

Отчёт по лабораторной работе 6

Архитектура компьютера

Хулер Александрович Оюн

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Задание для самостоятельной работы	20
3	Выводы	23

Список иллюстраций

2.1	Код программы lab6-1.asm	7
2.2	Компиляция и запуск программы lab6-1.asm	7
2.3	Код программы lab6-1.asm	9
2.4	Компиляция и запуск программы lab6-1.asm	10
2.5	Код программы lab6-2.asm	11
2.6	Компиляция и запуск программы lab6-2.asm	11
2.7	Код программы lab6-2.asm	12
2.8	Компиляция и запуск программы lab6-2.asm	13
2.9	Код программы lab6-2.asm	13
2.10	Компиляция и запуск программы lab6-2.asm	14
2.11	Код программы lab6-3.asm	15
2.12	Компиляция и запуск программы lab6-3.asm	15
2.13	Код программы lab6-3.asm	16
2.14	Компиляция и запуск программы lab6-3.asm	17
2.15	Код программы variant.asm	18
2.16	Компиляция и запуск программы variant.asm	18
2.17	Код программы program.asm	21
2.18	Компиляция и запуск программы program.asm	22

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

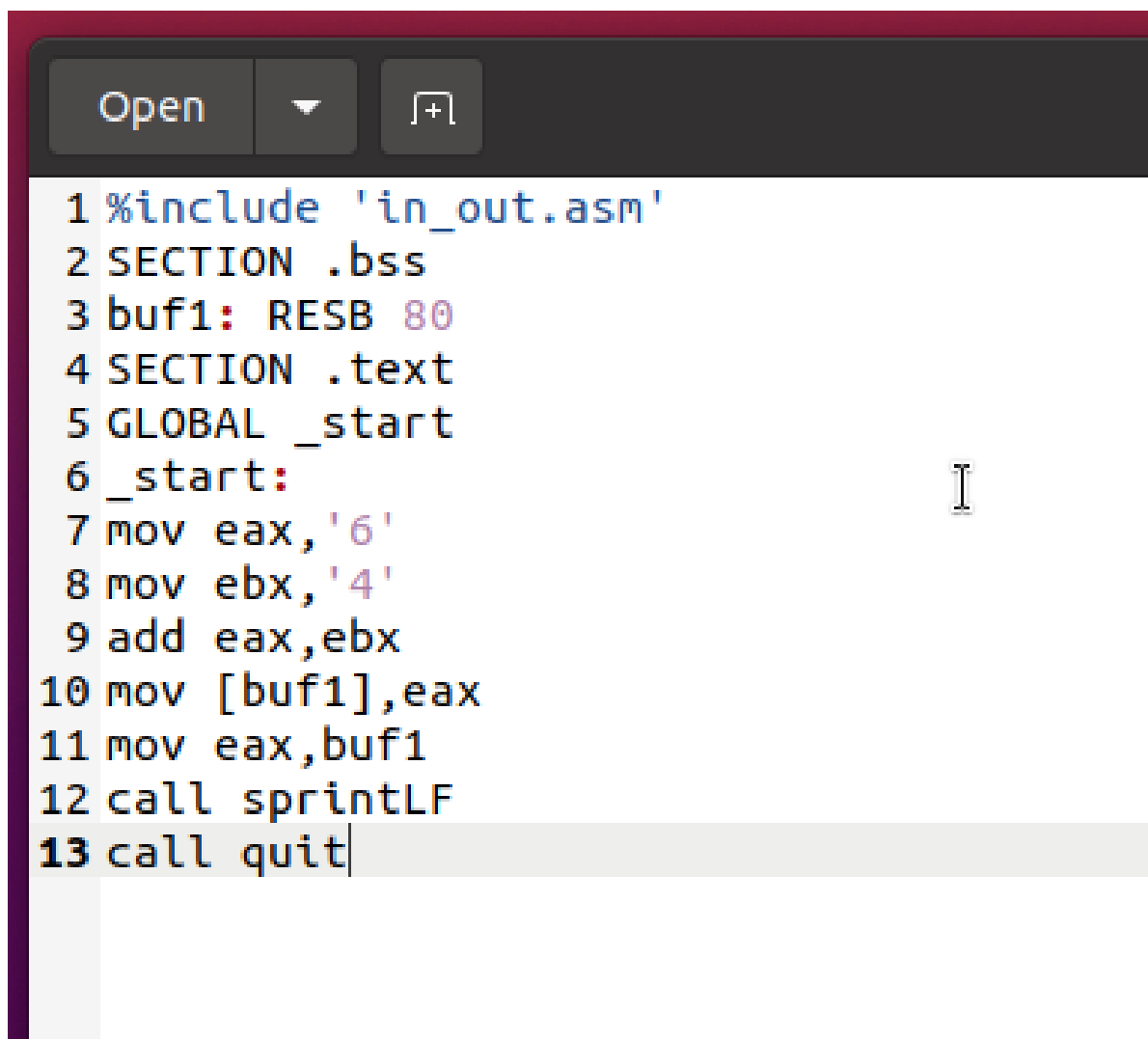
2 Выполнение лабораторной работы

Я создал папку для хранения файлов шестой лабораторной работы, перешел в нее и сформировал файл с исходным кодом lab6-1.asm.

В ходе этой лабораторной мы изучим примеры программ, демонстрирующих вывод символов и цифровых данных на экран. Эти программы будут оперировать данными, помещенными в регистр `eax`.

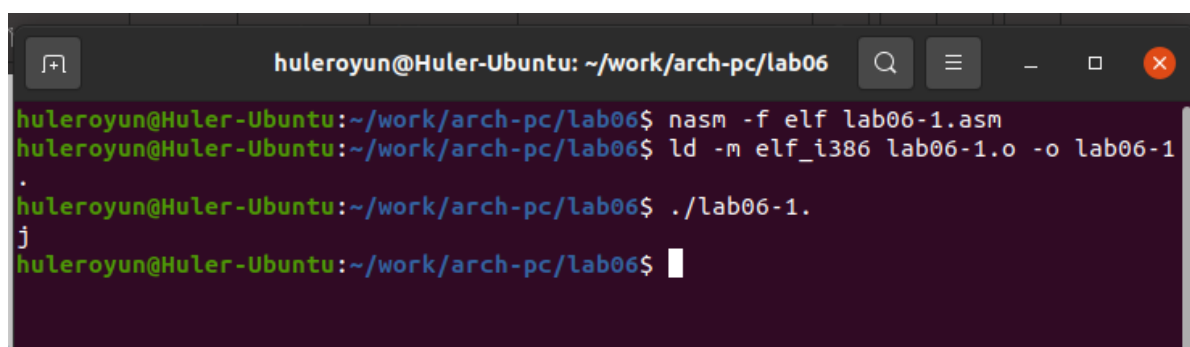
В одной из программ мы помещаем символ '6' в регистр `eax` с помощью инструкции (`mov eax, '6'`) и символ '4' в регистр `ebx` (`mov ebx, '4'`). После этого выполняем сложение значений, находящихся в регистрах `eax` и `ebx` (`add eax, ebx`), и отображаем полученный итог.

Чтобы воспользоваться функцией `sprintf`, которая ожидает адрес в регистре `eax`, мы применяем вспомогательную переменную. Сначала мы копируем содержимое регистра `eax` в переменную `buf1` (`mov [buf1], eax`), затем загружаем адрес `buf1` обратно в регистр `eax` (`mov eax, buf1`) и выполняем вызов функции `sprintf`.



```
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 _exit
```

Рис. 2.1: Код программы lab6-1.asm

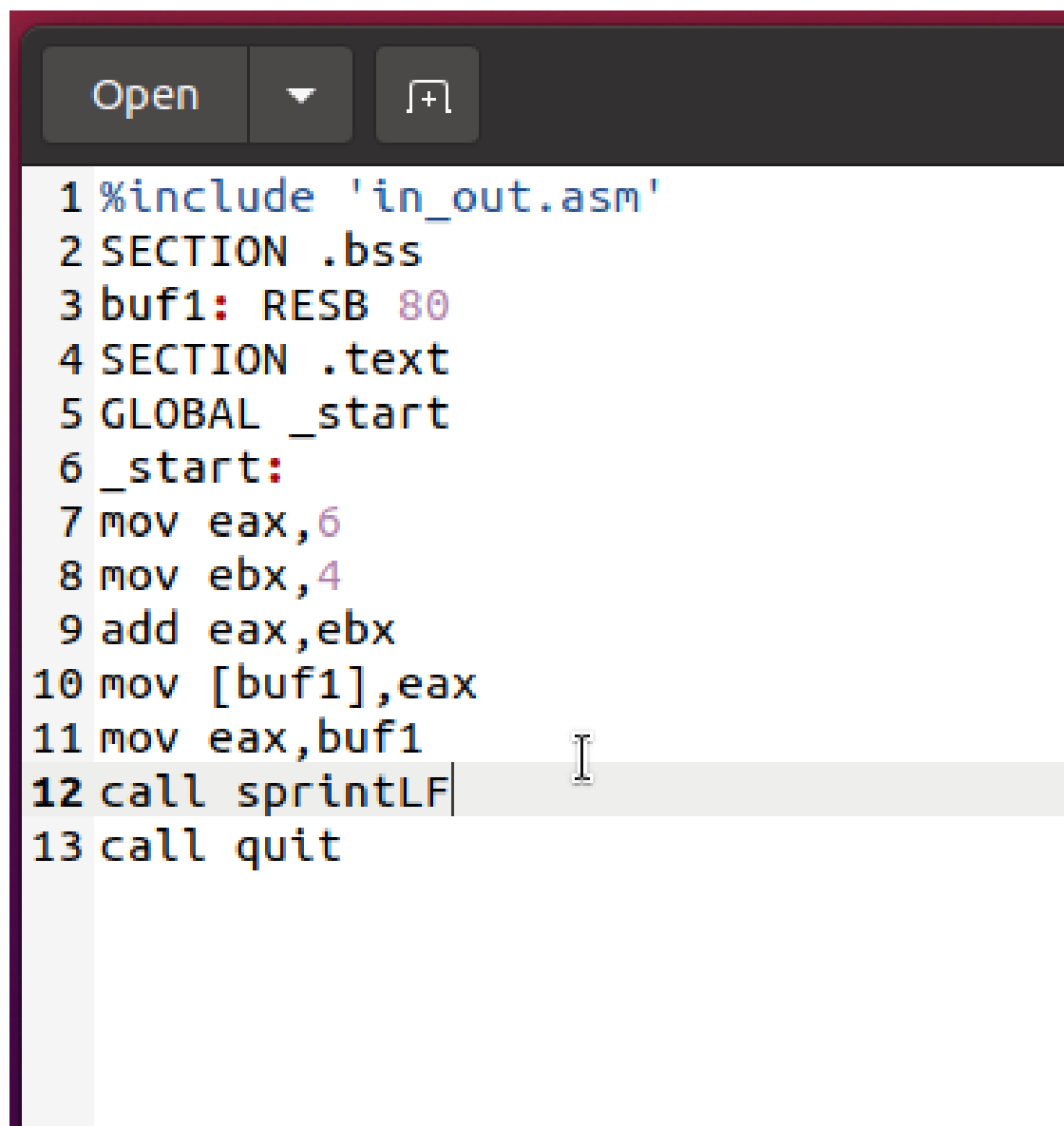


```
huleroyn@Huler-Ubuntu: ~/work/arch-pc/lab06
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ ./lab06-1.
j
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.2: Компиляция и запуск программы lab6-1.asm

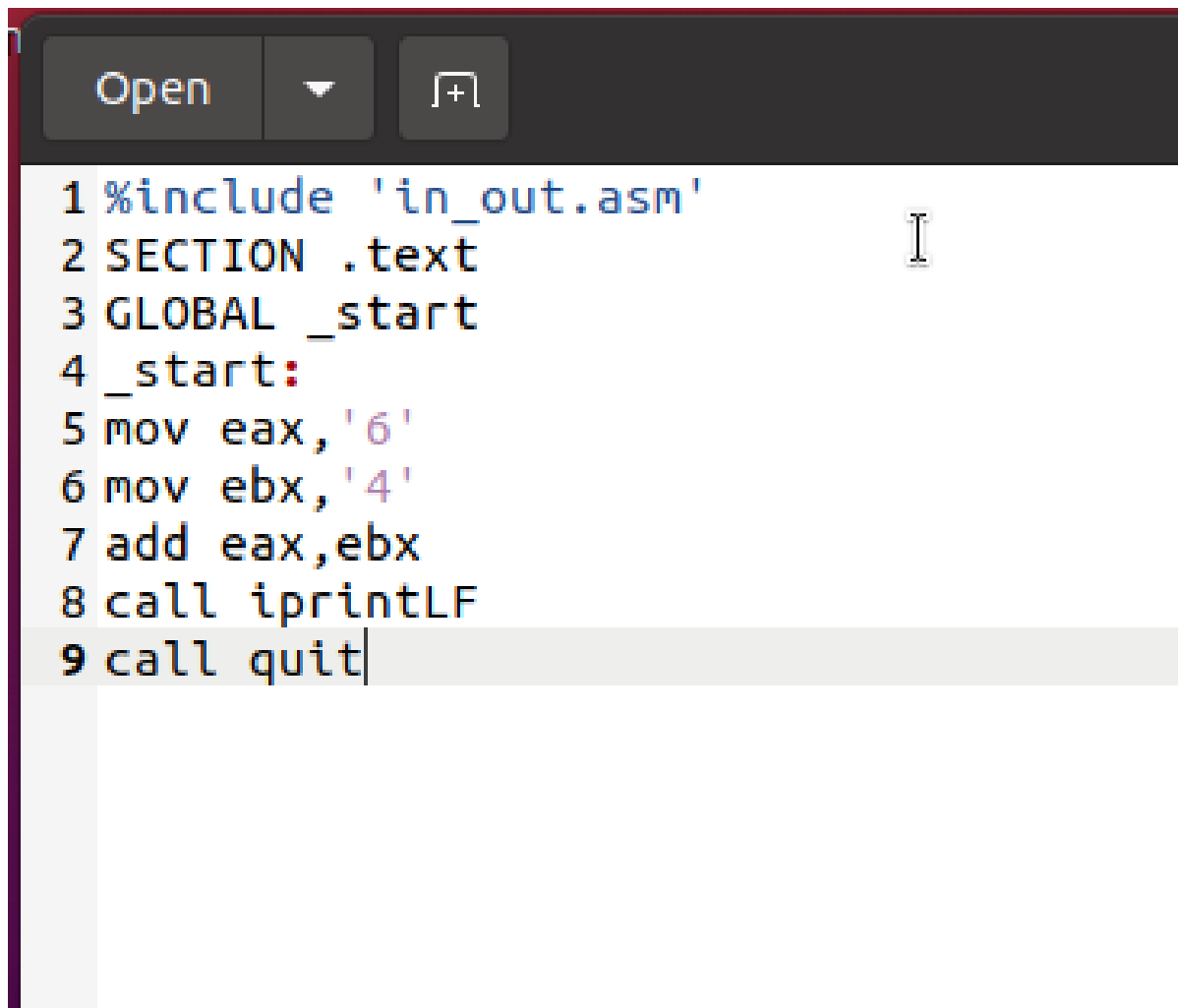
В этом примере мы ожидаем на выходе число 10 после сложения значений регистров `eax` и `ebx`. Тем не менее, на экране появится символ 'j', так как двоичный код символа '6' равен 00110110 (что соответствует 54 в десятичной системе), а символа '4' – 00110100 (или 52 в десятичной системе). Инструкция `add eax, ebx` приводит к записи в регистр `eax` суммы этих кодов – 01101010 (или 106 в десятичной системе), что соответствует коду символа 'j'.

После этого я внес изменения в код программы, заменив символы на числовые значения в регистрах.



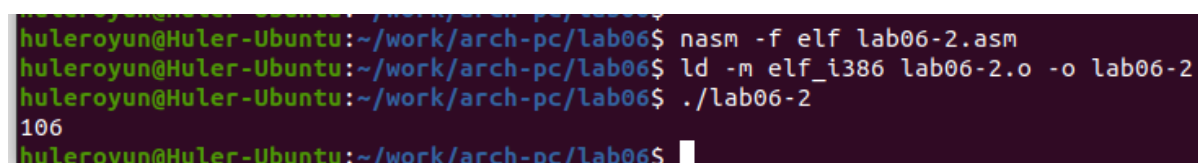
```
1 %include 'in_out.asm'
2 SECTION .bss
3 buf1: RESB 80
4 SECTION .text
5 GLOBAL _start
6 _start:
7 mov eax,6
8 mov ebx,4
9 add eax,ebx
10 mov [buf1],eax
11 mov eax,buf1
12 call sprintf|
13 call quit
```

Рис. 2.3: Код программы lab6-1.asm



```
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: Код программы lab6-2.asm



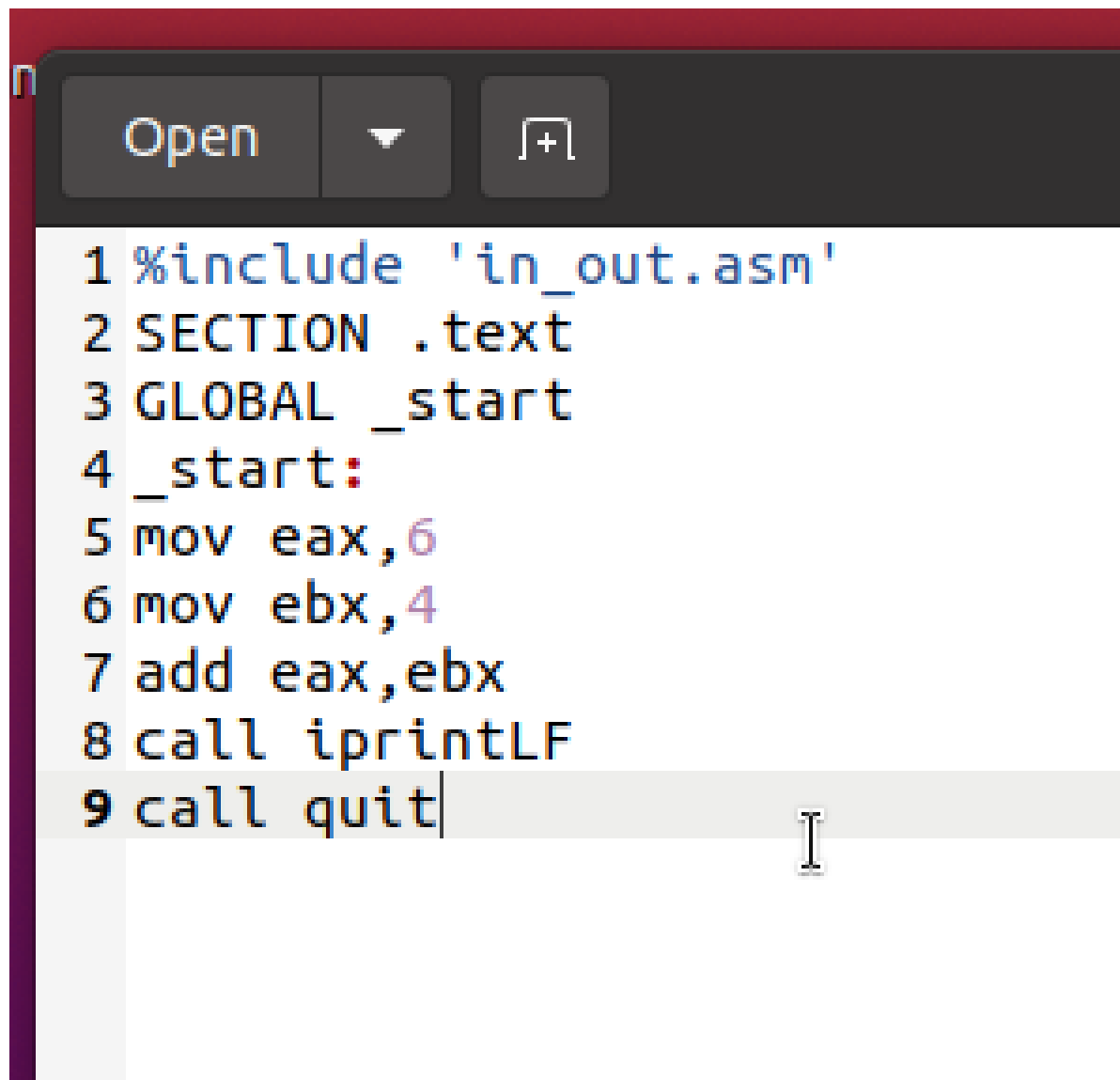
```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ ./lab06-2
106
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.6: Компиляция и запуск программы lab6-2.asm

После запуска программы на экран будет выведено число 106. В этом случае, подобно первому примеру, команда `add` суммирует численные значения символов '6' и '4' ($54+52=106$). Но в отличие от прошлой версии программы, функция

iprintLF позволяет отобразить именно число, а не символ с соответствующим числовым кодом.

Таким же образом, как и в предыдущем случае, мы провели замену символов на их числовые эквиваленты.



```
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: Код программы lab6-2.asm

Функция iprintLF предназначена для вывода числовых значений, и в данном контексте она работает с числами, а не с их символьными кодами. В результате мы видим на экране число 10.

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

Рис. 2.8: Компиляция и запуск программы lab6-2.asm

Функция `iprintLF` используется для отображения чисел, и в этой ситуации в качестве операндов выступают числа (не их символьные коды), что приводит к выводу числа 10.

Мы произвели замену функции `iprintLF` на `iprint`. Скомпилировали программу, создали исполняемый файл и запустили его. Разница в выводе заключается в отсутствии перевода строки.

```
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. 2.9: Код программы lab6-2.asm

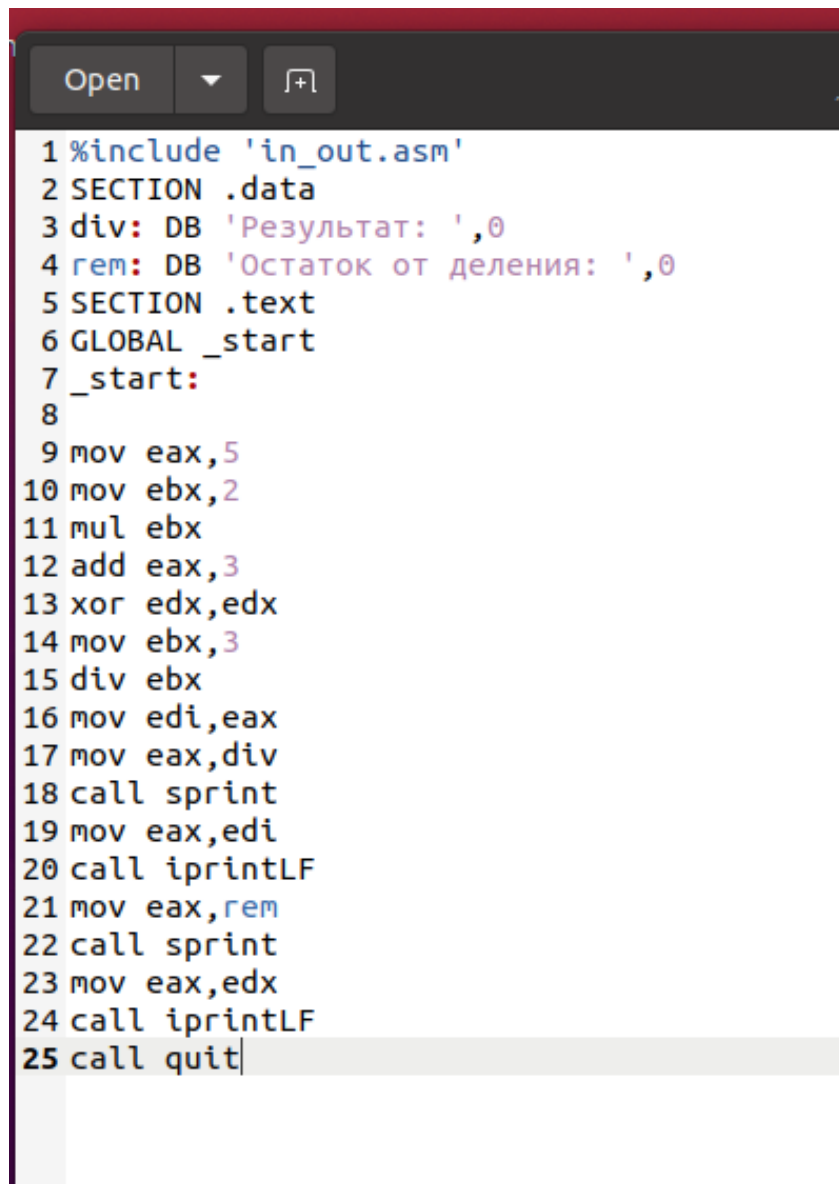
```
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$ ./lab06-2
10huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$
huleroyun@Huler-Ubuntu:~/work/arch-pc/lab06$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

Рис. 2.10: Компиляция и запуск программы lab6-2.asm

В рамках изучения выполнения арифметических действий в NASM была рассмотрена программа для расчета арифметической функции

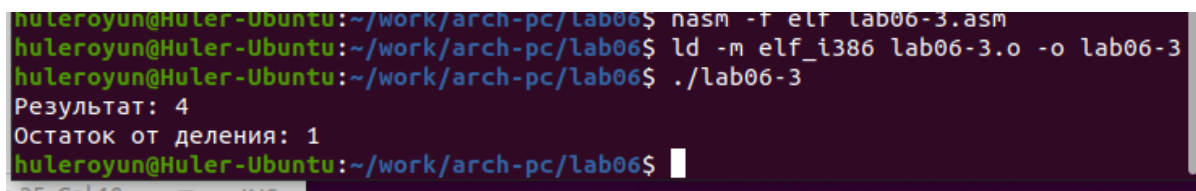
$$f(x) = (5 * 2 + 3) / 3$$

.



```
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. 2.11: Код программы lab6-3.asm

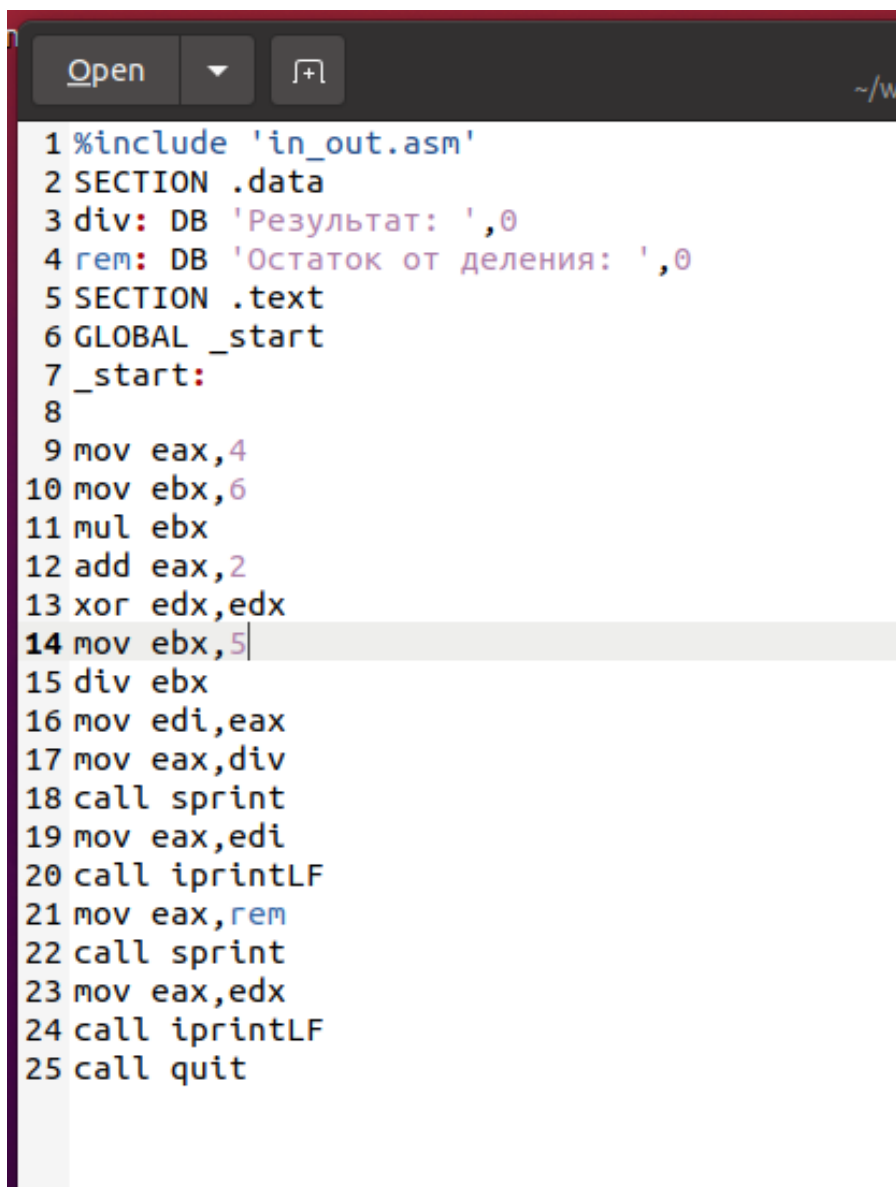


```
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
huleroyn@Huler-Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.12: Компиляция и запуск программы lab6-3.asm

Затем модифицировал код программы для того, чтобы она могла вычислять функцию. После создания исполняемого файла он провел его тестирование.

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



```
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
```

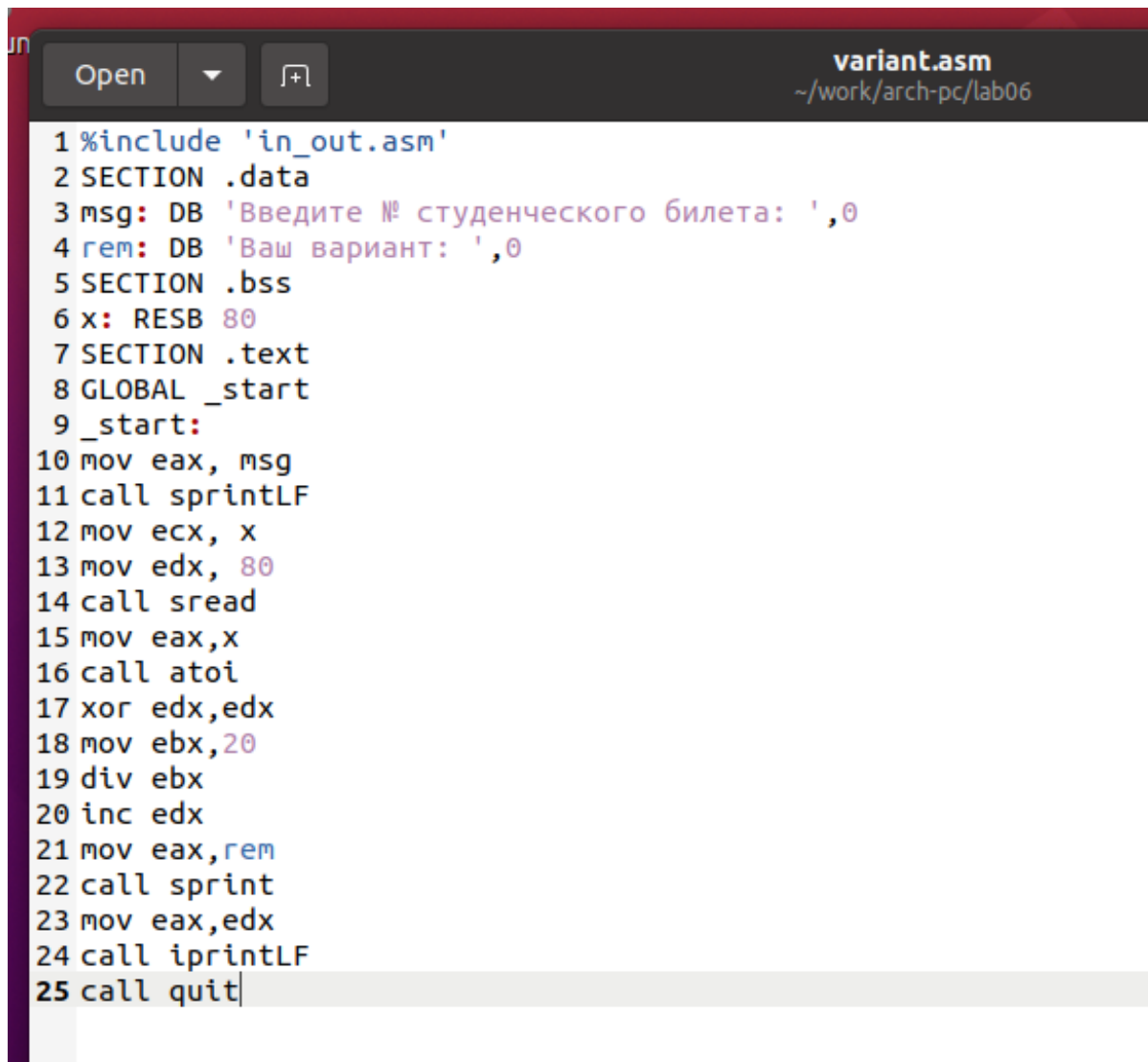
Рис. 2.13: Код программы lab6-3.asm


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

Рис. 2.14: Компиляция и запуск программы lab6-3.asm

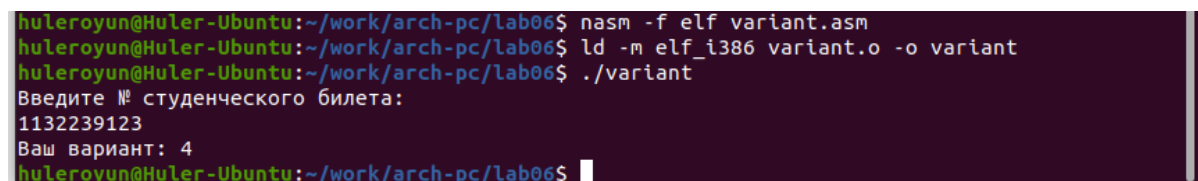
В качестве другого примера мы рассмотрели программу вычисления варианта задания по номеру студенческого билета.

В этом примере число для выполнения арифметических действий пользователь вводит с клавиатуры. Как было отмечено ранее, ввод осуществляется в виде символов, и для того чтобы арифметические операции в NASM были выполнены правильно, эти символы необходимо преобразовать в числовой формат. Для конвертации можно применить функцию `atoi`, которая содержится 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 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. 2.15: Код программы variant.asm



```
huleroyn@Huleroyn-Ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant.asm
huleroyn@Huleroyn-Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
huleroyn@Huleroyn-Ubuntu:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132239123
Ваш вариант: 4
huleroyn@Huleroyn-Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.16: Компиляция и 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 `eax` строку с текстом “Ваш вариант:”. После этого команда `call sprint` иницирует процедуру, которая выводит эту строку на экран.

2. Для чего используются следующие инструкции?

Команда `mov ecx, x` копирует значение из регистра `ecx` в переменную `x`. Команда `mov edx, 80` присваивает регистру `edx` число 80. Команда `call sread` запускает процедуру чтения данных из стандартного ввода.

3. Для чего используется инструкция “`call atoi`”?

Команда `call atoi` конвертирует строку символов в целое число.

4. Какие строки листинга отвечают за вычисления варианта?

Команды, выполняющие вычисление варианта, включают: `xor edx, edx` для обнуления регистра `edx`, `mov ebx, 20` для присвоения регистру `ebx` числа 20, `div ebx` для деления значения в аккумуляторе на значение в `ebx`, и `inc edx` для увеличения результата в регистре `edx` на единицу.

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

Остаток от деления после выполнения команды `div ebx` записывается в регистр `edx`.

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

Команда `inc edx` служит для инкремента, то есть увеличения значения в регистре `edx` на один. Это необходимо для корректного расчета варианта по заданной формуле, где к остатку от деления добавляется единица.

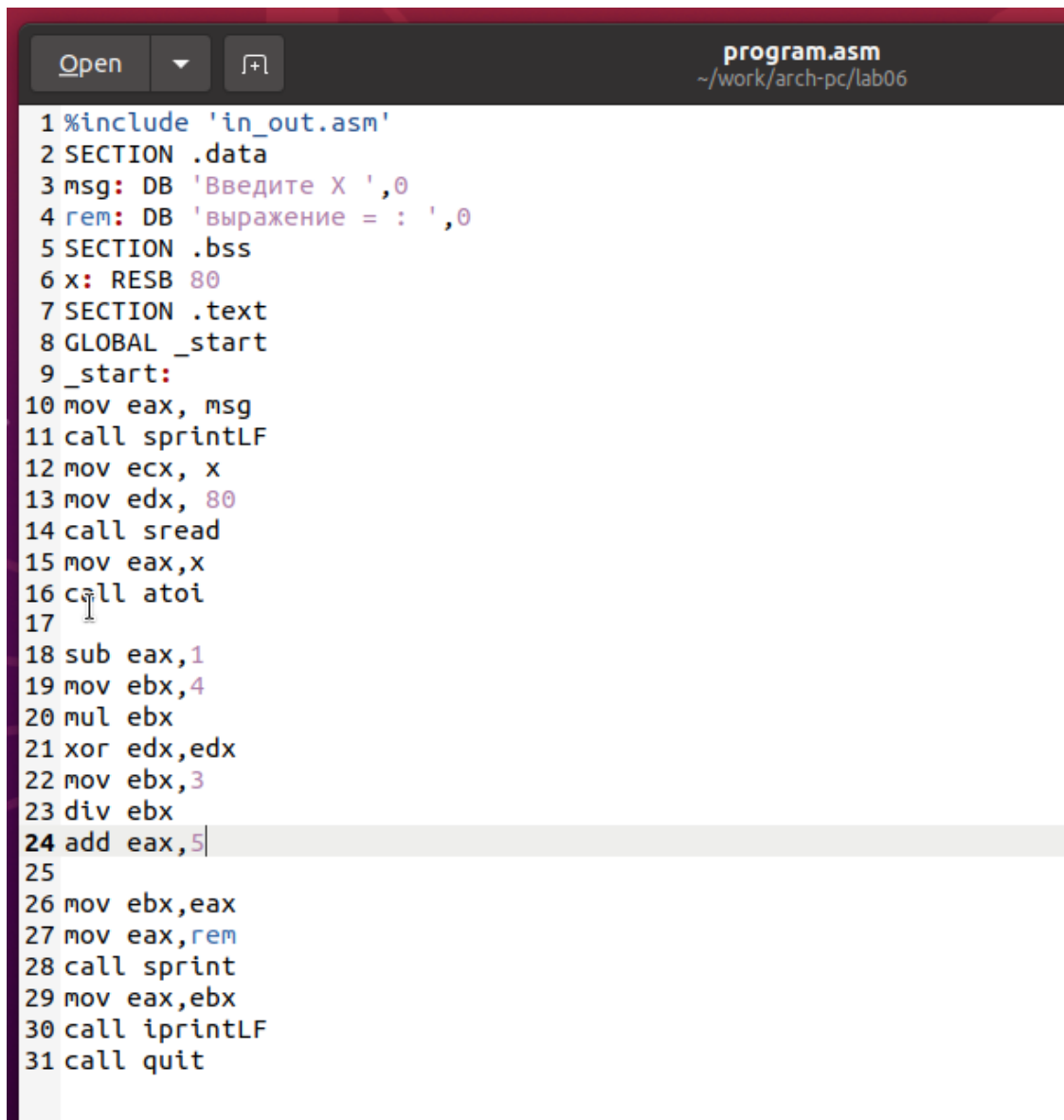
7. Какие строки листинга отвечают за вывод на экран результата вычислений?

Команда `mov eax, edx` помещает результат вычислений в регистр `eax`. Затем команда `call iprintLF` активирует процедуру, которая выводит результат на экран с переводом строки.

2.1 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.

Вариант - $\frac{4}{3}(x - 1) + 5$ для $x = 4, x = 10$



```
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 sprintf
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax, x
16 call atoi
17
18 sub eax, 1
19 mov ebx, 4
20 mul ebx
21 xor edx, edx
22 mov ebx, 3
23 div ebx
24 add eax, 5
25
26 mov ebx, eax
27 mov eax, rem
28 call sprintf
29 mov eax, ebx
30 call iprintLF
31 call quit
```

Рис. 2.17: Код программы program.asm

при $x = 4$ $f(x) = 10$

при $x = 10$ $f(x) = 17$

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

Рис. 2.18: Компиляция и запуск программы program.asm

Программа считает верно.

3 Выводы

Изучили работу с арифметическими операциями.