

Отчёт по лабораторной работе 6

Дисциплина: Архитектура компьютера

Надир Гасанли

Содержание

| | | |
|----------|--|-----------|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 2.1 | Ответы на вопросы по программе variant.asm | 17 |
| 2.2 | Самостоятельное задание | 18 |
| 3 | Выводы | 21 |

Список иллюстраций

| | | |
|------|---|----|
| 2.1 | Подготовил каталог | 6 |
| 2.2 | Программа в файле lab6-1.asm | 7 |
| 2.3 | Запуск программы lab6-1.asm | 8 |
| 2.4 | Программа в файле lab6-1.asm | 9 |
| 2.5 | Запуск программы lab6-1.asm | 9 |
| 2.6 | Программа в файле lab6-2.asm | 10 |
| 2.7 | Запуск программы lab6-2.asm | 10 |
| 2.8 | Программа в файле lab6-2.asm | 11 |
| 2.9 | Запуск программы lab6-2.asm | 12 |
| 2.10 | Запуск программы lab6-2.asm | 12 |
| 2.11 | Программа в файле lab6-3.asm | 13 |
| 2.12 | Запуск программы lab6-3.asm | 13 |
| 2.13 | Программа в файле lab6-3.asm | 14 |
| 2.14 | Запуск программы lab6-3.asm | 15 |
| 2.15 | Программа в файле variant.asm | 16 |
| 2.16 | Запуск программы variant.asm | 16 |
| 2.17 | Программа в файле task.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

Создал каталог для программ лабораторной работы № 6, перешел в него и создал файл lab6-1.asm. (рис. 2.1)

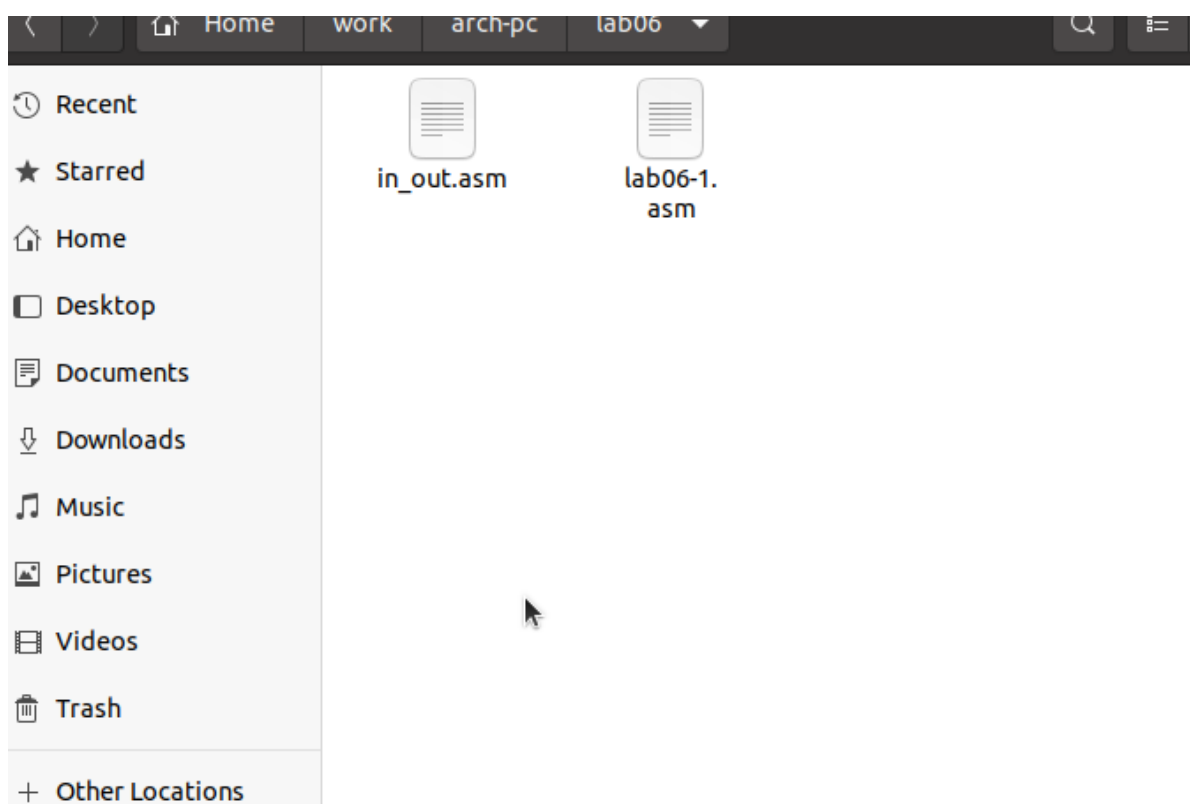
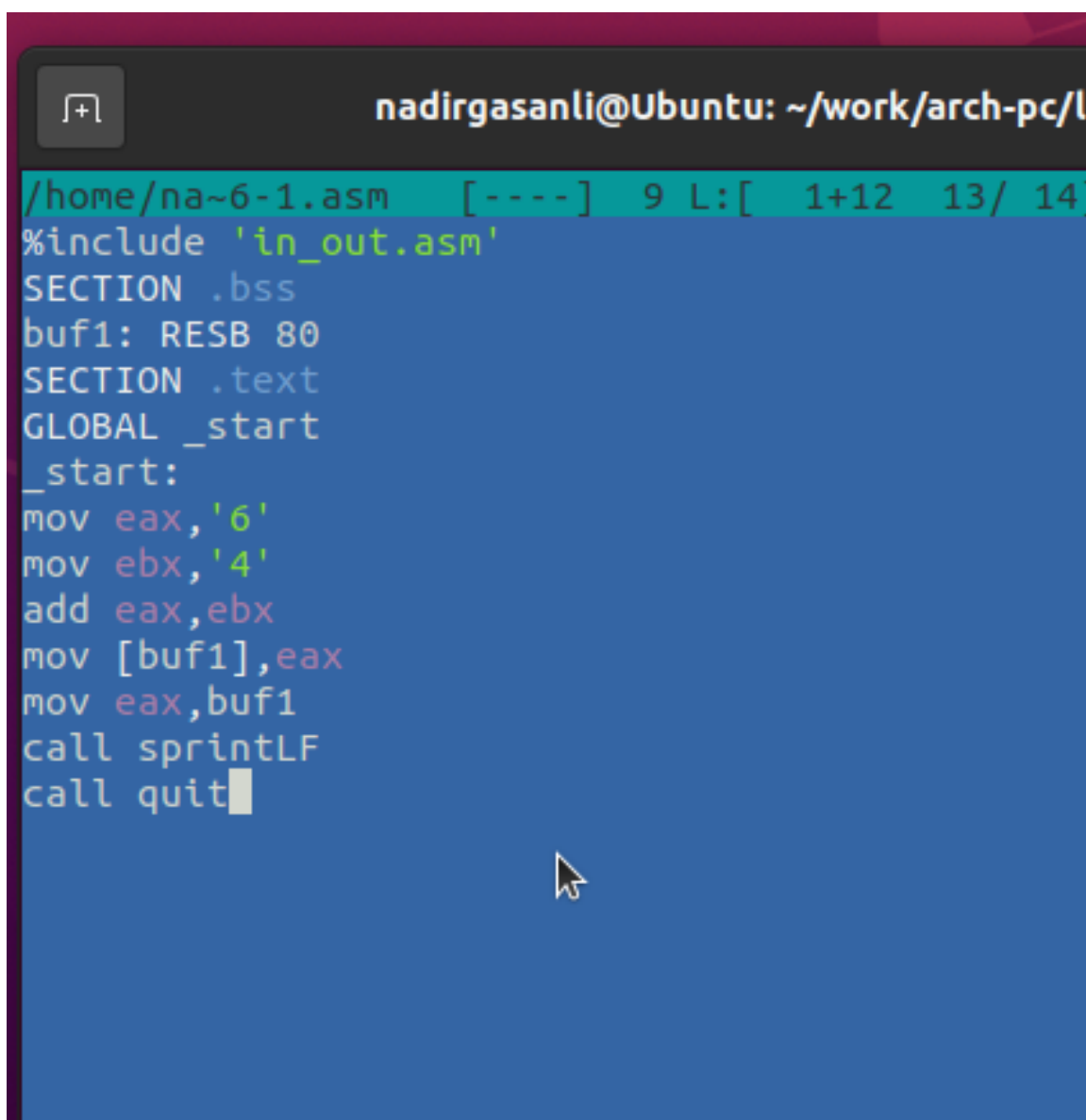


Рис. 2.1: Подготовил каталог

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

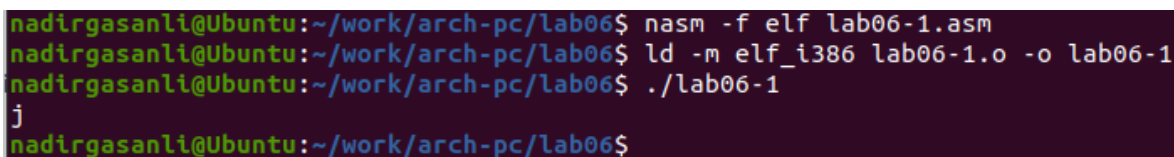


```
nadirgasanli@Ubuntu: ~/work/arch-pc/l
/home/na~6-1.asm [ - - - ] 9 L: [ 1+12 13/ 14
#include 'in_out.asm'
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx
mov [buf1], eax
mov eax, buf1
call sprintLF
call quit
```

Рис. 2.2: Программа в файле lab6-1.asm

В данной программе (рис. 2.2) мы записываем символ '6' в регистр `eax` (`mov eax, '6'`), а символ '4' в регистр `ebx` (`mov ebx, '4'`). Затем мы добавляем значение регистра `ebx` к значению в регистре `eax` (`add eax, ebx`, результат сложения записывается в регистр `eax`). После этого мы выводим результат. Однако, для использования функции `sprintLF`, необходимо, чтобы в регистре `eax` был записан адрес, поэтому мы используем дополнительную переменную. Мы записываем значение регистра `eax` в переменную `buf1` (`mov [buf1], eax`), а затем записываем

адрес переменной buf1 в регистр eax (mov eax, buf1) и вызываем функцию sprintLF.

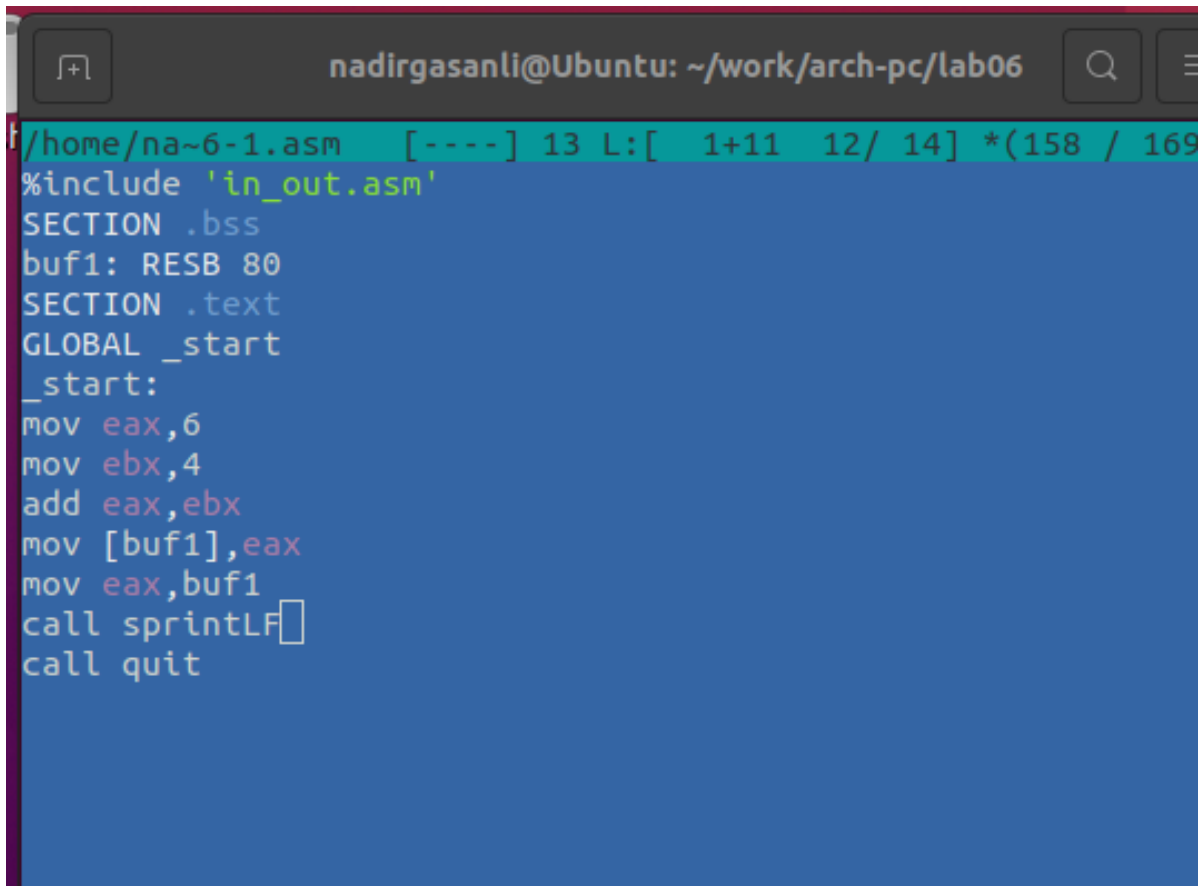


```
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./lab06-1
j
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.3: Запуск программы lab6-1.asm

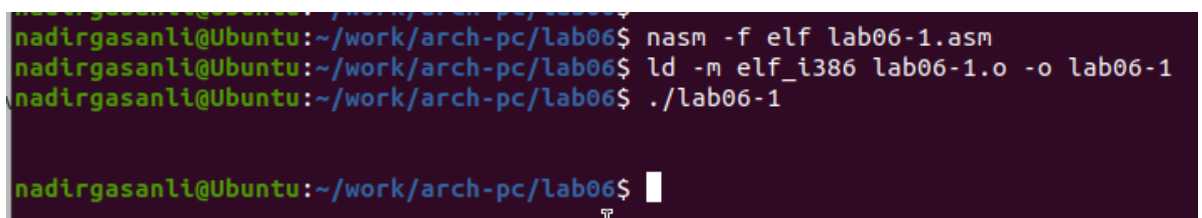
В данном случае, когда мы ожидаем увидеть число 10 при выводе значения регистра eax, фактическим результатом будет символ 'j'. Это происходит из-за того, что код символа '6' равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа '4' равен 00110100 (или 52 в десятичном представлении). Когда мы выполняем команду add eax, ebx, результатом будет сумма кодов - 01101010 (или 106 в десятичном представлении), который соответствует символу 'j'. (рис. 2.3)

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



```
nadirgasanli@Ubuntu: ~/work/arch-pc/lab06
/home/na~6-1.asm  [----] 13 L:[ 1+11 12/ 14] *(158 / 169
#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. 2.4: Программа в файле lab6-1.asm



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

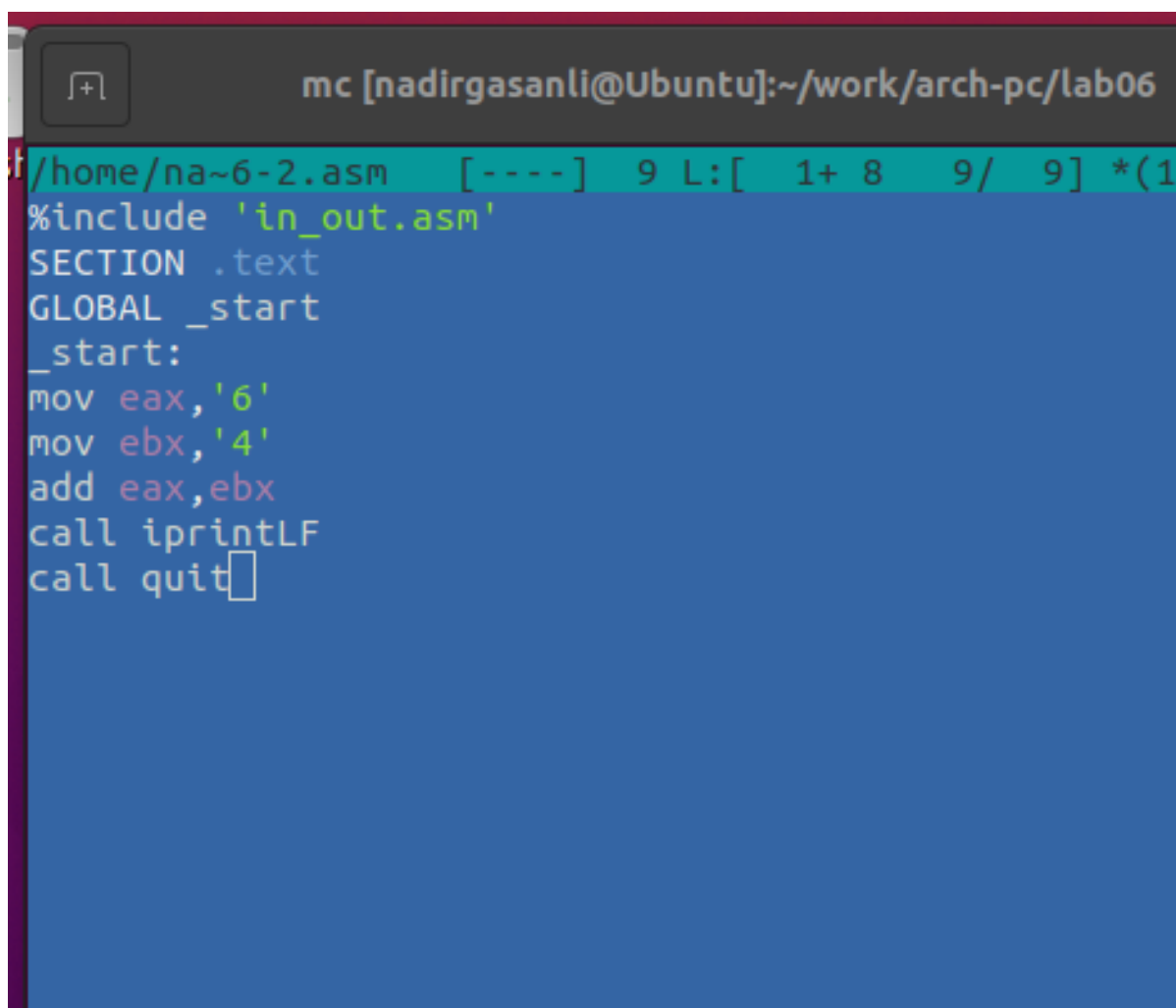
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.5: Запуск программы lab6-1.asm

Как и в предыдущем случае, при выполнении программы мы не получим число 10. Вместо этого выводится символ с кодом 10, который представляет собой символ конца строки (возврат каретки). (рис. 2.5) Этот символ не отображается в консоли, но он добавляет пустую строку.

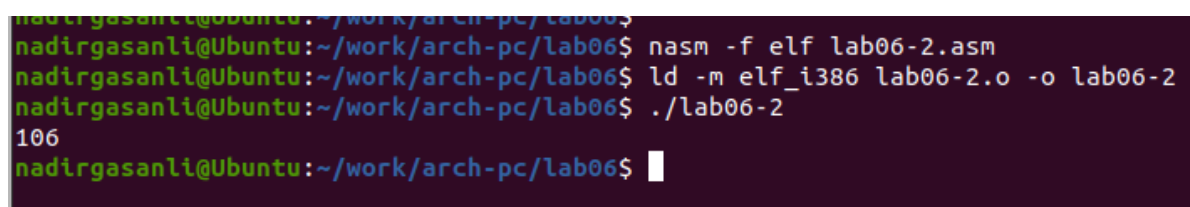
Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно.

Преобразовал текст программы с использованием этих функций. (рис. 2.6)

A screenshot of a code editor window titled 'mc [nadirgasanli@Ubuntu]:~/work/arch-pc/lab06'. The editor shows the content of a file named 'lab6-2.asm'. The code is as follows:

```
1 /home/na~6-2.asm [ - - - - ] 9 L: [ 1+ 8 9/ 9 ] *(1
2 %include 'in_out.asm'
3 SECTION .text
4 GLOBAL _start
5 _start:
6 mov eax,'6'
7 mov ebx,'4'
8 add eax,ebx
9 call iprintLF
10 call quit
```

Рис. 2.6: Программа в файле lab6-2.asm

A screenshot of a terminal window showing the compilation and execution of the assembly program. The commands and their outputs are:

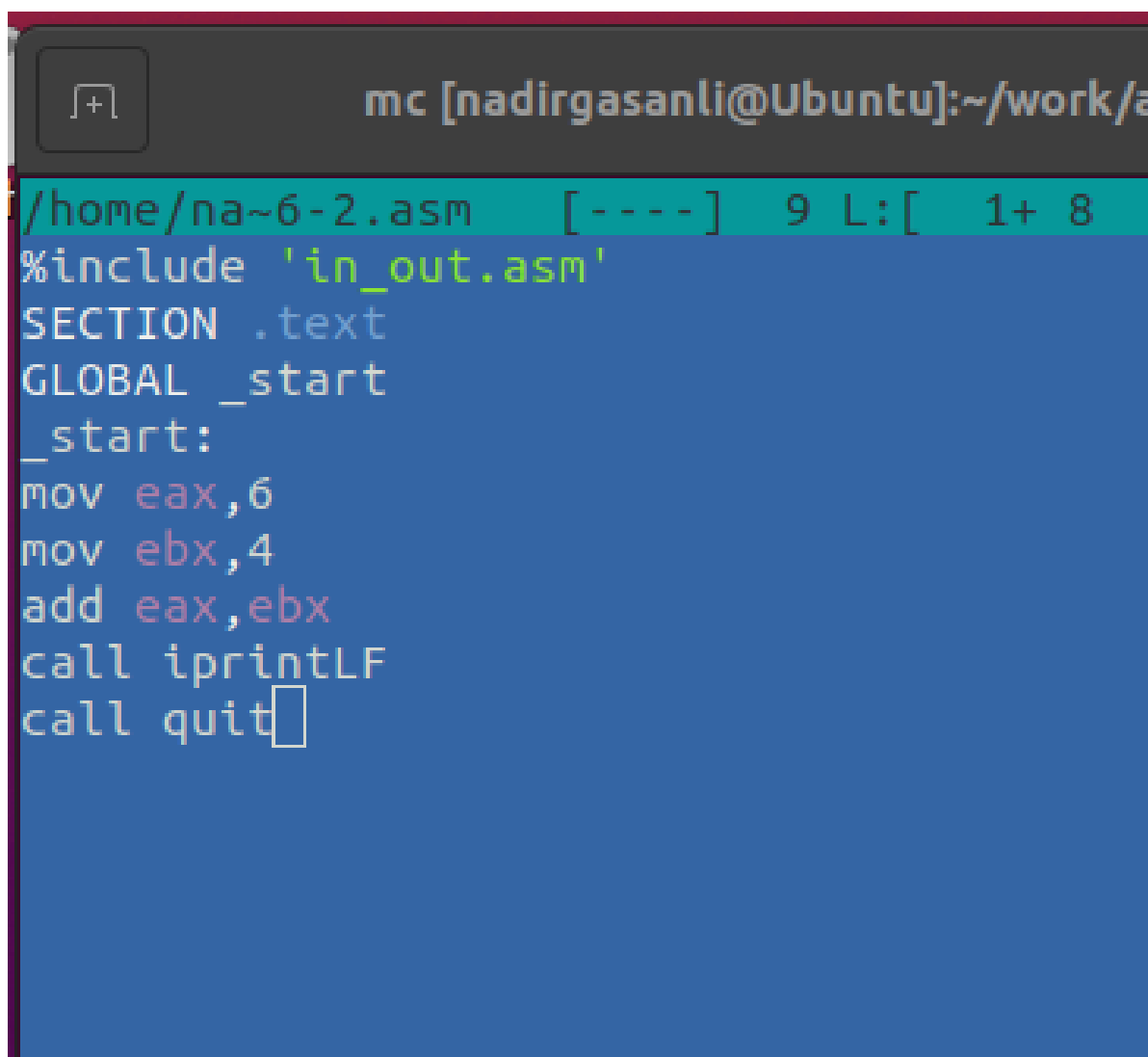
```
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./lab06-2
106
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.7: Запуск программы lab6-2.asm

В результате выполнения программы мы получим число 106. (рис. 2.7) В данном случае, как и в первом случае, команда add складывает коды символов '6'

и '4' ($54+52=106$). Однако, в отличие от предыдущей программы, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

Аналогично предыдущему примеру изменим символы на числа.(рис. 2.8)



```
mc [nadirgasanli@Ubuntu]:~/work/a
/home/na~6-2.asm  [ - - - - ]  9 L:[  1+ 8
%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: Программа в файле lab6-2.asm

Функция `iprintLF` позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10.(рис. 2.9)

```

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

```

Рис. 2.9: Запуск программы lab6-2.asm

Заменяю функцию `iprintLF` на `iprint`. Создаю исполняемый файл и запускаю его. Вывод отличается тем, что нет переноса строки. (рис. 2.10)

```

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

```

Рис. 2.10: Запуск программы lab6-2.asm

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

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

.

```
mc [nadirgasanli@Ubuntu]:~/work/arch-pc/lab06
/home/na~6-3.asm [----] 11 L:[ 1+17 18/ 26] *(258
%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: Программа в файле lab6-3.asm

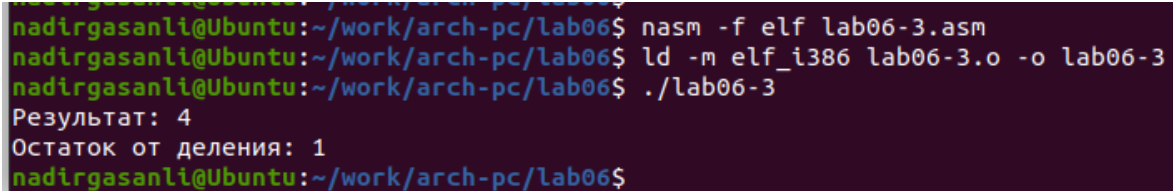
```
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.12: Запуск программы lab6-3.asm

Изменил текст программы для вычисления выражения

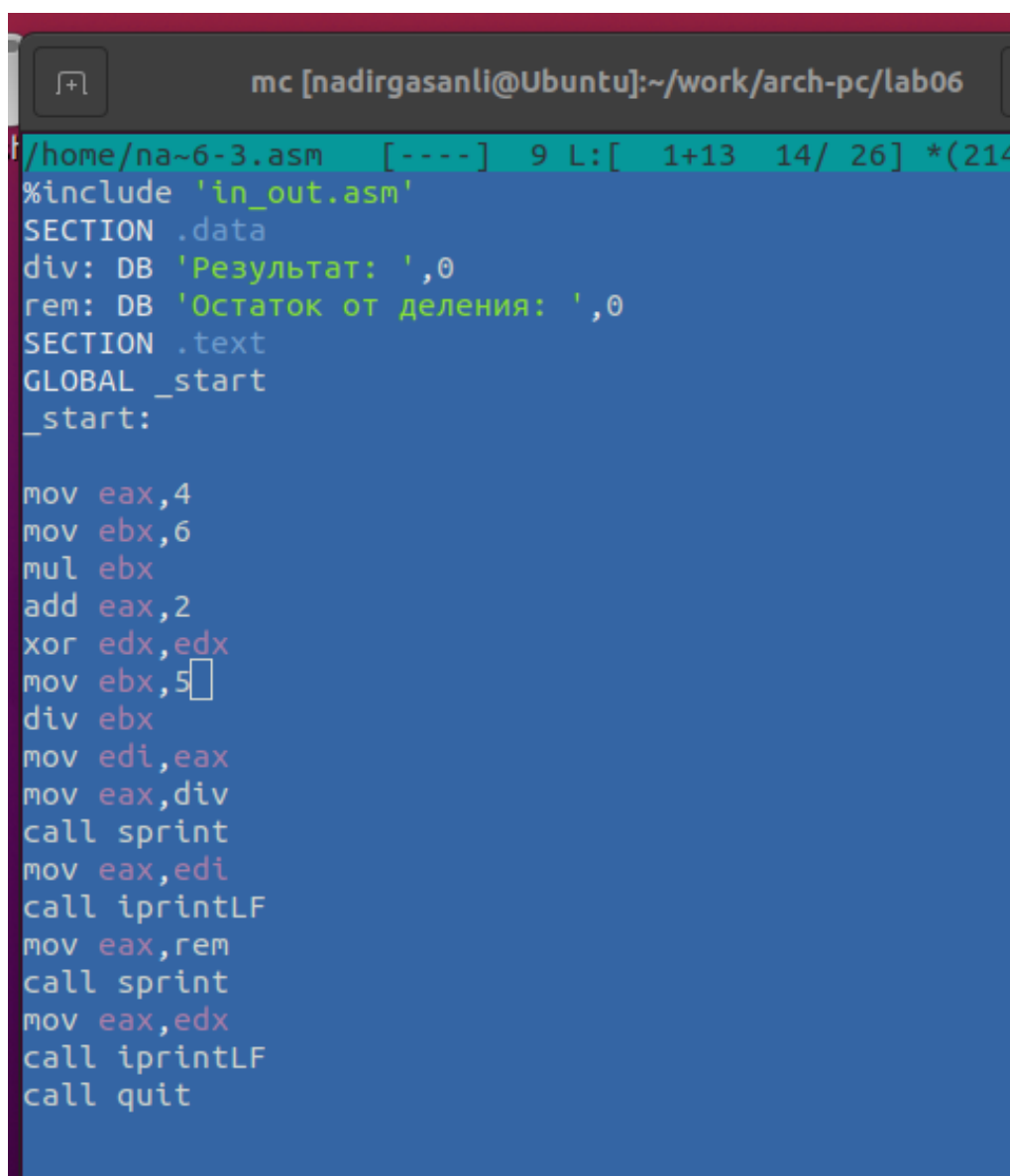
$$f(x) = (4 * 6 + 2) / 5$$

. Создал исполняемый файл и проверил его работу. (рис. 2.13) (рис. 2.14)



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

Рис. 2.13: Программа в файле lab6-3.asm



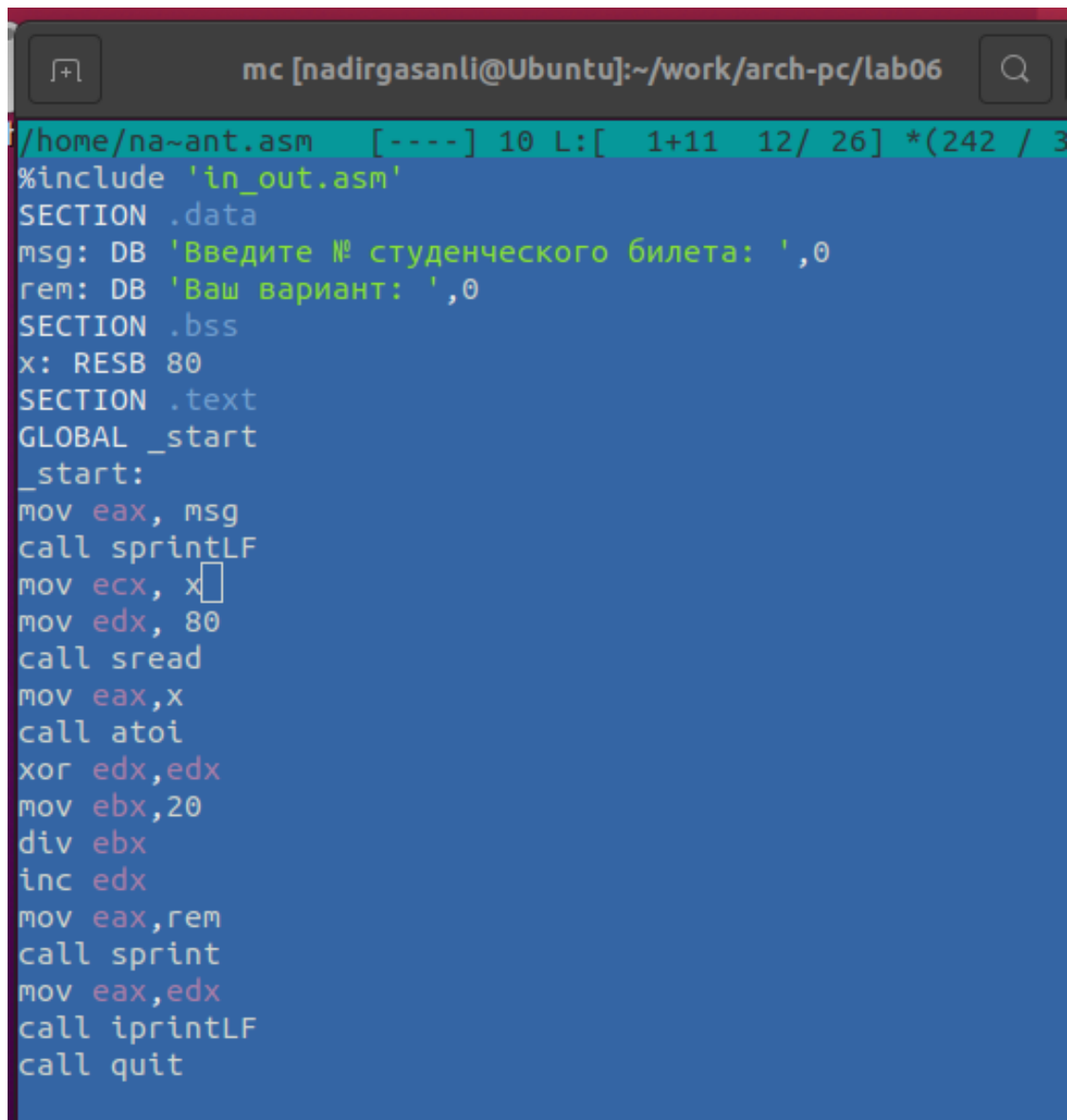
```
mc [nadirgasanli@Ubuntu]:~/work/arch-pc/lab06
/home/na~6-3.asm [----] 9 L:[ 1+13 14/ 26] *(214
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

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)

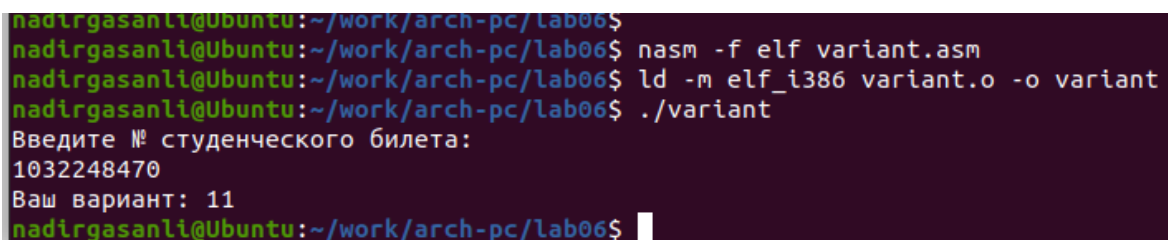
В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована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



The screenshot shows a text editor window titled 'mc [nadirgasanli@Ubuntu]:~/work/arch-pc/lab06'. The code is in assembly format and includes comments in Russian. It defines a data section with a message and a .bss section with a buffer 'x' of size 80. The .text section contains the main logic: it moves the message to 'eax', calls 'sprintLF', moves the buffer address to 'ecx', sets 'edx' to 80, calls 'sread', converts the input to integer with 'atoi', calculates the variant number using division, and finally prints the result with 'iprintLF' before calling 'quit'.

```
/home/na~ant.asm [----] 10 L:[ 1+11 12/ 26] *(242 / 3
#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. 2.15: Программа в файле variant.asm



The screenshot shows a terminal window with the following commands and output: the user runs 'nasm -f elf variant.asm' to compile the assembly code into an object file, then 'ld -m elf_i386 variant.o -o variant' to link it into an executable, and finally './variant' to run it. The program prompts for a student ID, which is entered as '1032248470', and then for a variant number, which is entered as '11'.

```
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant.asm
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1032248470
Ваш вариант: 11
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.16: Запуск программы variant.asm

2.1 Ответы на вопросы по программе variant.asm

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

Строка “mov eax, mem” перекладывает в регистр значение переменной с фразой “Ваш вариант:”

Строка “call sprint” вызывает подпрограмму вывода строки

2. Для чего используются следующие инструкции?

Инструкция “nasm” используется для компиляции кода на языке ассемблера NASM

Инструкция “mov ecx, x” используется для перемещения значения переменной x в регистр ecx

Инструкция “mov edx, 80” используется для перемещения значения 80 в регистр edx

Инструкция “call sread” вызывает подпрограмму для считывания значения студенческого билета из консоли

3. Для чего используется инструкция “call atoi”?

Инструкция “call atoi” используется для преобразования введенных символов в числовой формат

4. Какие строки листинга отвечают за вычисления варианта?

Строка “xor edx, edx” обнуляет регистр edx

Строка “mov ebx, 20” записывает значение 20 в регистр ebx

Строка “div ebx” выполняет деление номера студенческого билета на 20

Строка “inc edx” увеличивает значение регистра edx на 1

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

Остаток от деления записывается в регистр `edx`

6. Для чего используется инструкция `inc edx`?

Инструкция `inc edx` используется для увеличения значения в регистре `edx` на 1, в соответствии с формулой вычисления варианта

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

Строка `mov eax, edx` перекладывает результат вычислений в регистр `eax`

Строка `call iprintLF` вызывает подпрограмму для вывода значения на экран

2.2 Самостоятельное задание

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.

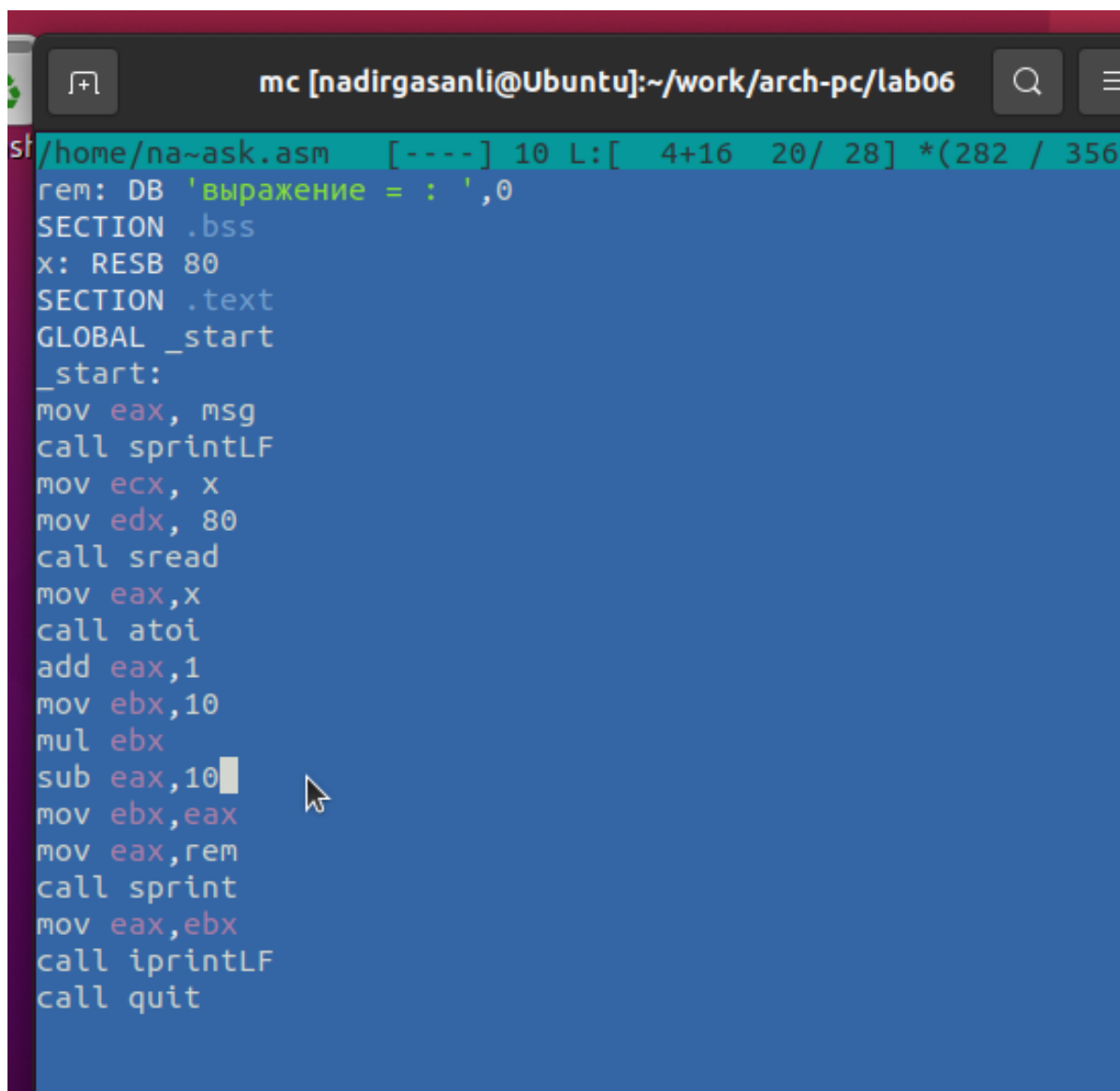
Получили вариант 11 -

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

для

$$x_1 = 1, x_2 = 7$$

(рис. 2.17) (рис. 2.18)



The image shows a terminal window with a dark blue background. The title bar at the top reads "mc [nadirgasanli@Ubuntu]:~/work/arch-pc/lab06". The terminal content displays assembly code for a file named "task.asm". The code includes a data definition for "rem", section directives for ".bss" and ".text", a global symbol "_start", and a series of assembly instructions for string handling and arithmetic. A mouse cursor is visible over the line "sub eax,10".

```
st/home/na~ask.asm [----] 10 L:[ 4+16 20/ 28] *(282 / 356
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,1
mov ebx,10
mul ebx
sub eax,10
mov ebx,eax
mov eax,rem
call sprintf
mov eax,ebx
call iprintLF
call quit
```

Рис. 2.17: Программа в файле task.asm

```
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ nasm -f elf task.asm
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./task
Введите X
1
выражение = : 10
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$ ./task
Введите X
7
выражение = : 70
nadirgasanli@Ubuntu:~/work/arch-pc/lab06$
```

Рис. 2.18: Запуск программы task.asm

3 Выводы

Изучили работу с арифметическими операциями.