

Отчёт по лабораторной работе 6

Архитектура компьютера

Тарасова Анна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	ответы на вопросы	17
2.2	Задание для самостоятельной работы	18
3	Выводы	21

Список иллюстраций

2.1	Редактирование файла lab6-1.asm	7
2.2	Компиляция и запуск файла lab6-1.asm	8
2.3	Редактирование файла lab6-1.asm	8
2.4	Компиляция и запуск файла lab6-1.asm	9
2.5	Редактирование файла lab6-2.asm	9
2.6	Компиляция и запуск файла lab6-2.asm	10
2.7	Редактирование файла lab6-2.asm	11
2.8	Компиляция и запуск файла lab6-2.asm	11
2.9	Компиляция и запуск файла lab6-2.asm	12
2.10	Редактирование файла lab6-3.asm	13
2.11	Компиляция и запуск файла lab6-3.asm	13
2.12	Редактирование файла lab6-3.asm	14
2.13	Компиляция и запуск файла lab6-3.asm	15
2.14	Редактирование файла variant.asm	16
2.15	Компиляция и запуск файла variant.asm	17
2.16	Редактирование файла task.asm	19
2.17	Компиляция и запуск файла task.asm	20

Список таблиц

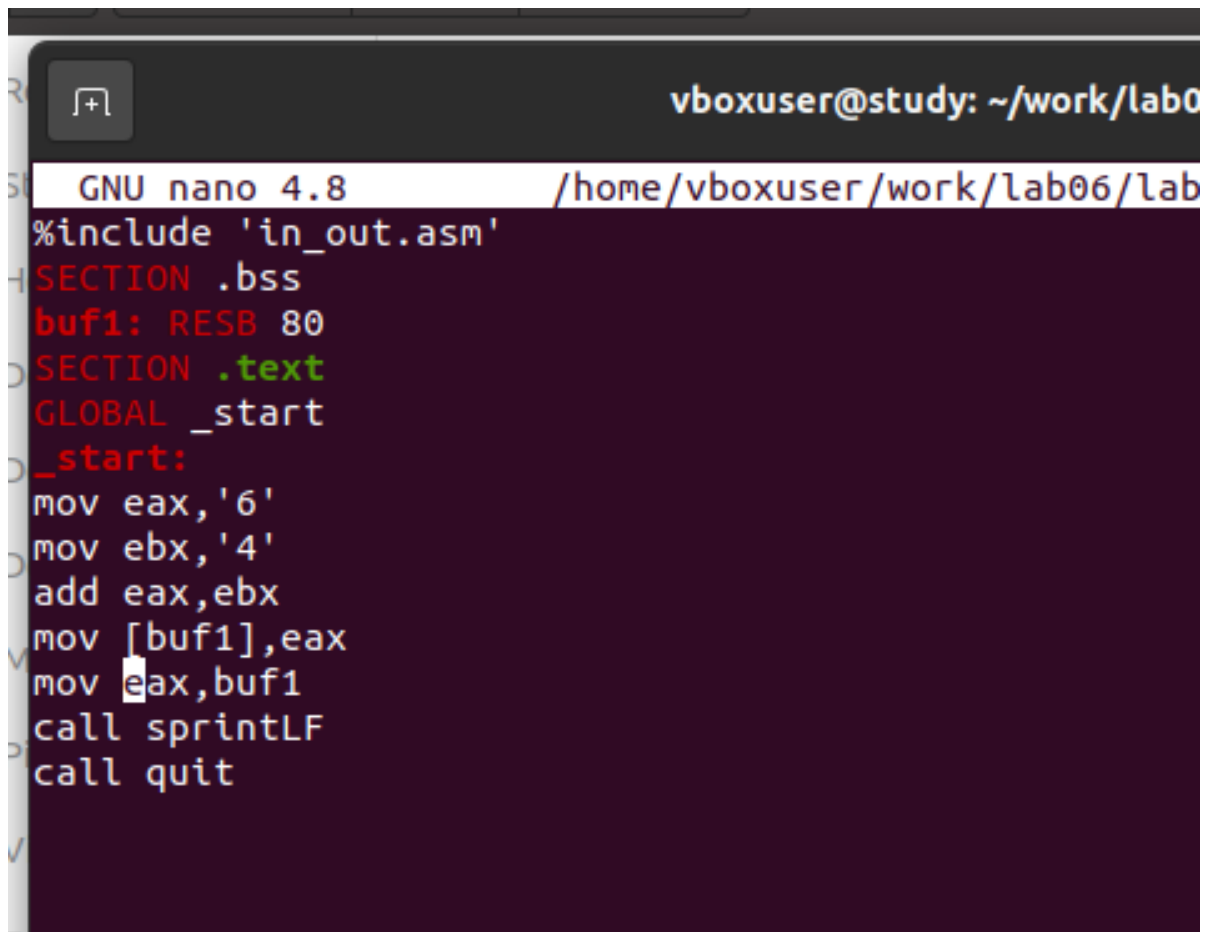
1 Цель работы

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 названием lab6-1.asm.
2. Давайте посмотрим на примеры программ, которые выводят символы и числовые данные. Эти программы будут отображать информацию, которую мы поместим в регистр `eax`.

В одной из программ я поместила символ '6' в регистр `eax`, используя команду `mov eax, '6'`, а затем записала символ '4' в регистр `ebx` с помощью команды `mov ebx, '4'`. После этого я сложила значения из регистров `eax` и `ebx`, применив команду `add eax, ebx`, и результат сложения сохранился в регистре `eax`. Затем я хотела вывести получившийся результат. Но поскольку для работы функции `sprintf` необходимо, чтобы в регистре `eax` был адрес, мне понадобилась дополнительная переменная. Я перенесла значение из регистра `eax` в переменную `buf1`, используя команду `mov [buf1], eax`. Потом я поместила адрес переменной `buf1` обратно в регистр `eax` с помощью команды `mov eax, buf1` и вызвала функцию `sprintf`, чтобы вывести результат на экран.



```
GNU nano 4.8 /home/vboxuser/work/lab06/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
```

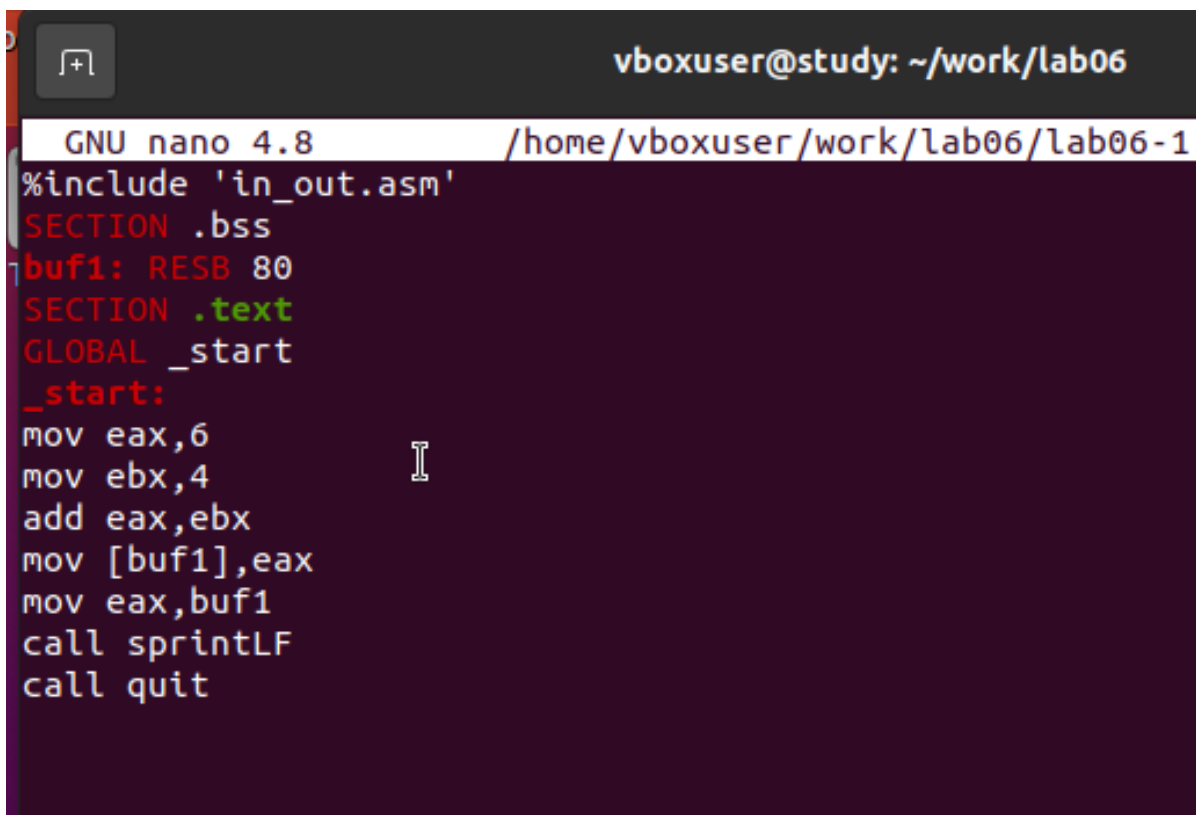
Рис. 2.1: Редактирование файла lab6-1.asm

В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число 10. Но вместо этого там оказывается символ 'j'. Это связано с тем, что в компьютерном представлении код символа '6' равен 00110110 в двоичном формате, что соответствует 54 в десятичной системе, а код символа '4' - это 00110100, или 52 в десятичных числах. Когда выполняется команда `add eax, ebx`, в регистр `eax` записывается сумма этих кодов, которая равна 01101010 в двоичной системе, или 106 в десятичной, и это именно код для символа 'j'.

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

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

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

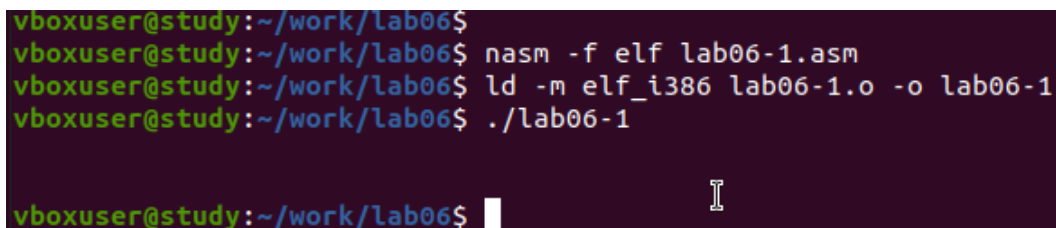


```
vboxuser@study: ~/work/lab06  
GNU nano 4.8 /home/vboxuser/work/lab06/lab06-1  
%include 'in_out.asm'  
SECTION .bss  
buf1: RESB 80  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
mov [buf1],eax  
mov eax,buf1  
call sprintf  
call quit
```

Рис. 2.3: Редактирование файла lab6-1.asm

Как и в прошлый раз, когда мы запускали программу, число 10 не появится. На этот раз на экране появится символ, который соответствует коду 10. Этот

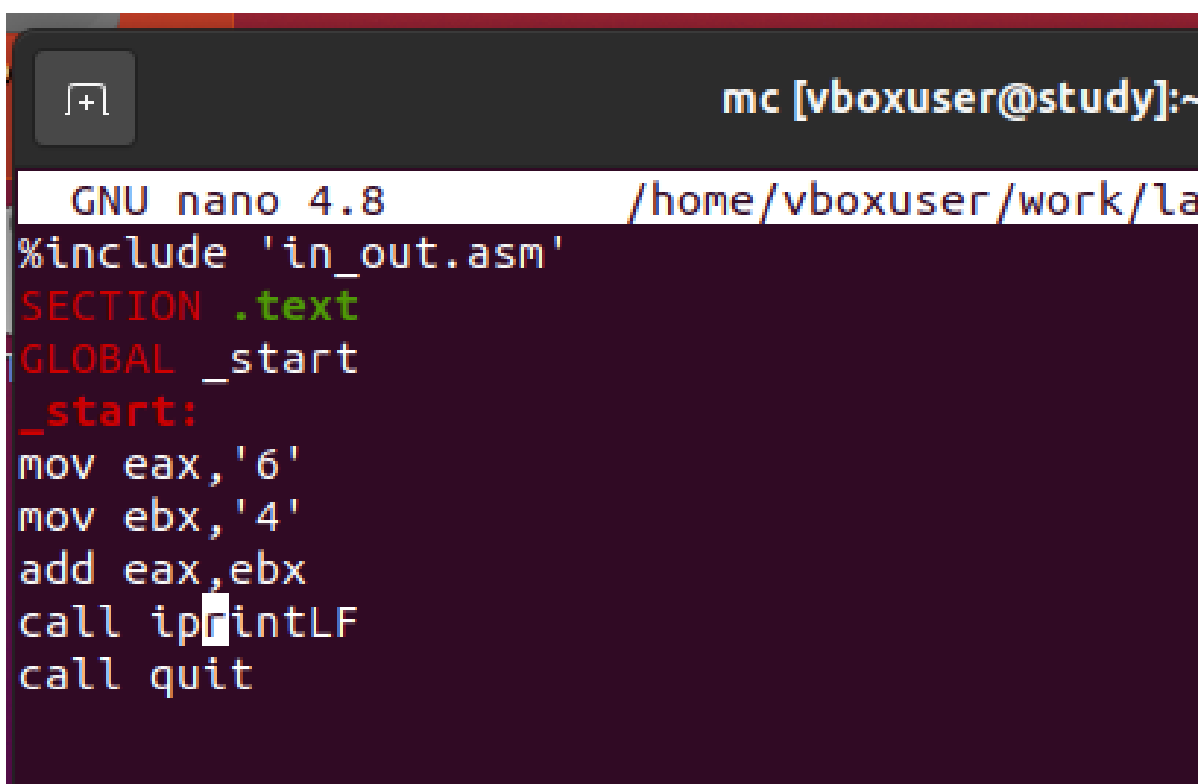
символ означает конец строки, или возврат каретки. В консоли он не виден, но он создает пустую строку в выводе.



```
vboxuser@study:~/work/lab06$  
vboxuser@study:~/work/lab06$ nasm -f elf lab06-1.asm  
vboxuser@study:~/work/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1  
vboxuser@study:~/work/lab06$ ./lab06-1  
  
vboxuser@study:~/work/lab06$
```

Рис. 2.4: Компиляция и запуск файла lab6-1.asm

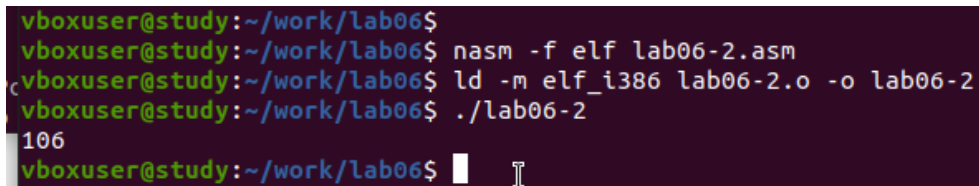
4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразовала текст программы с использованием этих функций.



```
mc [vboxuser@study]:~  
GNU nano 4.8 /home/vboxuser/work/la  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,'6'  
mov ebx,'4'  
add eax,ebx  
call iprintLF  
call quit
```

Рис. 2.5: Редактирование файла lab6-2.asm

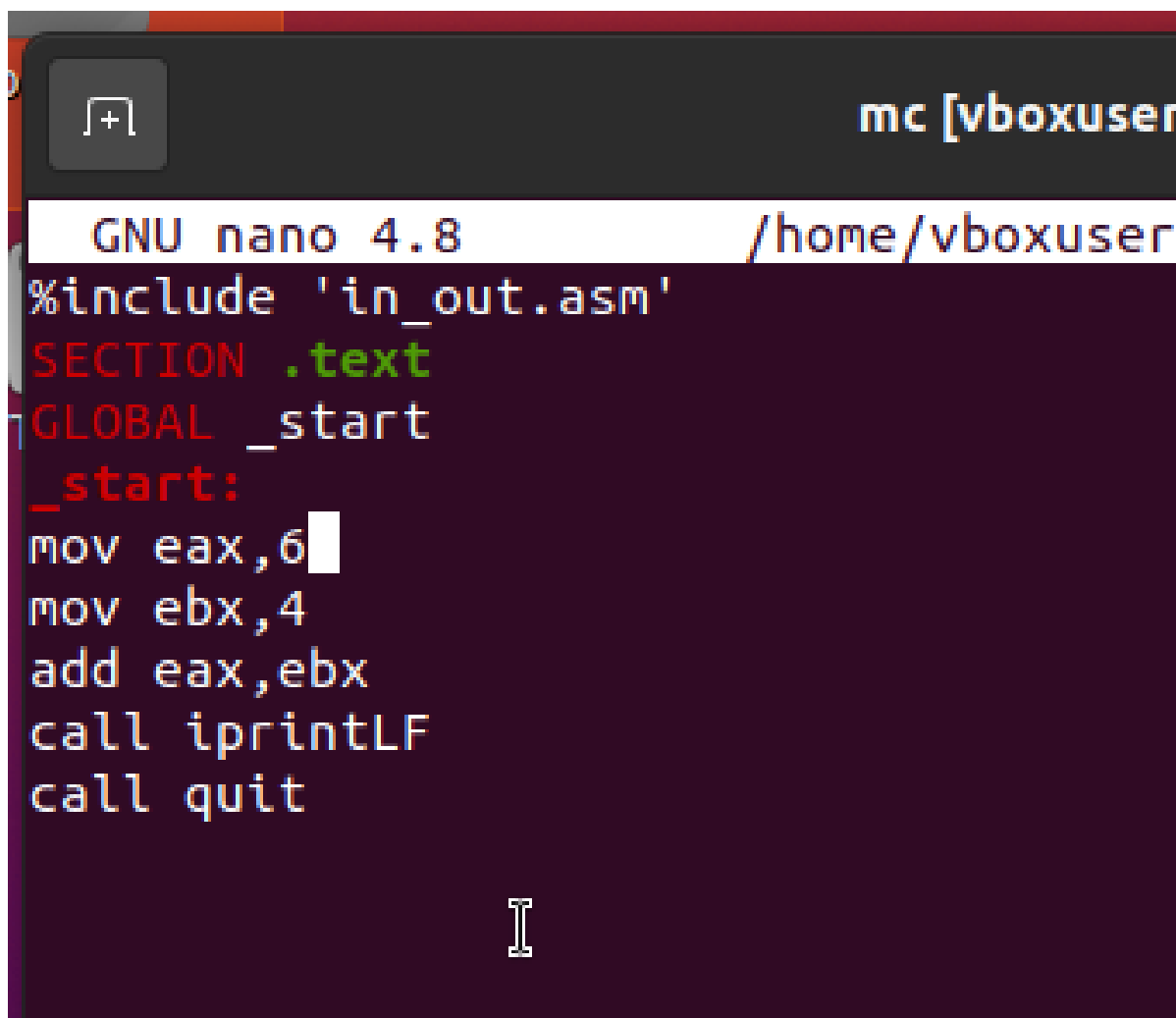
Когда я запустила программу, она выдала мне число 106. Это произошло потому, что команда `add` сложила ASCII коды символов '6' и '4', что в сумме дало 106. Важно отметить, что в этот раз, в отличие от предыдущей программы, я использовала функцию `iprintLF`, которая позволила мне вывести именно число, а не символ, соответствующий этому числовому коду.



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

Рис. 2.6: Компиляция и запуск файла `lab6-2.asm`

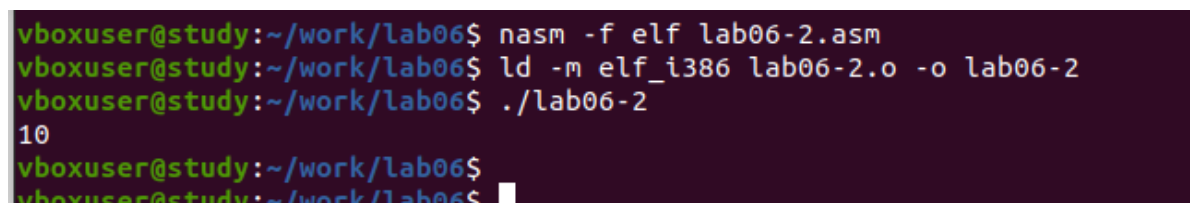
5. Аналогично предыдущему примеру изменим символы на числа.



```
mc [vboxuser
GNU nano 4.8 /home/vboxuser
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 2.7: Редактирование файла lab6-2.asm

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



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

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

Заменяла функцию `iprintLF` на `iprint`. Создала исполняемый файл и запустила его. Вывод отличается тем, что нет переноса строки.

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

Рис. 2.9: Компиляция и запуск файла `lab6-2.asm`

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

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

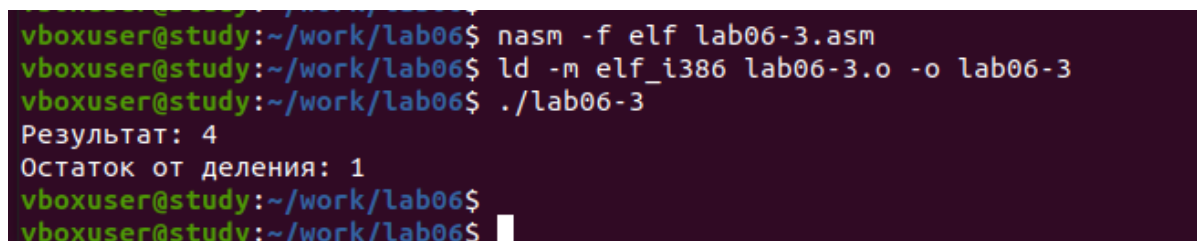
.



```
mc [vboxuser@study]:~/
GNU nano 4.8 /home/vboxuser/work/lab
%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
```

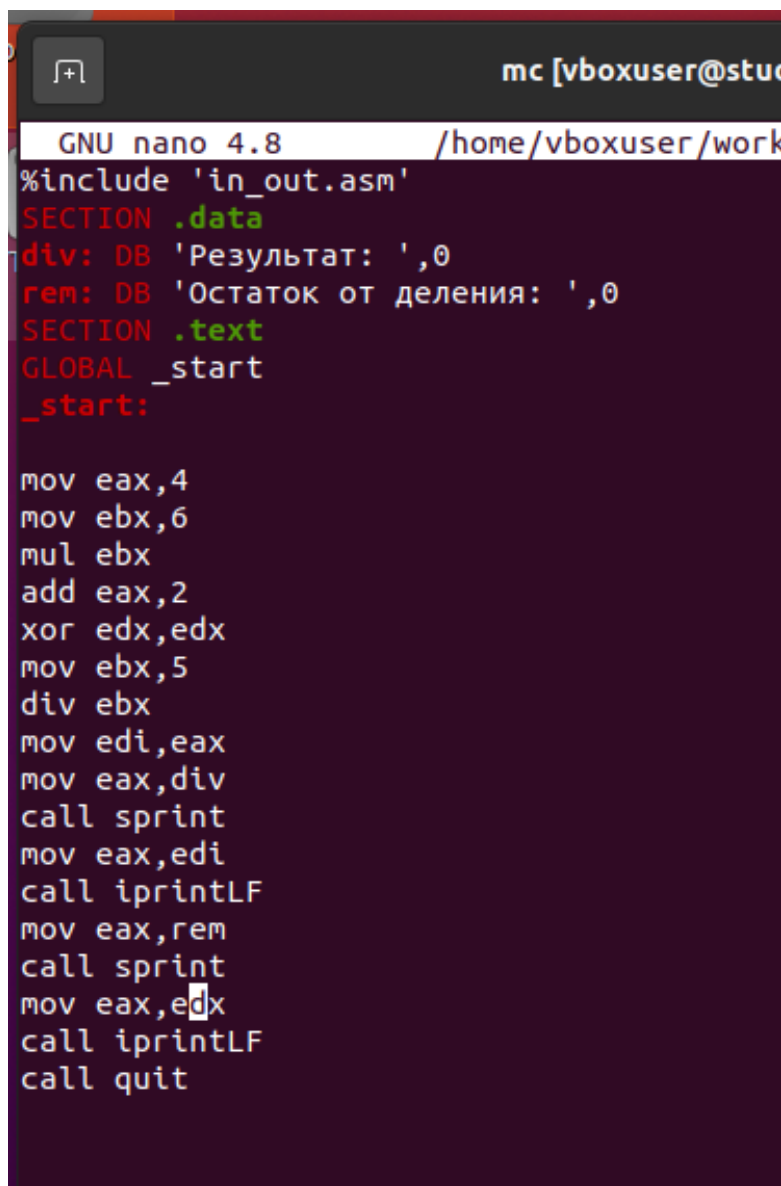
Рис. 2.10: Редактирование файла lab6-3.asm



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

Рис. 2.11: Компиляция и запуск файла lab6-3.asm

Изменила текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$.
Создала исполняемый файл и проверила его работу.



```
mc [vboxuser@stud
GNU nano 4.8 /home/vboxuser/work
%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
```

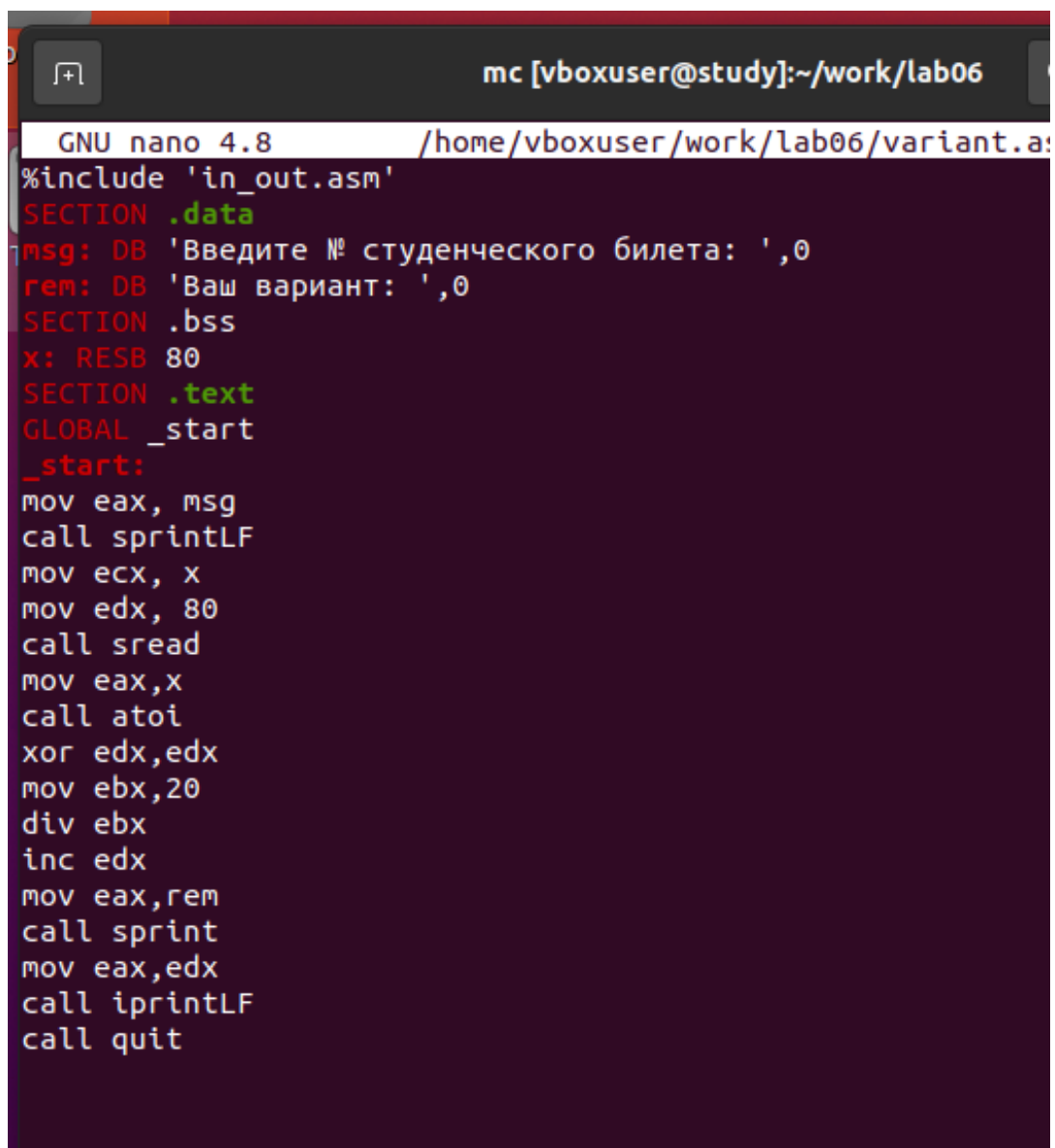
Рис. 2.12: Редактирование файла lab6-3.asm

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

Рис. 2.13: Компиляция и запуск файла lab6-3.asm

7. 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`.



```
mc [vboxuser@study]:~/work/lab06
GNU nano 4.8 /home/vboxuser/work/lab06/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 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. 2.14: Редактирование файла variant.asm


```
vboxuser@study:~/work/lab06$  
vboxuser@study:~/work/lab06$ nasm -f elf variant.asm  
vboxuser@study:~/work/lab06$ ld -m elf_i386 variant.o -o variant  
vboxuser@study:~/work/lab06$ ./variant  
Введите № студенческого билета:  
1132239107  
Ваш вариант: 8  
vboxuser@study:~/work/lab06$
```

Рис. 2.15: Компиляция и запуск файла variant.asm

2.1 ответы на вопросы

1. Какие строки листинга отвечают за вывод на экран сообщения ‘Ваш вариант:’?
 - Команда `mov eax, ptr` загружает в регистр `eax` адрес строки с текстом “Ваш вариант:”.
 - Использование `call sprint` запускает функцию, которая выводит строку на экран.
2. Для чего используются следующие инструкции?
 - `mov ecx, x` помещает в регистр `ecx` значение переменной `x`.
 - `mov edx, 80` устанавливает в регистре `edx` значение 80.
 - `call sread` активирует функцию чтения данных, которая считывает номер студенческого билета в переменную `x`.
3. Для чего используется инструкция “`call atoi`”?

Функция `atoi` конвертирует строковые данные в целочисленное значение.
4. Какие строки листинга отвечают за вычисления варианта?
 - `xor edx, edx` обнуляет регистр `edx`.

- `mov ebx, 20` помещает число 20 в регистр `ebx`.
 - `div ebx` выполняет деление аккумулятора на значение в `ebx`, результат в `eax`, остаток в `edx`.
 - `inc edx` увеличивает значение в регистре `edx` на единицу, что соответствует логике расчета варианта.
5. В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”?
- Остаток от деления сохраняется в регистре `edx`.
6. Для чего используется инструкция “`inc edx`”?
- Команда `inc edx` увеличивает значение в регистре `edx` на 1 для корректного расчета варианта по заданной формуле.
7. Какие строки листинга отвечают за вывод на экран результата вычислений?
- `mov eax, edx` переносит результат вычислений из регистра `edx` в регистр `eax`.
 - `call iprintLF` запускает функцию, которая выводит значение из регистра `eax` на экран с переводом строки.

2.2 Задание для самостоятельной работы

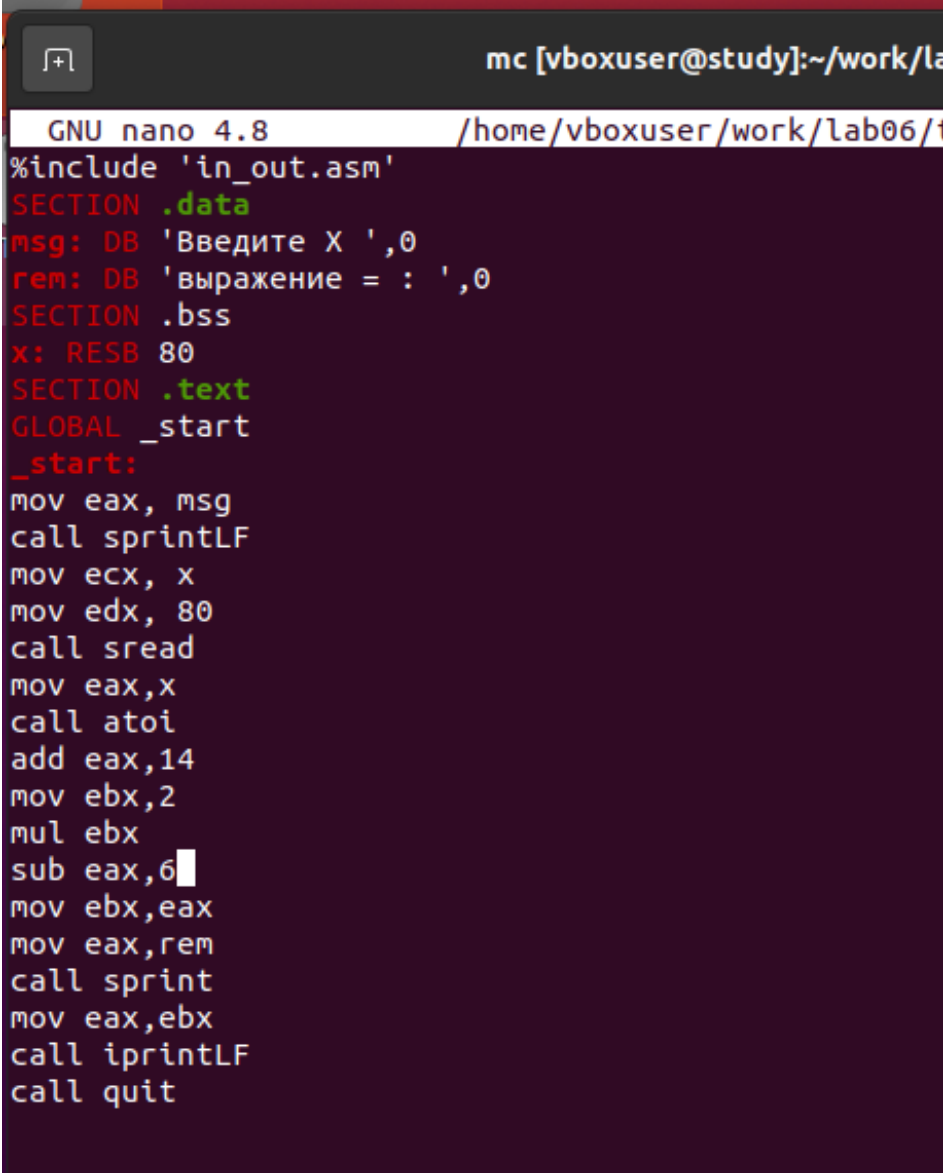
8. Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x , вычислять заданное выражение в зависимости от введенного x , выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы 6.3 вариантов заданий в соответствии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

$$(14 + x) * 2 - 6$$

для

$$x_1 = 1, x_2 = 9$$



```
mc [vboxuser@study]:~/work/lab06/1
GNU nano 4.8 /home/vboxuser/work/lab06/1
%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
add eax, 14
mov ebx, 2
mul ebx
sub eax, 6
mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.16: Редактирование файла task.asm

```
vboxuser@study:~/work/lab06$ nasm -f elf task.asm
vboxuser@study:~/work/lab06$ ld -m elf_i386 task.o -o task
vboxuser@study:~/work/lab06$ ./task
Введите X
1
выражение = : 24
vboxuser@study:~/work/lab06$ ./task
Введите X
9
выражение = : 40
vboxuser@study:~/work/lab06$
```

Рис. 2.17: Компиляция и запуск файла task.asm

3 Выводы

Изучили работу с арифметическими операциями.