

Отчёт по лабораторной работе 6

Архитектура компьютера

Гуламова Е.М. НПИбд-03-23

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	21

Список иллюстраций

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

Список таблиц

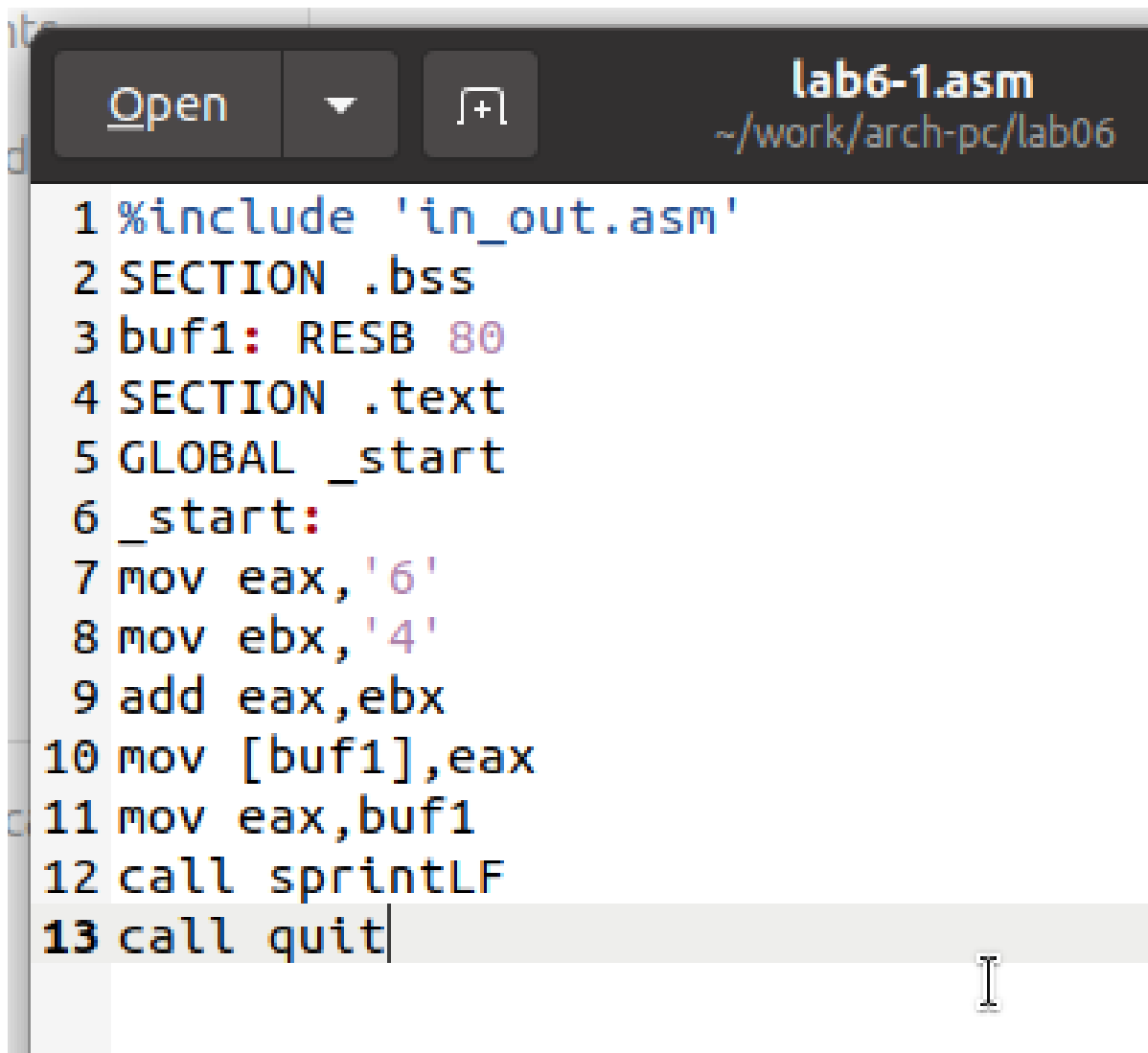
1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Выполнение лабораторной работы

1. Я создала папку для программ лабораторной работы номер шесть, затем перешла в неё и сформировала файл с именем lab6-1.asm.
2. Давайте рассмотрим примеры программ, которые отображают символы и числовые данные. Эти программы будут выводить информацию, которая была помещена в регистр `eax`.

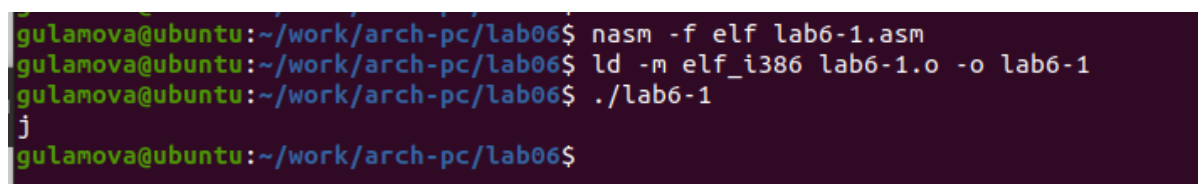
В одной из программ в регистр `eax` мы помещаем символ '6' (`mov eax, '6'`), а в регистр `ebx` символ '4' (`mov ebx, '4'`). После этого мы складываем значения, хранящиеся в регистрах `eax` и `ebx` (`add eax, ebx`, и результат сложения сохранится в `eax`). Затем мы выводим полученный результат на экран. Однако, поскольку функция `sprintf` требует, чтобы в регистре `eax` находился адрес, нам нужно воспользоваться дополнительной переменной. Сначала мы переносим значение из регистра `eax` в переменную `buf1` (`mov [buf1], eax`), а потом записываем адрес переменной `buf1` обратно в регистр `eax` (`mov eax, buf1`) и вызываем функцию `sprintf`.



```
lab6-1.asm
~/work/arch-pc/lab06

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 sprintf
13 call quit
```

Рис. 2.1: Программа в файле lab6-1.asm



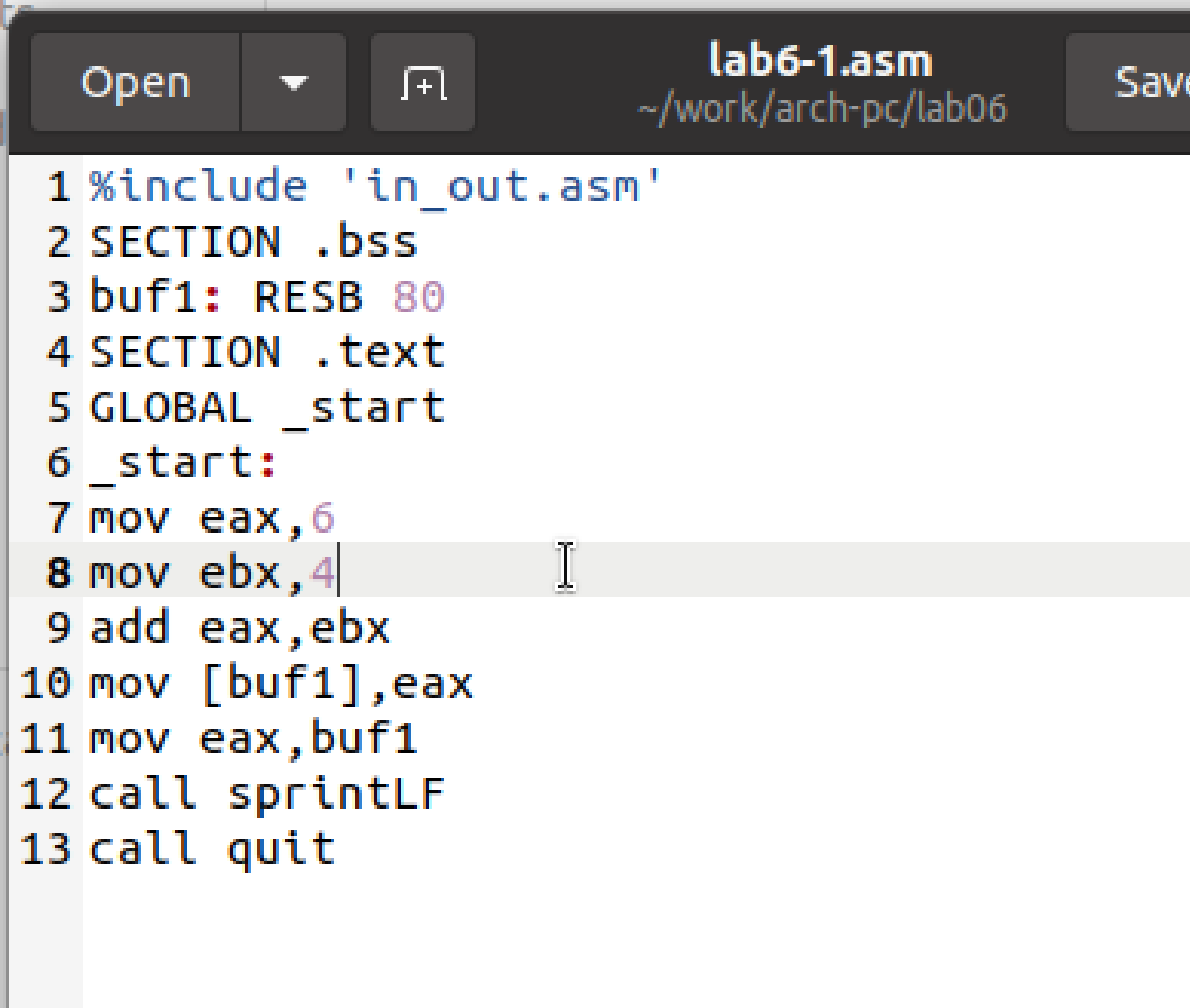
```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-1.o -o lab6-1
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-1
j
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.2: Запуск программы lab6-1.asm

Когда я смотрю на значение в регистре `eax`, я ожидаю увидеть цифру 10. Но вместо этого там отображается символ 'j'. Это происходит из-за того, что дво-

ичный код символа '6' равен 00110110, что соответствует числу 54, а двоичный код символа '4' – 00110100, или 52. Когда я использую команду `add eax, ebx`, то в регистр `eax` записывается их сумма – 01101010, что в десятичной системе равно 106, и это код для символа 'j'.

3. Затем я внесла изменения в программу, чтобы в регистры записывались числа, а не символы.



```
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 sprintf
13 call quit
```

Рис. 2.3: Программа в файле `lab6-1.asm`

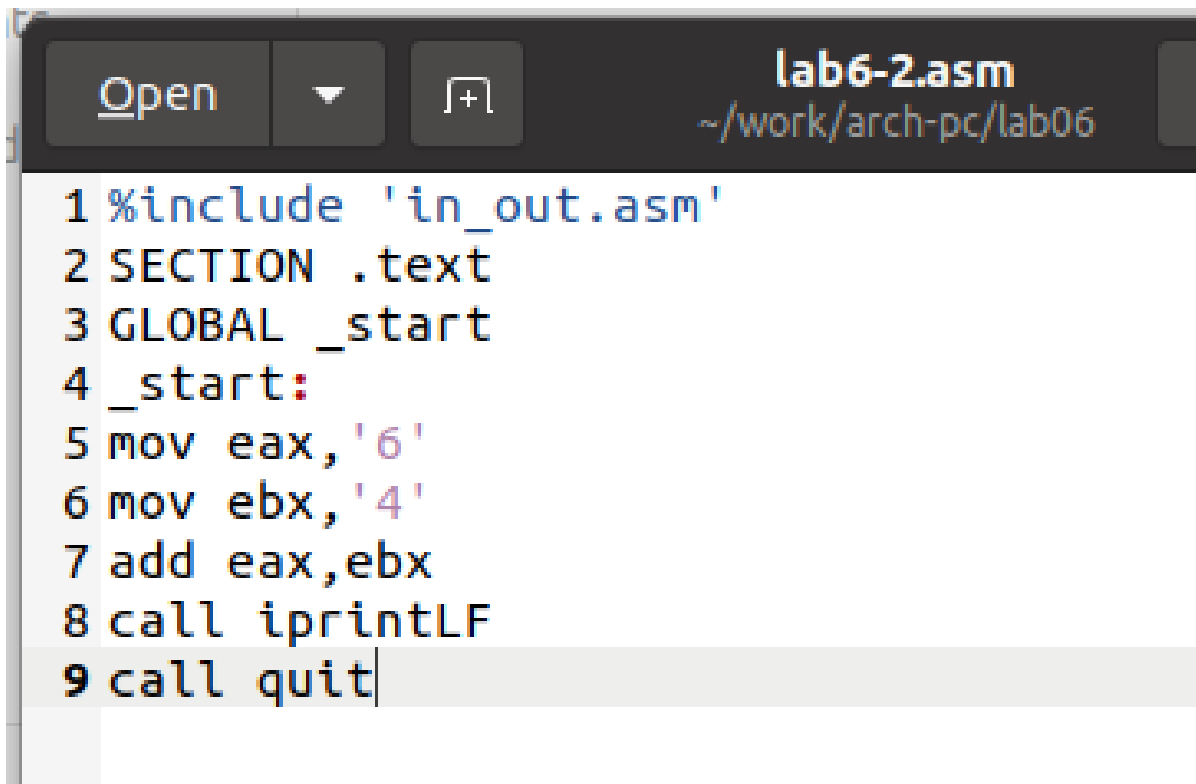

```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-1.o -o lab6-1
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-1
j
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-1.o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

gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.4: Запуск программы lab6-1.asm

Но даже после этих изменений, когда программа выполняется, она не показывает число 10. В этот раз выводится символ с кодом 10, который является символом конца строки. В консоли он не виден, но создает пустую строку.

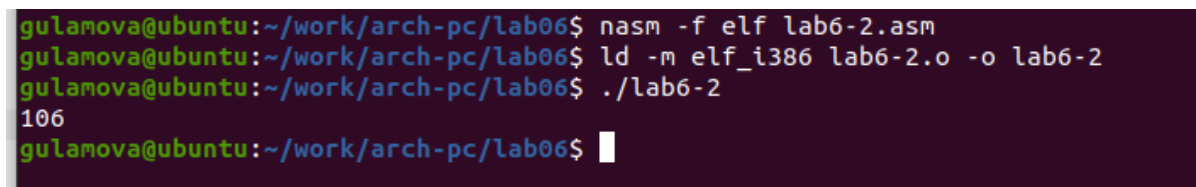
4. Как я уже упоминала, для работы с числами в файле `in_out.asm` были реализованы специальные подпрограммы, которые позволяют преобразовывать ASCII символы в числа и наоборот. Я использовала эти функции, чтобы изменить текст программы.



```
lab6-2.asm
~/work/arch-pc/lab06

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. 2.5: Программа в файле lab6-2.asm

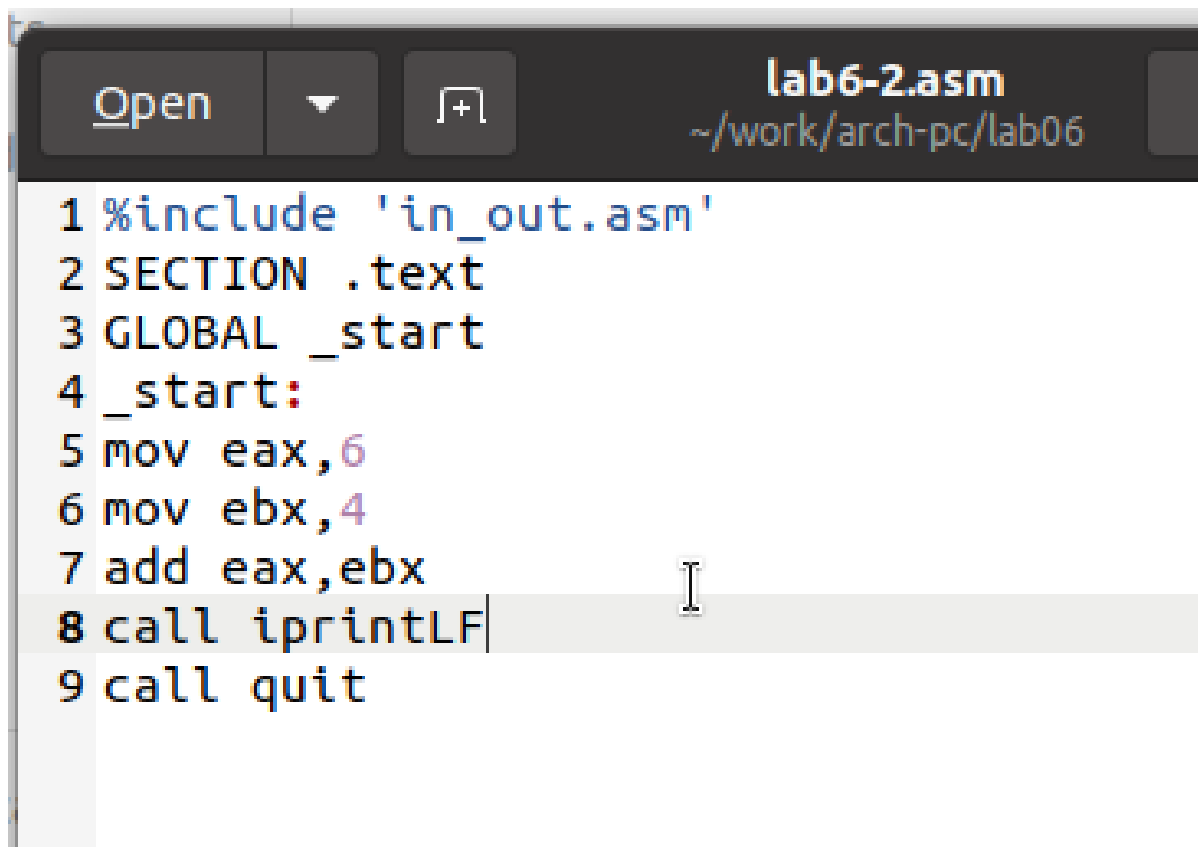


```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
106
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.6: Запуск программы lab6-2.asm

Когда я запустила программу, она выдала мне число 106. Так же, как в первом случае, здесь функция add суммирует коды символов '6' и '4', что в сумме даёт $54+52=106$. Но в отличие от предыдущей программы, здесь используется функция iprintLF, которая позволяет выводить число, а не символ, соответствующий этому числовому коду.

5. Подобно предыдущему примеру, я заменила символы на числа.

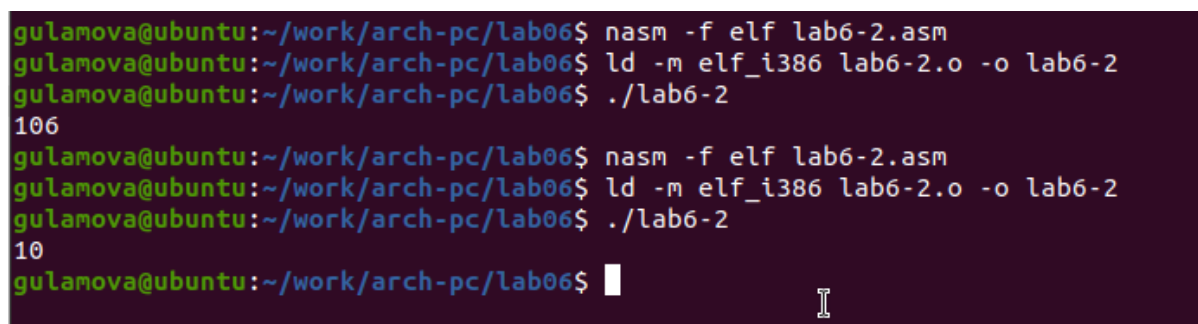


```
lab6-2.asm
~/work/arch-pc/lab06

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. 2.7: Программа в файле lab6-2.asm

Благодаря функции `iprintLF`, которая выводит числа, и тому, что в качестве операндов были использованы именно числа, а не коды символов, в результате получилось число 10.



```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
106
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
10
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.8: Запуск программы lab6-2.asm

Я изменила функцию `iprintLF` на `iprint`, собрала исполняемый файл и запусти-

ла его. Отличие заключалось в том, что теперь вывод не сопровождался переносом строки.

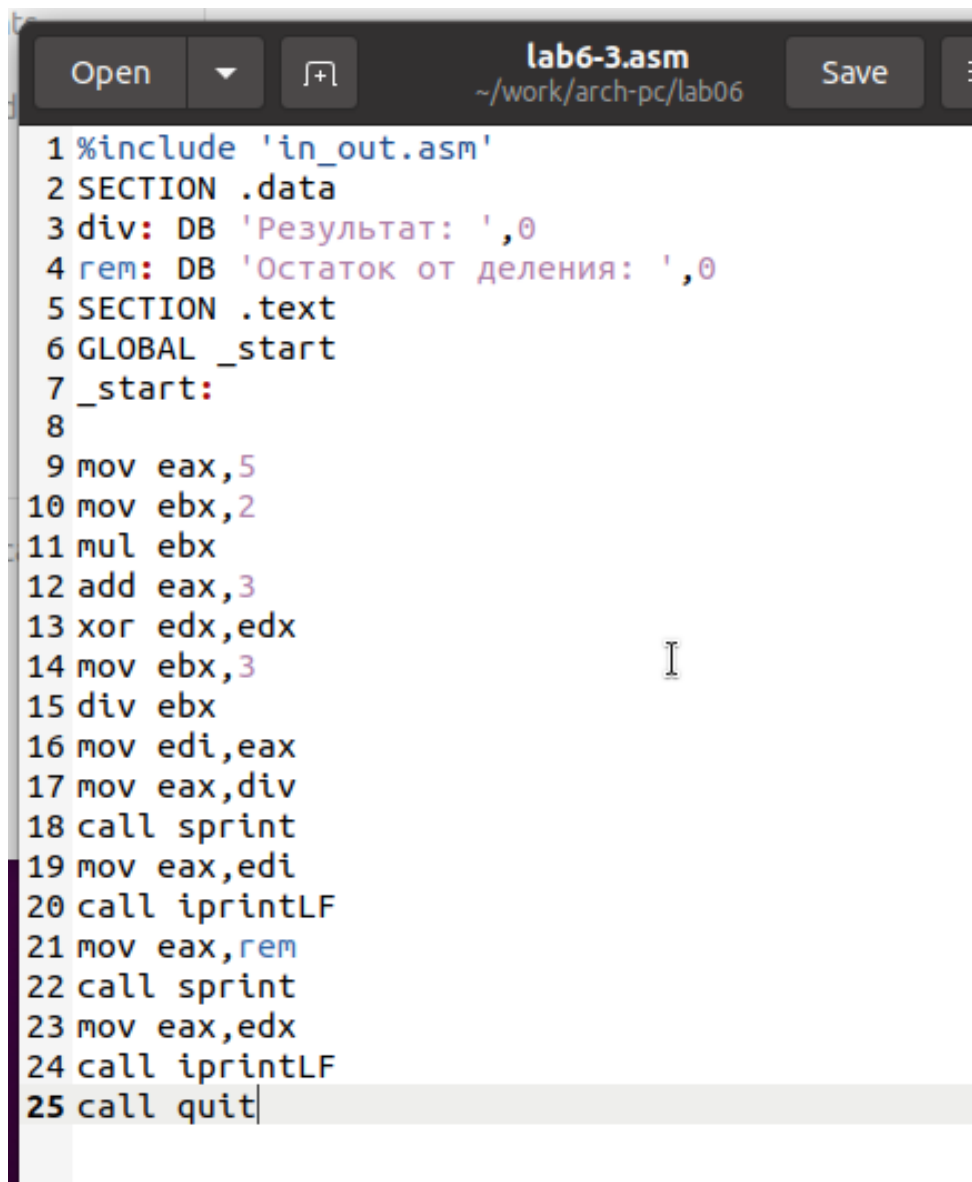
```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
106
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
10
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-2.o -o lab6-2
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-2
10gulamova@ubuntu:~/work/arch-pc/lab06$
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.9: Запуск программы lab6-2.asm

6. В качестве примера, демонстрирующего выполнение арифметических операций в NASM, я написала программу для вычисления арифметического выражения

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

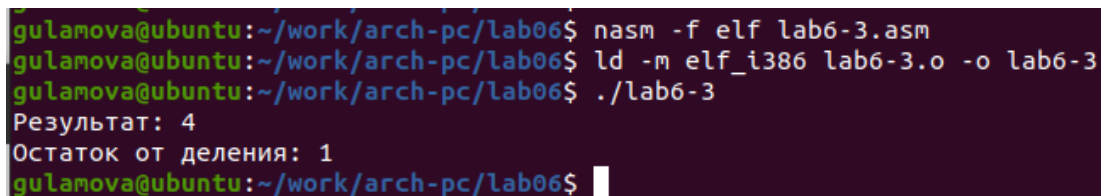
.



```
lab6-3.asm
~/work/arch-pc/lab06

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. 2.10: Программа в файле lab6-3.asm



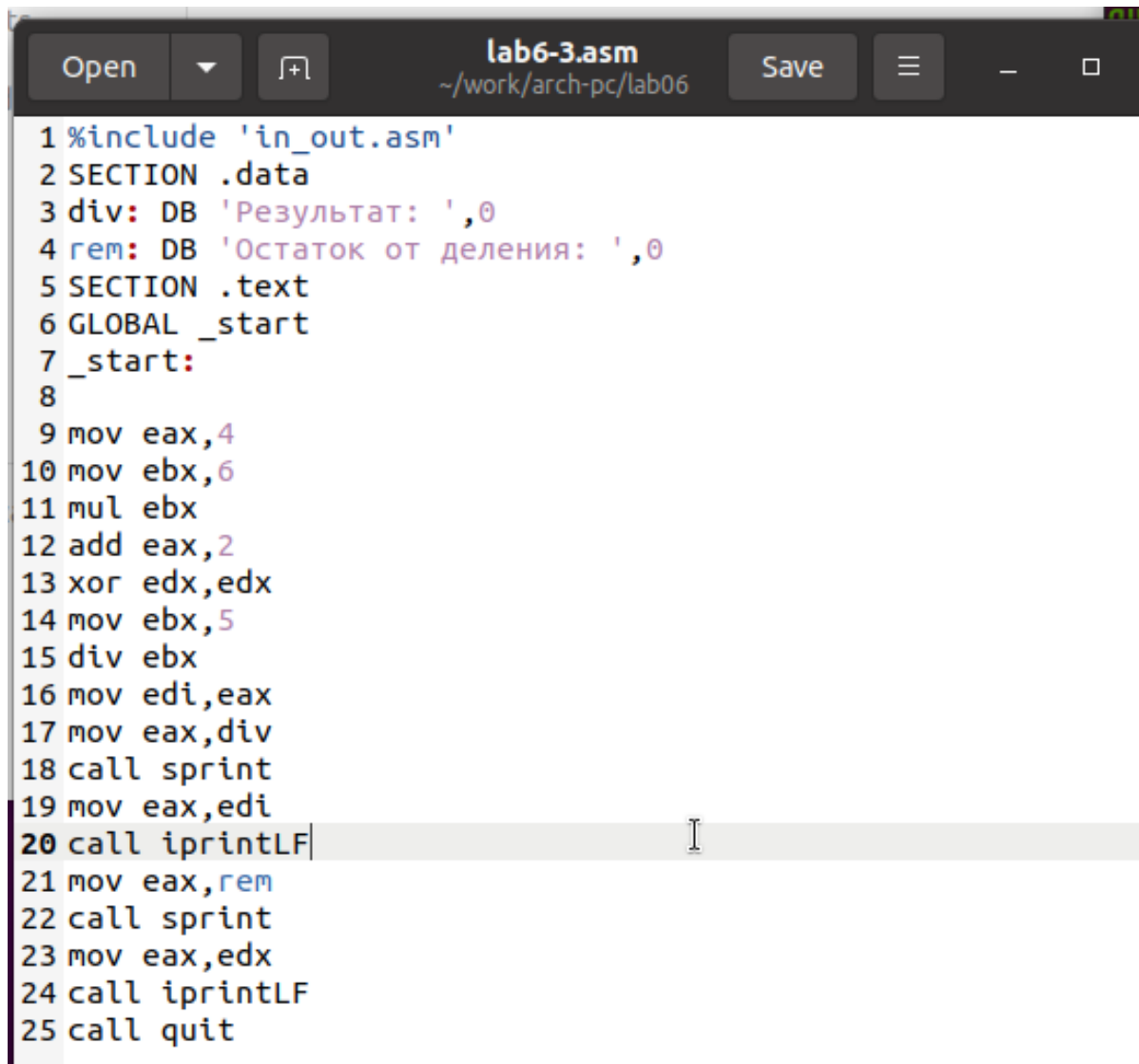
```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.11: Запуск программы lab6-3.asm

Затем я изменила код программы, чтобы она вычисляла выражение

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

. После создания исполняемого файла я проверила, как он работает.



```
lab6-3.asm
~/work/arch-pc/lab06

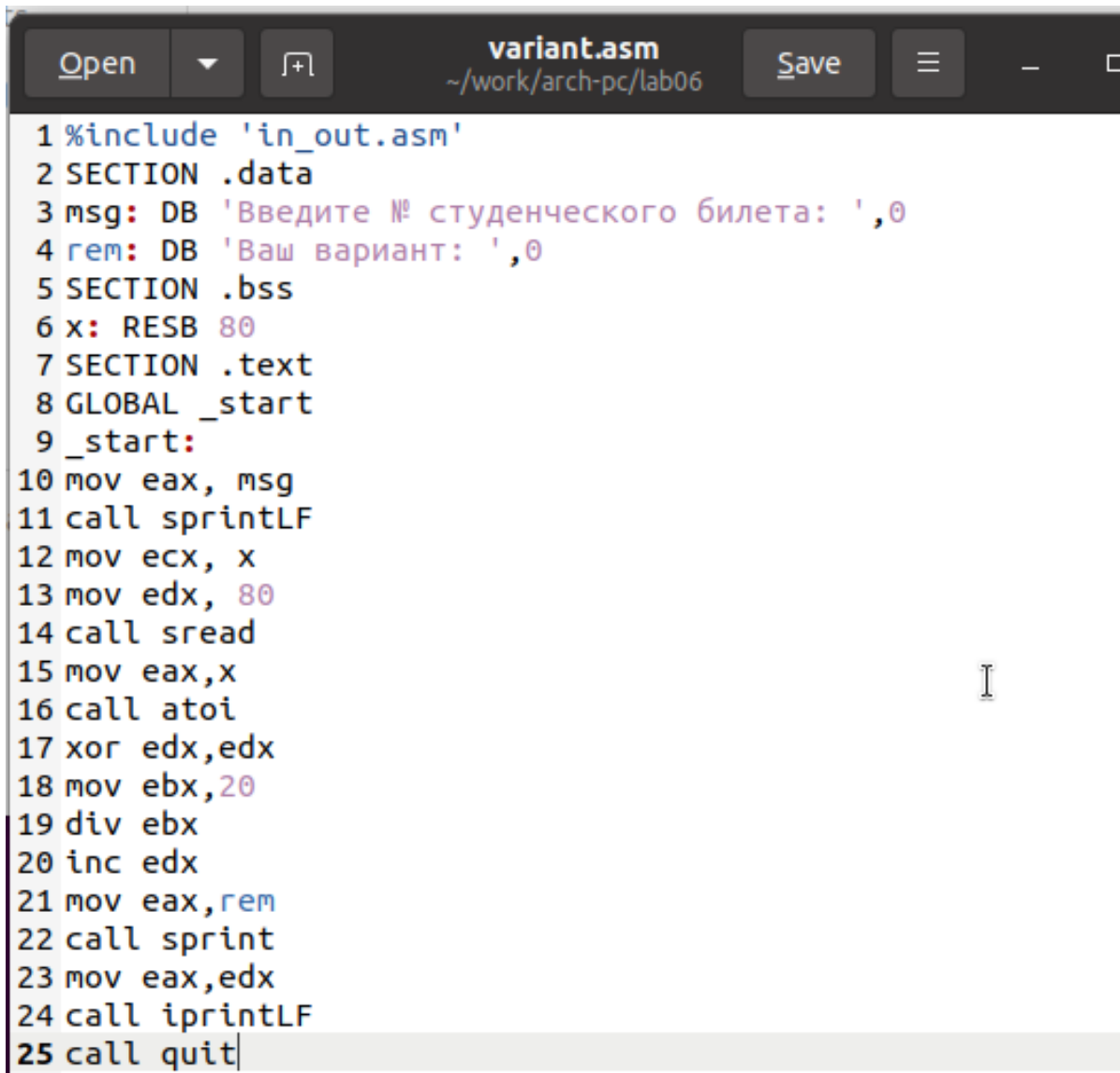
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5 SECTION .text
6 GLOBAL _start
7 _start:
8
9 mov eax,4
10 mov ebx,6
11 mul ebx
12 add eax,2
13 xor edx,edx
14 mov ebx,5
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call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

Рис. 2.12: Программа в файле lab6-3.asm

```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
gulamova@ubuntu:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
gulamova@ubuntu:~/work/arch-pc/lab06$
```

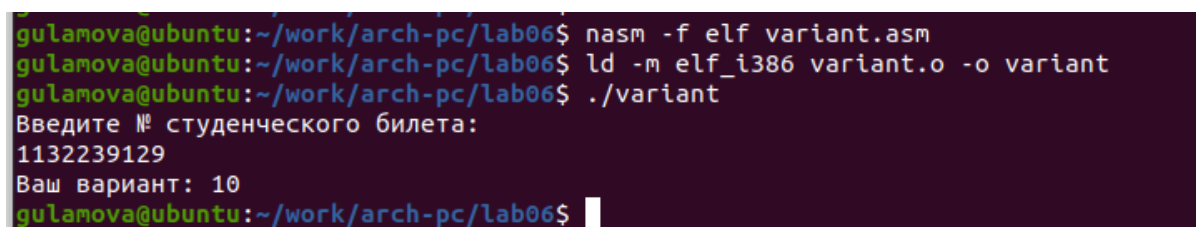
Рис. 2.13: Запуск программы lab6-3.asm

7. Давайте возьмем для примера задачу, где нужно вычислить вариант упражнения на основе номера студенческого билета. Здесь нам придется работать с числом, которое мы введем через клавиатуру. Как я уже упоминала ранее, вводимые данные поступают в виде символов, и чтобы выполнять с ними математические операции в NASM, их нужно преобразовать в числовой формат. В этом может помочь функция `atoi`, которую можно найти в файле `in_out.asm`.



```
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 sprintf
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. 2.14: Программа в файле variant.asm



```
gulamova@ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant.asm
gulamova@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
gulamova@ubuntu:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132239129
Ваш вариант: 10
gulamova@ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.15: Запуск программы variant.asm

ответы на вопросы

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

- Команда `mov eax, ret` загружает в регистр значение, соответствующее строке “Ваш вариант:”.
- Использование `call sprint` приводит к выполнению функции, отображающей строку.

2. Для чего используются следующие инструкции?

```
mov ecx, x
mov edx, 80
call sread
```

Они используются для ввода номера студенческого билета и его сохранения в переменной X через терминал.

3. Для чего используется инструкция “call atoi”?

Данная функция преобразует введенные пользователем символы в числовое значение.

4. Какие строки листинга отвечают за вычисления варианта?

```
xor edx,edx
mov ebx,20
div ebx
inc edx
```

Эти команды выполняют операцию деления номера студенческого билета на 20 и увеличивают остаток от деления на единицу.

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

Остаток от деления помещается в регистр edx.

6. Для чего используется инструкция “inc edx”?

Эта команда увеличивает значение в регистре edx на единицу, что необходимо для расчёта номера варианта по заданной формуле.

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

- mov eax, edx – перемещает результат вычислений в регистр eax.
- call iprintLF – иницирует функцию, которая выводит результат на экран с переводом строки.

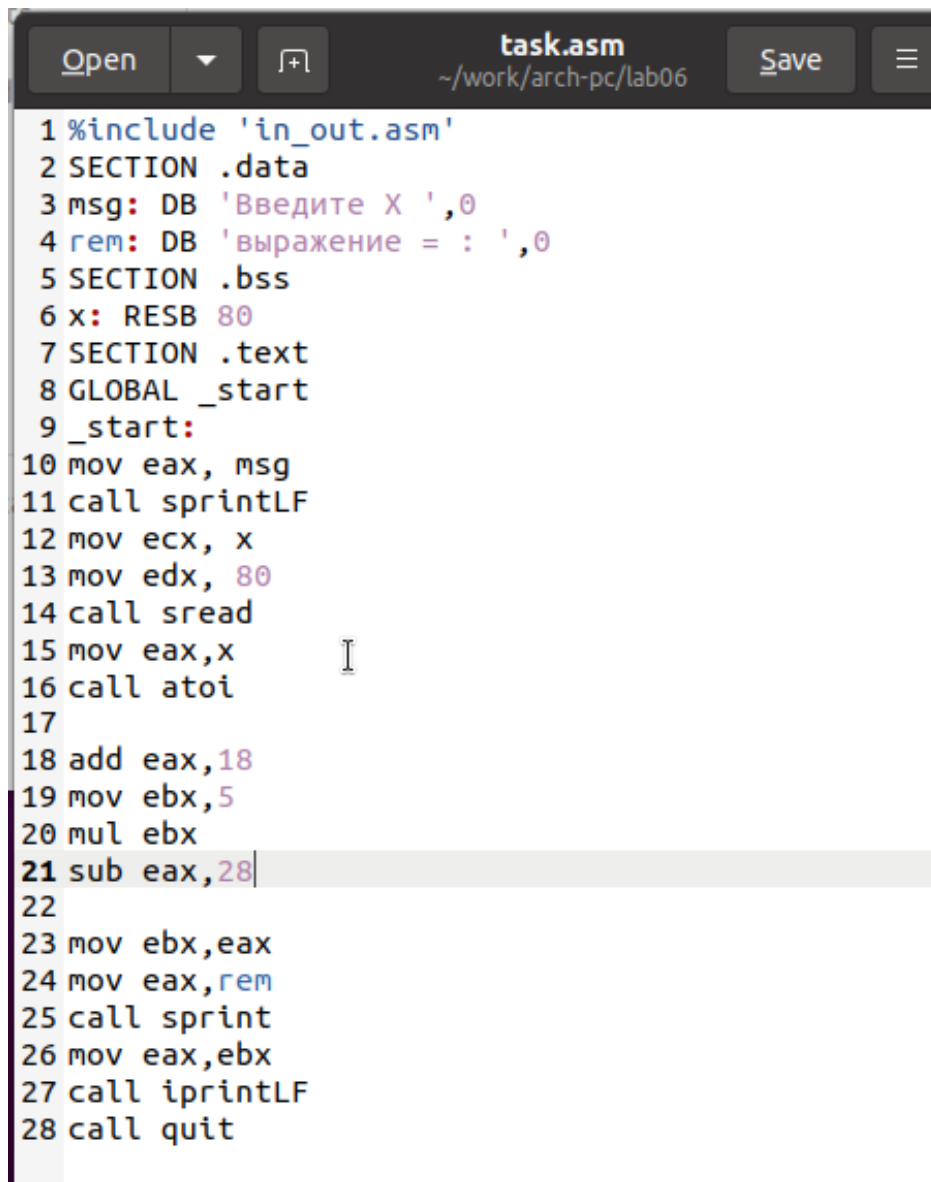
8. Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x , вычислять заданное выражение в зависимости от введенного x , выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы 6.3 вариантов заданий в соответствии 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 10 -

$$5(x + 18) - 28$$

для

$$x_1 = 2, x_2 = 3$$



```
task.asm
~/work/arch-pc/lab06

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. 2.16: Программа в файле task.asm

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

Рис. 2.17: Запуск программы task.asm

3 Выводы

Изучили работу с арифметическими операциями.