла `in_out.asm`. Она конвертирует строку символов в эквивалентное десятичное число.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите № студенческого билета: ',0
4 rem: DB 'Ваш вариант: ',0
5 SECTION .bss
6 x: RESB 80
7 SECTION .text
8 GLOBAL _start
9 _start:
10 mov eax, msg
11 call sprintfLF
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax, x
16 call atoi
17 xor edx, edx
18 mov ebx, 20
19 div ebx
20 inc edx
21 mov eax, rem
22 call sprintf
23 mov eax, edx
24 call iprintLF
25 call quit
```

Рис. 4.15: Редактирование файла `variant.asm`

```
leila@ubuntu:~/work/lab06$ nasm -f elf variant.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 variant.o -o variant
leila@ubuntu:~/work/lab06$ ./variant
Введите № студенческого билета:
1032235809
Ваш вариант: 10
leila@ubuntu:~/work/lab06$
```

Рис. 4.16: Проверка кода variant.asm

4.2.1 Ответы на вопросы по программе variant.asm

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

Ответ: Строки, отвечающие за вывод сообщения “Ваш вариант:”, - это строки, где происходит перемещение фразы в регистры с помощью инструкции `mov`, `get`, а затем вызов подпрограммы вывода строки с помощью инструкции `call sprint`.

2. Для чего используются следующие инструкции?

Ответ:

- `mov ecx, x`: Инструкция `mov ecx, x` используется для сохранения значения регистра `ecx` в переменной `x`.
- `mov edx, 80`: Инструкция `mov edx, 80` используется для присваивания значения 80 регистру `edx`.
- `call sread`: Инструкция `call sread` используется для вызова подпрограммы, которая считывает данные из консоли.

3. Для чего используется инструкция “call atoi”?

Ответ: Инструкция `call atoi` используется для преобразования введенных символов в числовой формат.

4. Какие строки листинга отвечают за вычисления варианта?

Ответ: Строки, отвечающие за вычисление варианта, включают следующие инструкции:

- `xor edx, edx`: Инструкция `xor edx, edx` используется для обнуления регистра `edx`.
- `mov ebx, 20`: Инструкция `mov ebx, 20` используется для присваивания значения 20 регистру `ebx`.
- `div ebx`: Инструкция `div ebx` используется для деления номера студента на 20.
- `inc edx`: Инструкция `inc edx` используется для увеличения значения регистра `edx` на 1.

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

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

6. Для чего используется инструкция “`inc edx`”?

Ответ: Инструкция `inc edx` используется для увеличения значения регистра `edx` на 1, что необходимо для вычисления варианта по формуле.

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

Ответ: Строки, отвечающие за вывод на экран результата вычислений, включают следующие инструкции:

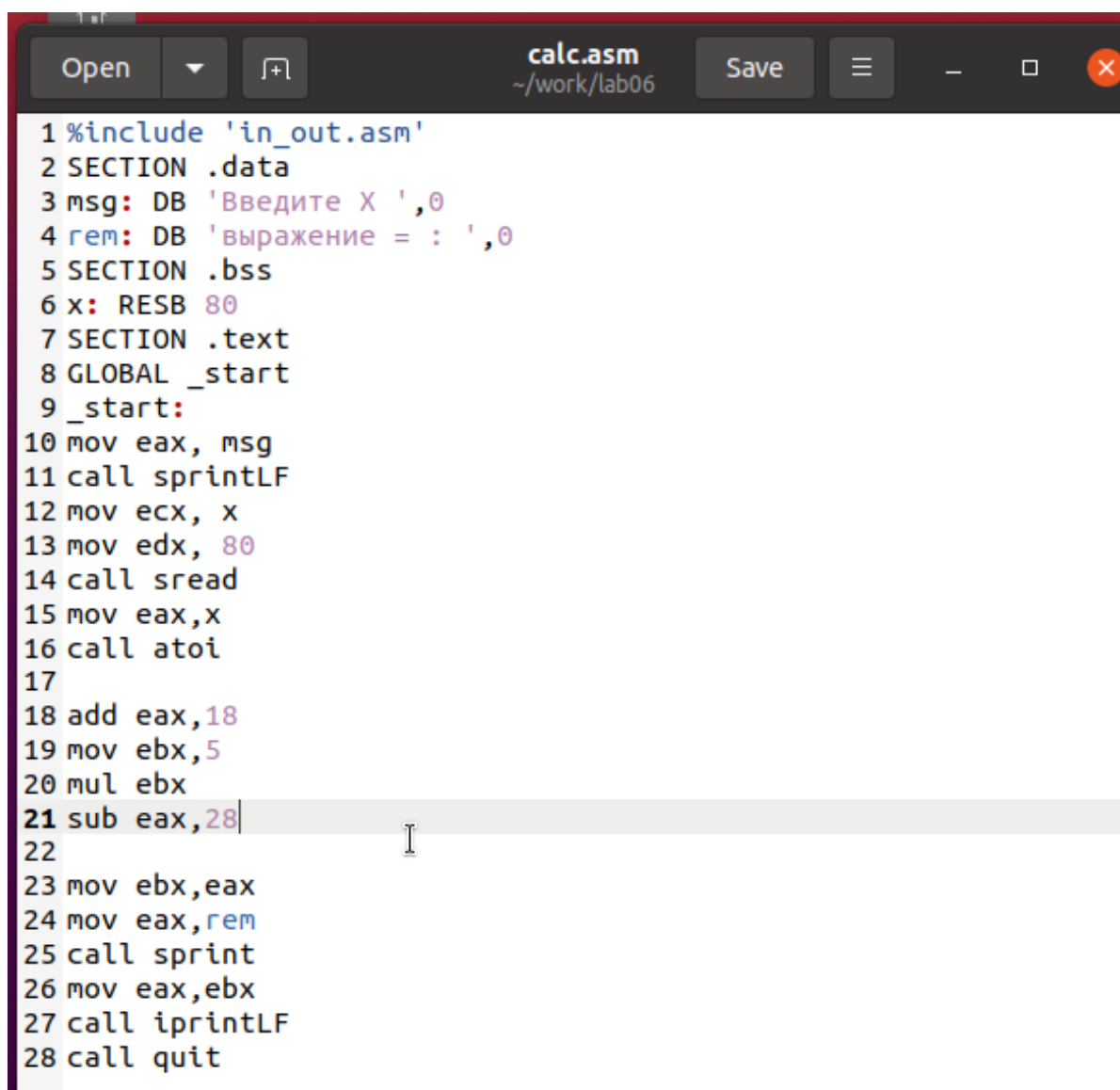
- `mov eax, edx`: Инструкция `mov eax, edx` используется для помещения результата в регистр `eax`.
- `call iprintLF`: Инструкция `call iprintLF` используется для вызова подпрограммы вывода результата.

4.3 Выполнение заданий для самостоятельной работы

Написала программу для вычисления выражения $y = f(x)$. Программа выводит выражение для вычисления, запрашивает ввод значения x , вычисляет заданное выражение в зависимости от введенного x и выводит результат вычислений. Для выбора вида функции $f(x)$ использовала таблицу 6.3 вариантов заданий, в соответствии с номером, полученным при выполнении лабораторной работы.

Создала исполняемый файл и проверила его работу для значений x_1 и x_2 из таблицы 6.3.

Вариант 10 - $5 * (x + 18) - 28$ для $x = 2, x = 3$



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите X ',0
4 rem: DB 'выражение = : ',0
5 SECTION .bss
6 x: RESB 80
7 SECTION .text
8 GLOBAL _start
9 _start:
10 mov eax, msg
11 call sprintLF
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax,x
16 call atoi
17
18 add eax,18
19 mov ebx,5
20 mul ebx
21 sub eax,28
22
23 mov ebx,eax
24 mov eax,rem
25 call sprint
26 mov eax,ebx
27 call iprintLF
28 call quit
```

Рис. 4.17: Редактирование файла calc.asm

```
leila@ubuntu:~/work/lab06$ nasm -f elf calc.asm
leila@ubuntu:~/work/lab06$ ld -m elf_i386 calc.o -o calc
leila@ubuntu:~/work/lab06$ ./calc
Введите X
2
выражение = : 72
leila@ubuntu:~/work/lab06$ ./calc
Введите X
3
выражение = : 77
leila@ubuntu:~/work/lab06$
```

Рис. 4.18: Проверка кода calc.asm

Программа считает верно.

5 Выводы

Изучили работу с арифметическими операциями.