

# **Отчёт по лабораторной работе №7. Арифметические операции в Nasm**

**дисциплина: Архитектура компьютера**

Ибатулина Дарья Эдуардовна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
3.1	Адресация в Nasm . . . . .	6
3.2	Арифметические операции в NASM . . . . .	7
3.2.1	Целочисленное сложение add . . . . .	7
3.2.2	Целочисленное вычитание sub . . . . .	7
3.2.3	Команды инкремента и декремента. . . . .	8
3.2.4	Команда изменения знака операнда neg . . . . .	8
3.2.5	Команды умножения mul и imul . . . . .	8
3.2.6	Команды деления div и idiv . . . . .	9
3.3	Перевод символа числа в десятичную символьную запись . . . . .	9
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Ответы на вопросы . . . . .	19
4.2	Выполнение заданий для самостоятельной работы . . . . .	20
<b>5</b>	<b>Выводы</b>	<b>24</b>
	<b>Список литературы</b>	<b>25</b>

## Список иллюстраций

4.1	Создание каталога для лабораторной работы и переход в него, создание файла . . . . .	11
4.2	Копирование внешнего файла в каталог с программами . . . . .	11
4.3	Ввод текста программы в файл . . . . .	12
4.4	Создание и запуск исполняемого файла . . . . .	12
4.5	Изменение текста программы в файле . . . . .	13
4.6	Создание и запуск исполняемого файла . . . . .	13
4.7	Создание файла . . . . .	13
4.8	Внесение изменений в файл . . . . .	14
4.9	Создание и запуск исполняемого файла . . . . .	14
4.10	Изменение текста программы . . . . .	15
4.11	Изменение текста программы . . . . .	15
4.12	Изменение текста программы . . . . .	16
4.13	Создание и запуск исполняемого файла . . . . .	16
4.14	Создание файла . . . . .	16
4.15	Ввод текста программы для вычисления арифметического выражения	17
4.16	Создание и запуск исполняемого файла . . . . .	17
4.17	Внесение изменений в текст программы . . . . .	18
4.18	Создание и запуск исполняемого файла . . . . .	18
4.19	Создание файла . . . . .	18
4.20	Ввод текста программы . . . . .	19
4.21	Создание и запуск исполняемого файла . . . . .	19
4.22	Создание файла . . . . .	21
4.23	Ввод кода программы в файл . . . . .	21
4.24	Создание и запуск исполняемого файла . . . . .	21

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

## 2 Задание

1. Написание программ вывода символьных и численных значений;
2. Написание программ вычисления значений арифметических выражений;
3. Выполнение заданий для самостоятельной работы.

## 3 Теоретическое введение

### 3.1 Адресация в Nasm

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.

Например, определим переменную `intg DD 3` – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой `intg`. В таком случае, команда

```
mov eax,[intg]
```

копирует из памяти по адресу *intg* данные в регистр *eax*.

В свою очередь, команда

```
mov [intg],eax
```

запишет в память по адресу *intg* данные из регистра *eax*.

Также рассмотрим команду

```
mov eax,intg
```

В этом случае в регистр *eax* запишется адрес *intg*. Допустим, для *intg* выделена память начиная с ячейки с адресом *0x600144*, тогда команда *mov eax,intg* аналогична команде *mov eax,0x600144* – т.е. эта команда запишет в регистр *eax* число *0x600144*.

## 3.2 Арифметические операции в NASM

### 3.2.1 Целочисленное сложение *add*

Схема команды целочисленного сложения *add* (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда *add* работает как с числами со знаком, так и без знака и выглядит следующим образом:

```
add <операнд_1>, <операнд_2>
```

### 3.2.2 Целочисленное вычитание *sub*

Команда целочисленного вычитания *sub* (от англ. subtraction – вычитание) работает аналогично команде *add* и выглядит следующим образом:

```
sub <операнд_1>, <операнд_2>
```

### 3.2.3 Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: *inc* (от англ. *increment*) и *dec* (от англ. *decrement*), которые увеличивают и уменьшают на 1 свой операнд.

Эти команды содержат один операнд и имеет следующий вид:

```
inc <операнд>
```

```
dec <операнд>
```

### 3.2.4 Команда изменения знака операнда *neg*

Еще одна команда, которую можно отнести к арифметическим командам, это команда изменения знака *neg*:

```
neg <операнд>
```

### 3.2.5 Команды умножения *mul* и *imul*

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):

```
mul <операнд>
```

Для знакового умножения используется команда *imul*:

```
imul <операнд>
```



### 3.2.6 Команды деления `div` и `idiv`

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

`div <делитель>` ; Беззнаковое деление

`idiv <делитель>` ; Знаковое деление

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом.

## 3.3 Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом.

Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы.

Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций.

Для выполнения лабораторных работ в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- *iprint* – вывод на экран чисел в формате ASCII, перед вызовом *iprint* в регистр *eax* необходимо записать выводимое число (*mov eax,*).
- *iprintLF* – работает аналогично *iprint*, но при выводе на экран после числа добавляет к символ перевода строки.
- *atoi* – функция преобразует *ascii*-код символа в целое число и записывает результат в регистр *eax*, перед вызовом *atoi* в регистр *eax* необходимо записать число (*mov eax,*).

## 4 Выполнение лабораторной работы

1. Создаём каталог для программ лабораторной работы № 7, переходим в него и создаём файл lab7-1.asm (рис. 4.1):

```
[deibatulina@10 ~]$ mkdir work/arch-pc/lab07
[deibatulina@10 ~]$ cd work/arch-pc/lab07
[deibatulina@10 lab07]$ touch lab7-1.asm
[deibatulina@10 lab07]$ ls
lab7-1.asm
[deibatulina@10 lab07]$
```

Рис. 4.1: Создание каталога для лабораторной работы и переход в него, создание файла

Скопируем внешний файл in\_out.asm в наш каталог для лабораторной работы, поскольку он будет использоваться в дальнейших программах (рис. 4.2):

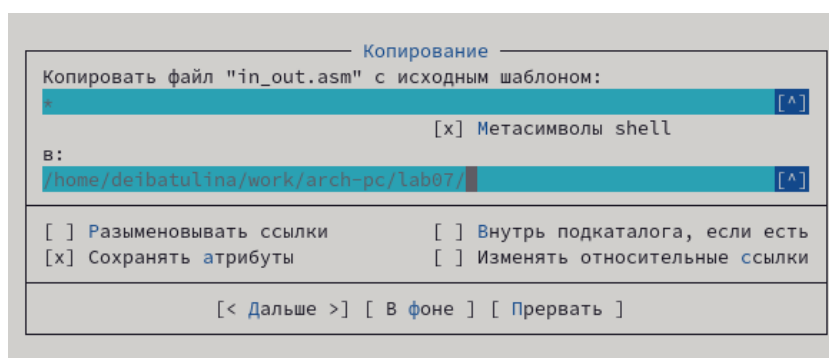
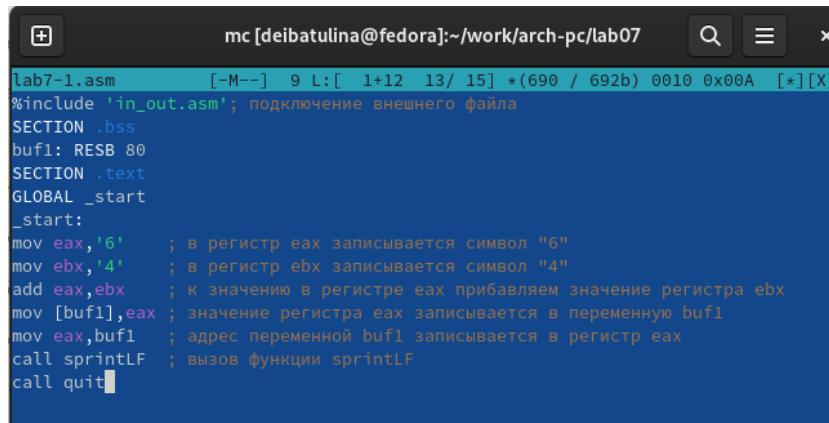


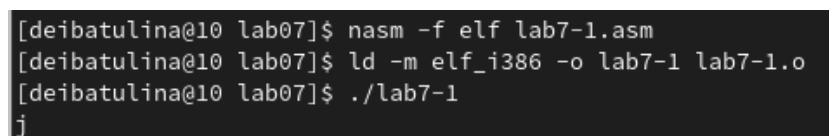
Рис. 4.2: Копирование внешнего файла в каталог с программами

2. Введём в файл lab7-1.asm текст программы из листинга (рис. 4.3), создадим исполняемый файл и запустим его (рис. 4.4):



```
mc [deibatulina@fedora]:~/work/arch-pc/lab07
lab7-1.asm [-M--] 9 L: [ 1+12 13/ 15] *(690 / 692b) 0010 0x00A [*][X]
#include 'in_out.asm'; подключение внешнего файла
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,'6' ; в регистр eax записывается символ "6"
mov ebx,'4' ; в регистр ebx записывается символ "4"
add eax,ebx ; к значению в регистре eax прибавляем значение регистра ebx
mov [buf1],eax ; значение регистра eax записывается в переменную buf1
mov eax,buf1 ; адрес переменной buf1 записывается в регистр eax
call printf ; вызов функции printf
call quit
```

Рис. 4.3: Ввод текста программы в файл

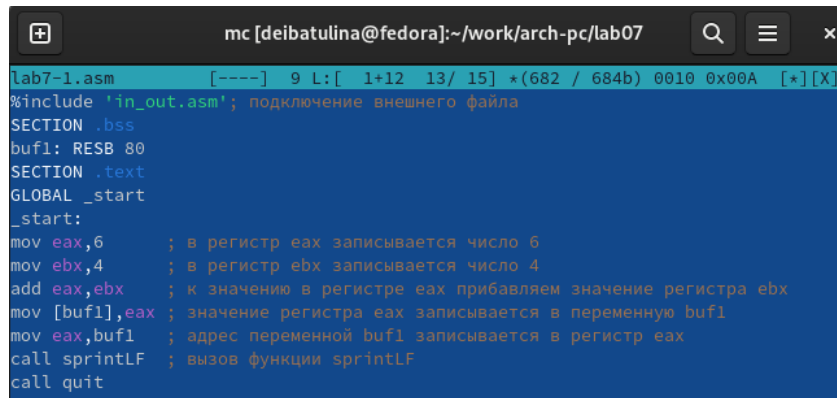


```
[deibatulina@10 lab07]$ nasm -f elf lab7-1.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[deibatulina@10 lab07]$ ./lab7-1
j
```

Рис. 4.4: Создание и запуск исполняемого файла

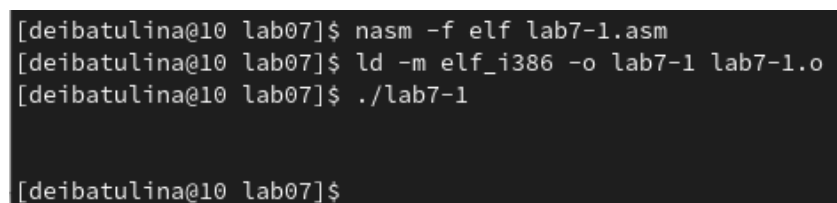
В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число 10. Однако результатом будет символ `j`. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда `add eax,ebx` запишет в регистр `eax` сумму кодов – 01101010 (106), что в свою очередь является кодом символа `j`.

3. Теперь изменим текст программы, заменив символы “6” и “4” на числа 6 и 4 (рис. 4.5). Создадим исполняемый файл и запустим его (рис. 4.6):



```
lab7-1.asm  [----]  9  L: [ 1+12 13/ 15] *(682 / 684b) 0010 0x00A [*][X]
#include 'in_out.asm'; подключение внешнего файла
SECTION .bss
buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,6      ; в регистр eax записывается число 6
mov ebx,4      ; в регистр ebx записывается число 4
add eax,ebx    ; к значению в регистре eax прибавляем значение регистра ebx
mov [buf1],eax ; значение регистра eax записывается в переменную buf1
mov eax,buf1   ; адрес переменной buf1 записывается в регистр eax
call printf    ; вызов функции printf
call quit
```

Рис. 4.5: Изменение текста программы в файле



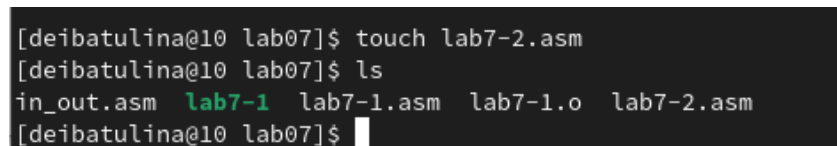
```
[deibatulina@10 lab07]$ nasm -f elf lab7-1.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[deibatulina@10 lab07]$ ./lab7-1

[deibatulina@10 lab07]$
```

Рис. 4.6: Создание и запуск исполняемого файла

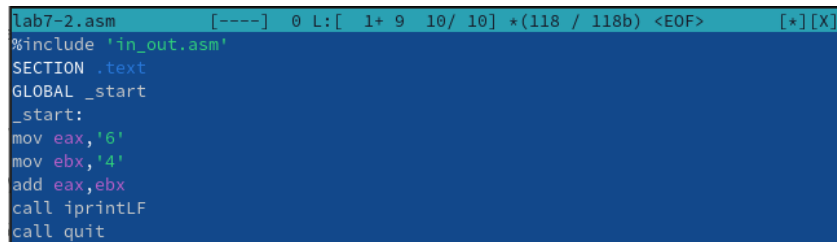
Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. В соответствии с таблицей, приложенной к руководству по лабораторной работе №7, это символ перевода строки. При выводе на экран он, как видно по скриншоту, не отображается.

4. Создаём файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.7) и введите в него текст программы из листинга (рис. 4.8). Создаём исполняемый файл и запускаем его рис. (4.9):



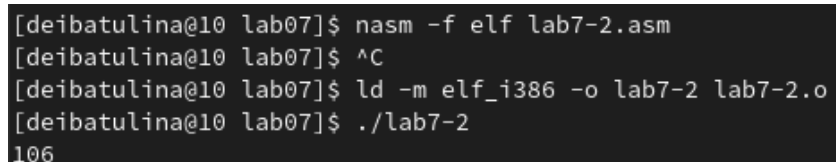
```
[deibatulina@10 lab07]$ touch lab7-2.asm
[deibatulina@10 lab07]$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
[deibatulina@10 lab07]$
```

Рис. 4.7: Создание файла



```
lab7-2.asm  [----]  0 L: [ 1+ 9 10/ 10] *(118 / 118b) <EOF>  [*][X]
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Рис. 4.8: Внесение изменений в файл



```
[deibatulina@10 lab07]$ nasm -f elf lab7-2.asm
[deibatulina@10 lab07]$ ^C
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[deibatulina@10 lab07]$ ./lab7-2
106
```

Рис. 4.9: Создание и запуск исполняемого файла

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 предыдущей программы, функция *iprintLF* позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменим символы на числа (рис. 4.10). Создадим исполняемый файл и запустим его рис. (4.11):

```
lab7-2.asm [-----]
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.10: Изменение текста программы

```
[deibatulina@10 lab07]$ nasm -f elf lab7-2.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[deibatulina@10 lab07]$ ./lab7-2
10
```

Рис. 4.11: Изменение текста программы

Заменяем функцию `iprintLF` на `iprint` (рис. 4.12). Создаём исполняемый файл и запускаем его (рис. 4.13). Выводы функций `iprintLF` и `iprint` отличаются тем, что `iprintLF` делает перевод на новую строку, а `iprint` - нет.

```
lab7-2.asm [-M--]
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

Рис. 4.12: Изменение текста программы

```
[deibatulina@10 lab07]$ nasm -f elf lab7-2.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[deibatulina@10 lab07]$ ./lab7-2
10[deibatulina@10 lab07]$
```

Рис. 4.13: Создание и запуск исполняемого файла

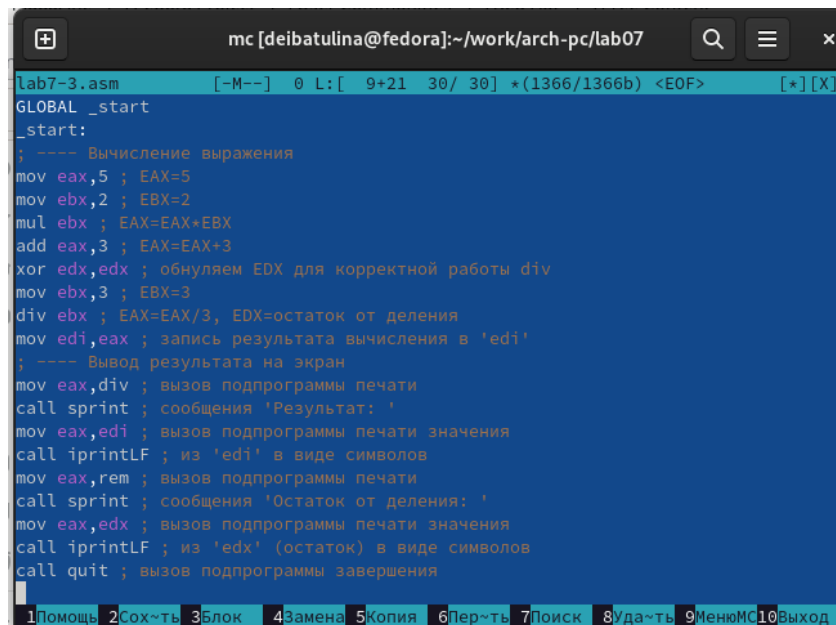
6. Создаём файл lab7-3.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.14):

```
10[deibatulina@10 lab07]touch lab7-3.asm
[deibatulina@10 lab07]$
```

Рис. 4.14: Создание файла

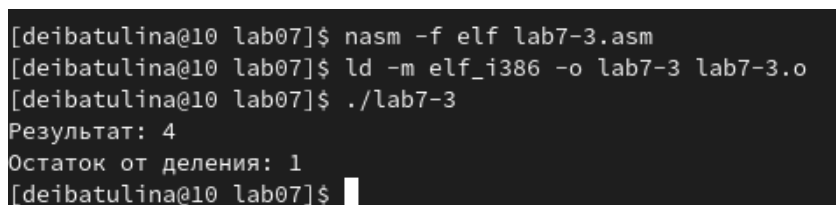
Вводим в данный файл текст программы из листинга (рис. 4.15). Данная программа вычисляет значение арифметического выражения  $(5 * 2 + 3) / 3$  (рис. 4.16):





```
lab7-3.asm [-M--] 0 L: [ 9+21 30/ 30] *(1366/1366b) <EOF> [*] [X]
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintfLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения
call iprintfLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
```

Рис. 4.15: Ввод текста программы для вычисления арифметического выражения

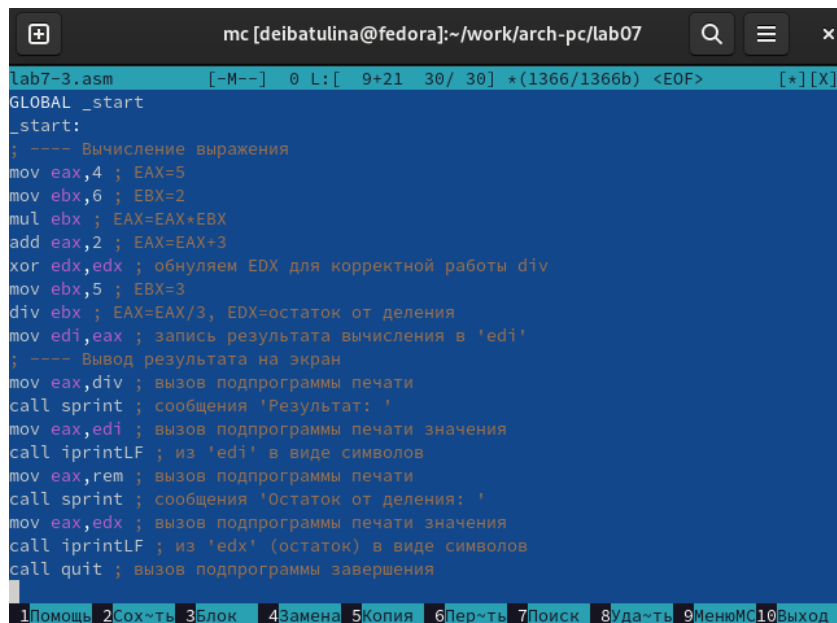


```
[deibatulina@10 lab07]$ nasm -f elf lab7-3.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[deibatulina@10 lab07]$ ./lab7-3
Результат: 4
Остаток от деления: 1
[deibatulina@10 lab07]$
```

Рис. 4.16: Создание и запуск исполняемого файла

Вывод программы корректен.

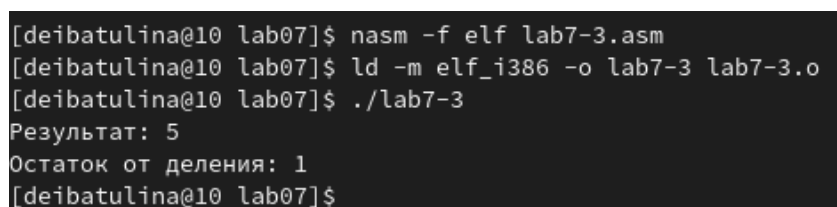
Теперь заменим некоторые числа в этой программе (рис. 4.17): так, теперь она будет вычислять значение выражения  $(4 * 6 + 2) / 5$  (рис. 4.18):



```
lab7-3.asm [-M--] 0 L:[ 9+21 30/ 30] *(1366/1366b) <EOF> [*][X]
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call printf ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call printf ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call printf ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения
call printf ; из 'edx' (остаток) в виде символов
call exit ; вызов подпрограммы завершения

1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Вда-ть 9МенюMC10Выход
```

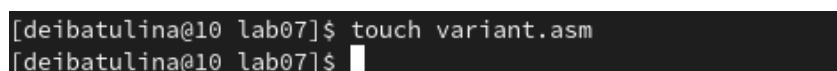
Рис. 4.17: Внесение изменений в текст программы



```
[deibatulina@10 lab07]$ nasm -f elf lab7-3.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[deibatulina@10 lab07]$ ./lab7-3
Результат: 5
Остаток от деления: 1
[deibatulina@10 lab07]$
```

Рис. 4.18: Создание и запуск исполняемого файла

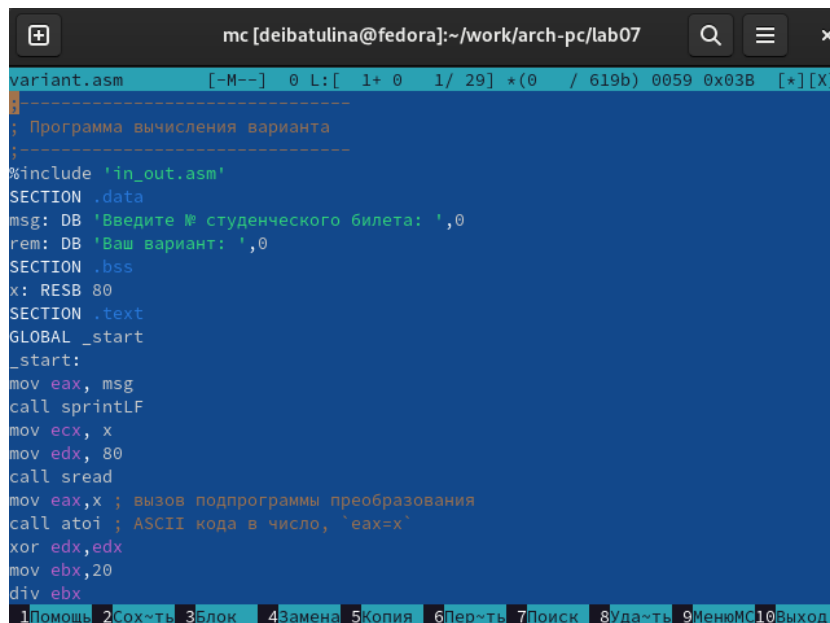
7. Создаём файл variant.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.19):



```
[deibatulina@10 lab07]$ touch variant.asm
[deibatulina@10 lab07]$
```

Рис. 4.19: Создание файла

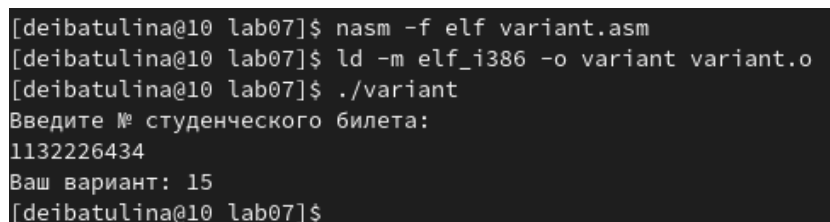
Ввожу в файл текст программы (рис. 4.20):



```
variant.asm [-M--] 0 L: [ 1+ 0 1/ 29] *(0 / 619b) 0059 0x03B [*][X]
;-----
; Программа вычисления варианта
;-----
#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
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Ввести 9МенюMC10Выход
```

Рис. 4.20: Ввод текста программы

Создаём исполняемый файл и, вычислив номер варианта аналитически, приходим к выводу, что программа вычислила номер варианта корректно (рис. 4.21):



```
[deibatulina@10 lab07]$ nasm -f elf variant.asm
[deibatulina@10 lab07]$ ld -m elf_i386 -o variant variant.o
[deibatulina@10 lab07]$ ./variant
Введите № студенческого билета:
1132226434
Ваш вариант: 15
[deibatulina@10 lab07]$
```

Рис. 4.21: Создание и запуск исполняемого файла

## 4.1 Ответы на вопросы

1. За вывод на экран сообщения “Ваш вариант” отвечают следующие строки кода программы:

```
mov eax, rem
call sprint
```

2. Инструкция *mov ecx,x* используется для помещения адреса вводимого значения *x* в регистр *ecx*. Инструкция *mov edx,80* предназначена для помещения длины вводимого с клавиатуры значения. Инструкция *call read* относится ко внешнему файлу и позволяет прочитать введенное пользователем значение.
3. Функция *atoi* преобразует ASCII-код символа в целое число и записывает результат в регистр *eax*, перед вызовом *atoi* в регистр *eax* необходимо записать число (*mov eax,*).
4. За вычисление номера варианта отвечают следующие строки кода программы:

```
xor edx,edx ; обнуляем edx для корректной работы div
mov ebx,20 ; помещаем в регистр ebx число 20: ebx = 20
div ebx ; eax = eax / 20
inc edx ; edx = edx + 1
```

5. При выполнении инструкции *div ebx* остаток от деления записывается в регистр *edx*.
6. Инструкция *inc edx* увеличивает значение регистра *edx* на 1. Прибавление единицы называется инкрементом.
7. За вывод результата вычисления на экран отвечают следующие строки кода программы:

```
mov eax,edx
call iprintLF
```

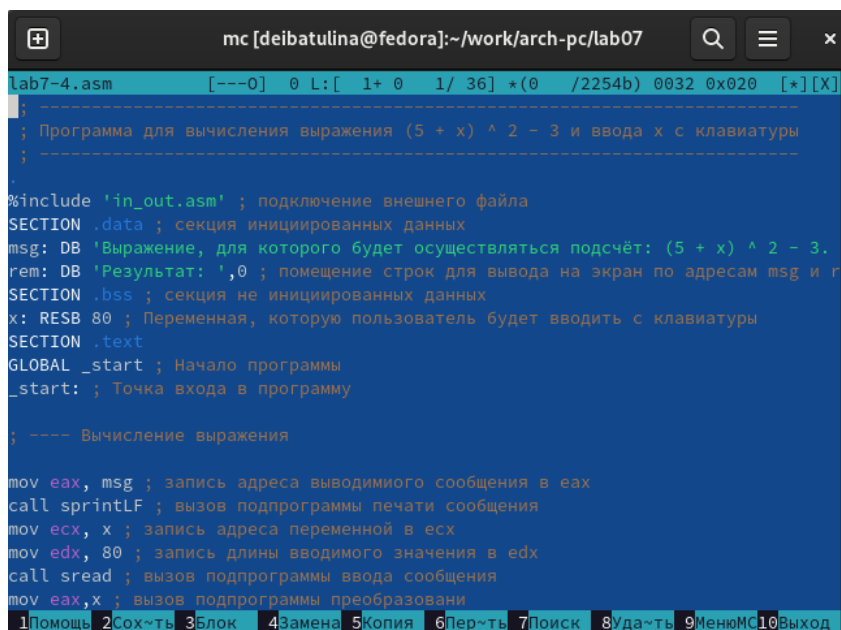
## 4.2 Выполнение заданий для самостоятельной работы

1. В соответствии с номером моего варианта необходимо написать программу для вычисления значения арифметического выражения. Мой вариант №15,

$f(x) = (5 + x)^2 - 3$ . Создаю файл для данной программы (рис. 4.22, ввожу текст программы (рис. 4.23, создаю и запускаю исполняемый файл, проверяю работу программы для заданных значений  $x$ : 5 и 1 (рис. 4.24):

```
[deibatulina@i0 lab07]$ touch lab7-4.asm
[deibatulina@i0 lab07]$
```

Рис. 4.22: Создание файла



```
lab7-4.asm  [--O]  0 L: [ 1+ 0  1/ 36] *(0  /2254b) 0032 0x020  [*] [X]
; -----
; Программа для вычисления выражения (5 + x) ^ 2 - 3 и ввода x с клавиатуры
; -----
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; секция инициализированных данных
msg: DB 'Выражение, для которого будет осуществляться подсчёт: (5 + x) ^ 2 - 3.
rem: DB 'Результат: ',0 ; помещение строк для вывода на экран по адресам msg и r
SECTION .bss ; секция не инициализированных данных
x: RESB 80 ; Переменная, которую пользователь будет вводить с клавиатуры
SECTION .text
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу

; ---- Вычисление выражения

mov eax, msg ; запись адреса выводимого сообщения в eax
call sprintf ; вызов подпрограммы печати сообщения
mov ecx, x ; запись адреса переменной в ecx
mov edx, 80 ; запись длины вводимого значения в edx
call sread ; вызов подпрограммы ввода сообщения
mov eax, x ; вызов подпрограммы преобразования
```

Рис. 4.23: Ввод кода программы в файл

```
[deibatulina@i0 lab07]$ nasm -f elf lab7-4.asm
[deibatulina@i0 lab07]$ ld -m elf_i386 -o lab7-4 lab7-4.o
[deibatulina@i0 lab07]$ ./lab7-4
Выражение, для которого будет осуществляться подсчёт: (5 + x) ^ 2 - 3. Введите
значение переменной x:
5
Результат: 97
[deibatulina@i0 lab07]$ ./lab7-4
Выражение, для которого будет осуществляться подсчёт: (5 + x) ^ 2 - 3. Введите
значение переменной x:
1
Результат: 33
[deibatulina@i0 lab07]$
```

Рис. 4.24: Создание и запуск исполняемого файла

Приведу ниже листинг программы для вычисления значения арифметического выражения  $(5 + x)^2 - 3$ :

**Листинг программы для вычисления значения арифметического выражения  $(5 + x)^2 - 3$  (Вариант 15)**

```
; -----  
; Программа для вычисления выражения  $(5 + x)^2 - 3$  и ввода x с клавиатуры  
; -----  
  
%include 'in_out.asm' ; подключение внешнего файла  
SECTION .data ; секция инициированных данных  
msg: DB 'Выражение, для которого будет осуществляться подсчёт:  $(5 + x)^2 - 3$ .  
      Введите значение переменной x: ',0  
res: DB 'Результат: ',0 ; помещение строк для вывода на экран по адресам msg и res  
SECTION .bss ; секция не инициированных данных  
x: RESB 80 ; Переменная, которую пользователь будет вводить с клавиатуры  
SECTION .text  
GLOBAL _start ; Начало программы  
_start: ; Точка входа в программу  
  
; ---- Вычисление выражения  
  
mov eax, msg ; запись адреса выводимого сообщения в eax  
call sprintf ; вызов подпрограммы печати сообщения  
mov ecx, x ; запись адреса переменной в ecx  
mov edx, 80 ; запись длины вводимого значения в edx  
call sread ; вызов подпрограммы ввода сообщения  
mov eax,x ; вызов подпрограммы преобразования  
call atoi ; ASCII кода в число, `eax = x`  
add eax,5;  $eax = eax + 5 = x + 5$   
mul eax ; возведение выражения  $(x + 5)$  в квадрат  
add eax,-3 ;  $eax = eax - 3 = (x + 5)^2 - 3$ 
```

```
mov edi,eax ; запись результата вычисления в 'edi'
```

```
; ---- Вывод результата на экран
```

```
mov eax,rem ; вызов подпрограммы печати
```

```
call sprint ; сообщения 'Результат: '
```

```
mov eax,edi ; вызов подпрограммы печати значения
```

```
call iprintLF ; из 'edi' в виде символов
```

```
call quit ; вызов подпрограммы завершения
```

Выполнив подсчёты аналитически, я пришла к выводу, что программа работает корректно.

## 5 Выводы

При выполнении лабораторной работы я освоила арифметические инструкции языка Ассемблер.



## Список литературы

- Руководство по выполнению лабораторной работы №7.