

Отчёт по лабораторной работе №6

Дисциплина: Архитектура компьютера

Осина Виктория Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Освоение вывода символьных и численных значений	9
4.2	Выполнение арифметических операций	14
4.3	Выполнение задания для самостоятельной работы	17
5	Выводы	19
6	Список литературы	20

Список иллюстраций

4.1	Создание каталога lab06 и файла lab6-1.asm в нем	9
4.2	Копирование файла in_out.asm	9
4.3	Проверка, что файл находится в нужном каталоге	10
4.4	Ввод текста программы в файл	10
4.5	Создание исполняемого файла и его запуск	11
4.6	Изменение в тексте программы символов на числа	11
4.7	Создание исполняемого файла и его запуск	11
4.8	Создание файла lab6-2.asm	12
4.9	Ввод текста программы в файл	12
4.10	Создание исполняемого файла и его запуск	12
4.11	Изменение в тексте программы символов на числа	13
4.12	Создание исполняемого файла и его запуск	13
4.13	Изменение в тексте программы функции iprintLF на iprint	13
4.14	Создание исполняемого файла и его запуск	14
4.15	Создание файла lab6-3.asm	14
4.16	Ввод текста программы	14
4.17	Создание исполняемого файла и его запуск	15
4.18	Изменение текста программы для вычисления другого выражения	15
4.19	Создание исполняемого файла и его запуск	15
4.20	Создание файла variant.asm	16
4.21	Ввод текста программы	16
4.22	Создание исполняемого файла и его запуск	16
4.23	Создание файла var7.asm	18
4.24	Ввод текста программы	18
4.25	Создание исполняемого файла и его запуск	18

Список таблиц

1 Цель работы

Целью данной лабораторной работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Освоение вывода символьных и численных значений.
2. Выполнение арифметических операций.
3. Выполнение задания 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. 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 `add` работает как с числами со знаком, так и без знака.

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add`.

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.

Команда `neg` рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный.

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 знакового умножения используется команда `imul`.

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

для преобразования ASCII символов в числа и обратно реализованы следующие подпрограммы:

- `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

4.1 Освоение вывода символьных и численных значений

Создаю каталог для программ лабораторной работы №6, перехожу в него и создаю файл lab6-1.asm (рис. 4.1)

```
[vaosina@fedora ~]$ mkdir ~/work/arch-pc/lab06  
[vaosina@fedora ~]$ cd ~/work/arch-pc/lab06  
[vaosina@fedora lab06]$ touch lab6-1.asm
```

Рис. 4.1: Создание каталога lab06 и файла lab6-1.asm в нем

Перед работой с программами копирую файл in_out.asm в каталог(рис. 4.2) (рис. 4.3)

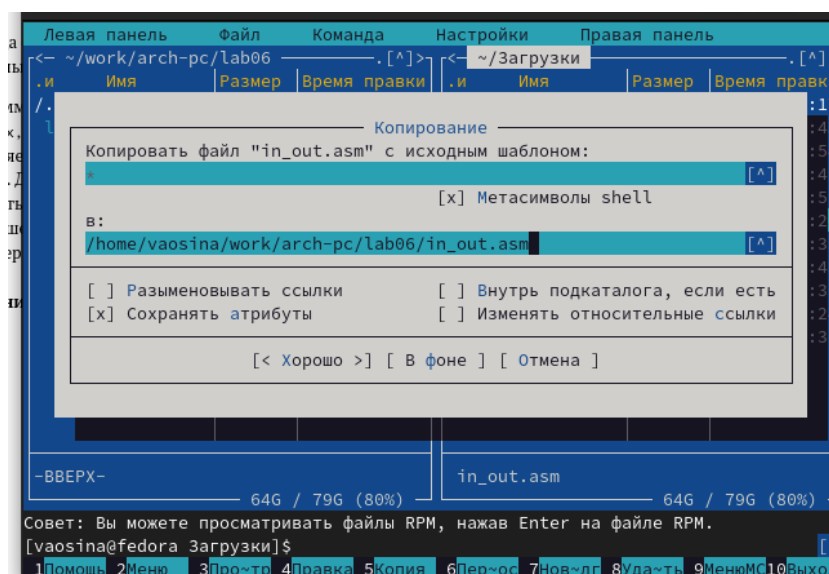


Рис. 4.2: Копирование файла in_out.asm

Левая панель	Файл	Команда	
<- ~/work/arch-pc/lab06 .[^]>			
.и	Имя	Размер	Время правки
/..		-ВВЕРХ-	ноя 15 17:06
	in_out.asm	3942	ноя 11 14:25
	lab6-1.asm	0	ноя 15 17:07

Рис. 4.3: Проверка, что файл находится в нужном каталоге

Ввожу в файл lab6-1.asm текст программы (рис. 4.4).

```
lab6-1.asm [-M--] 9 L: [ 1+16 17/ 17] *(181 / 181b) <EOF> [*][X]
#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. 4.4: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его (рис. 4.5). 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

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

Рис. 4.5: Создание исполняемого файла и его запуск

Изменяю текст программы и вместо символов, записываю в регистры числа (рис. 4.6).

```
lab6-1.asm  [-M--] 10 L: [ 1+10 11/ 17] *(109 / 177b) 0010 0x00A [*][X]
#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

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.6: Изменение в тексте программы символов на числа

Создаю исполняемый файл и запускаю его. (рис. 4.7) Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10, при этом этот символ не отображается при выводе на экран. Код 10 соответствует символу переноса строки

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

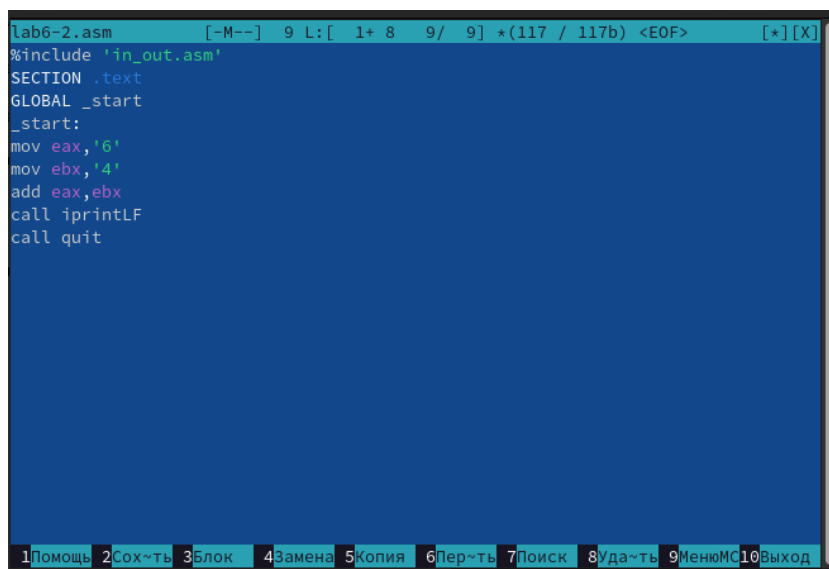
[vaosina@fedora lab06]$
```

Рис. 4.7: Создание исполняемого файла и его запуск

Создаю файл lab6-2.asm в каталоге ~/work/arch-рс/lab06 и ввожу в него текст программы (рис. 4.8) (рис. 4.9).

```
[vaosina@fedora lab06]$ touch lab6-2.asm
```

Рис. 4.8: Создание файла lab6-2.asm



```
lab6-2.asm [-M--] 9 L: [ 1+ 8 9/ 9] *(117 / 117b) <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. 4.9: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его. (рис. 4.10). В результате работы программы получаем число 106. В данном случае, как и в первом, команда add складывает коды символов '6' и '4' ($54+52=106$). Однако функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

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

Рис. 4.10: Создание исполняемого файла и его запуск

Изменяю текст программы и вместо символов, записываю в регистры числа (рис. ??) Создаю исполняемый файл и запускаю его. (рис. ??). Программа складывает

вает не коды символов, а сами числа, поэтому в результате работы программы получаем 10.

```
lab6-2.asm  [----] 13 L: [ 1+ 7 8/ 10] *(103 / 114b) 0010 0x00A [*][X]  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit
```

Рис. 4.11: Изменение в тексте программы символов на числа

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

Рис. 4.12: Создание исполняемого файла и его запуск

Заменяю функцию iprintLF на iprint. (рис. 4.13)

```
lab6-2.asm  [-M--] 11 L: [ 1+ 7 8/ 10] *(101 / 112b) 0010 0x00A [*][X]  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprint  
call quit
```

Рис. 4.13: Изменение в тексте программы функции iprintLF на iprint

Создаю исполняемый файл и запускаю его. (рис. 4.14) Вывод функций iprintLF и iprint отличается тем, что функция iprintLF после числа добавляет символ переноса строки, а iprint нет.

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

Рис. 4.14: Создание исполняемого файла и его запуск

4.2 Выполнение арифметических операций

Создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рис. 4.15).

```
[vaosina@fedora lab06]$ touch lab6-3.asm
[vaosina@fedora lab06]$
```

Рис. 4.15: Создание файла lab6-3.asm

Ввожу текст программы для вычисления арифметического выражения $\boxed{\boxed{\boxed{5}} \cdot \boxed{\boxed{2}} + \boxed{\boxed{3}}} / \boxed{\boxed{3}}$ = $(5 \cdot 2 + 3) / 3$. (рис. 4.16).

```
lab6-3.asm  [-M--] 23 L: [ 1+ 2 3/ 26] *(121 /1236b) 0010 0x00A [*][X]
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
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 iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
```

Рис. 4.16: Ввод текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.17).

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

Рис. 4.17: Создание исполняемого файла и его запуск

Изменяю текст программы для вычисления выражения $\frac{4 \cdot 6 + 2}{5}$ (рис. 4.18)

```
lab6-3.asm  [----]  9 L: [ 4+10 14/ 26] *(423 /1236b) 0032 0x020 [*][X]
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,4 ; EAX=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
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 iprintLF ; 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 iprintLF ; из 'edx' (остаток) в виде символов
1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС10Выход
```

Рис. 4.18: Изменение текста программы для вычисления другого выражения

Создаю исполняемый файл и запускаю его. (рис. 4.19).

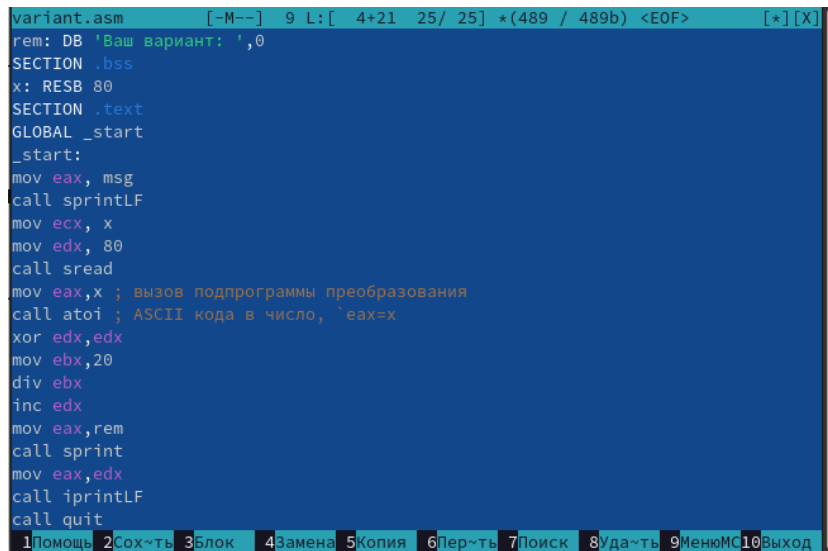
```
[vaosina@fedora lab06]$ nasm -f elf lab6-3.asm
[vaosina@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[vaosina@fedora lab06]$ ./lab6-3
Результат: 5
Остаток от деления: 1
[vaosina@fedora lab06]$
```

Рис. 4.19: Создание исполняемого файла и его запуск

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 (рис. 4.20). Ввожу текст программы для вычисления варианта задания по номеру студенческого билета (рис. 4.21)

```
[vaosina@fedora lab06]$ touch variant.asm
```

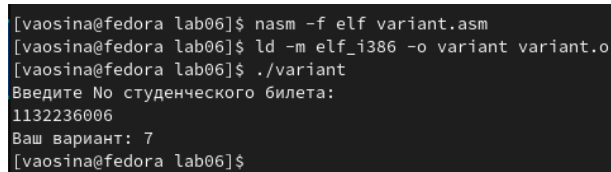
Рис. 4.20: Создание файла variant.asm



```
variant.asm      [-M--]  9 L: [ 4+21 25/ 25] *(489 / 489b) <EOF>      [*][X]
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 sprint
mov eax, edx
call iprintLF
call quit
1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС10Выход
```

Рис. 4.21: Ввод текста программы

Создаю исполняемый файл и запускаю его.(рис. 4.22)



```
[vaosina@fedora lab06]$ nasm -f elf variant.asm
[vaosina@fedora lab06]$ ld -m elf_i386 -o variant variant.o
[vaosina@fedora lab06]$ ./variant
Введите No студенческого билета:
1132236006
Ваш вариант: 7
[vaosina@fedora lab06]$
```

Рис. 4.22: Создание исполняемого файла и его запуск

Проверяю результат работы программы вычислив номер варианта аналитически: 1132236006 делится на 20, в остатке остается 6, к этому значению прибавляется 1, получается 7

1. Какие строки отвечают за вывод на экран сообщения 'Ваш вариант: '? За вывод сообщения на экран отвечают строки:

“ mov eax,rem call sprint “

2. Для чего используются следующие инструкции?

`mov ecx, x` - для того, чтобы поместить адрес вводимой строки `x` в регистр `ecx`
`mov edx, 80` - запись в регистр `edx` длины вводимой строки
`call sread` - вызов подпрограммы, обеспечивающей ввод сообщения с клавиатуры

3. Для чего используется инструкция `call atoi`?

Для преобразования ASCII кода в число

4. Какие строки отвечают за вычисления варианта?

За вычисление варианта отвечают:

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

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

Остаток от деления при выполнении функции `div` записывается в регистр `edx`

6. Для чего используется инструкция `inc edx`?

Инструкция `inc edx` используется для увеличения значения `edx` на 1

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

За вывод на экран результата вычислений отвечают:

“`mov eax,edx call iprintLF`”

4.3 Выполнение задания для самостоятельной работы


Необходимо написать программу вычисления выражения $x = x(x)$. Согласно моему варианту, полученному при выполнении лабораторной работы, это выражение $5(x-1)^2$. Проверить работу программы вычисления нужно, используя значение переменных `x1` (3) и `x2` (5).

Создаю файл var7.asm в каталоге ~/work/arch-pc/lab06 (рис. 4.23)

```
[vaosina@fedora lab06]$ touch var7.asm
[vaosina@fedora lab06]$ gedit var7.asm
```

Рис. 4.23: Создание файла var7.asm

Ввожу текст программы (рис. 4.24).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB "Введите значение переменной x: ",0
5 rem: DB "Результат: ",0
6
7 SECTION .bss
8 x: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13
14 ; ---- Вычисление выражения 5(x - 1)^2
15
16 mov eax, msg
17 call sprintf
18 mov ecx, x
19 mov edx, 80
20 call sread
21 mov eax, x ; вызов подпрограммы преобразования
22 call atoi ; ASCII кода в число, 'eax=x'
23 dec eax ; EAX = EAX - 1
24 mul eax ; EAX = EAX * EAX
25 mov ebx, 5 ; EBX=5
26 mul ebx ; EAX=EAX*EBX
27 mov edi, eax ; запись результата вычисления в edi
28
29 ; ---- Вывод результата на экран
30
31 mov eax, rem
32 call sprintf
33 mov eax, edi ; вызов подпрограммы печати значения
34 call iprintf
35 call quit ; вызов подпрограммы завершения
```

Рис. 4.24: Ввод текста программы

Создаю исполняемый файл и запускаю его. Проверяю работу программы, используя сначала $x_1 = 3$, а затем $x_2 = 5$ (рис. 4.25).

```
[vaosina@fedora lab06]$ nasm -f elf var7.asm
[vaosina@fedora lab06]$ ld -m elf_i386 -o var7 var7.o
[vaosina@fedora lab06]$ ./var7
Введите значение переменной x:
3
Результат: 20
[vaosina@fedora lab06]$ ./var7
Введите значение переменной x:
5
Результат: 80
```

Рис. 4.25: Создание исполняемого файла и его запуск

Действительно, если подставить 3, то $5(3-1)^2 = 5^2 \cdot 2 = 20$. Если подставить 5: $5(5-1)^2 = 5^2 \cdot 4 = 80$.

5 Выводы

При выполнении лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

6 Список литературы

1. Архитектура ЭВМ