

Отчёт по лабораторной работе 6

Архитектура компьютеров и операционные системы

Алина Молокова

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Символьные и численные данные в NASM	6
2.2	Выполнение арифметических операций в NASM	12
2.3	Ответы на вопросы	17
2.4	Задание для самостоятельной работы	18
3	Выводы	21

Список иллюстраций

2.1	Каталог	6
2.2	Редактирую файл lab6-1.asm	7
2.3	Запуск программы lab6-1.asm	8
2.4	Редактирую файл lab6-1.asm	8
2.5	Запуск программы lab6-1.asm	9
2.6	Редактирую файл lab6-2.asm	10
2.7	Запуск программы lab6-2.asm	10
2.8	Редактирую файл lab6-2.asm	11
2.9	Запуск программы lab6-2.asm	11
2.10	Запуск программы lab6-2.asm	12
2.11	Редактирую файл lab6-3.asm	13
2.12	Запуск программы lab6-3.asm	14
2.13	Редактирую файл lab6-3.asm	14
2.14	Запуск программы lab6-3.asm	15
2.15	Редактирую файл variant.asm	16
2.16	Запуск программы variant.asm	17
2.17	Редактирую файл work.asm	19
2.18	Запуск программы task.asm	20

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 создала папку для хранения файлов, связанных с шестой лабораторной работой, и перешла в эту папку. Затем я создала файл с кодом программы, который назвала lab6-1.asm. (рис. [2.1])

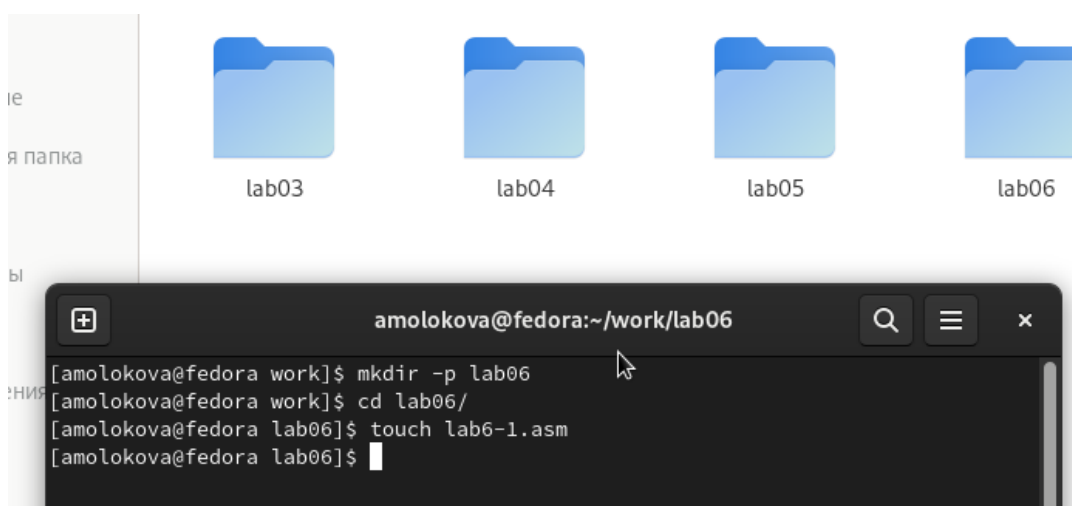
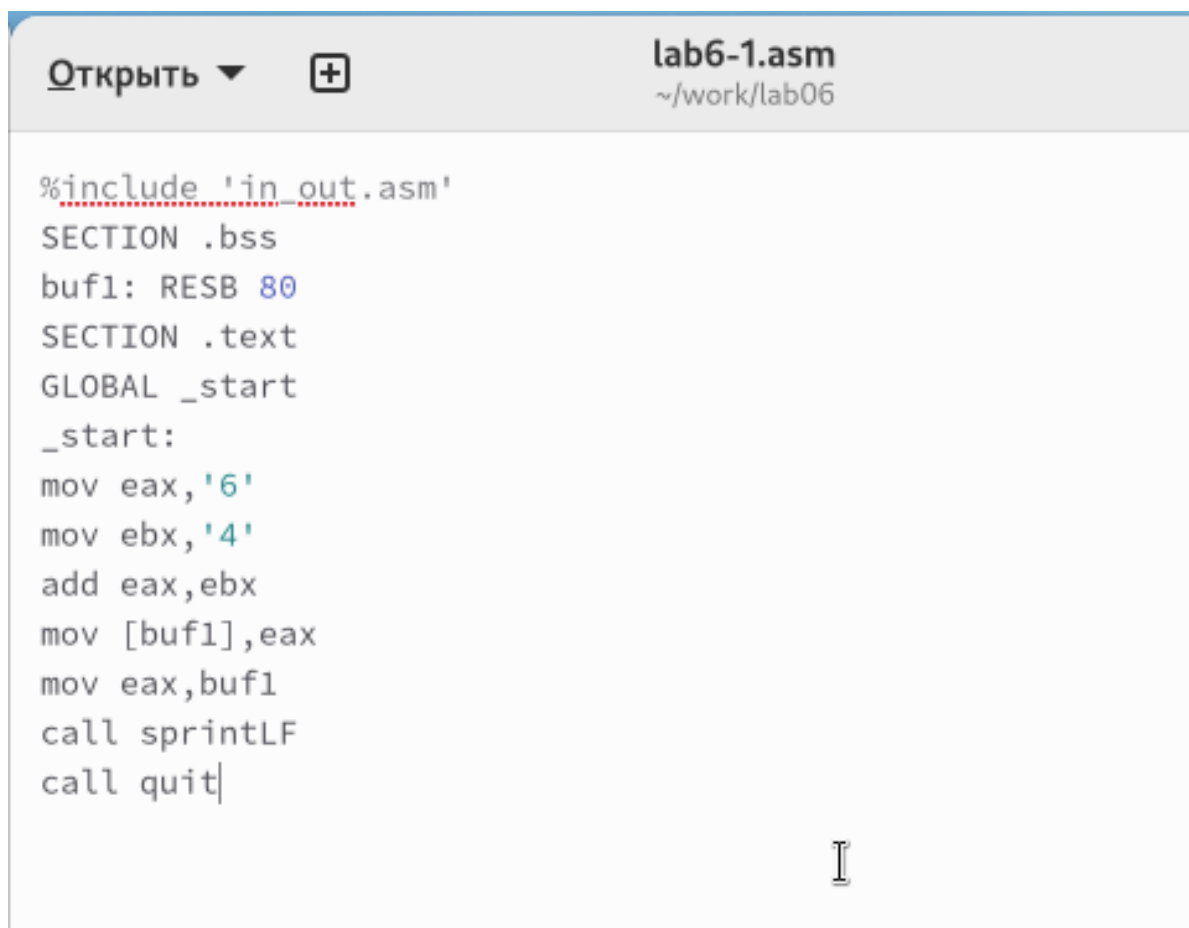


Рис. 2.1: Каталог

На следующем рисунке (рис. [2.2]) показана программа, где я помещаю символ '6' в регистр eax с помощью команды mov eax, '6', а символ '4' - в регистр ebx с помощью команды mov ebx, '4'. После этого я складываю значения из регистров eax и ebx, используя команду add eax, ebx, и результат сложения сохраняется в регистре eax. Чтобы вывести результат на экран, мне нужно использовать

функцию `sprintf`, которая требует адреса в регистре `eax`. Поэтому я сохраняю результат из `eax` в переменную `buf1` с помощью команды `mov [buf1], eax`, а затем загружаю адрес переменной `buf1` обратно в регистр `eax` командой `mov eax, buf1` перед вызовом функции `sprintf`.



```
%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 sprintf
call quit
```

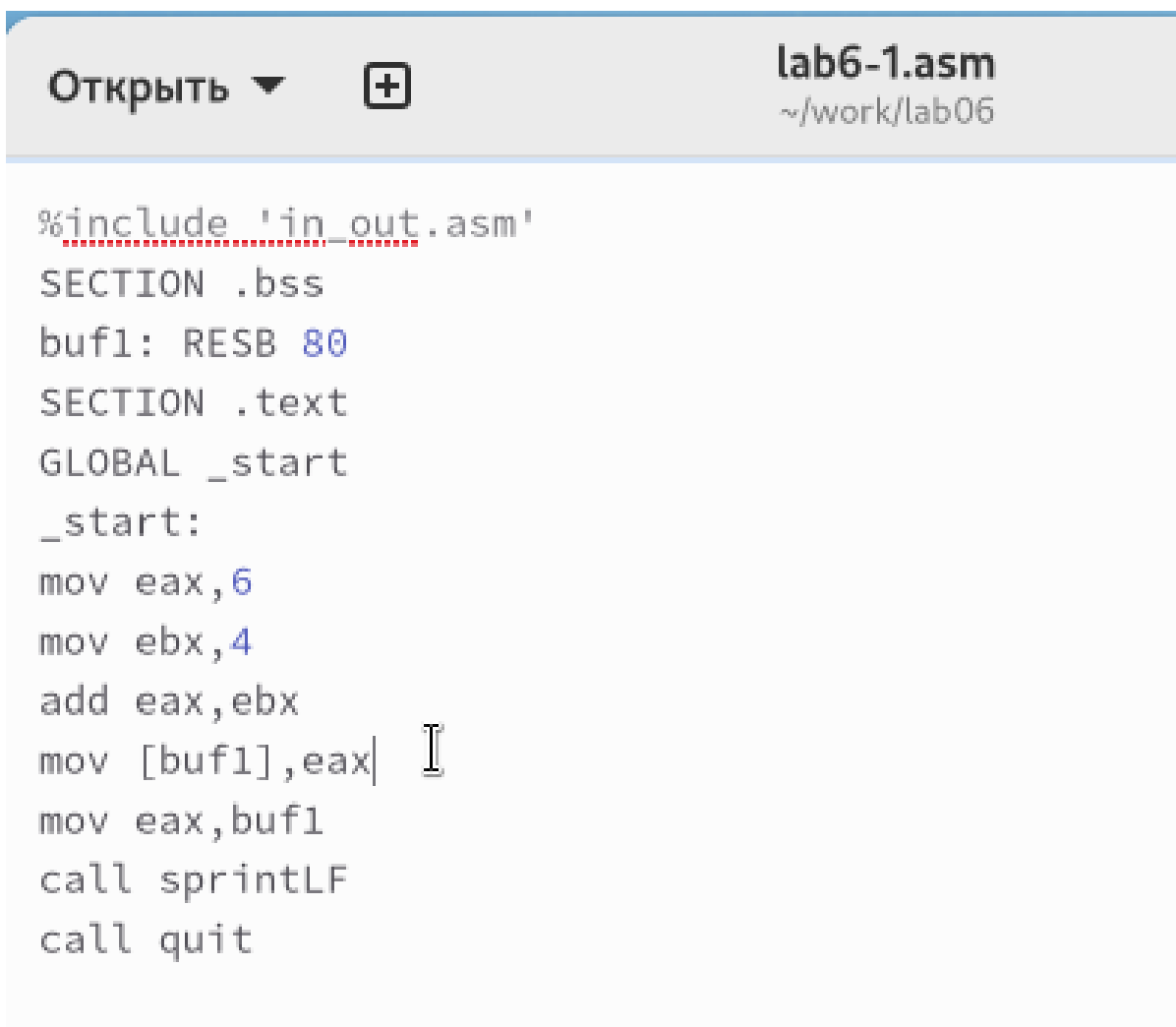
Рис. 2.2: Редактирую файл `lab6-1.asm`

Однако, когда я ожидаю увидеть число 10 на экране после вывода содержимого регистра `eax`, на деле я вижу символ 'j'. Это происходит потому, что символ '6' имеет двоичный код 00110110 (или 54 в десятичной системе), а символ '4' - двоичный код 00110100 (или 52 в десятичной системе). Поэтому, когда я выполняю сложение с помощью команды `add eax, ebx`, в результате получается 01101010 (или 106 в десятичной системе), что соответствует символу 'j'. (рис. [2.3])

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

Рис. 2.3: Запуск программы lab6-1.asm

Далее изменяю текст программы и вместо символов, запишем в регистры числа. (рис. [2.4])



```
%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. 2.4: Редактирую файл lab6-1.asm


Как и в прошлый раз, когда я запустила программу, ожидаемое число 10 не появилось на экране. Вместо этого был выведен символ с кодом 10, который является символом конца строки или возвратом каретки (рис. [2.5]). Этот символ не виден в консоли, но он создает пустую строку между строками текста.



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

Рис. 2.5: Запуск программы lab6-1.asm

Как я уже упоминала ранее, в файле in_out.asm были написаны специальные подпрограммы, чтобы можно было работать с числами, преобразовывая ASCII символы в числа и наоборот. Я внесла изменения в текст программы, чтобы использовать эти функции. (рис. [2.6])

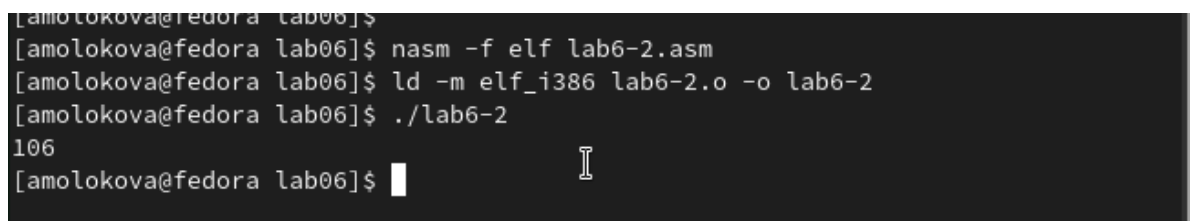


```
Открыть ▾ + lab6-2.asm
~/work/lab06

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

Рис. 2.6: Редактирую файл lab6-2.asm

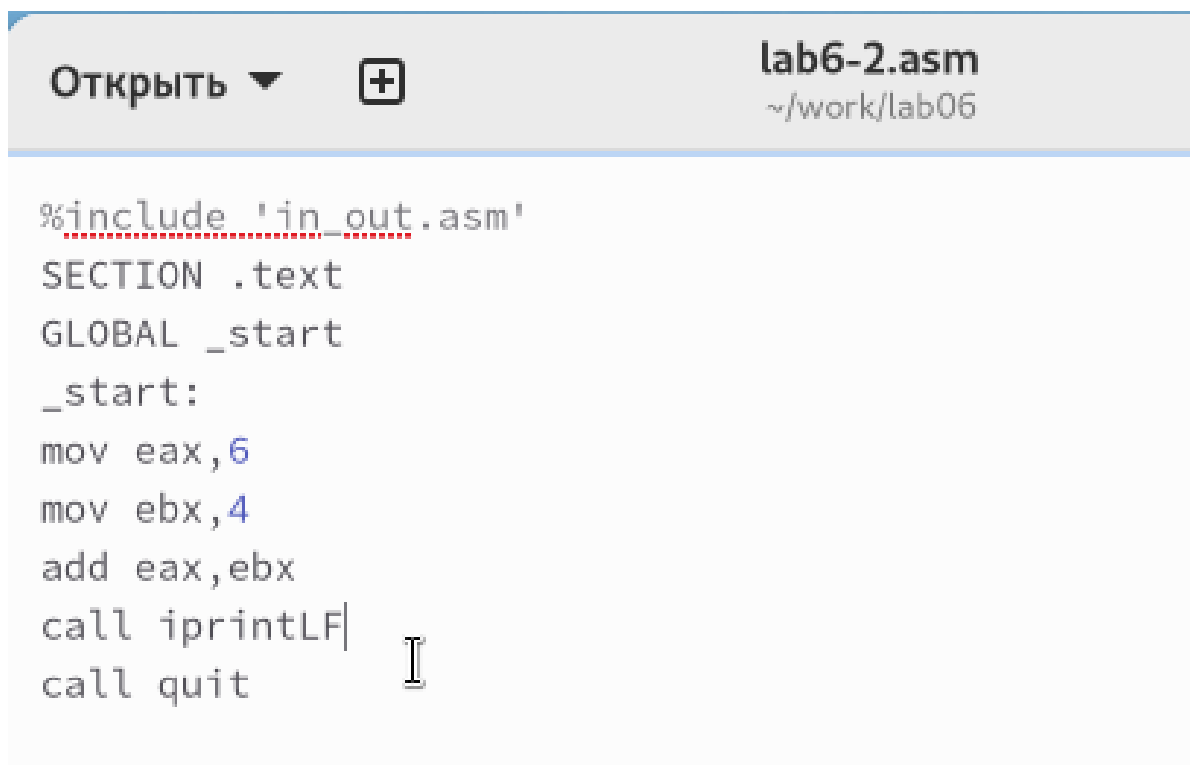
После запуска измененной программы на экране появилось число 106 (рис. [2.7]). В этом случае, как и в первом примере, команда `add` складывает коды символов '6' и '4' ($54+52=106$), но в отличие от предыдущей версии программы, функция `iprintLF` позволяет вывести на экран именно число, а не символ с соответствующим кодом.



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

Рис. 2.7: Запуск программы lab6-2.asm

Также я изменила символы на числа, как это было в предыдущем примере. (рис. [2.8])

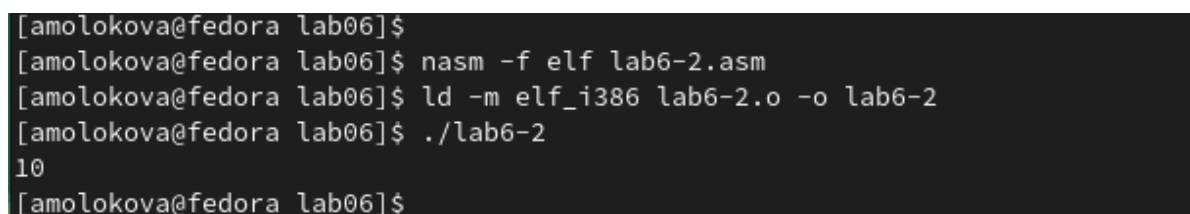


```
Открыть ▼  +  lab6-2.asm
~/work/lab06

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

Рис. 2.8: Редактирую файл lab6-2.asm

Благодаря функции `iprintLF`, которая выводит число, и тому, что в качестве операндов использовались числа, а не коды символов, на экране появилось число 10. (рис. [2.9])



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

Рис. 2.9: Запуск программы lab6-2.asm

Затем я заменила функцию `iprintLF` на `iprint`, собрала исполняемый файл и запустила его. В этот раз результат отличался отсутствием переноса строки после выводимого числа. (рис. [2.10])

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


Рис. 2.10: Запуск программы lab6-2.asm

2.2 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения (рис. [2.11]) (рис. [2.12])

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

.

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. 2.11: Редактирую файл lab6-3.asm

```

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

```

Рис. 2.12: Запуск программы lab6-3.asm

Изменила текст программы для вычисления выражения

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

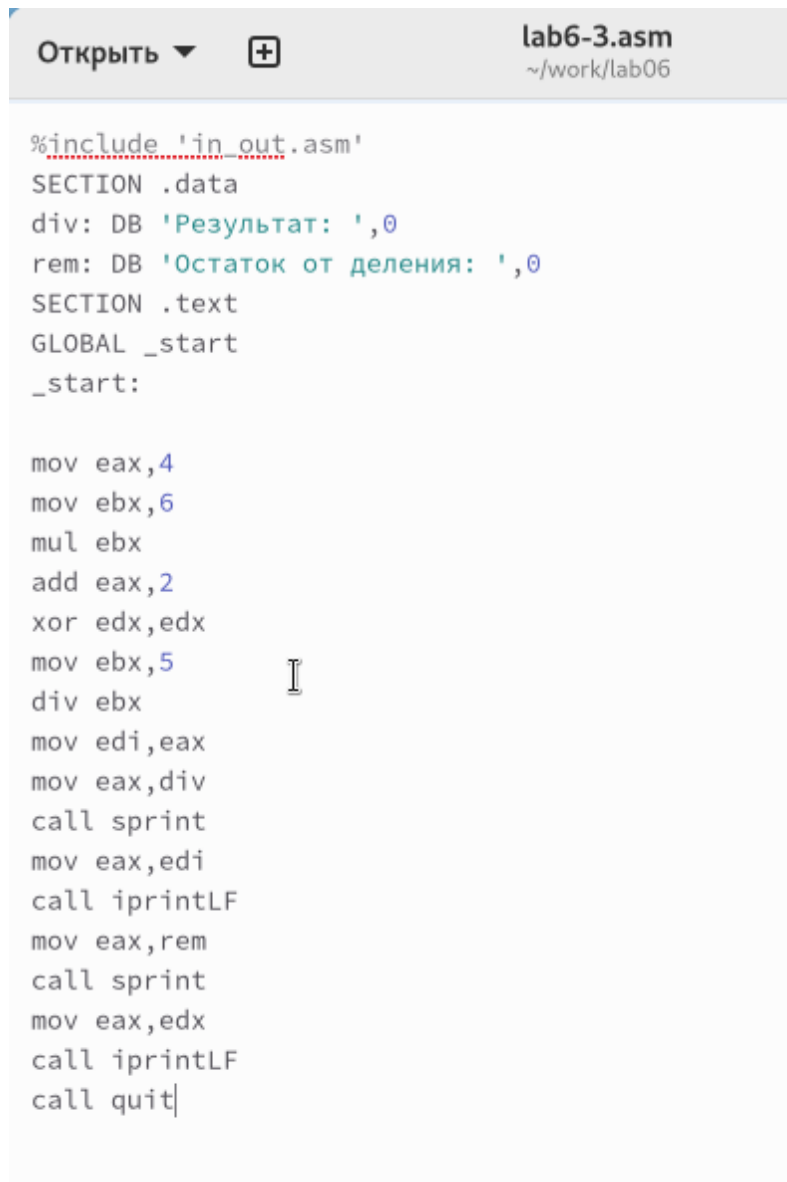
. Создала исполняемый файл и проверил его работу. (рис. [2.13]) (рис. [2.14])

```

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

```

Рис. 2.13: Редактирую файл lab6-3.asm



```
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:

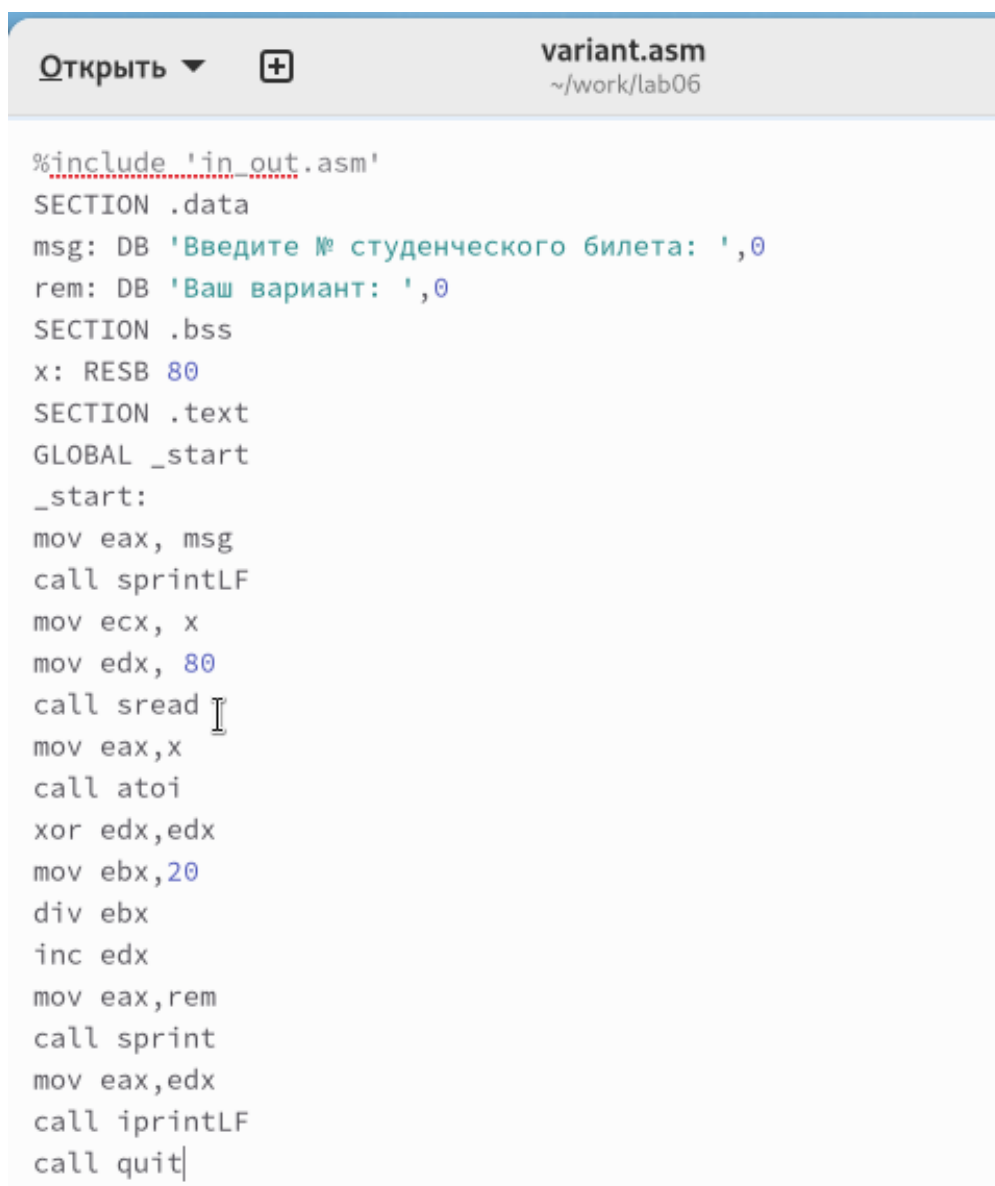
mov eax,4
mov ebx,6
mul ebx
add eax,2
xor edx,edx
mov ebx,5
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 2.14: Запуск программы lab6-3.asm

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета. (рис. [2.15]) (рис. [2.16])

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может

быть использована функция `atoi` из файла `in_out.asm`.



```
%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: Редактирую файл `variant.asm`


```

[amolokova@fedora lab06]$
[amolokova@fedora lab06]$ nasm -f elf variant.asm
[amolokova@fedora lab06]$ ld -m elf_i386 variant.o -o variant
[amolokova@fedora lab06]$ ./variant
Введите № студенческого билета:
1032235470
Ваш вариант: 11
[amolokova@fedora lab06]$

```

Рис. 2.16: Запуск программы variant.asm

2.3 Ответы на вопросы

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

Команда “mov eax, rem” загружает в регистр значение, соответствующее строке “Ваш вариант:”.

Команда “call sprint” инициирует процедуру, которая выводит строку на экран.

2. Для чего используются следующие инструкции?

Инструкция “mov ecx, x” копирует значение из переменной x в регистр ecx.

Инструкция “mov edx, 80” помещает число 80 в регистр edx.

Инструкция “call sread” активирует функцию чтения данных студенческого билета через консоль.

3. Для чего используется инструкция “call atoi”?

Команда “call atoi” преобразует строковые символы в целочисленное значение.

4. Какие строки листинга отвечают за вычисления варианта?

Команда “xor edx, edx” очищает содержимое регистра edx.

Команда “mov ebx, 20” помещает число 20 в регистр ebx.

Команда “div ebx” делит число студенческого билета на 20.

Команда “inc edx” увеличивает содержимое регистра edx на единицу.

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

Результат остатка от деления помещается в регистр edx.

6. Для чего используется инструкция “inc edx”?

Команда “inc edx” прибавляет единицу к содержимому регистра edx, что необходимо для расчета номера варианта.

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

Команда “mov eax, edx” переносит результат расчетов в регистр eax.

Команда “call iprintLF” запускает подпрограмму, которая выводит результат на экран.

2.4 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.

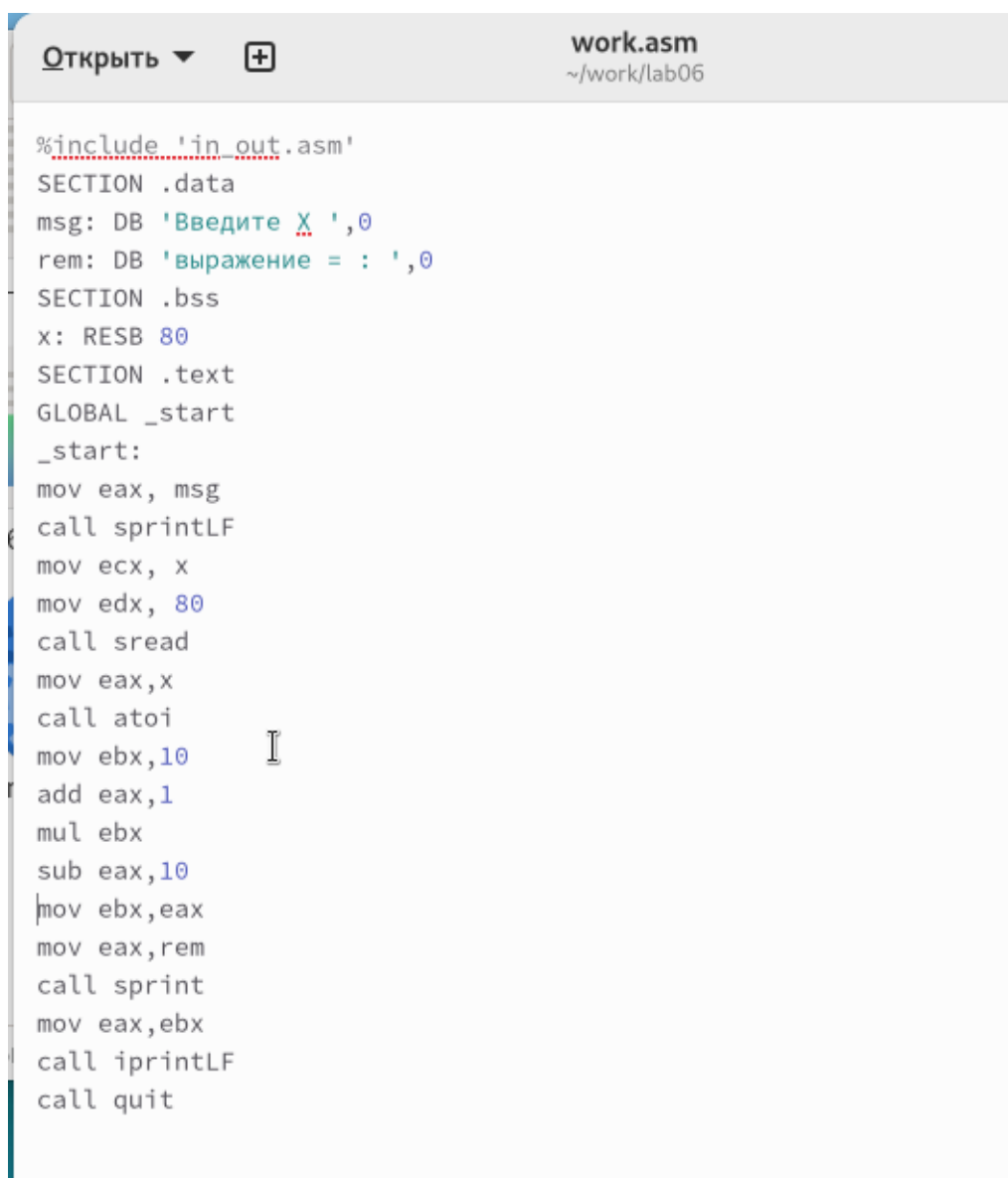
Получили вариант 11 -

$$10 * (x + 1) - 10$$

для

$$x = 1, x = 7$$

(рис. [2.17]) (рис. [2.18])



```
%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
mov ebx, 10
add eax, 1
mul ebx
sub eax, 10
mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.17: Редактирую файл work.asm

```
[amolokova@fedora lab06]$ nasm -f elf work.asm
[amolokova@fedora lab06]$ ld -m elf_i386 work.o -o work
[amolokova@fedora lab06]$ ./work
Введите X
1
выражение = : 10
[amolokova@fedora lab06]$ ./work
Введите X
7
выражение = : 70
[amolokova@fedora lab06]$
```

Рис. 2.18: Запуск программы task.asm

3 Выводы

Изучили работу с арифметическими операциями.