

Отчёт по лабораторной работе 6

Дисциплина: Архитектура компьютеров

Баранов Георгий Павлович

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
3.1	Символьные и численные данные в NASM	8
3.2	Выполнение арифметических операций в NASM	14
3.3	Задание для самостоятельной работы	20
4	Выводы	23

Список иллюстраций

3.1	Программа в файле lab6-1.asm	9
3.2	Запуск программы lab6-1.asm	9
3.3	Программа в файле lab6-1.asm	10
3.4	Запуск программы lab6-1.asm	11
3.5	Программа в файле lab6-2.asm	12
3.6	Запуск программы lab6-2.asm	12
3.7	Программа в файле lab6-2.asm	13
3.8	Запуск программы lab6-2.asm	13
3.9	Запуск программы lab6-2.asm	14
3.10	Программа в файле lab6-3.asm	15
3.11	Запуск программы lab6-3.asm	15
3.12	Программа в файле lab6-3.asm	16
3.13	Запуск программы lab6-3.asm	17
3.14	Программа в файле variant.asm	18
3.15	Запуск программы variant.asm	18
3.16	Программа в файле task.asm	21
3.17	Запуск программы task.asm	22

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 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. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации. Существует три основных способа адресации:

- Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`.
- Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,2`.
- Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

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`.

Существуют специальные команды: `inc` (от англ. *increment*) и `dec` (от англ. *decrement*), которые увеличивают и уменьшают на 1 свой операнд.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производятся по-разному, поэтому существуют различные

команды. Для беззнакового умножения используется команда `mul` (от англ. `multiply` – умножение), для знакового умножения используется команда `imul`.

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

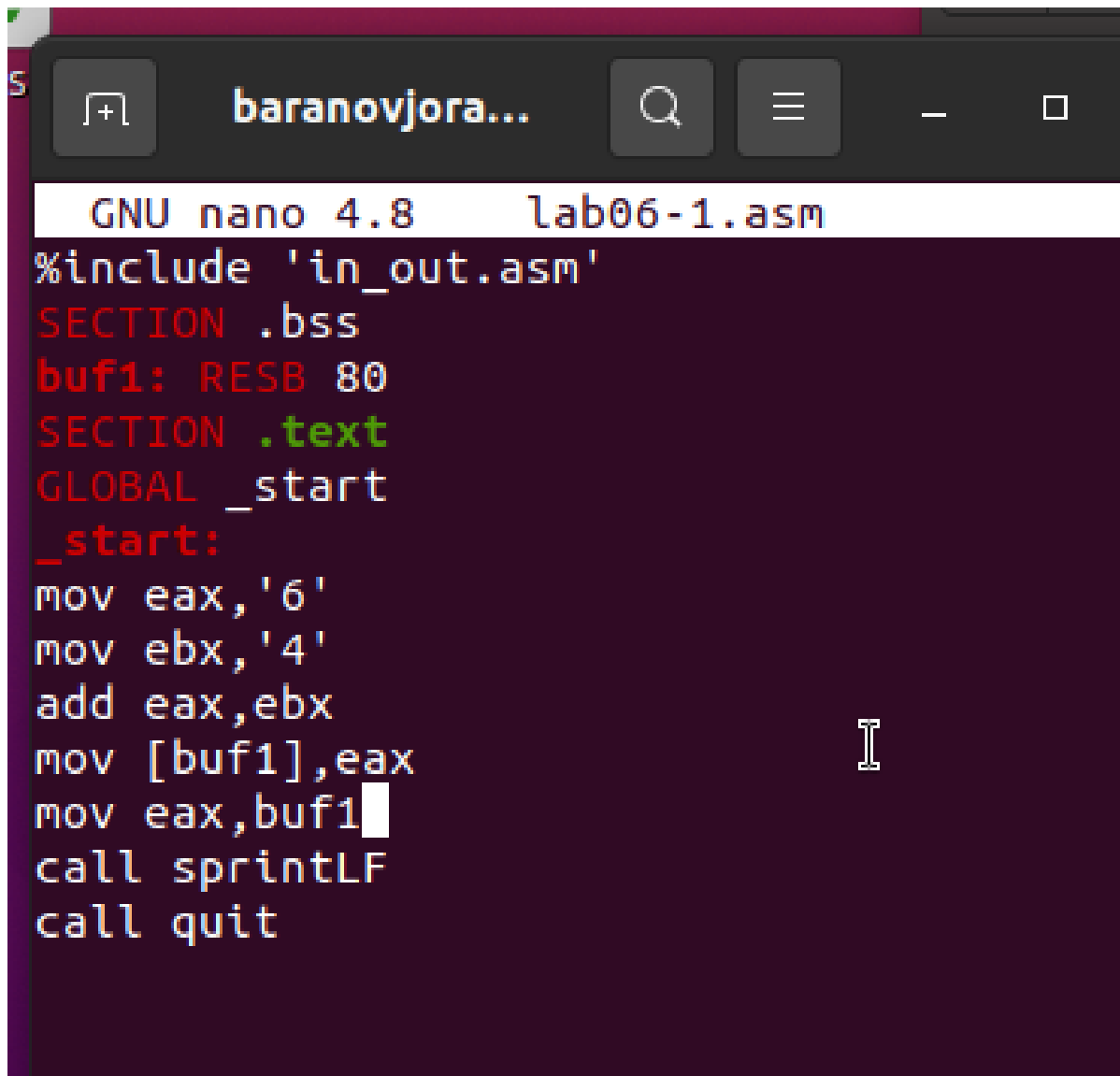
3 Выполнение лабораторной работы

3.1 Символьные и численные данные в NASM

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

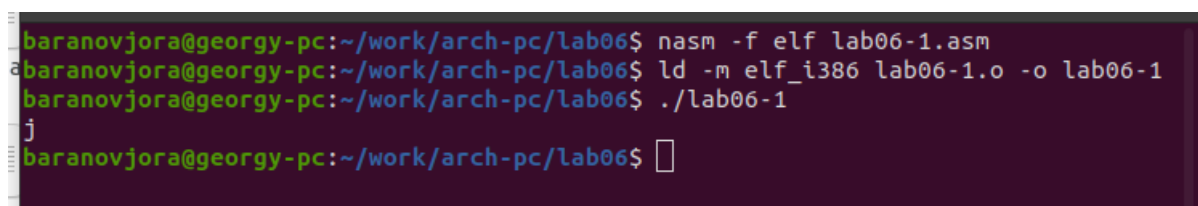
Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр `eax`.

В данной программе в регистр `eax` записывается символ 6 (`mov eax,'6'`), в регистр `ebx` символ 4 (`mov ebx,'4'`). Далее к значению в регистре `eax` прибавляем значение регистра `ebx` (`add eax,ebx`, результат сложения запишется в регистр `eax`). Далее выводим результат. Так как для работы функции `sprintLF` в регистр `eax` должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра `eax` в переменную `buf1` (`mov [buf1],eax`), а затем запишем адрес переменной `buf1` в регистр `eax` (`mov eax,buf1`) и вызовем функцию `sprintLF`.



```
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 sprintf
call quit
```

Рис. 3.1: Программа в файле lab6-1.asm



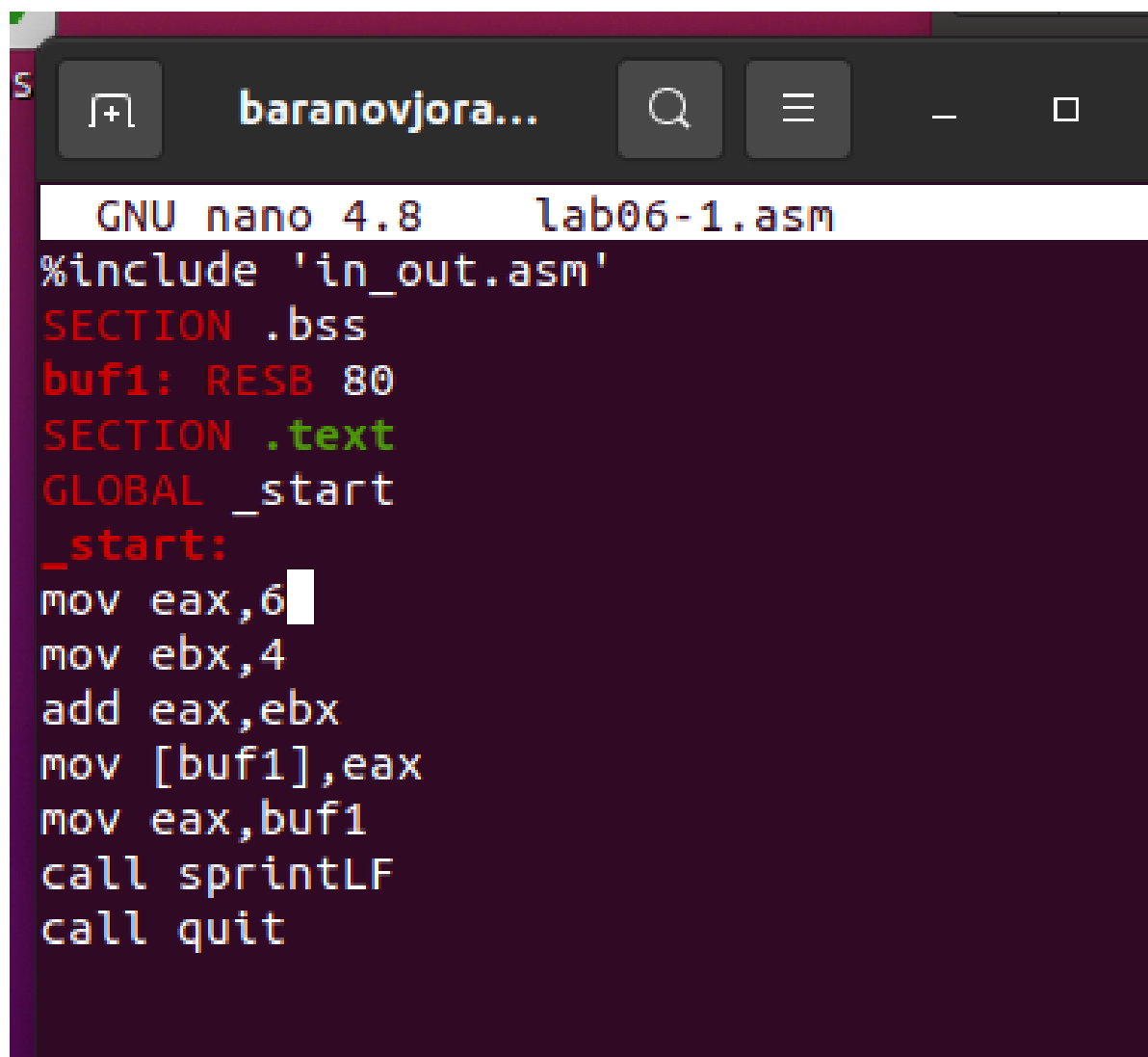
```
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./lab06-1
j
baranovjora@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.2: Запуск программы lab6-1.asm

В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число 10. Однако результатом будет символ `j`. Это происходит 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` запишет в регистр `eax` сумму кодов – 01101010 (106), что в свою очередь является кодом символа `j`.

Далее изменяю текст программы и вместо символов, запишем в регистры числа.



```
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. 3.3: Программа в файле lab6-1.asm

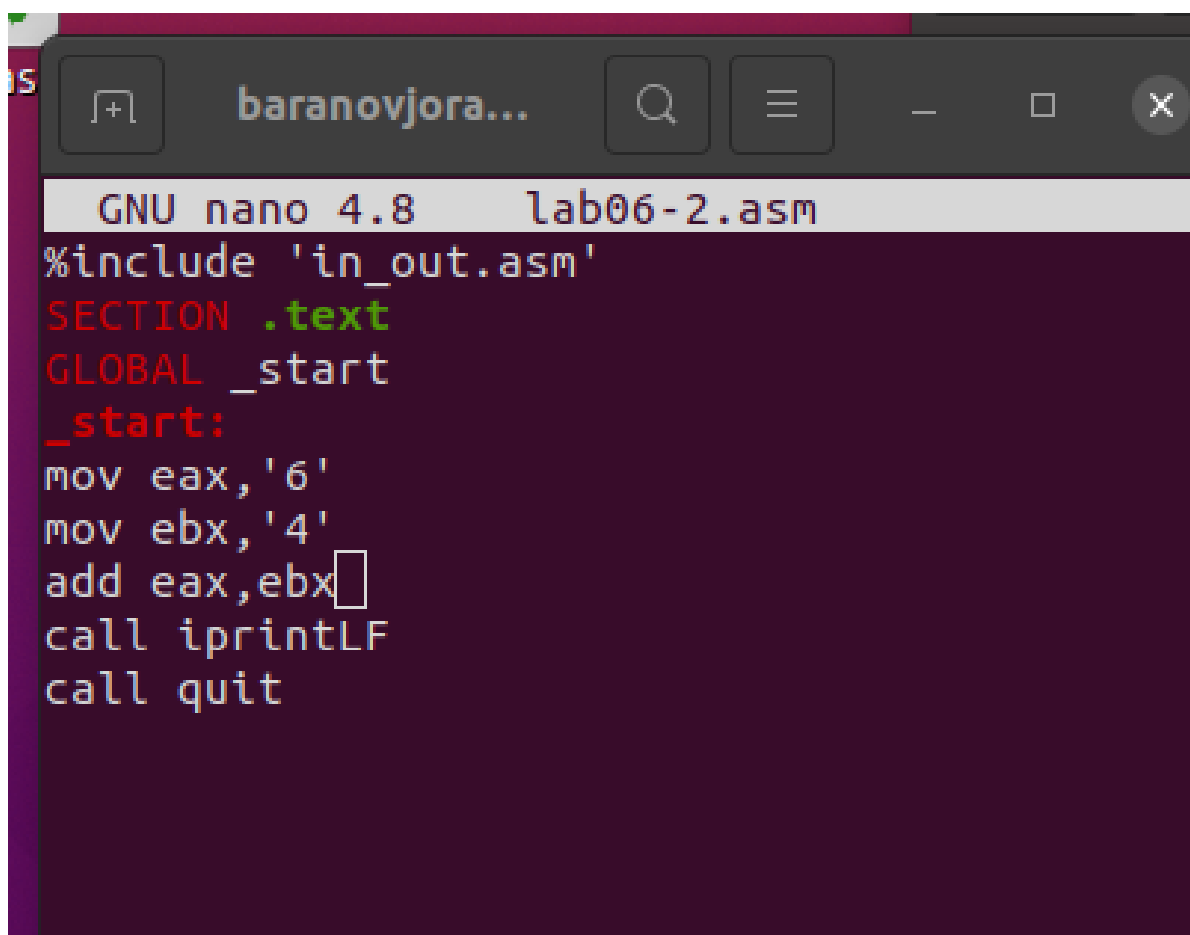
```
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./lab06-1
j
baranovjora@georgy-pc:~/work/arch-pc/lab06$
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm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

baranovjora@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.4: Запуск программы lab6-1.asm

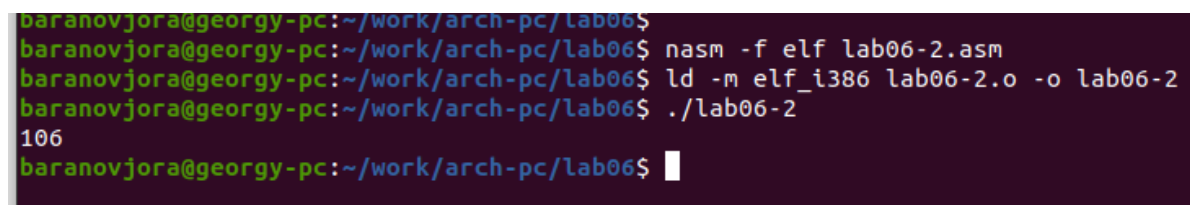
Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Это символ конца строки (возврат каретки). В консоле он не отображается, но добавляет пустую строку.

Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразовал текст программы с использованием этих функций.



```
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. 3.5: Программа в файле lab6-2.asm

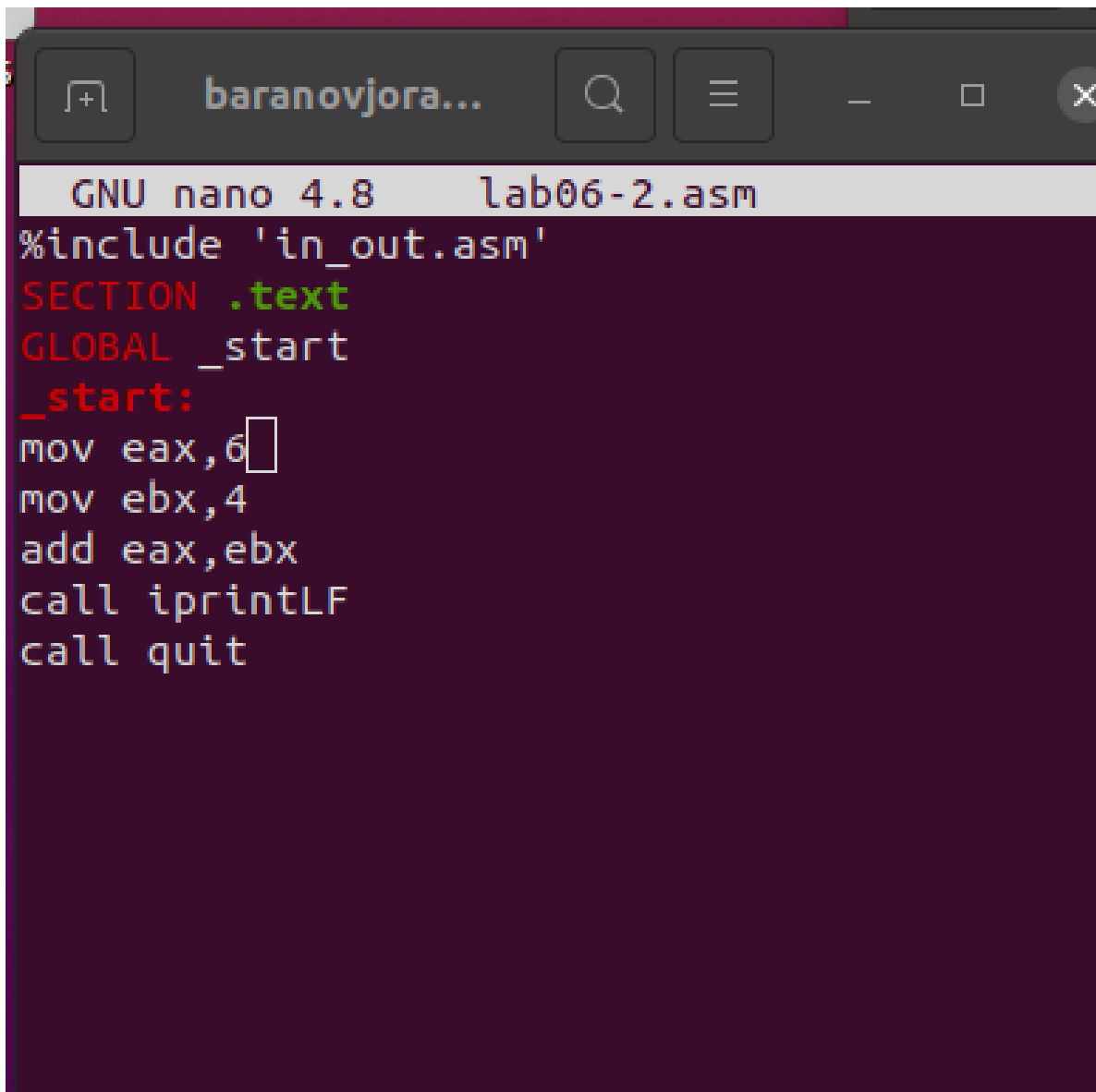


```
baranovjora@georgy-pc:~/work/arch-pc/lab06$
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./lab06-2
106
baranovjora@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.6: Запуск программы lab6-2.asm

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда `add` складывает коды символов '6' и '4' ($54+52=106$). Однако, в отличие от прошлой программы, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

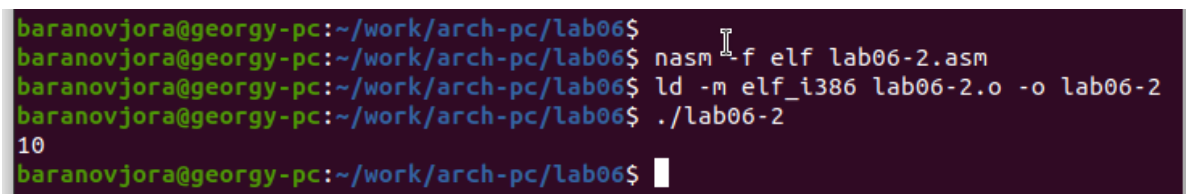
Аналогично предыдущему примеру изменим символы на числа.



```
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. 3.7: Программа в файле lab6-2.asm

Функция `iprintLF` позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10.



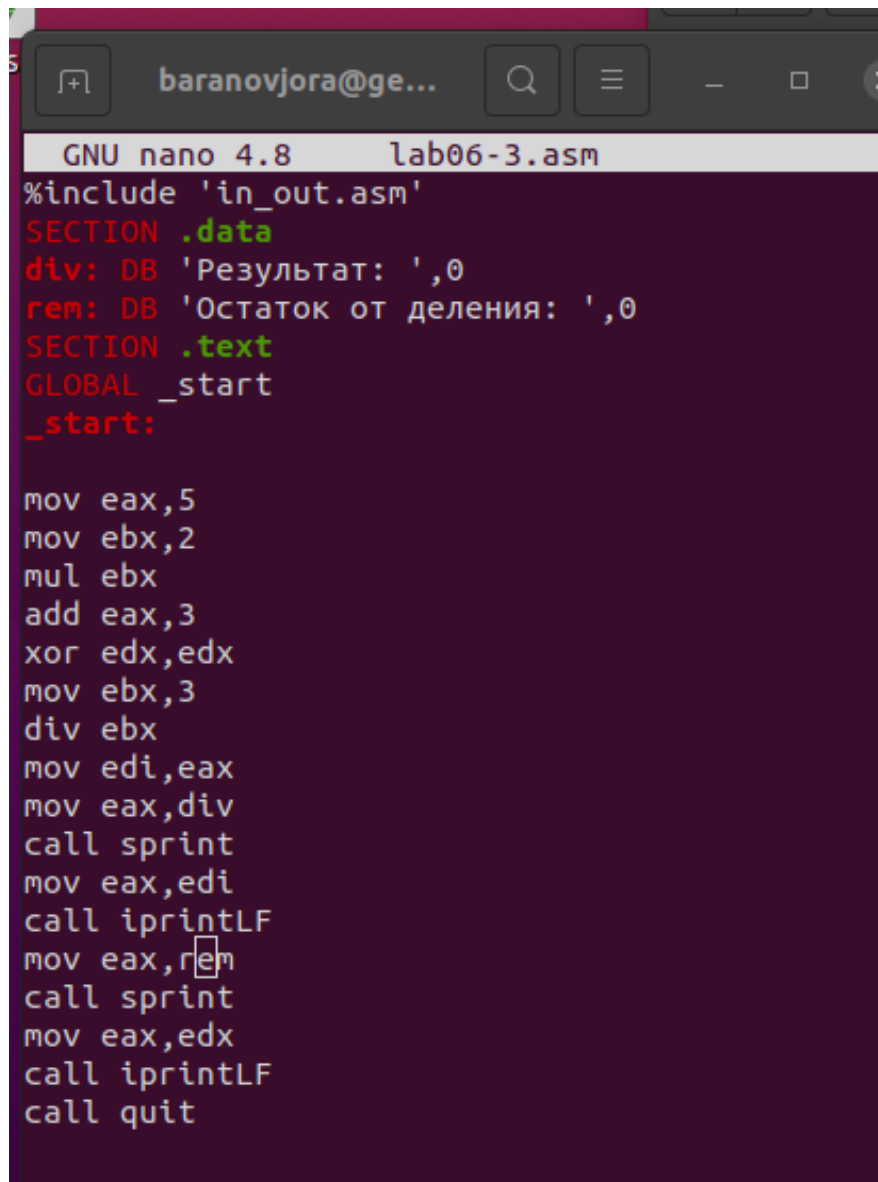
```
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./lab06-2
10
baranovjora@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.8: Запуск программы lab6-2.asm

Заменяю функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Вывод отличается тем, что нет переноса строки.

Рис. 3.9: Запуск программы lab6-2.asm

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

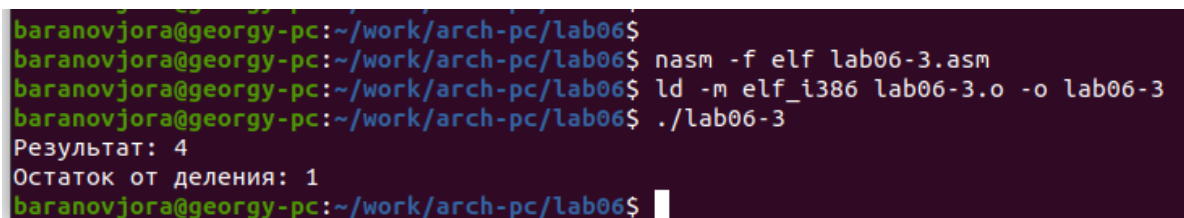


The screenshot shows a terminal window with the GNU nano 4.8 editor open to the file lab06-3.asm. The code is as follows:

```
%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. 3.10: Программа в файле lab6-3.asm



The screenshot shows a terminal window with the following commands and output:

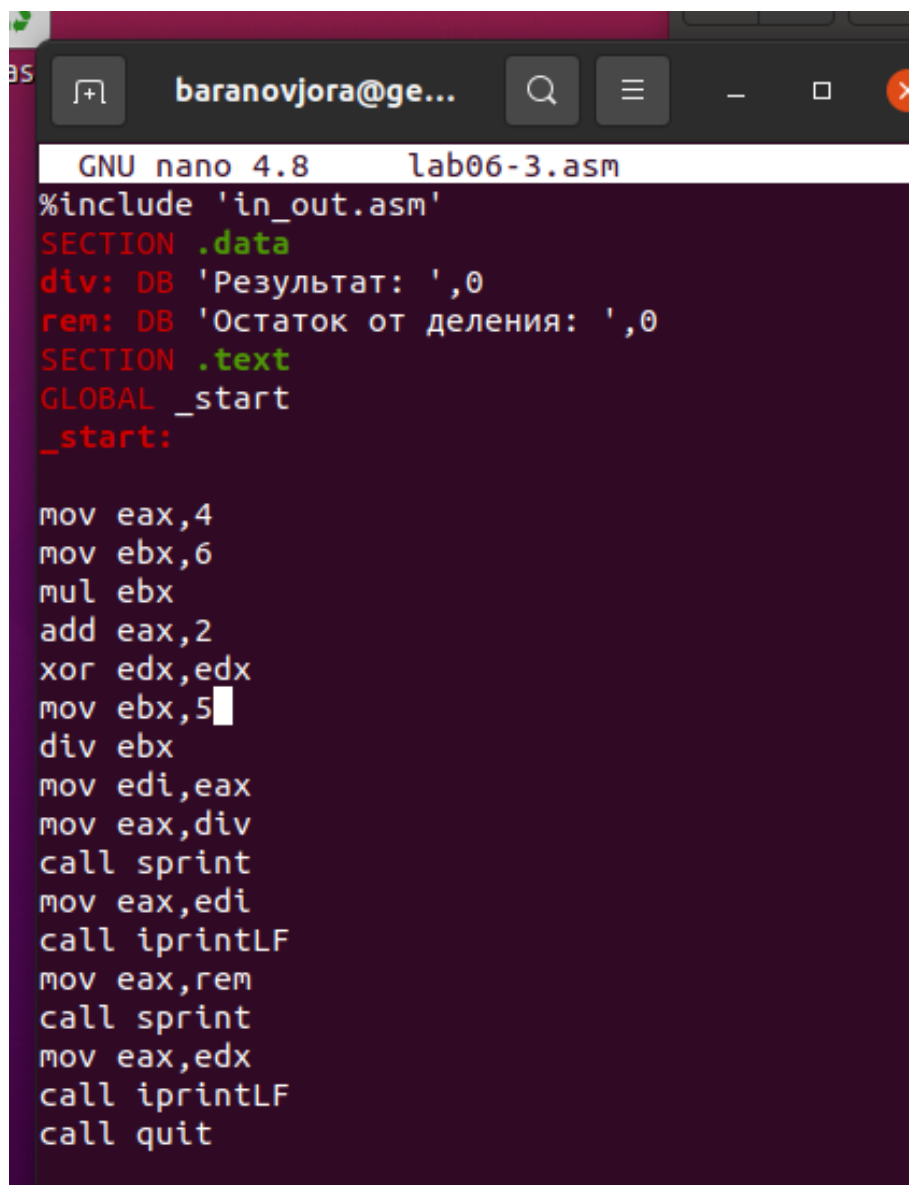
```
baranovjora@georgy-pc:~/work/arch-pc/lab06$
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
baranovjora@georgy-pc:~/work/arch-pc/lab06$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

Рис. 3.11: Запуск программы lab6-3.asm

Изменил текст программы для вычисления выражения

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

. Создал исполняемый файл и проверил его работу.



```
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 iprintlnLF
mov eax,rem
call sprint
mov eax,edx
call iprintlnLF
call quit
```

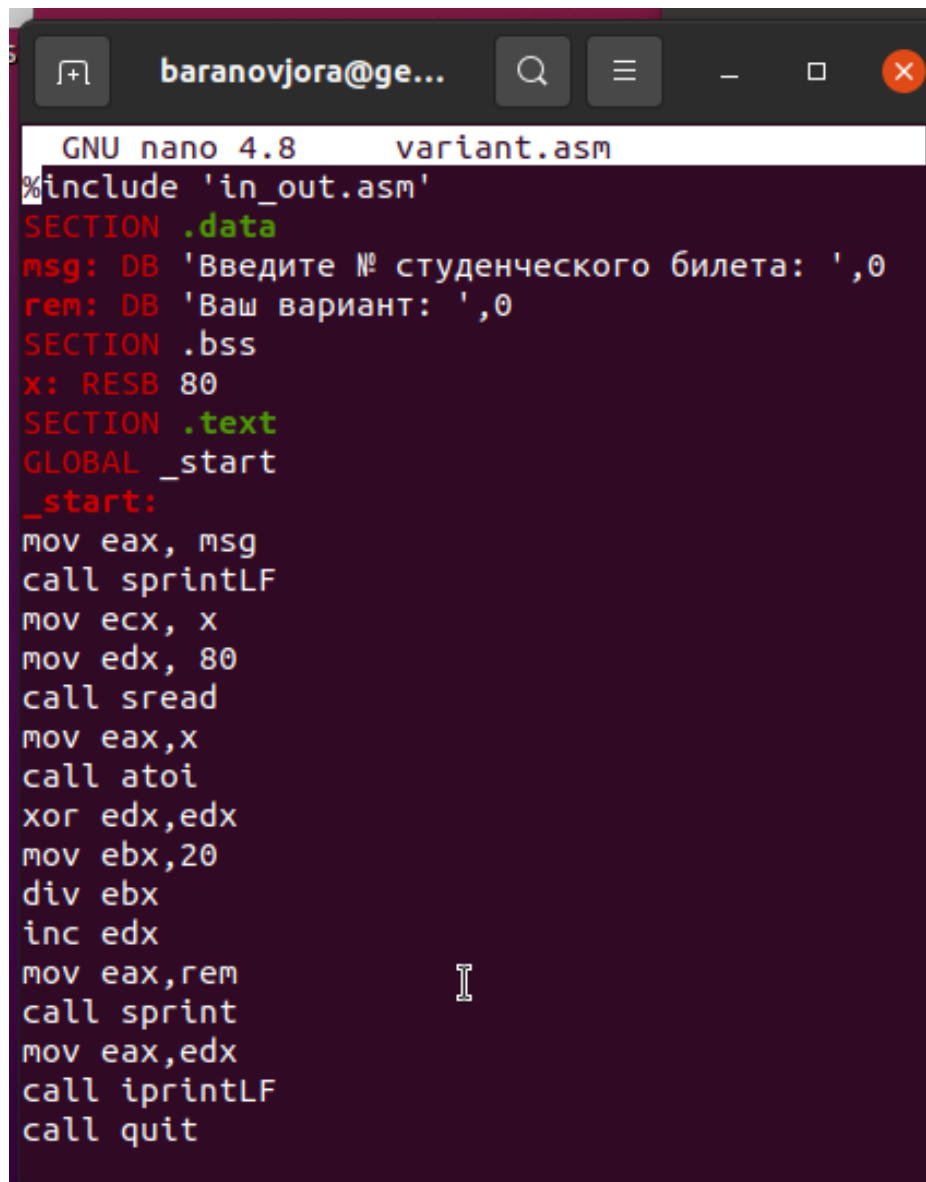
Рис. 3.12: Программа в файле lab6-3.asm


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

Рис. 3.13: Запуск программы lab6-3.asm

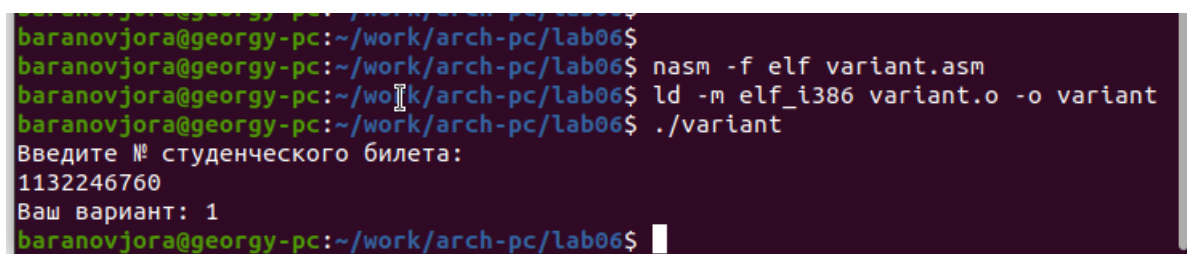
В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета.

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`.



```
GNU nano 4.8 variant.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 sprintLF
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 sprint
mov eax, edx
call iprintLF
call quit
```

Рис. 3.14: Программа в файле variant.asm



```
baranovjora@georgy-pc:~/work/arch-pc/lab06$
baranovjora@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf variant.asm
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
baranovjora@georgy-pc:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132246760
Ваш вариант: 1
baranovjora@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.15: Запуск программы variant.asm

ответы на вопросы

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

`mov eax,rem` – перекладывает в регистр значение переменной с фразой ‘Ваш вариант:’

`call sprint` – вызов подпрограммы вывода строки

2. Для чего используются следующие инструкции?

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

Считывает значение студбилета в переменную X из консоли

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

Эта подпрограмма переводит введенные символы в числовой формат.

4. Какие строки листинга отвечают за вычисления варианта?

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

Здесь происходит деление номера студ билета на 20. В регистре `edx` хранится остаток, к нему прибавляется 1.

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

регистр `edx`

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

по формуле вычисления варианта нужно прибавить единицу

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

`mov eax,edx` – результат перекладывается в регистр `eax`

`call iprintLF` – вызов подпрограммы вывода

3.3 Задание для самостоятельной работы

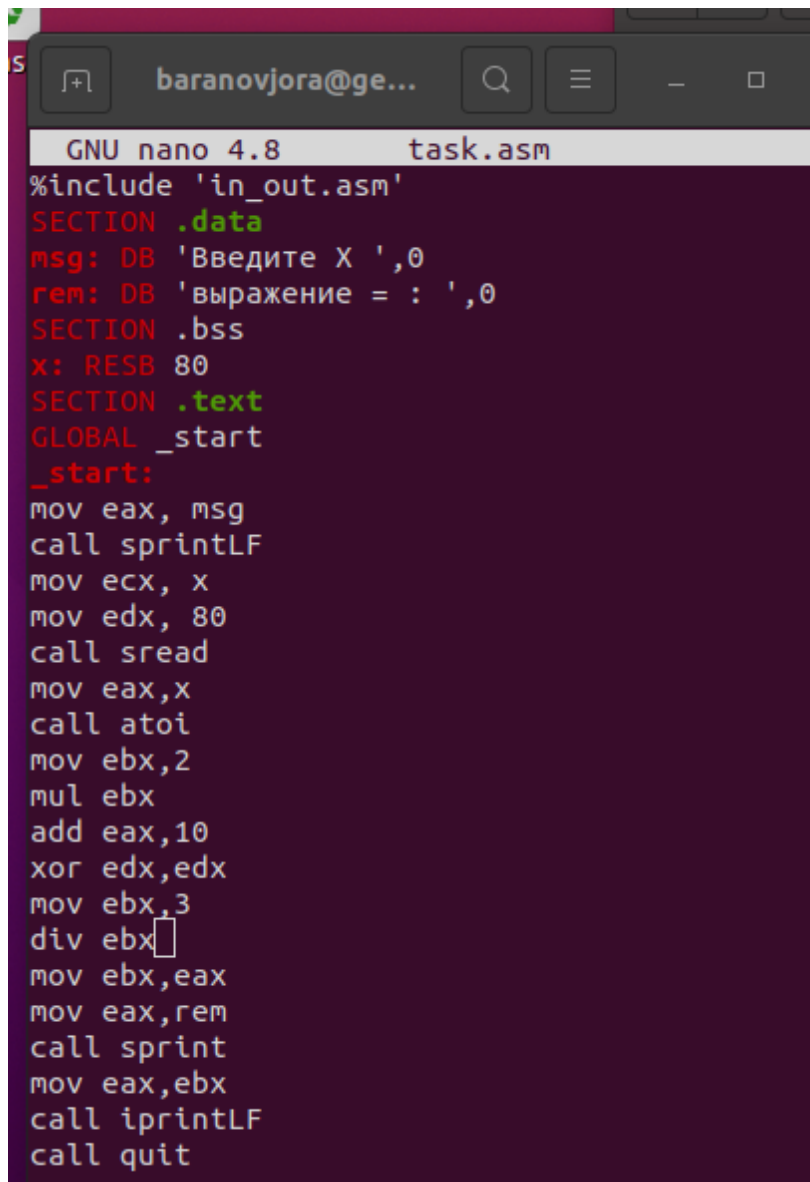
Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x , вычислять заданное выражение в зависимости от введенного x , выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы 6.3 вариантов заданий в соответствии 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 1 -

$$(10 + 2x)/3$$

для

$$x_1 = 1, x_2 = 10$$



```
GNU nano 4.8 task.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
mov ebx, 2
mul ebx
add eax, 10
xor edx, edx
mov ebx, 3
div ebx
mov ebx, eax
mov eax, rem
call sprint
mov eax, ebx
call iprintLF
call quit
```

Рис. 3.16: Программа в файле task.asm

Если подставить 1 получается 4.

Если подставить 10 получается 10.

```
baranovjoga@georgy-pc:~/work/arch-pc/lab06$  
baranovjoga@georgy-pc:~/work/arch-pc/lab06$ nasm -f elf task.asm  
baranovjoga@georgy-pc:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task  
nt. baranovjoga@georgy-pc:~/work/arch-pc/lab06$ ./task  
Введите X  
1  
выражение = : 4  
baranovjoga@georgy-pc:~/work/arch-pc/lab06$ ./task  
Введите X  
10  
выражение = : 10  
baranovjoga@georgy-pc:~/work/arch-pc/lab06$
```

Рис. 3.17: Запуск программы task.asm

Программа считает верно.

4 Выводы

Изучили работу с арифметическими операциями.