

Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Акмурадов Тимур

Содержание

Цель работы	1
Задание	1
Теоретическое введение	1
Выполнение лабораторной работы	3
Создание программы Hello world!	3
Работа с транслятором NASM.....	4
Работа с расширенным синтаксисом командной строки NASM.....	4
Работа с компоновщиком LD	5
Запуск исполняемого файла	5
Выполнение заданий для самостоятельной работы.	5
Выводы.....	8
Список литературы	8

Цель работы

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

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
6. Выполнение заданий для самостоятельной работы.

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 долговременного 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.

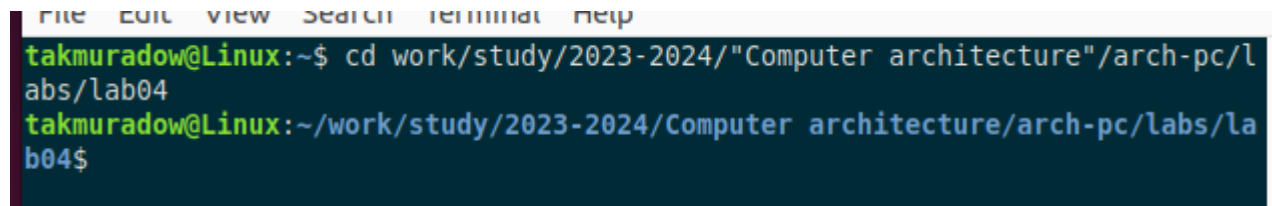
Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении 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.

Выполнение лабораторной работы

Создание программы Hello world!

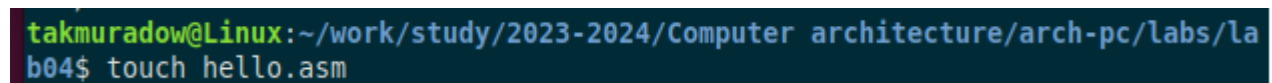
С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 01).



```
File Edit View Search Terminal Help
takmuradow@Linux:~$ cd work/study/2023-2024/"Computer architecture"/arch-pc/labs/lab04
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$
```

Перемещение между директориями

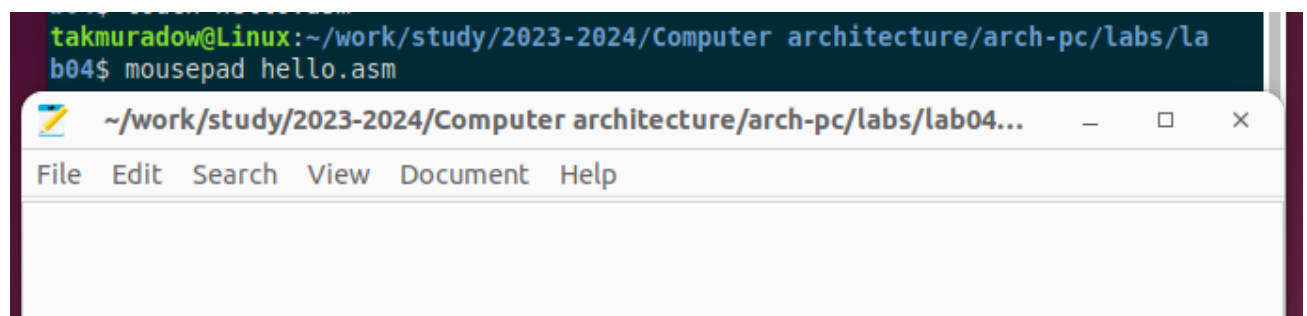
Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 02).



```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ touch hello.asm
```

Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 03).



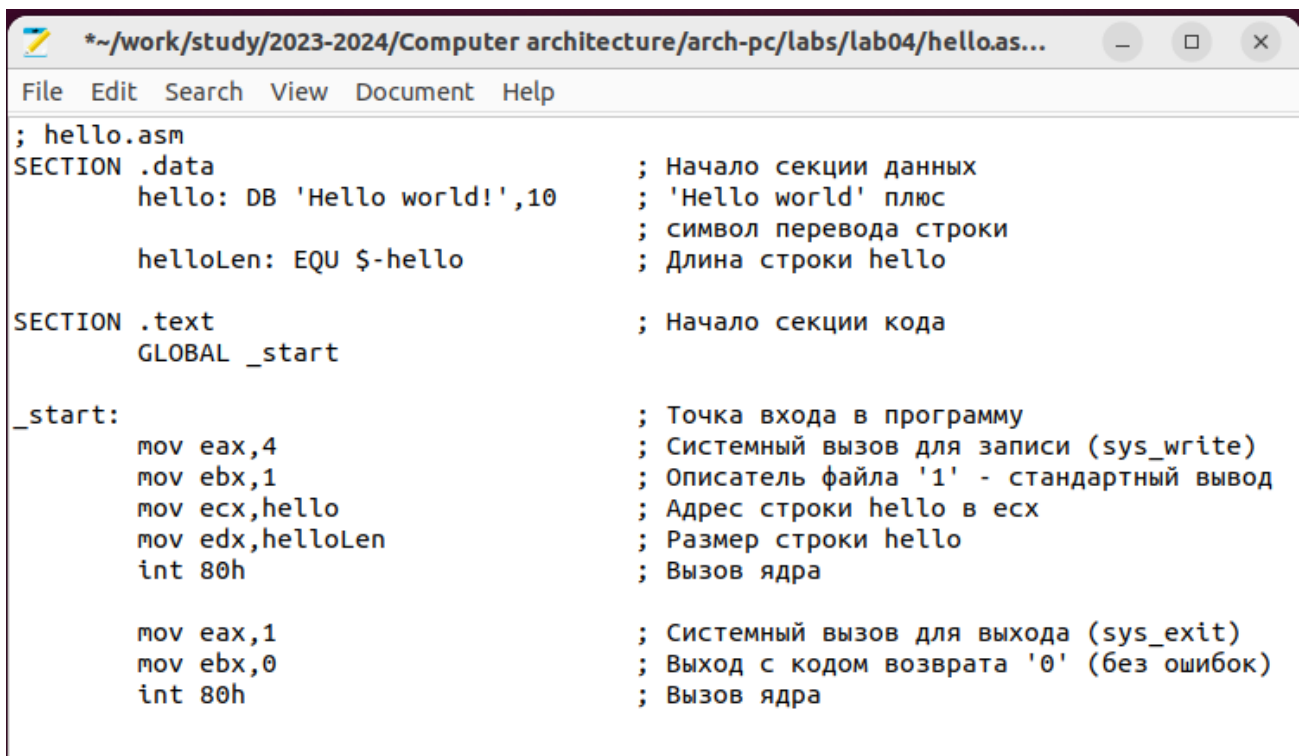
```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ mousepad hello.asm
```

~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04...

File Edit Search View Document Help

Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода "Hello word!". Так, как ассемблер не является высокоуровневым языком, каждая команда размещается на отдельной строке, так же обращаю внимание на регистр, так как Assembly чувствителен к нему. (рис. 04).



```
; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Hello world!',10             ; 'Hello world' плюс
                                              ; символ перевода строки
    helloLen: EQU $-hello                   ; Длина строки hello

SECTION .text                                ; Начало секции кода
    GLOBAL _start

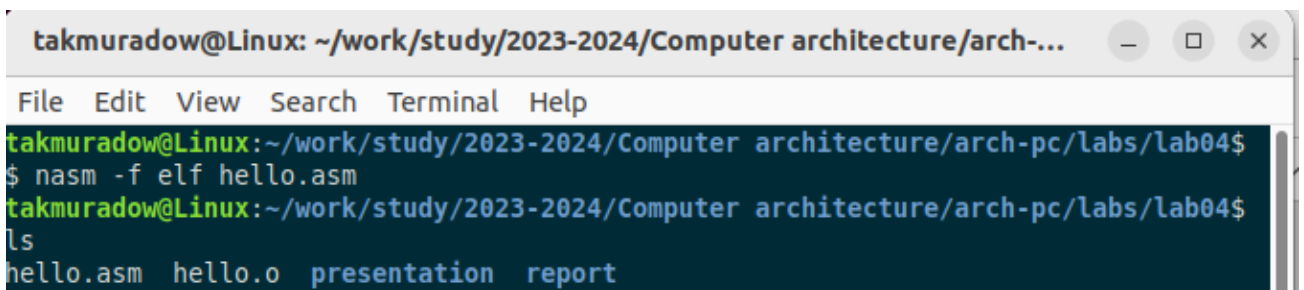
_start:                                      ; Точка входа в программу
    mov eax,4                               ; Системный вызов для записи (sys_write)
    mov ebx,1                               ; Описатель файла '1' - стандартный вывод
    mov ecx,hello                           ; Адрес строки hello в ecx
    mov edx,helloLen                        ; Размер строки hello
    int 80h                                ; Вызов ядра

    mov eax,1                               ; Системный вызов для выхода (sys_exit)
    mov ebx,0                               ; Выход с кодом возврата '0' (без ошибок)
    int 80h                                ; Вызов ядра
```

Заполнение файла

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. 05). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
takmuradow@Linux: ~/work/study/2023-2024/Computer architecture/arch-...
File Edit View Search Terminal Help
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$
$ nasm -f elf hello.asm
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$
ls
hello.asm hello.o presentation report
```

Компиляция текста программы

Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, используя ключ `-o` который задает имя объектному файлу, так же в файл будут включены символы для отладки (ключ `-g`), с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 06). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ nasm -o obj.o -f elf -g -l list.lst hello.asm
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ls
hello.asm hello.o list.lst obj.o presentation report
```

Компиляция текста программы

Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. 07). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ls
hello hello.asm hello.o list.lst obj.o presentation report
```

Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 08). 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

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ld -m elf_i386 obj.o -o main
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
```

Передача объектного файла на обработку компоновщику

Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 09).

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ./hello
Hello world!
```

Запуск исполняемого файла

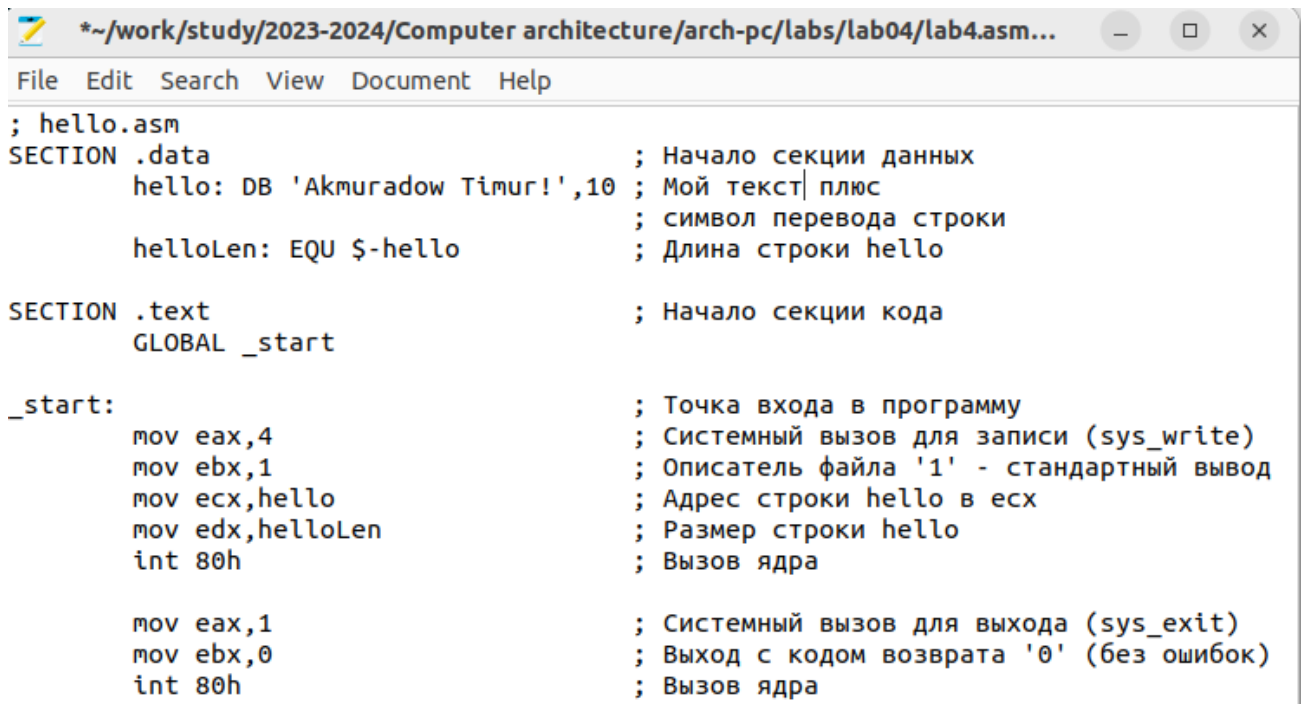
Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm (рис. 10).

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ cp hello.asm lab4.asm
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04
$ ls
hello hello.o list.lst obj.o report
hello.asm lab4.asm main presentation
```

Создание копии файла

С помощью текстового редактора mousepad открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).



```
*~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04/lab4.asm...
File Edit Search View Document Help

; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Akmuradow Timur!',10         ; Мой текст| плюс
                                           ; символ перевода строки
    helloLen: EQU $-hello                  ; Длина строки hello

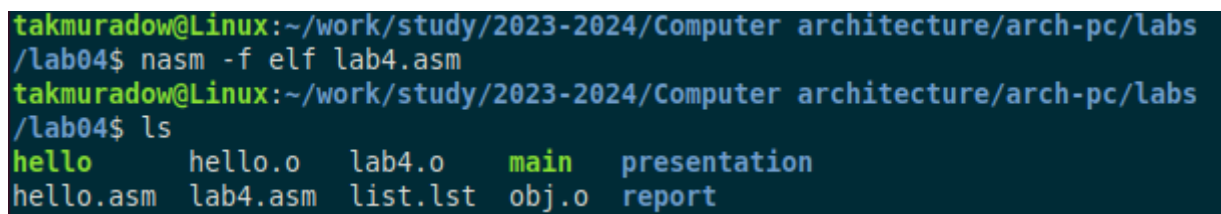
SECTION .text                                ; Начало секции кода
    GLOBAL _start

_start:                                     ; Точка входа в программу
    mov eax,4                               ; Системный вызов для записи (sys_write)
    mov ebx,1                               ; Описатель файла '1' - стандартный вывод
    mov ecx,hello                           ; Адрес строки hello в ecx
    mov edx,helloLen                       ; Размер строки hello
    int 80h                                ; Вызов ядра

    mov eax,1                               ; Системный вызов для выхода (sys_exit)
    mov ebx,0                               ; Выход с кодом возврата '0' (без ошибок)
    int 80h                                ; Вызов ядра
```

Изменение программы

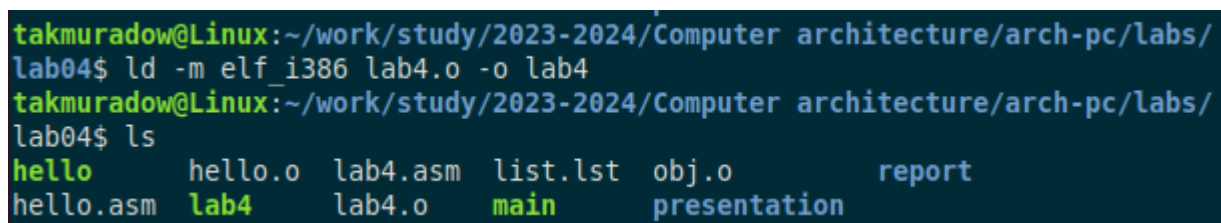
Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab4.o создан.



```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ nasm -f elf lab4.asm
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ ls
hello      hello.o   lab4.o    main      presentation
hello.asm  lab4.asm  list.lst  obj.o     report
```

Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 13).



```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ ld -m elf_i386 lab4.o -o lab4
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04$ ls
hello      hello.o   lab4.asm  list.lst  obj.o     report
hello.asm  lab4      lab4.o    main      presentation
```

Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 14).

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ ./lab4
Akmuradow Timur!
```

Запуск исполняемого файла

К сожалению, я начала работу не в том каталоге, поэтому создаю другую директорию lab04 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.

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ mkdir ~/work/study/2023-2024/"Computer architecture"/arch-pc/lab04
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ cp * ~/work/study/2023-2024/"Computer architecture"/arch-pc/lab04
cp: -r not specified; omitting directory 'presentation'
cp: -r not specified; omitting directory 'report'
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ ls ~/work/study/2023-2024/"Computer architecture"/arch-pc/lab04
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
```

Создании копии файлов в новом каталоге 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).

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ rm hello hello.o lab4 lab4.o list.lst main obj.o
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc/labs/
lab04$ ls
hello.asm lab4.asm presentation report
```

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).

```
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc$ git
add .
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc$ git
commit -m "Add files for lab04"
[master 2edec63] Add files for lab04
9 files changed, 58 insertions(+)
```

Добавление файлов на GitHub

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

```
created mode 100644 tabs/tabs/report/image/images1.png
takmuradow@Linux:~/work/study/2023-2024/Computer architecture/arch-pc$ git push
Enumerating objects: 29, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 6 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (21/21), 267.69 KiB | 2.25 MiB/s, done.
Total 21 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), completed with 4 local objects.
To github.com:takmuradow/study_2023-2024_arch-pc.git
 42c13c3..6276145 master -> master
```

Отправка файлов

Выводы

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

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf