

Лабораторная работа № 6

Архитектура компьютера

Тойчубекова Асель Нурлановна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
3.1	Адресация в NASM	8
3.2	Целочисленное сложение add	8
3.3	Целочисленное вычитание sub	8
3.4	Команды инкремента и декремента	9
3.5	Команда изменения знака операнда neg	9
3.6	Команды умножения mul и imul	9
3.7	Команды деления div и idiv	9
3.8	Перевод символа числа в десятичную символьную запись	10
4	Выполнение лабораторной работы	11
4.1	Символьные и численные данные в NASM	11
4.2	Выполнение арифметических операций в NASM	16
4.3	Ответы на вопросы	19
4.4	Задание для самостоятельной работы	20
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	РИС.1 Создание каталога и файла в нем.	11
4.2	РИС.2 Копирование файла	11
4.3	РИС.3 Редактирование файла	12
4.4	РИС.4 Запуск исполняемого файла	12
4.5	РИС.5 Редактирование файла	13
4.6	РИС.6 Запуск исполняемого файла	13
4.7	РИС.7 Создание файла	13
4.8	РИС.8 Редактирование файла	14
4.9	РИС.9 Запуск исполняемого файла	14
4.10	РИС.10 Редактирование файла	14
4.11	РИС.11 Запуск нового исполняемого файла	15
4.12	РИС.12 Редактирование файла	15
4.13	РИС.13 Запуск измененного исполняемого файла	15
4.14	РИС.14 Создание файла lab6-3.asm	16
4.15	РИС.15 Редактирование файла	16
4.16	РИС.16 Запуск исполняемого файла	17
4.17	РИС.17 Редактирование файла	17
4.18	РИС.18 Запуск исполняемого файла	17
4.19	РИС.19 Создание файла	18
4.20	РИС.20 Редактирование файла	18
4.21	РИС.21 Запуск исполняемого файла	19
4.22	РИС.22 Создания файла	20
4.23	РИС.23 Написание программы	22
4.24	РИС.24 Запуск исполняемого файла	22
4.25	РИС.25 Запуск исполняемого файла	23

Список таблиц

1 Цель работы

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

2 Задание

- Познакомиться с адресацией в NASM.
 - Понять как производятся арифметические операции в NASM: -
Целочисленное сложение add;
-Целочисленное вычитание sub;
-Команды инкремента и декремента;
-Команда изменения знака операнда neg;
-Команда умножения mul и imul;
-Команда деления div и idiv.
 - Изучить как происходит перевод символа числа в десятичную символьную запись.
 - Научиться писать программу вывода значения регистра eax, используя пример.
 - Научиться писать программу, которая вычисляет выражение $F(x) = (5x^2 + 3)/3$.
 - Научиться писать программу, вычисляющая вычисления варианта задания по номеру студенческого билета.
 - Задание для самостоятельной работы:
1. 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.

3 Теоретическое введение

3.1 Адресация в NASM

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.

3.2 Целочисленное сложение `add`

Команда `add` производит сложение двух операндов и записывает результат по адресу первого операнда. Команда работает как с числами со знаком, так и без знака и выглядит следующим образом:

`add ,`

3.3 Целочисленное вычитание `sub`

Команда целочисленного вычитания `sub` работает аналогично команде `add` и выглядит следующим образом:

`sub ,`

3.4 Команды инкремента и декремента

Это команда отвечает за прибавление единицы - инкремент(int) и за ее вычитание - декремент(dec). Эти команды имеют следующий вид:

inc

dec .

3.5 Команда изменения знака операнда neg

Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Это команда имеет вид:

mov ax,1

neg ax.

3.6 Команды умножения mul и imul

Для беззнакового умножения используется команда mul:
mul . Для знакового умножения используется команда imul:
imul .

3.7 Команды деления div и idiv

Для деления, как и для умножения, существует 2 команды div и idiv: div - Беззнаковое деление;
idiv - Знаковое деление.

3.8 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. Согласно стандарту ASCII каждый символ кодируется одним байтом. Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться.

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 решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

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,< int >`). - `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,< int >`).

4 Выполнение лабораторной работы

4.1 Символьные и численные данные в NASM

Сперва создадим каталог для программы лабораторной работы №6 и перейдем в него, далее создадим файл lab6-1.asm. Введя команду ls можем видеть что файл успешно был создан (РИС.1)

```
antoychubekova@dk6n52 ~ $ mkdir ~/work/arch-pc/lab06
antoychubekova@dk6n52 ~ $ cd ~/work/arch-pc/lab06
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ touch lab6-1.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ls
lab6-1.asm
```

Рис. 4.1: РИС.1 Создание каталога и файла в нем.

Перейдя в Midnight Commander с командой mc и используя функциональную клавишу F5 скопируем в текущий каталог файл in_out.asm для ее дальнейшего применения. (РИС.2)

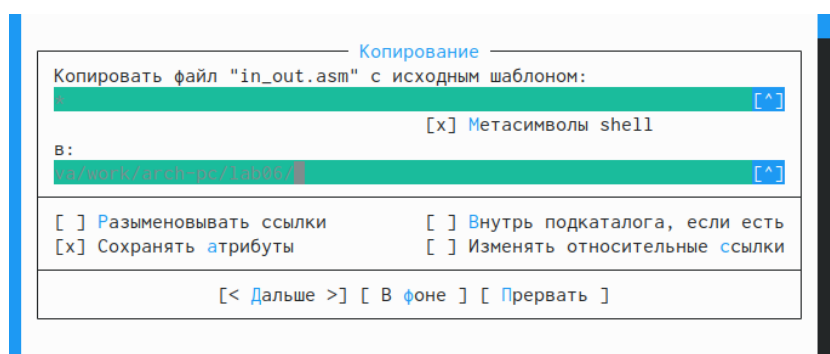
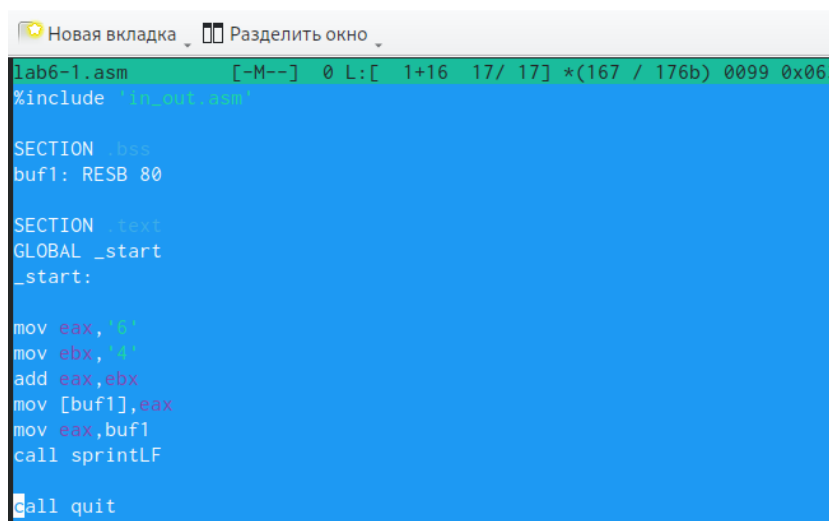


Рис. 4.2: РИС.2 Копирование файла

Откроем созданный файл lab6-1.asm и напишем в него программу вывода значения регистра eax.(РИС 3)



```
lab6-1.asm [-M--] 0 L: [ 1+16 17/ 17] *(167 / 176b) 0099 0x063
#include "in_out.asm"

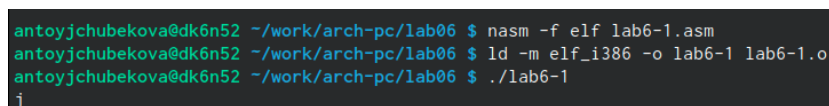
SECTION .bss
buf1: RESB 80

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: РИС.3 Редактирование файла

Создаем исполняемый файл и запускаем его.(РИС.4). Система выводит j, это происходит потому что программа вывела символ, соответствующий по системе ASCII сумме “6(код символа равен 54)+4(код символа равен 52)”, то есть число 106, что в свою очередь является кодом символа j.



```
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-1
j
```

Рис. 4.4: РИС.4 Запуск исполняемого файла

Теперь изменим текст программы заменяя символы ‘6’ и ‘4’ на числа 6 и 4.(РИС.5)

```

lab6-1.asm      [-M--]  8 L:[ 1+16 17/ 17] *(171 / 172b) 0116 0x0
#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. 4.5: РИС.5 Редактирование файла

Создадим новый исполняемый файл программы и запустим его.(РИС.6).В кодировке ASCII символ Char 10 представлен шестнадцатеричным значением 0x0A или восьмеричным значением 012. На экран ничего не отображается

```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-1

```

Рис. 4.6: РИС.6 Запуск исполняемого файла

Создадим файл lab6-2.asm с помощью утилита touch в исходный каталог. С помощью ls видим, что файл был удачно создан

```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ls
in_out.asm  lab6-1  lab6-1.asm  lab6-1.o  lab6-2.asm

```

Рис. 4.7: РИС.7 Создание файла

Введем в файл новую программу с использованием iprintLF для вывода значения регистра eax(РИС.8)

```
lab6-2.asm      [-M--]  9 L:[ 1+11 12/ 12] *(120 / 120b) <EOF>
#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: РИС.8 Редактирование файла

Создадим исполняемый файл и запускаем его.(РИС.9) Теперь на экран выводится число 106, в данном случае команда add складывает символы, но функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

```
antoyjchubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
antoyjchubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
antoyjchubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-2
106
```

Рис. 4.9: РИС.9 Запуск исполняемого файла

Аналогично предыдущему примеру изменим символы на числа, заменяя '6' и '4' на 6 и 4 в команде mov.(РИС.10)

```
lab6-2.asm      [-M--]  9 L:[ 1+11 12/ 12] *(116 / 116b) <EOF>
#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: РИС.10 Редактирование файла

Создадим и запустим новый исполняемый файл(РИС.11). Программа складывает сами числа, а не соответствующие символам коды в системе ASCII и в итоге получаем число 10.

```
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

Рис. 4.11: РИС.11 Запуск нового исполняемого файла

Заменим в исходной программе функцию `iprintLF` на функцию `iprint`.(РИС.12)

```
lab6-2.asm  [-M--]  9 L:[ 1+11 12/ 12] *(114 / 114b) <EOF>
#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: РИС.12 Редактирование файла

Создадим и запустим отредактированный файл(РИС.13). Мы видим, что программа выполнилась без переноса на новую строку. в этом и заключается разница между двумя функциями `iprintLF`-запрашивает перенос на новую строку, `iprint`-не запрашивает перенос на новую строку.

```
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-2
10antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.asm
```

Рис. 4.13: РИС.13 Запуск измененного исполняемого файла

4.2 Выполнение арифметических операций в NASM

Используя утилит touch создадим файл lab6-3.asm, в котором дальше будем работать.(РИС.14).Введя утилит ls видим, что файл удачно создан.

```
10antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3.asm
```

Рис. 4.14: РИС.14 Создание файла lab6-3.asm

Внимательно изучив текст программы вычисления выражения $F(x) = (5x^2 + 3)/3$ запишем ее в файл lab6-3.asm.(РИС.15)

```
lab6-3.asm [-M--] 0 L: [ 1+ 2 3/ 30] *(24 / 363b) 0010 0x00A
%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. 4.15: РИС.15 Редактирование файла

Создадим и запустим исполняемый файл(РИС.16). Можно заметить, что программа сработала и вывела на экран правильные значения.


```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1

```

Рис. 4.16: РИС.16 Запуск исполняемого файла

Изменим текст программы так, чтобы она вычисляла выражение $F(x) = (4*6 + 2)/5$ (РИС.17).

```

lab6-3.asm [----] 9 L: [ 1+25 26/ 26] *(355 / 355b) <EOF>
#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. 4.17: РИС.17 Редактирование файла

Создадим и запустим исполняемый файл (РИС.18). Посчитав самостоятельно, мы видим, что программа работает правильно.

```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1

```

Рис. 4.18: РИС.18 Запуск исполняемого файла

Пользуясь утилитой touch создадим файл variant.asm.(РИС.19). Пользуясь командой ls видим, что файл создан.

```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/variant.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3 lab6-3.asm lab6-3.o variant.asm

```

Рис. 4.19: РИС.19 Создание файла

Откроем этот файл с помощью функциональной клавиши F4 и запишем в него программу вычисления вычисления варианта задания по номеру студенческого билета.(РИС.20)

```

variant.asm [-M--] 9 L:[ 1+32 33/ 33] *(393 / 393b) <EOF>
#include "in_out.asm"

SECTION .data
msg: DB "Введите No 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. 4.20: РИС.20 Редактирование файла

Создадим и запустим исполняемый файл(РИС.21). Посчитав используя калькулятор, мы видим, что программа правильно выдает вариант.

```

antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
antoychubekova@dk6n52 ~/work/arch-pc/lab06 $ ./variant
Введите No студенческого билета:
1032235033
Ваш вариант: 14

```

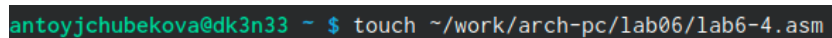
Рис. 4.21: РИС.21 Запуск исполняемого файла

4.3 Ответы на вопросы

1. За вывод сообщения “Ваш вариант” отвечает строки кода:
`mov eax, rem`
`call sprint.`
2. `mov ecx, x` - перемещает адрес вводимой строки x в регистр ecx
`mov edx, 80` - запись в регистр edx длину вводимой строки
`call sread` - вызов подпрограммы из внешнего файла, обеспечивающий ввод сообщения с клавиатуры.
3. `call atoi` - используется для вызова подпрограммы из внешнего файла, которая преобразует ASCII код символа в целое число и записывает результат в регистр eax.
4. За вычисления варианта отвечают строки:
`xor edx,edx` - обнуление edx для корректной работы `div`
`mov ebx,20` - `ebx=20`
`div ebx` - `eax=eax/20`, `edx`-остаток от деления
`inc edx` - `edx=edx+1`.
5. При выполнении инструкции “`div ebx`” остаток от деления записывается в регистр `edx`.
6. Инструкция “`inc edx`” увеличивают значения регистра `edx` на 1.
7. За вывод на экран результата вычислений отвечают строки:
`mov eax, edx`
`call iprintLF.`

4.4 Задание для самостоятельной работы

Создадим файл lab6-4.asm с помощью утилита touch.(РИС.22)



```
antoyjchubekova@dk3n33 ~ $ touch ~/work/arch-pc/lab06/lab6-4.asm
```

Рис. 4.22: РИС.22 Создания файла

Откроем созданный файл и вводим в него текст программы, которая вычисляет значения выражения $(x/2 + 8)^3$, где x вводит пользователь - вариант 14(РИС.23).

Исходная программа выглядит следующим образом:

```
%include 'in_out.asm'
```

```
SECTION .data
msg: DB 'Введите x:',0
rem: DB 'Результат:',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,2
div ebx
add eax,8
mov ebx,3
mul ebx
mov edi,eax
mov eax,rem
call sprint
mov eax,edi
call iprintLF
call quit.
```

```

lab6-4.asm [----] 9 L:[ 1+32 33/ 34] *(380 / 381b) 0010 0x00A
#include "in_out.asm"

SECTION .data
msg: DB 'Введите x: ',0
rem: DB 'Результат: ',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, 2
div ebx
add eax, 8
mov ebx, 3
mul ebx
mov edi, eax
mov eax, rem
call sprintf
mov eax, edi
call iprintf
call quit

```

Рис. 4.23: РИС.23 Написание программы

Создадим и запустим исполняемый файл(РИС.24). При вводе значения 1, выводится правильный ответ-24.

```

antoyjchubekova@dk3n33 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm
antoyjchubekova@dk3n33 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o
antoyjchubekova@dk3n33 ~/work/arch-pc/lab06 $ ./lab6-4
Введите x:
1
Результат: 24

```

Рис. 4.24: РИС.24 Запуск исполняемого файла

Проверим работу программы введя значения 4(РИС.25). Выводится правильный ответ-30

```
Результат: 24  
antoychubekova@dk3n33 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm  
antoychubekova@dk3n33 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o  
antoychubekova@dk3n33 ~/work/arch-pc/lab06 $ ./lab6-4  
Введите x:  
4  
Результат: 30
```

Рис. 4.25: РИС.25 Запуск исполняемого файла

5 Выводы

Выполняя лабораторную работу № 6 я освоила арифметические инструкции языка ассемблера NASM. Используя полученные навыки я написала программу, которая вычисляет значения выражения $(x/2 + 8)^3$, где x вводит пользователь.

Список литературы

-<https://esystem.rudn.ru>.