

мехмет эфе кантоз моя группа НКАбд-04-23

Лабораторная работа №4

1) Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2) Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла

Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

Основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора входят следующие устройства: - арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; - устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; - регистры — сверхбыстрая оперативная память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и

специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора.

существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между регистрами и памятью, преобразование (арифметические или логические операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита. В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ): - RAX, RCX, RDX, RBX, RSI, RDI — 64-битные - EAX, ECX, EDX, EBX, ESI, EDI — 32-битные - AX, CX, DX, BX, SI, DI — 16-битные - AH, AL, CH, CL, DH, DL, BH, BL — 8-битные

Другим важным узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для длительного хранения больших объёмов данных. - устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой.

В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает, что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы.

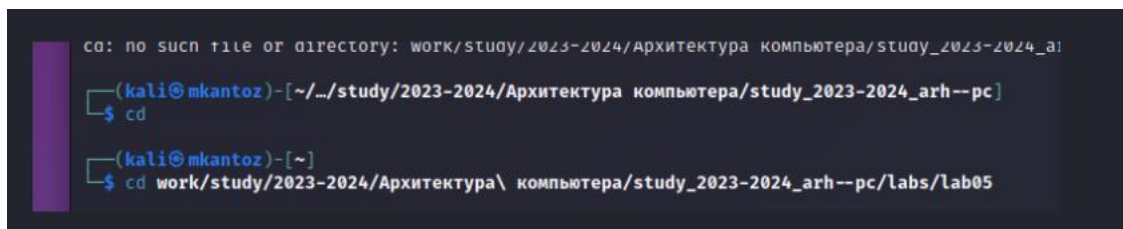
Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. Он заключается в следующем: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. переход к следующей команде.

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

4)Выполнение лабораторной работы

4.1 Создание программы HELLO WORLD!

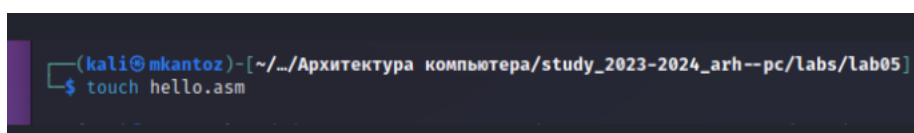
С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 1).



```
cd: no such file or directory: work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh
(kali@mkantoz)-[~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh--pc]
$ cd
(kali@mkantoz)-[~]
$ cd work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh--pc/labs/lab05
```

Рис. 1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 2).

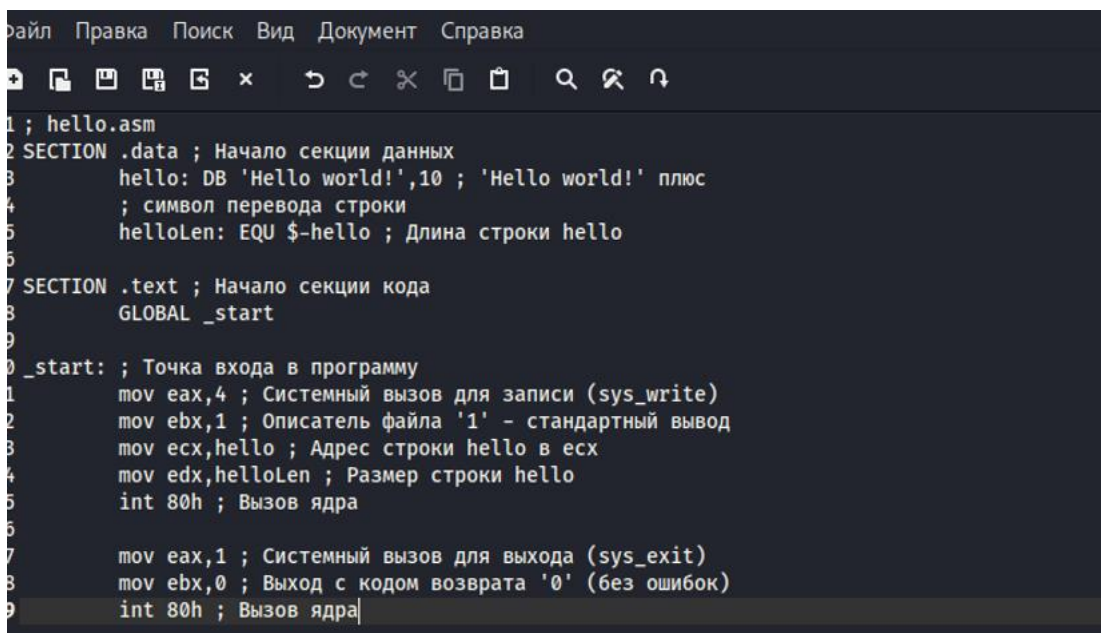


```
(kali@mkantoz)-[~/work/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ touch hello.asm
```

Рис. 2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 3). Рис. 3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4).



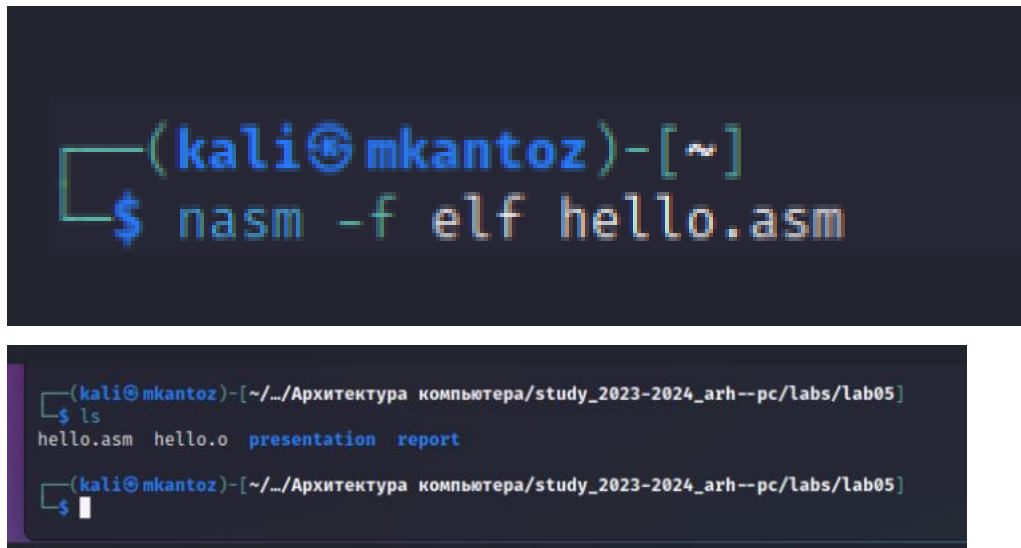
```
Файл Правка Поиск Вид Документ Справка
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3     hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4     ; символ перевода строки
5     helloLen: EQU $-hello ; Длина строки hello
6
7 SECTION .text ; Начало секции кода
8     GLOBAL _start
9
10 _start: ; Точка входа в программу
11     mov eax,4 ; Системный вызов для записи (sys_write)
12     mov ebx,1 ; Описатель файла '1' - стандартный вывод
13     mov ecx,hello ; Адрес строки hello в ecx
14     mov edx,helloLen ; Размер строки hello
15     int 80h ; Вызов ядра
16
17     mov eax,1 ; Системный вызов для выхода (sys_exit)
18     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19     int 80h ; Вызов ядра
```

Рис. 4: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает

транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл `"hello.o"`.

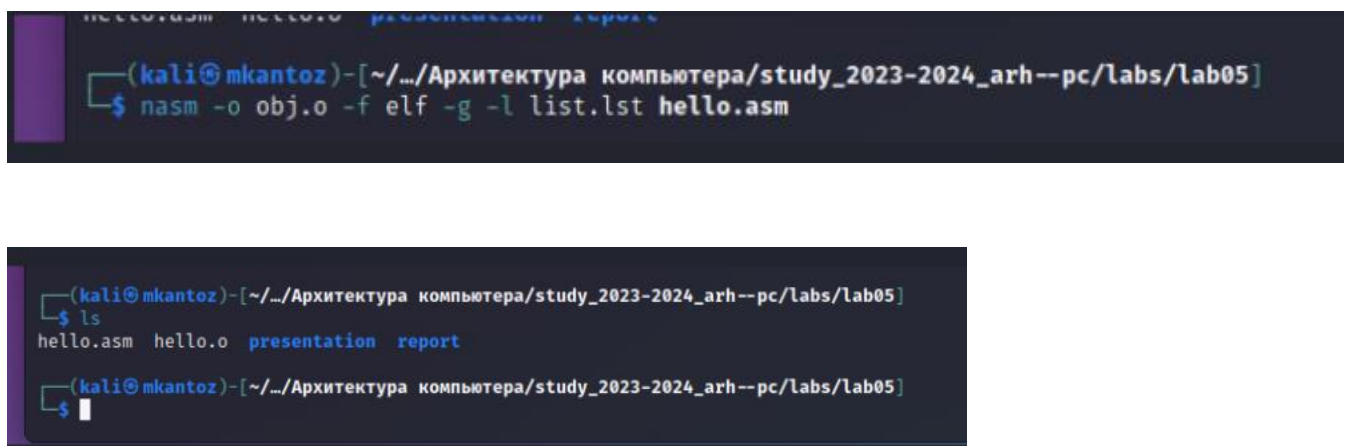


```
(kali@mkantoz)-[~]  
$ nasm -f elf hello.asm  
  
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$ ls  
hello.asm hello.o presentation report  
  
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$
```

Рис. 5: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки **NASM**

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, при этом в файл будут включены символы для отладки (ключ `-g`), также с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

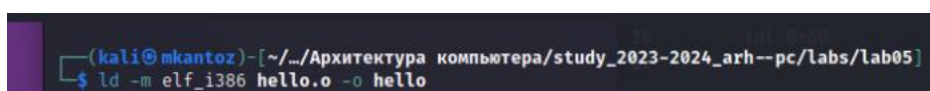


```
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$ nasm -o obj.o -f elf -g -l list.lst hello.asm  
  
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$ ls  
hello.asm hello.o presentation report  
  
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$
```

Рис. 6: Компиляция текста программы

4.4 Работа с компоновщиком **LD**

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 7). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.



```
(kali@mkantoz)-[~/Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]  
$ ld -m elf_i386 hello.o -o hello
```

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ls
hello.asm hello.o list.lst obj.o presentation report
```

Рис. 7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 8). Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ld -m elf_i386 obj.o -o main
```

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
$
```

Рис. 8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла..

Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ./hello
Hello world!
```

Рис. 9: Запуск исполняемого файла

4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab5.asm (рис. 10).

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ cp hello.asm lab5.asm
```

Рис. 10: Создание копии файла

С помощью текстового редактора mousepad открываю файл lab5.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. .

(11)

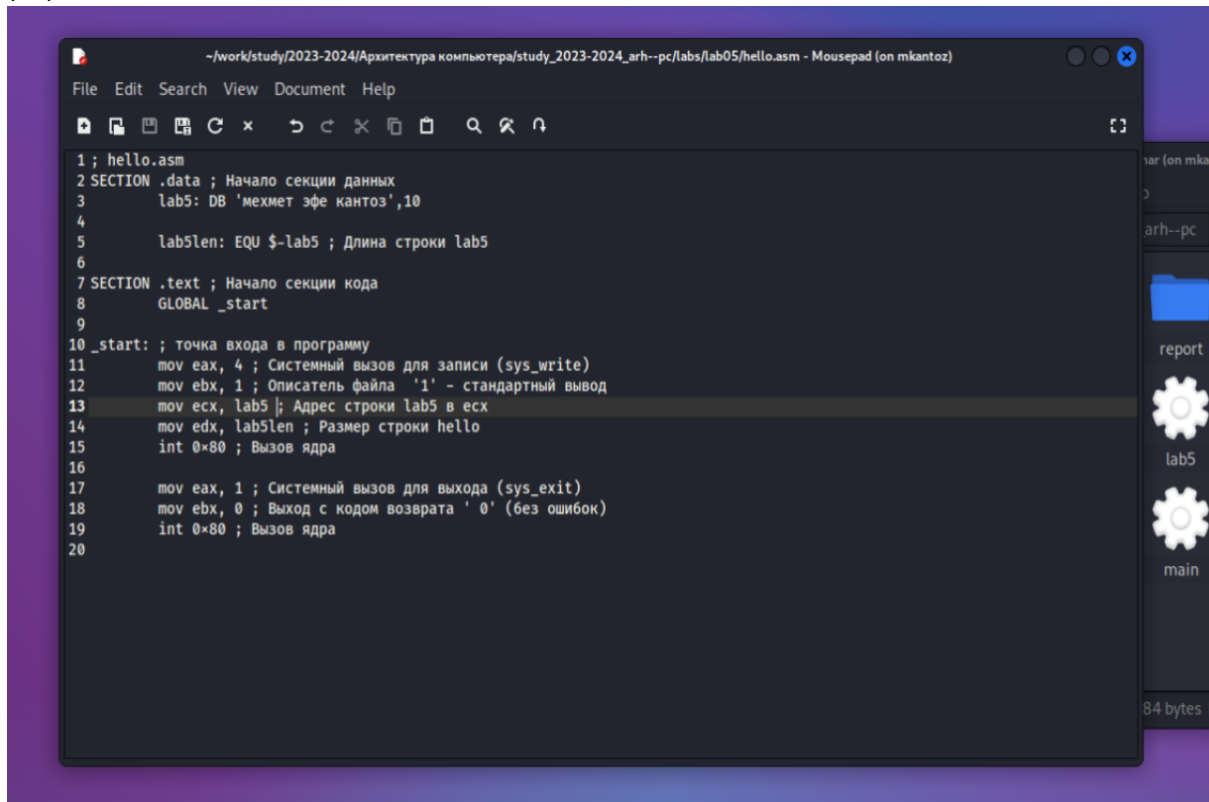


Рис. 11: Изменение программы

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab5.o создан

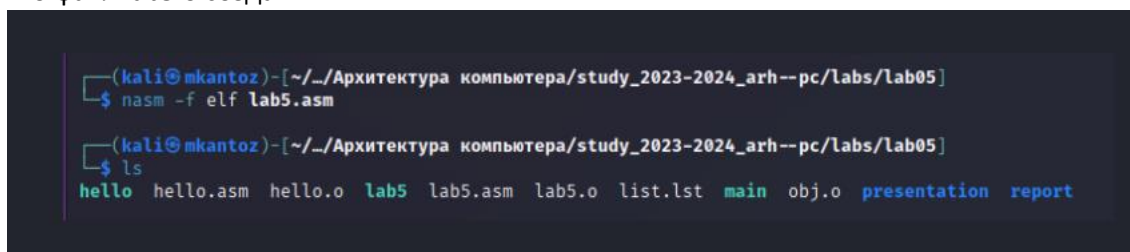


Рис. 12: Компиляция текста программы

Передаю объектный файл lab5.o на обработку компоновщику LD, чтобы получить исполняемый файл lab5 (рис. 13).

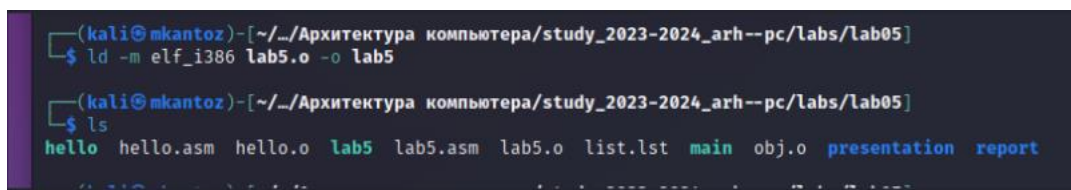


Рис. 13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab5, на экран действительно выводятся мои имя и фамилия .

Рис. 14: Запуск исполняемого файла,

К сожалению, я начал работу не в том каталоге, поэтому создаю другую директорию lab05 с помощью mkdir, прописывая полный путь к каталогу, в котором хочу создать эту директорию. Далее копирую из текущего каталога файлы, созданные в процессе выполнения лабораторной работы, с помощью утилиты cp, указывая вместо имени файла символ *, чтобы скопировать все файлы. Команда проигнорирует директории в этом каталоге, т. к. не указан ключ -r, это мне и нужно (рис. 15). Проверяю с помощью утилиты ls правильность выполнения команды.

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh--pc/lab05

(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ cp * ~/work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh--pc/lab05
cp: -r not specified; omitting directory 'presentation'
cp: -r not specified; omitting directory 'report'

(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ls ~/work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh--pc/lab05
hello hello.asm hello.o lab5 lab5.asm lab5.o list.lst main obj.o
```

Рис. 15: Создании копии файлов в новом каталоге

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm, ведь копии файлов остались в другой директории (рис. 16).

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ rm hello hello.o lab5 lab5.o list.lst main obj.o

(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ ls
hello.asm lab5.asm presentation report
```

Рис. 16: Удаление лишних файлов в текущем каталоге С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис. 17).

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ git add .

(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ git commit -m "Add fales for lab05"
[master 3ec6d2e] Add fales for lab05
2 files changed, 38 insertions(+)
create mode 100644 labs/lab05/hello.asm
create mode 100644 labs/lab05/lab5.asm
```

Рис. 17: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 18).

```
(kali@mkantoz)-[~/../Архитектура компьютера/study_2023-2024_arh--pc/labs/lab05]
$ git push
Username for 'https://github.com': mkantoz1
Password for 'https://mkantoz1@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/mkantoz1/study_2023-2024_arh--pc.git/'
```

Рис. 18: Отправка файлов

5) Выводы

При выполнении данной лабораторной работы я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.

