

Отчёт по лабораторной работе 6

Дисциплина: архитектура компьютера

Синюшко Элза Камировна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Символьные и численные данные в NASM	9
4.2	Выполнение арифметических операций в NASM	17
4.2.1	Ответы на вопросы по программе variant.asm	22
4.3	Выполнение заданий для самостоятельной работы	23
5	Источники	26

Список иллюстраций

4.1	Изменение кода lab6-1.asm	10
4.2	Компиляция текста программы lab6-1.asm	11
4.3	Изменение кода lab6-1.asm	12
4.4	Компиляция текста программы lab6-1.asm	12
4.5	Изменение кода lab6-2.asm	13
4.6	Компиляция текста программы lab6-2.asm	14
4.7	Изменение кода lab6-2.asm	15
4.8	Компиляция текста программы lab6-2.asm	16
4.9	Изменение кода lab6-2.asm	16
4.10	Компиляция текста программы lab6-2.asm	17
4.11	Изменение кода lab6-3.asm	18
4.12	Компиляция текста программы lab6-3.asm	18
4.13	Изменение кода lab6-3.asm	19
4.14	Компиляция текста программы lab6-3.asm	20
4.15	Изменение кода variant.asm	21
4.16	Компиляция текста программы variant.asm	22
4.17	Изменение кода task.asm	24
4.18	Компиляция текста программы task.asm	25

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Изучение арифметических операций в ассемблере
2. Изучение типов данных в ассемблере
3. Выполнение заданий, рассмотрение примеров
4. Выполнение заданий для самостоятельной работы

3 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.

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`.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда `mul` (от англ. `multiply` – умножение).

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре `EAX, AX` или `AL`, а результат помещается в регистры `EDX:EAX`, `DX:AX` или `AX`, в зависимости от размера операнда.

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

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера

делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 6.2.

4 Выполнение лабораторной работы

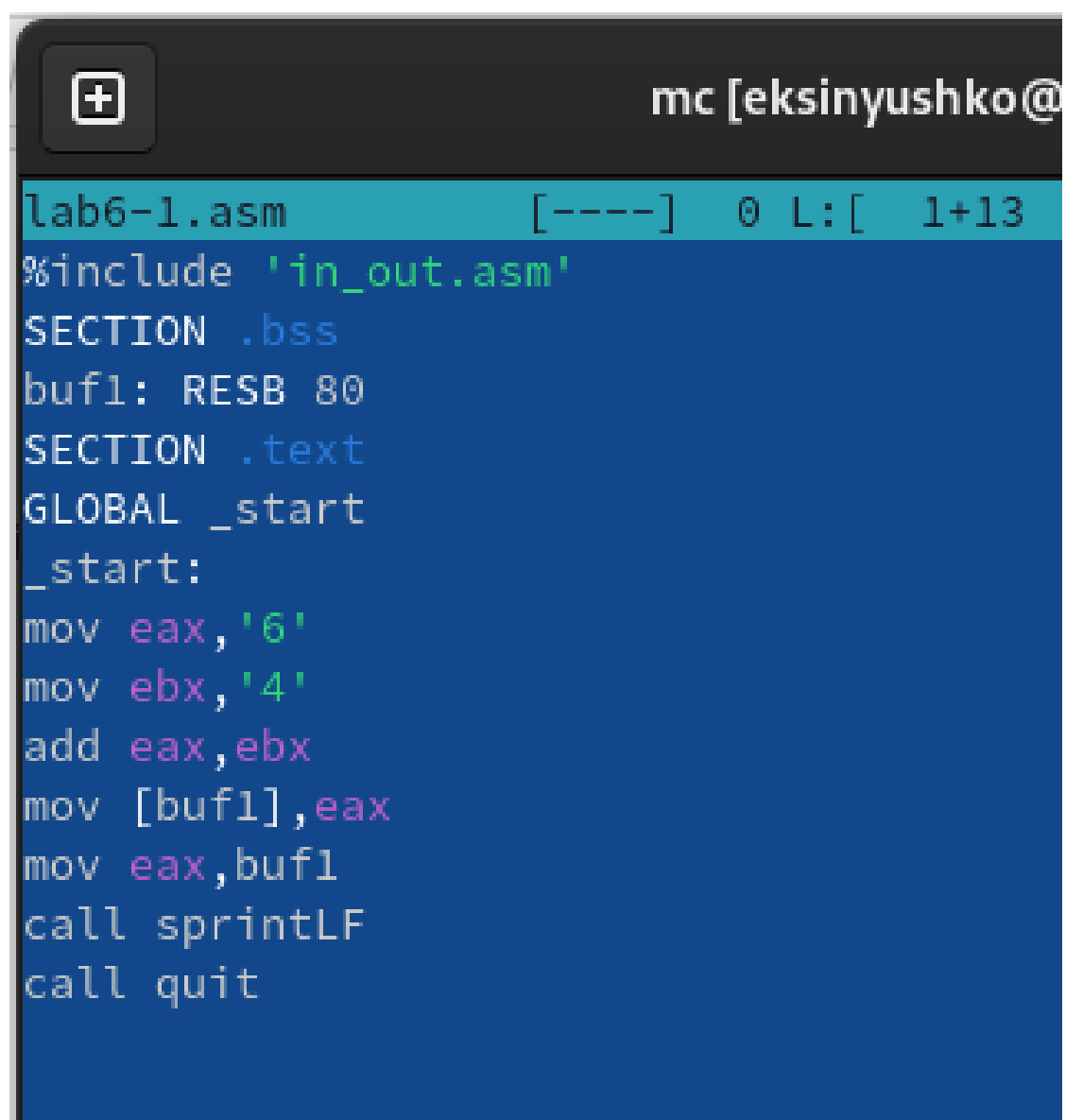
4.1 Символьные и численные данные в NASM

Я создала каталог для программ лабораторной работы № 6, перешла в него и создала файл lab6-1.asm.

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

В данной программе (рис. [4.1]) в регистр еах записан символ 6 (mov еах, '6'), а в регистр ебх символ 4 (mov ебх, '4'). Затем прибавляем значение регистра ебх к значению в регистре еах (add еах, ебх, результат сложения запишется в регистр еах). Затем вывод результата (рис. [4.2]).

Так как для работы функции sprintLF в регистр еах должен быть записан адрес, используем дополнительную переменную. Записали значение регистра еах в переменную buf1 (mov [buf1], еах), а затем записали адрес переменной buf1 в регистр еах (mov еах, buf1) и вызвали функцию sprintLF.



```
lab6-1.asm [----] 0 L:[ 1+13
#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
```

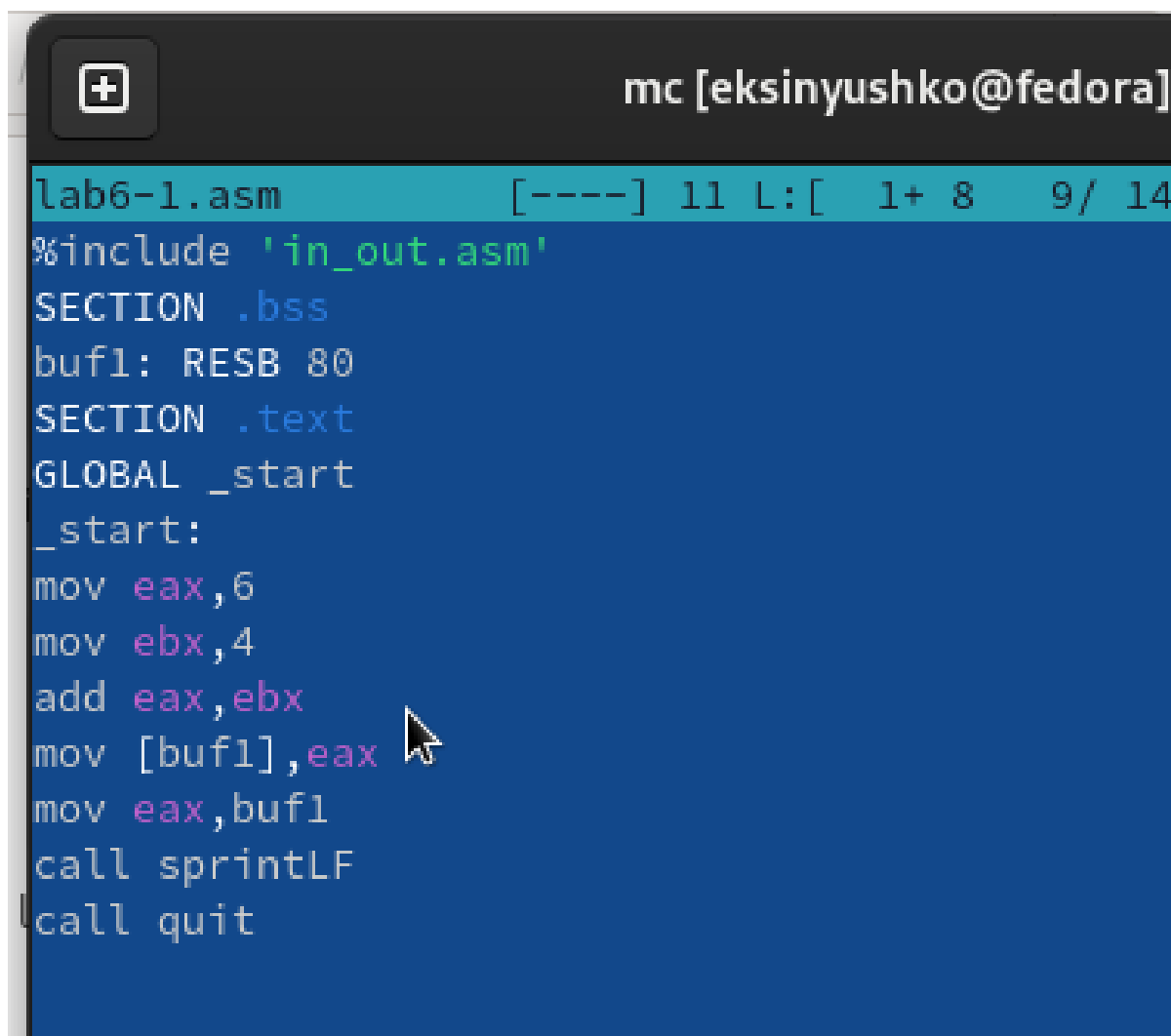
Рис. 4.1: Изменение кода lab6-1.asm

```
[eksinyushko@fedora lab06]$ nasm -f elf lab6-1.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1
[eksinyushko@fedora lab06]$ ./lab6-1
j
[eksinyushko@fedora lab06]$
```

Рис. 4.2: Компиляция текста программы lab6-1.asm

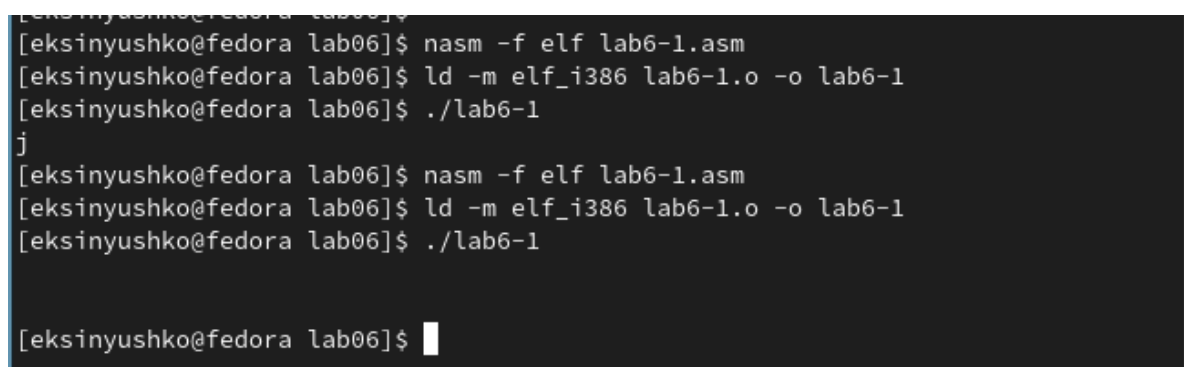
В данном случае, при выводе значения регистра еах, я ожидала увидеть число 10. Однако, результатом был символ 'j'. Это произошло потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда add еах,ебх записала в регистр еах сумму кодов – 01101010 (106), что в свою очередь является кодом символа 'j'.

Далее я изменила текст программы и вместо символов записала в регистры числа (рис. [4.3])



```
lab6-1.asm [----] 11 L:[ 1+ 8 9/ 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 sprintf
call quit
```

Рис. 4.3: Изменение кода lab6-1.asm



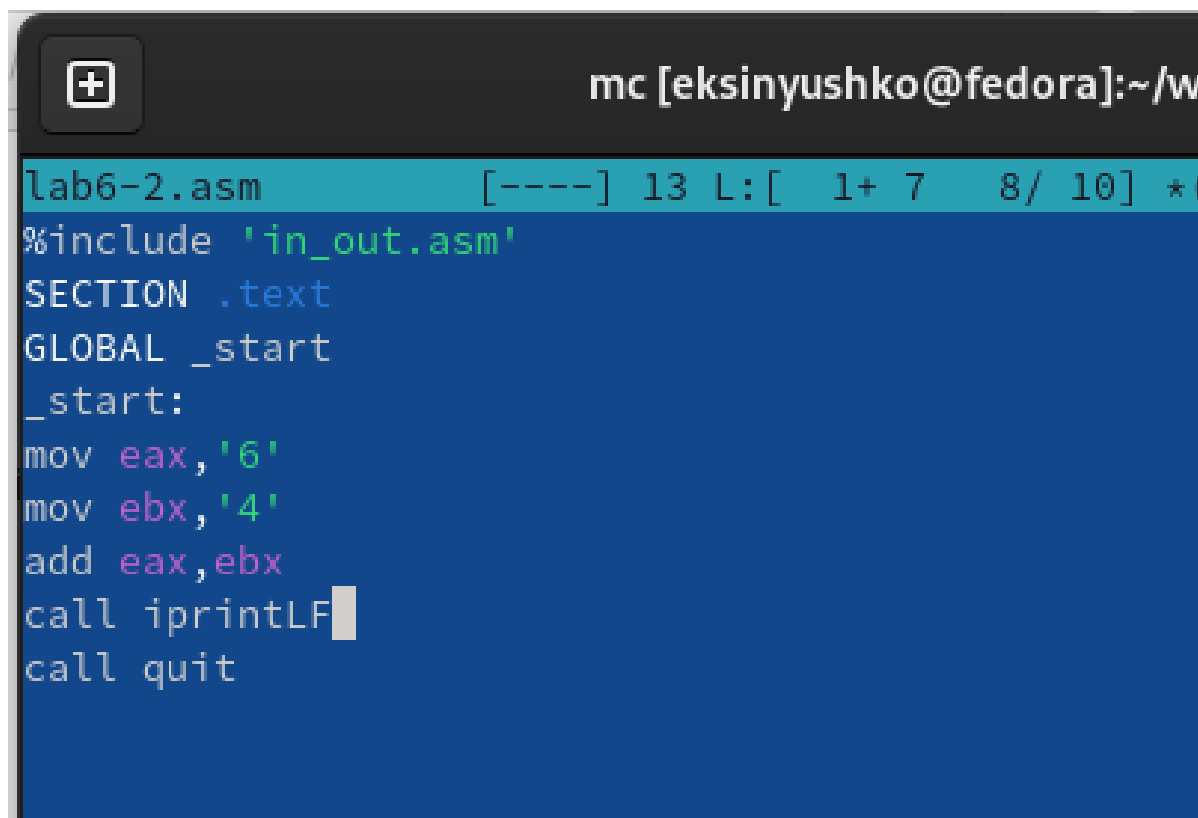
```
[eksinyushko@fedora lab06]$ nasm -f elf lab6-1.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1
[eksinyushko@fedora lab06]$ ./lab6-1
j
[eksinyushko@fedora lab06]$ nasm -f elf lab6-1.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1
[eksinyushko@fedora lab06]$ ./lab6-1

[eksinyushko@fedora lab06]$
```

Рис. 4.4: Компиляция текста программы lab6-1.asm

В процессе выполнения программы мы не получили ожидаемое число 10. Вместо этого был выведен символ с кодом 10 (рис. [4.4]). Это символ конца строки (возврат каретки), который в консоли не отображается, но добавляет пустую строку.

В файле `in_out.asm` реализованы подпрограммы для работы с числами и преобразования символов ASCII. Я модифицировала текст программы с использованием этих функций (рис. [4.5]).



```
mc [eksinyushko@fedora]:~/w
lab6-2.asm [-----] 13 L: [ 1+ 7 8/ 10] *
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

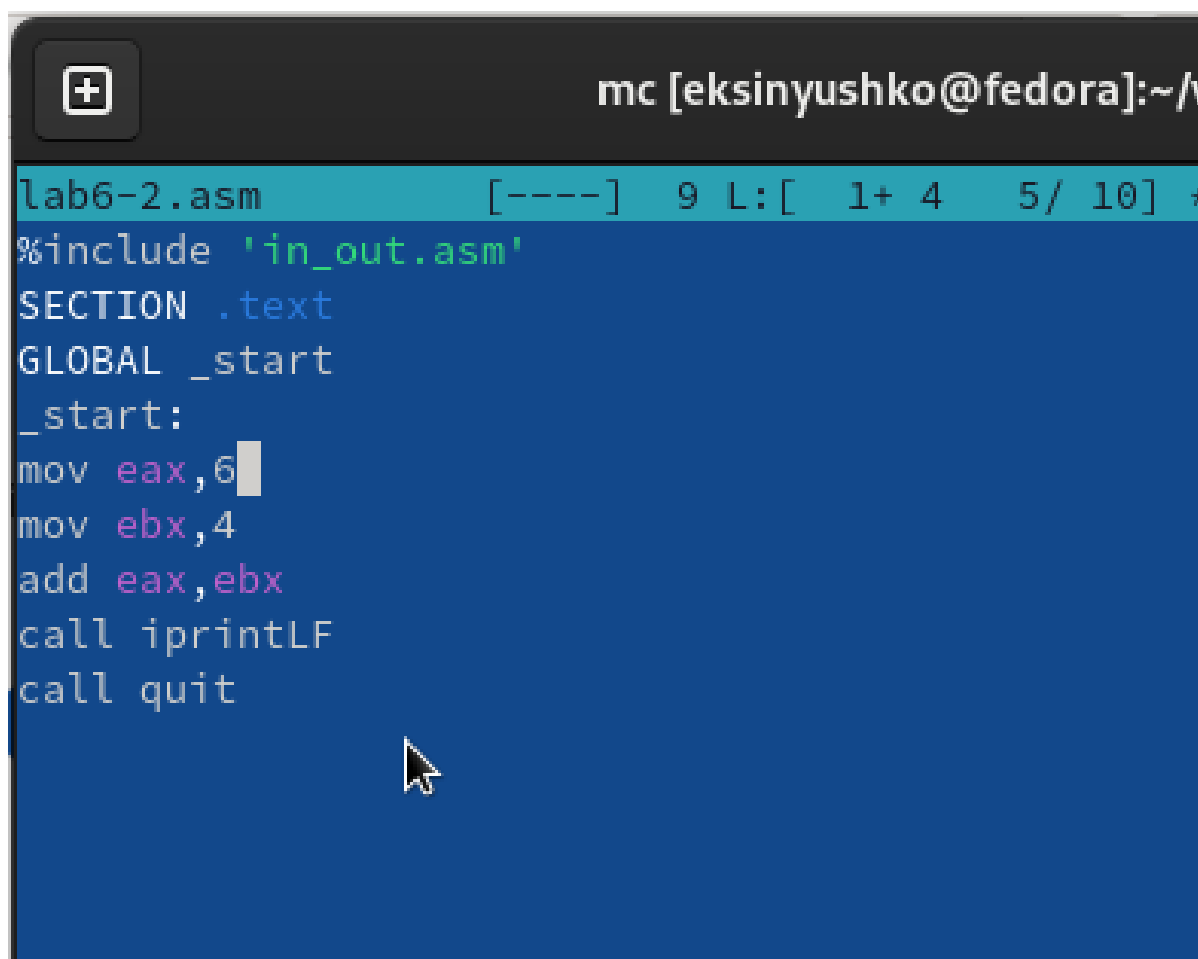
Рис. 4.5: Изменение кода lab6-2.asm

```
[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
106
[eksinyushko@fedora lab06]$
```

Рис. 4.6: Компиляция текста программы lab6-2.asm

В результате выполнения обновленной программы было выведено число 106 (рис. [4.6]). Здесь, как и в первом случае, команда `add` складывает коды символов '6' и '4' ($54 + 52 = 106$). Но в отличие от предыдущей версии, функция `iprintLF` позволяет напечатать само число, а не символ с соответствующим кодом.

По аналогии с предыдущим примером, я заменила символы на числа (рис. [4.7]).



```
lab6-2.asm [-----] 9 L: [ 1+ 4 5/ 10] *
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.7: Изменение кода lab6-2.asm

Функция iprintLF позволяет выводить числа, и на этот раз в качестве операндов использовались именно числа, а не коды символов. В результате мы получили число 10 (рис. [4.8]).

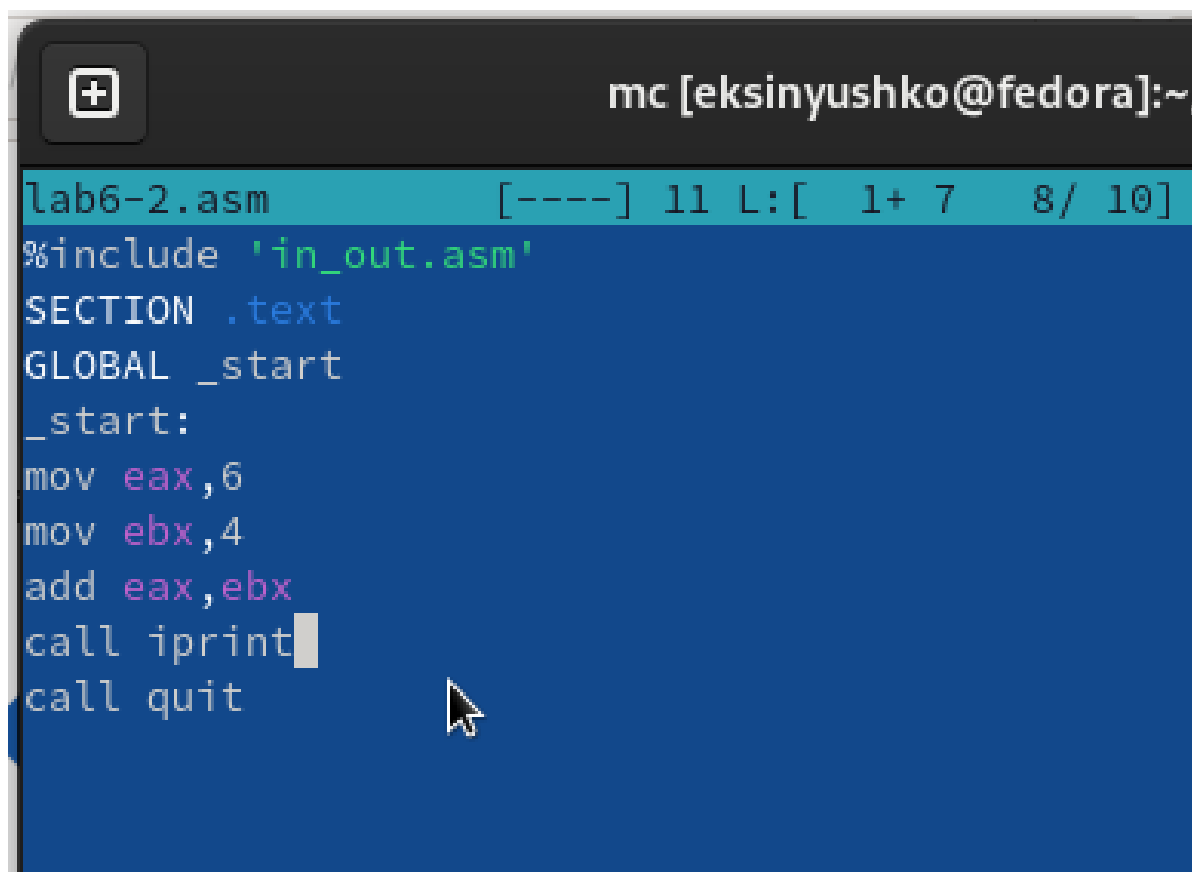
```

[eksinyushko@fedora lab06]$
[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
106
[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
10
[eksinyushko@fedora lab06]$

```

Рис. 4.8: Компиляция текста программы lab6-2.asm

Далее я заменила функцию `iprintLF` на `iprint`, создала исполняемый файл и запустила его. Вывод теперь отличается отсутствием перехода на новую строку (рис. [4.9]).



```

mc [eksinyushko@fedora]:~/
lab6-2.asm [-----] 11 L:[ 1+ 7 8/ 10]
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit

```

Рис. 4.9: Изменение кода lab6-2.asm


```

[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
106
[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
10
[eksinyushko@fedora lab06]$ nasm -f elf lab6-2.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[eksinyushko@fedora lab06]$ ./lab6-2
10[eksinyushko@fedora lab06]$
[eksinyushko@fedora lab06]$

```

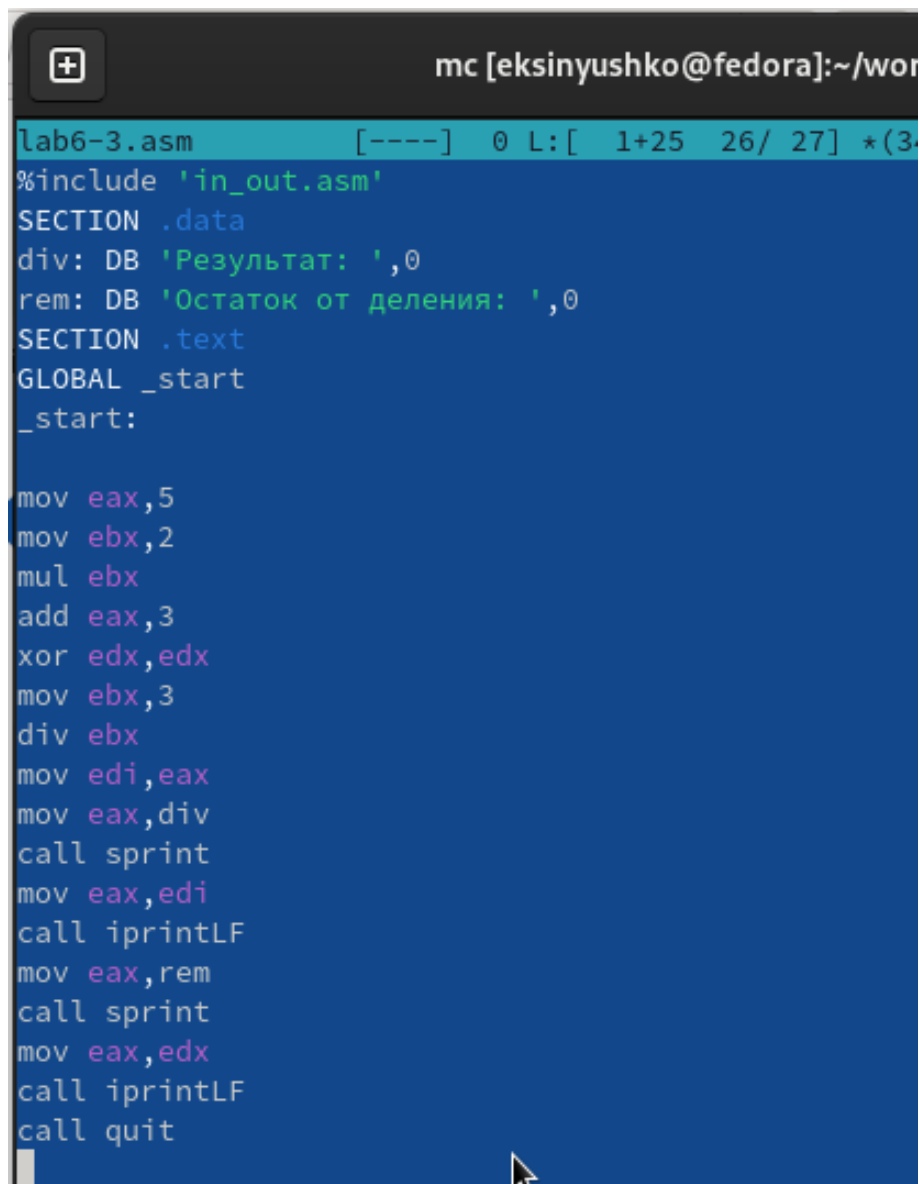
Рис. 4.10: Компиляция текста программы lab6-2.asm

4.2 Выполнение арифметических операций в NASM

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

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

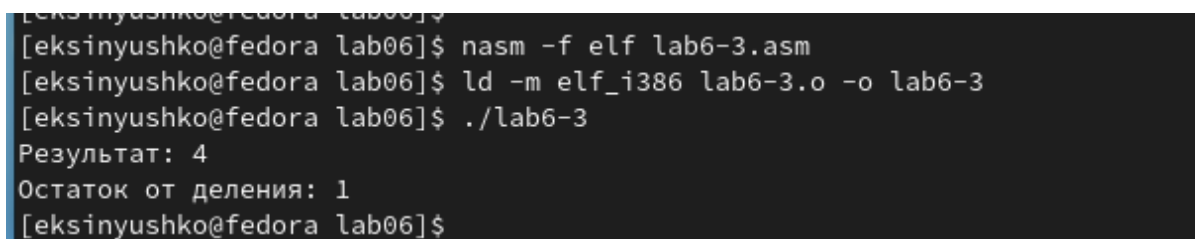
. (рис. [4.11]) (рис. [4.12])



```
mc [eksinyushko@fedora]:~/wor
lab6-3.asm [----] 0 L: [ 1+25 26/ 27] *(3.
#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. 4.11: Изменение кода lab6-3.asm



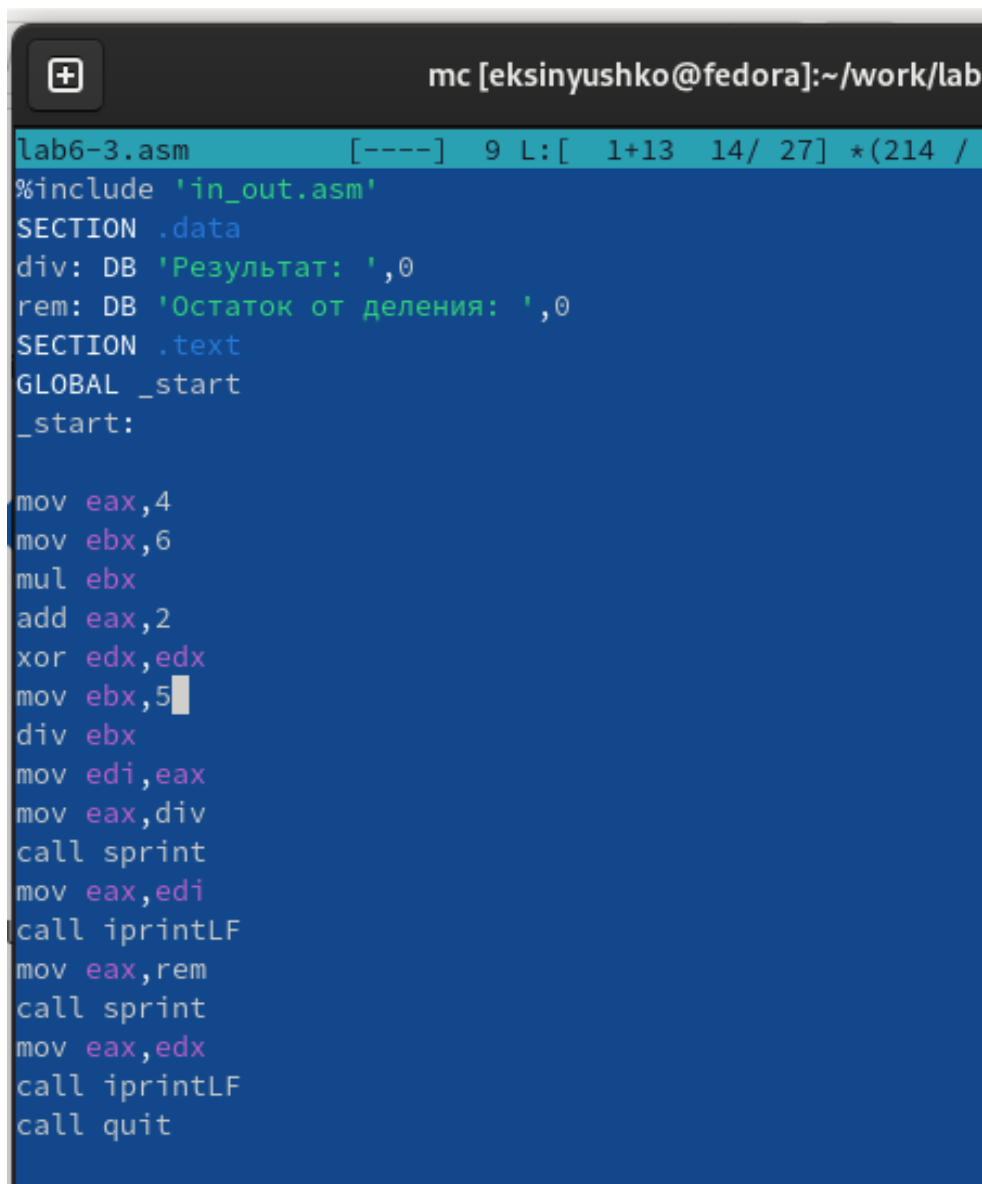
```
[eksinyushko@fedora lab06]$
[eksinyushko@fedora lab06]$ nasm -f elf lab6-3.asm
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3
[eksinyushko@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[eksinyushko@fedora lab06]$
```

Рис. 4.12: Компиляция текста программы lab6-3.asm

Изменила текст программы для вычисления выражения

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

. Создал исполняемый файл и проверил его работу. (рис. [4.13]) (рис. [4.14])



```
mc [eksinyushko@fedora]:~/work/lab
lab6-3.asm [----] 9 L: [ 1+13 14/ 27] *(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
```

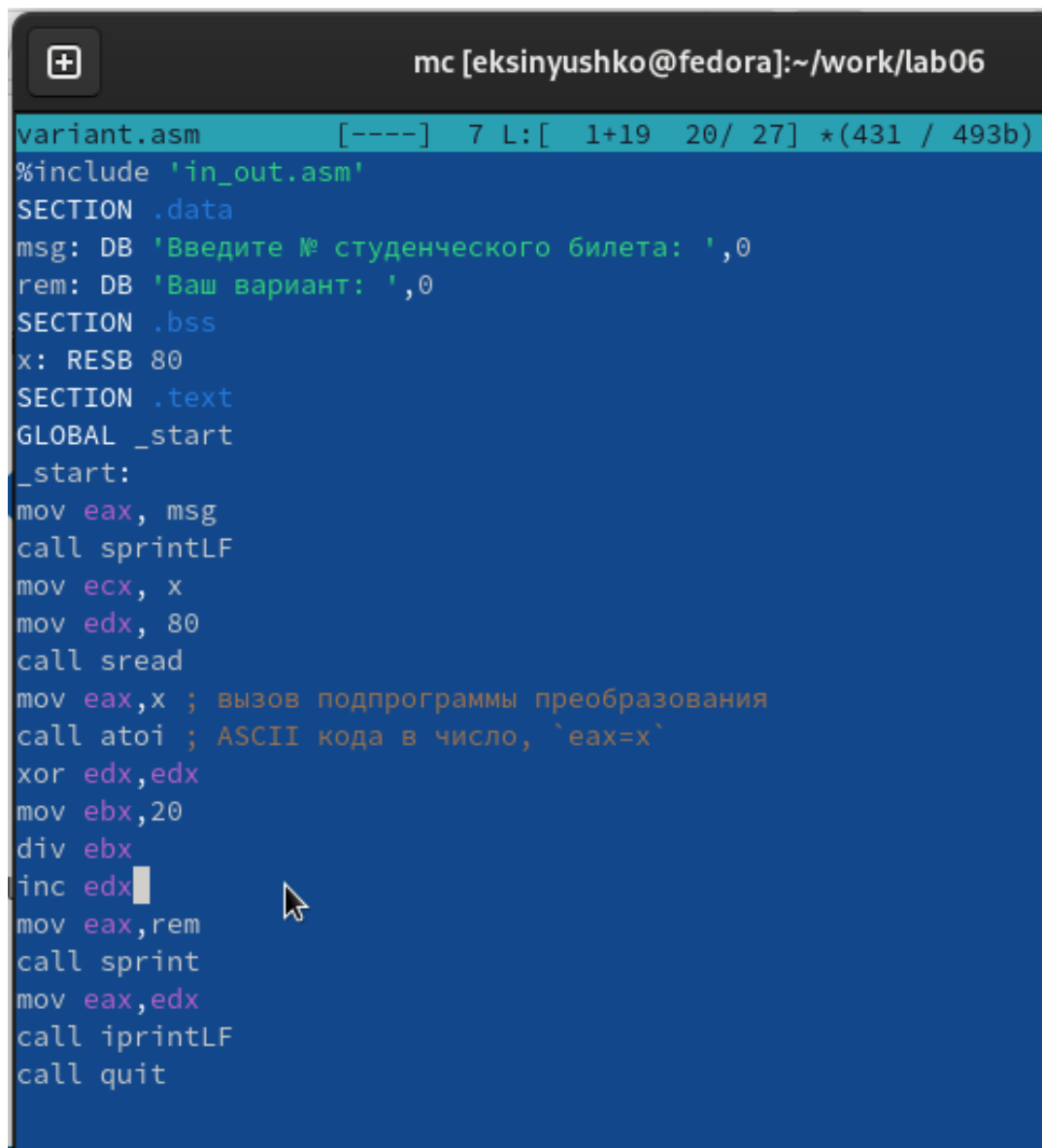
Рис. 4.13: Изменение кода lab6-3.asm

```
[eksinyushko@fedora lab06]$  
[eksinyushko@fedora lab06]$ nasm -f elf lab6-3.asm  
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3  
[eksinyushko@fedora lab06]$ ./lab6-3  
Результат: 4  
Остаток от деления: 1  
[eksinyushko@fedora lab06]$  
[eksinyushko@fedora lab06]$ nasm -f elf lab6-3.asm  
[eksinyushko@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3  
[eksinyushko@fedora lab06]$ ./lab6-3  
Результат: 5  
Остаток от деления: 1  
[eksinyushko@fedora lab06]$
```

Рис. 4.14: Компиляция текста программы lab6-3.asm

В качестве еще одного примера давайте рассмотрим программу для вычисления варианта задания на основе номера студенческого билета.

В этом случае число, над которым нужно выполнять арифметические операции, вводится с клавиатуры. Как я уже отмечала ранее, ввод с клавиатуры осуществляется в символьном виде. Для корректной работы арифметических операций в NASM эти символы необходимо преобразовать в числовой формат. С этой целью можно использовать функцию `atoi` из файла `in_out.asm` (рис. [4.15]) (рис. [4.16]). Она конвертирует строку символов в эквивалентное decimal число.



```
mc [eksinyushko@fedora]:~/work/lab06
variant.asm [-----] 7 L:[ 1+19 20/ 27] *(431 / 493b)
#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 ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
```

Рис. 4.15: Изменение кода variant.asm

```
[eksinyushko@fedora lab06]$  
[eksinyushko@fedora lab06]$ nasm -f elf variant.asm  
[eksinyushko@fedora lab06]$ ld -m elf_i386 variant.o -o variant  
[eksinyushko@fedora lab06]$ ./variant  
Введите № студенческого билета:  
1132239099  
Ваш вариант: 20  
[eksinyushko@fedora lab06]$
```

Рис. 4.16: Компиляция текста программы variant.asm

4.2.1 Ответы на вопросы по программе variant.asm

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

Строки, отвечающие за вывод сообщения “Ваш вариант:”, - это `mov eax, 7em` для помещения фразы в регистр `eax` и `call sprint` для вызова подпрограммы вывода строки.

2. Для чего используются следующие инструкции?

- `mov ecx, x` - сохранение регистра `ecx` в переменной `x`
- `mov edx, 80` - присваивание значения 80 регистру `edx`
- `call sread` - вызов подпрограммы для считывания данных из консоли

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

Инструкция `call atoi` используется для преобразования введённых символов в числовой формат.

4. Какие строки листинга отвечают за вычисления варианта?

- `xor edx, edx` - обнуление регистра `edx`

- `mov ebx, 20` - присваивание значения 20 регистру `ebx`
- `div ebx` - деление номера студента на 20
- `inc edx` - увеличение `edx` на 1

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

При выполнении `div ebx` остаток от деления помещается в регистр `edx`.

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

Инструкция `inc edx` увеличивает значение регистра `edx` на 1, что нужно для вычисления варианта по формуле.

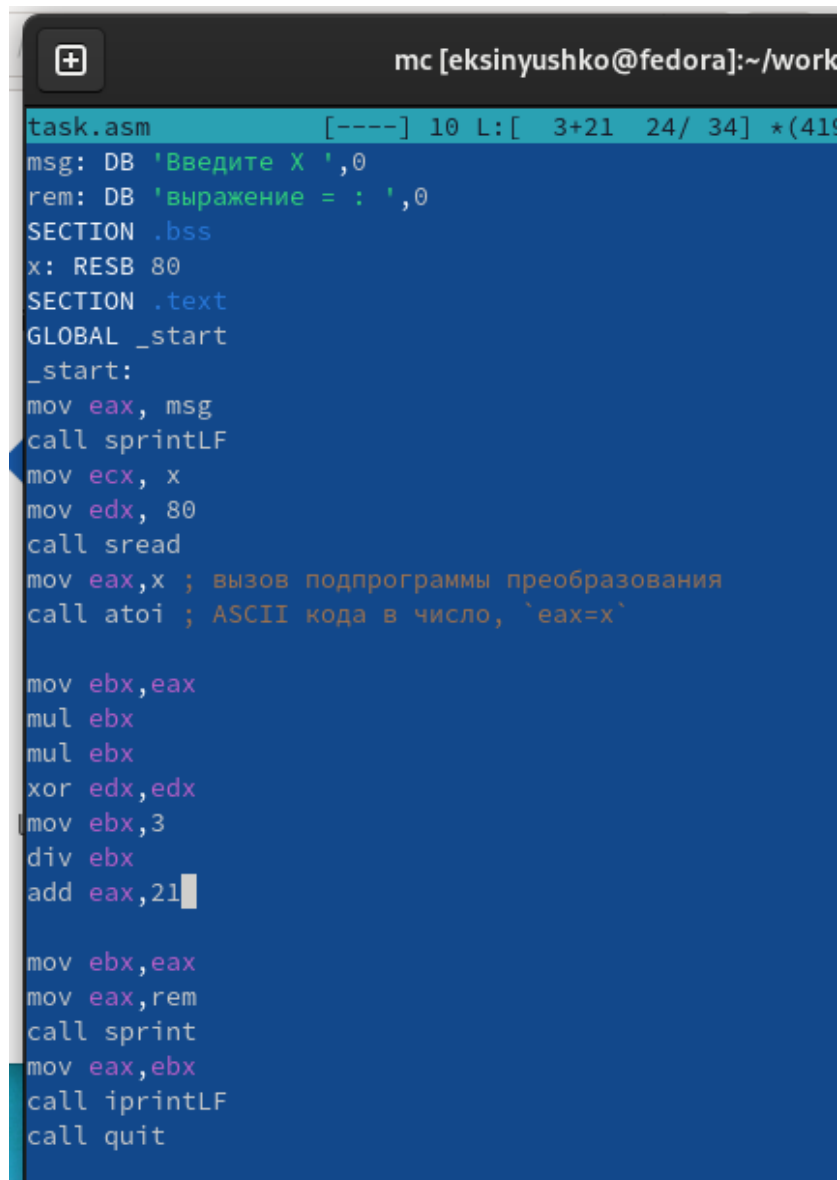
7. Какие строки листинга отвечают за вывод на экран результата вычислений?

- `mov eax, edx` - помещение результата в регистр `eax`
- `call iprintLF` - вызов подпрограммы вывода

4.3 Выполнение заданий 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. (рис. [4.17]) (рис. [4.18]) Создайте исполняемый файл и проверьте его работу для значений x_1 и x_2 из 6.3.

Вариант 20 - $x^3/3 + 21$ для $x = 1, x = 3$



```
task.asm [----] 10 L:[ 3+21 24/ 34] *(419
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 ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`

mov ebx, eax
mul ebx
mul ebx
xor edx, edx
mov ebx, 3
div ebx
add eax, 21

mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 4.17: Изменение кода task.asm


```
[eksinyushko@fedora lab06]$  
[eksinyushko@fedora lab06]$ nasm -f elf task.asm  
[eksinyushko@fedora lab06]$ ld -m elf_i386 task.o -o task  
[eksinyushko@fedora lab06]$ ./task  
Введите X  
1  
выражение = : 21  
[eksinyushko@fedora lab06]$ ./task  
Введите X  
3  
выражение = : 30  
[eksinyushko@fedora lab06]$  
[eksinyushko@fedora lab06]$
```

Рис. 4.18: Компиляция текста программы task.asm

Программа считает верно.

5 Источники

1. Архитектура ЭВМ - Материалы курса