

ОТЧЕТ

по лабораторной работе №4

"Создание и процесс обработки программ на языке ассемблера NASM"

Выполнил: Нхари Хатим

Группа: НБИбд-03-25

Дата: 18.01.2026

1. Цель работы

Освоение процедуры компиляции (трансляции) и сборки (компоновки) программ, написанных на ассемблере NASM.

2. Выполнение лабораторной работы

2.1. Создание рабочего каталога lab04


Команда:

```
mkdir -p ~/work/arch-pc/lab04
```

Результат:

Каталог `~/work/arch-pc/lab04` создан.

Скриншот 1:



```
user@HP:~/work$ cd
user@HP:~$ mkdir -p ~/work/arch-pc/lab04
user@HP:~$
```

Комментарий:

Каталог используется для хранения файлов лабораторной работы №4.

2.2. Переход в рабочий каталог


Команда:

```
cd ~/work/arch-pc/lab04  
pwd
```

Результат:

```
/home/user/rk/arch-pc/lab04
```

Скриншот 2:



```
user@HP:~$ cd ~/work/arch-pc/lab04  
pwd  
/home/user/work/arch-pc/lab04  
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Переходим в каталог, где будут размещены исходные файлы на ассемблере.

2.3. Создание файла hello.asm

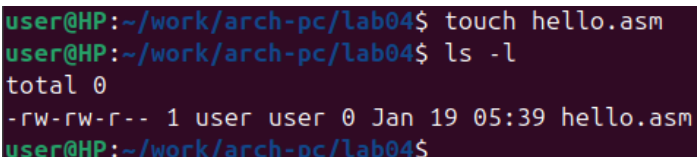
Команда:

```
touch hello.asm  
ls -l
```

Результат (пример):

```
-rw-rw-r-- 1 user user 0 Jan 19 05:39 hello.asm
```

Скриншот 3:



```
user@HP:~/work/arch-pc/lab04$ touch hello.asm  
user@HP:~/work/arch-pc/lab04$ ls -l  
total 0  
-rw-rw-r-- 1 user user 0 Jan 19 05:39 hello.asm  
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Файл `hello.asm` будет содержать программу "Hello world!" на NASM.

2.4. Ввод текста программы Hello world!

Команда (открытие в редакторе):

```
gedit hello.asm
```

Листинг hello.asm:

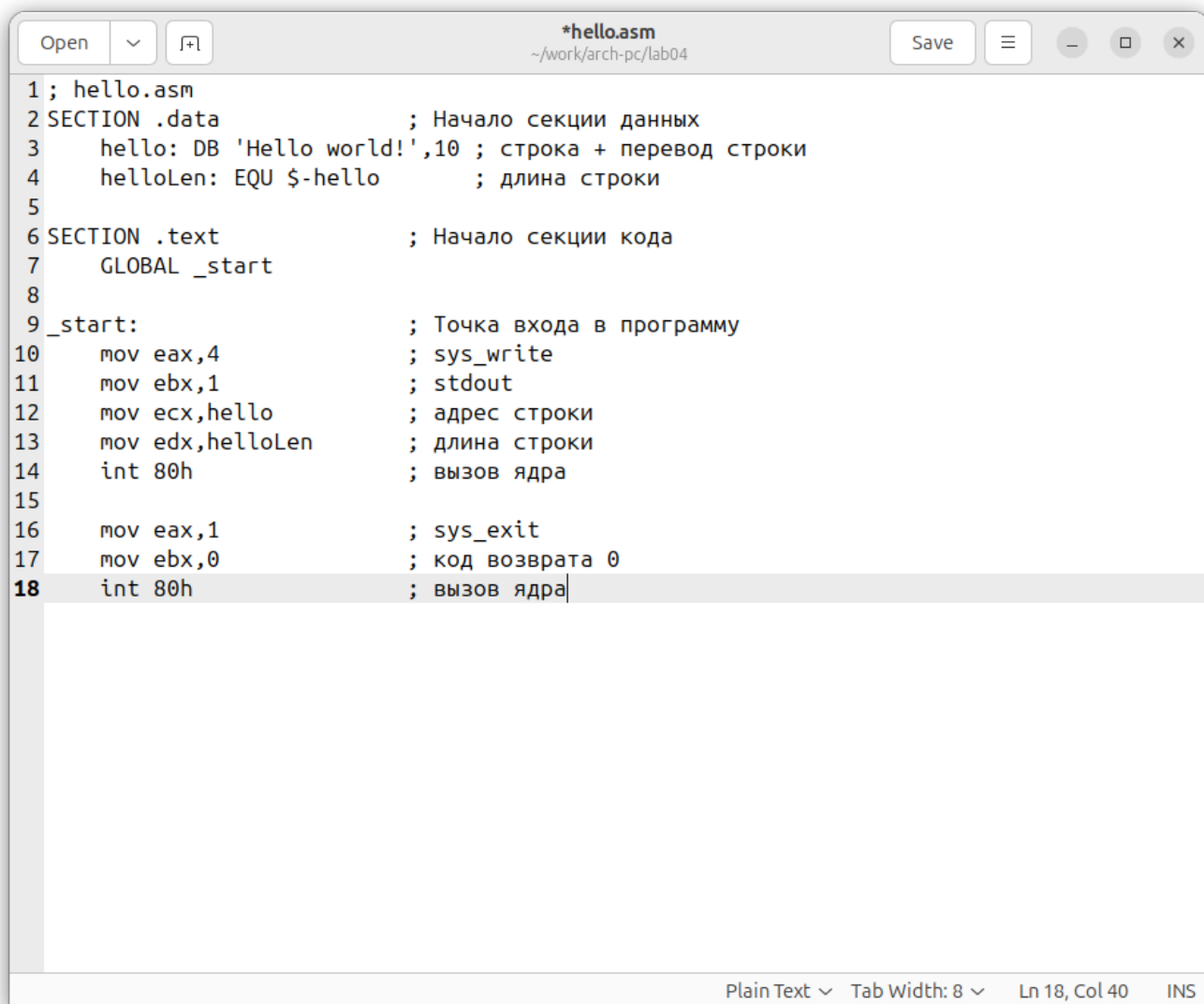
```
; hello.asm
SECTION .data                ; Начало секции данных
    hello: DB 'Hello world!',10 ; строка + перевод строки
    helloLen: EQU $-hello      ; длина строки

SECTION .text                ; Начало секции кода
    GLOBAL _start

_start:                      ; Точка входа в программу
    mov eax,4                ; sys_write
    mov ebx,1                ; stdout
    mov ecx,hello            ; адрес строки
    mov edx,helloLen         ; длина строки
    int 80h                  ; вызов ядра

    mov eax,1                ; sys_exit
    mov ebx,0                ; код возврата 0
    int 80h                  ; вызов ядра
```

Скриншот 4:



The screenshot shows a text editor window titled '*hello.asm' with the path '~/.work/arch-pc/lab04'. The editor contains the following assembly code:

```
1 ; hello.asm
2 SECTION .data          ; Начало секции данных
3     hello: DB 'Hello world!',10 ; строка + перевод строки
4     helloLen: EQU $-hello      ; длина строки
5
6 SECTION .text          ; Начало секции кода
7     GLOBAL _start
8
9 _start:                ; Точка входа в программу
10    mov eax,4           ; sys_write
11    mov ebx,1           ; stdout
12    mov ecx,hello       ; адрес строки
13    mov edx,helloLen    ; длина строки
14    int 80h            ; вызов ядра
15
16    mov eax,1           ; sys_exit
17    mov ebx,0           ; код возврата 0
18    int 80h            ; вызов ядра
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 18, Col 40', and 'INS'.

Комментарий:

Программа состоит из секции данных (`.data`) и секции кода (`.text`), выводит строку на экран через системный вызов `write` и завершается системным вызовом `exit`.

2.5. Трансляция программы NASM (получение объектного файла)

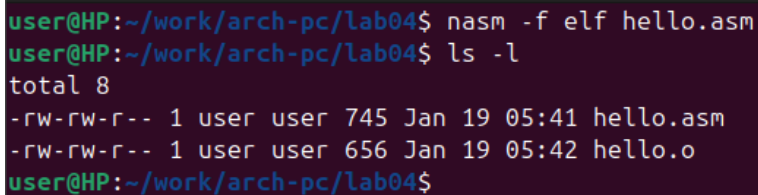
Команда:

```
nasm -f elf hello.asm
ls -l
```

Результат (пример):

```
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
```

Скриншот 5:



```
user@HP:~/work/arch-pc/lab04$ nasm -f elf hello.asm
user@HP:~/work/arch-pc/lab04$ ls -l
total 8
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Команда `nasm -f elf hello.asm` транслирует исходный код в объектный файл формата ELF для 32-битной сборки (`hello.o`).

2.6. Расширенная команда NASM: свой объектный файл + листинг + отладочные символы

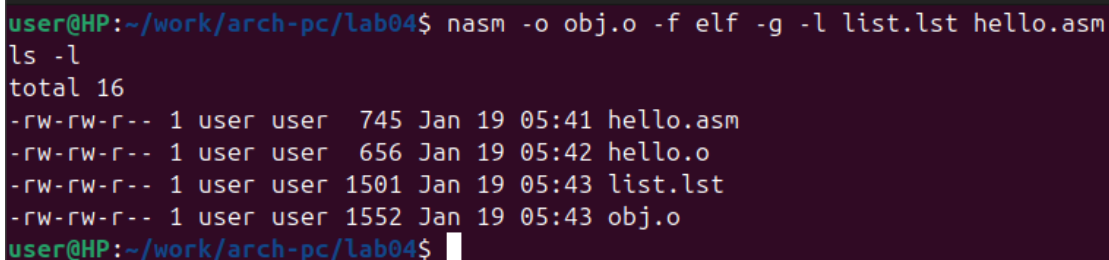
Команда:

```
nasm -o obj.o -f elf -g -l list.lst hello.asm
ls -l
```

Результат (пример):

```
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
```

Скриншот 6:



```
user@HP:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
ls -l
total 16
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

- `-o obj.o` задаёт имя объектного файла

- `-g` добавляет отладочные символы
 - `-l list.lst` создаёт файл листинга, где содержится текст программы и доп. информация транслятора
-

2.7. Компоновка (линковка) объектного файла hello.o

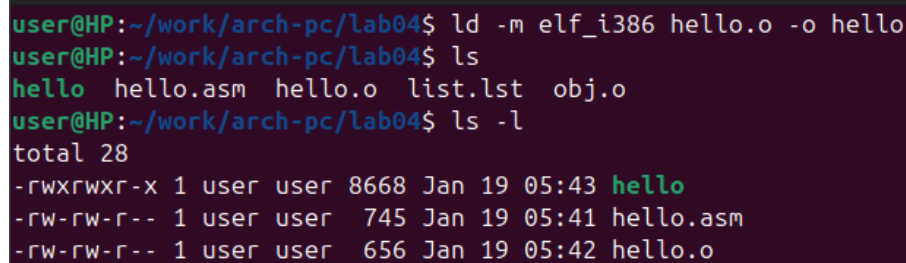
Команда:

```
ld -m elf_i386 hello.o -o hello
ls -l
```

Результат (пример):

```
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
```

Скриншот 7:



```
user@HP:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
user@HP:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst obj.o
user@HP:~/work/arch-pc/lab04$ ls -l
total 28
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
```

Комментарий:

Компоновщик `ld` собирает исполняемый файл из объектного. Ключ `-m elf_i386` указывает на 32-битный формат.

2.8. Компоновка obj.o в исполняемый файл main

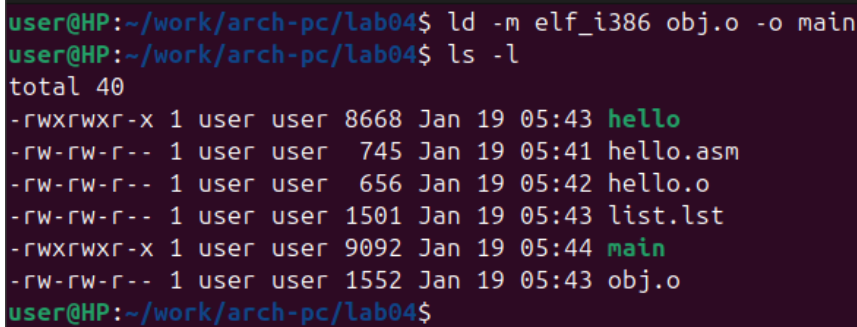
Команда:

```
ld -m elf_i386 obj.o -o main
ls -l
```

Результат (пример):

```
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
-rw-rw-r-- 1 user user  745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user  656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rwxrwxr-x 1 user user 9092 Jan 19 05:44 main
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
```

Скриншот 8:



```
user@HP:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
user@HP:~/work/arch-pc/lab04$ ls -l
total 40
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
-rw-rw-r-- 1 user user  745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user  656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rwxrwxr-x 1 user user 9092 Jan 19 05:44 main
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Исполняемый файл называется `main`, объектный файл — `obj.o` (он был создан расширенной командой NASM).

2.9. Запуск исполняемых файлов

Команды:

```
./hello
./main
```

Результат:

```
Hello world!
Hello world!
```

Скриншот 9:

```
user@HP:~/work/arch-pc/lab04$ ./hello
Hello world!
user@HP:~/work/arch-pc/lab04$ ./main
Hello world!
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Оба исполняемых файла выводят одно и то же сообщение, т.к. собраны из одинакового исходного текста `hello.asm`.

3. Задание для самостоятельной работы

3.1. Создание копии `hello.asm` → `lab4.asm`

Задание:

Создать копию исходного файла `hello.asm` с именем `lab4.asm`.

Команда:

```
cp hello.asm lab4.asm
ls -l
```

Результат (пример):

```
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
-rw-rw-r-- 1 user user  745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user  656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user  745 Jan 19 05:45 lab4.asm
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rwxrwxr-x 1 user user 9092 Jan 19 05:44 main
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
```


Скриншот 10:

```
user@HP:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
user@HP:~/work/arch-pc/lab04$ ls -l
total 44
-rwxrwxr-x 1 user user 8668 Jan 19 05:43 hello
-rw-rw-r-- 1 user user 745 Jan 19 05:41 hello.asm
-rw-rw-r-- 1 user user 656 Jan 19 05:42 hello.o
-rw-rw-r-- 1 user user 745 Jan 19 05:45 lab4.asm
-rw-rw-r-- 1 user user 1501 Jan 19 05:43 list.lst
-rwxrwxr-x 1 user user 9092 Jan 19 05:44 main
-rw-rw-r-- 1 user user 1552 Jan 19 05:43 obj.o
user@HP:~/work/arch-pc/lab04$
```

3.2. Изменение текста программы: вывод Фамилии и Имени

Команда (открытие редактора):

```
gedit lab4.asm
```

Листинг lab4.asm (изменённая строка вывода):

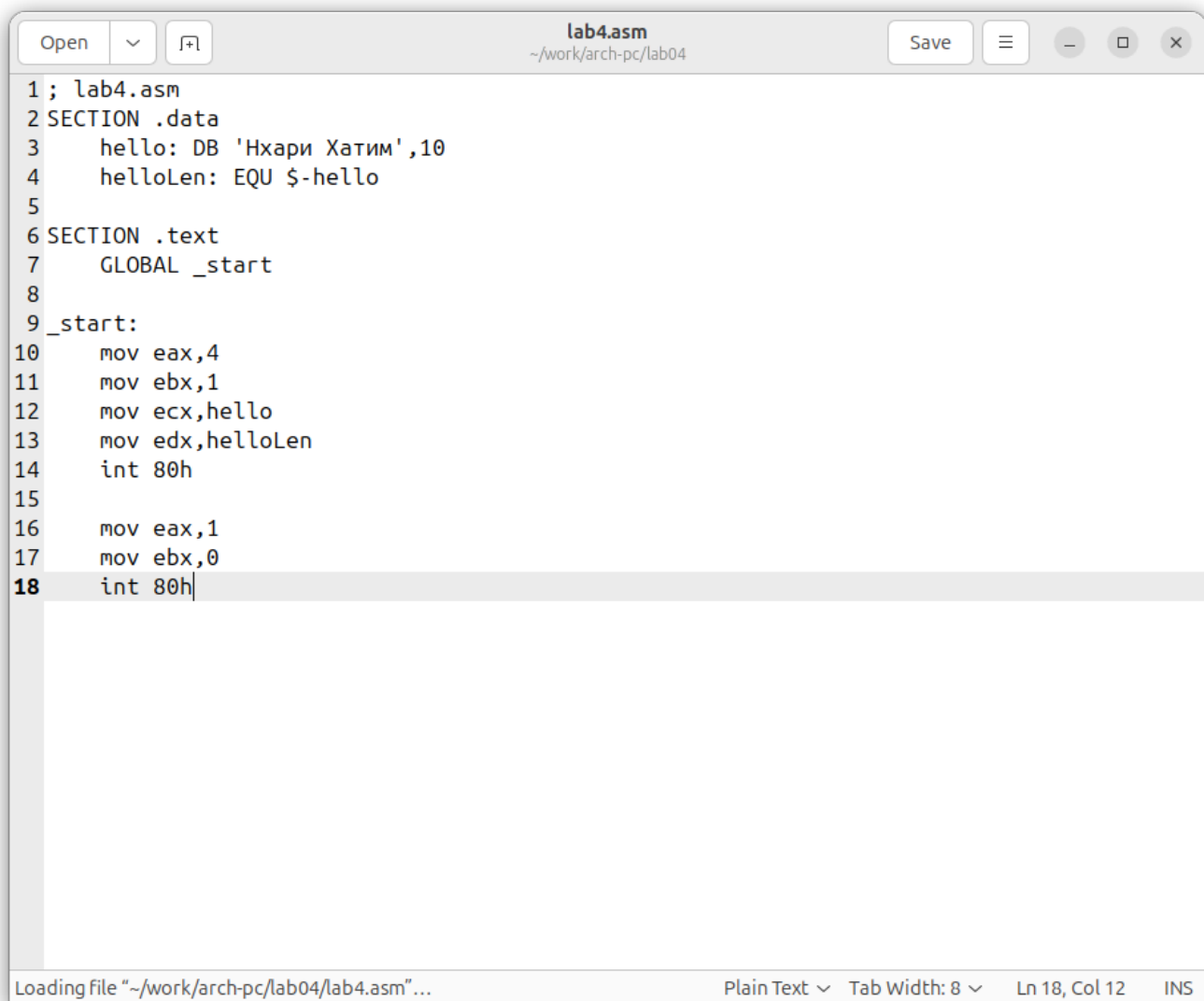
```
; lab4.asm
SECTION .data
    hello: DB 'Нхари Хатим',10
    helloLen: EQU $-hello

SECTION .text
    GLOBAL _start

_start:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h

    mov eax,1
    mov ebx,0
    int 80h
```

Скриншот 11:



```
1 ; lab4.asm
2 SECTION .data
3     hello: DB 'Нхари Хатим',10
4     helloLen: EQU $-hello
5
6 SECTION .text
7     GLOBAL _start
8
9 _start:
10    mov eax,4
11    mov ebx,1
12    mov ecx,hello
13    mov edx,helloLen
14    int 80h
15
16    mov eax,1
17    mov ebx,0
18    int 80h
```

Loading file "~/work/arch-pc/lab04/lab4.asm"...

Plain Text ▾ Tab Width: 8 ▾ Ln 18, Col 12 INS

Комментарий:

Изменена строковая константа: теперь вместо `Hello world!` выводятся фамилия и имя.

3.3. Трансляция, компоновка и запуск программы lab4.asm

Команды:

```
nasm -f elf lab4.asm
ld -m elf_i386 lab4.o -o lab4
./lab4
```

Результат:

Скриншот 12:

```
user@HP:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
ld -m elf_i386 lab4.o -o lab4
./lab4
Нхари Хатим
user@HP:~/work/arch-pc/lab04$
```

Комментарий:

Программа корректно выводит заданную строку и завершает работу.

3.4. Копирование файлов в локальный репозиторий и загрузка на GitHub

Действия:

Файлы `hello.asm` и `lab4.asm` были скопированы в каталог лабораторной работы внутри репозитория курса.

Команды (пример для структуры репозитория как в предыдущих работах):

```
# Переход в репозиторий курса
cd ~/work/study/2025-2026/"Архитектура компьютера"/arch-pc

# Копирование файлов исходников
cp ~/work/arch-pc/lab04/hello.asm labs/lab04/
cp ~/work/arch-pc/lab04/lab4.asm labs/lab04/

# Проверка
ls -l labs/lab04/
```

Скриншот 13:

```
user@HP:~/work/arch-pc/lab04$ cd ~/work/study/2025-2026/"Архитектура компьютера"/arch-pc
user@HP:~/work/study/2025-2026/Архитектура компьютера/arch-pc$ cp ~/work/arch-pc/lab04/hello.asm labs/lab04/
cp ~/work/arch-pc/lab04/lab4.asm labs/lab04/
user@HP:~/work/study/2025-2026/Архитектура компьютера/arch-pc$ ls -l labs/lab04/
total 16
-rw-rw-r-- 1 user user 745 Jan 19 05:47 hello.asm
-rw-rw-r-- 1 user user 254 Jan 19 05:47 lab4.asm
drwxrwxr-x 5 user user 4096 Jan 18 18:48 presentation
drwxrwxr-x 6 user user 4096 Jan 18 18:48 report
user@HP:~/work/study/2025-2026/Архитектура компьютера/arch-pc$
```

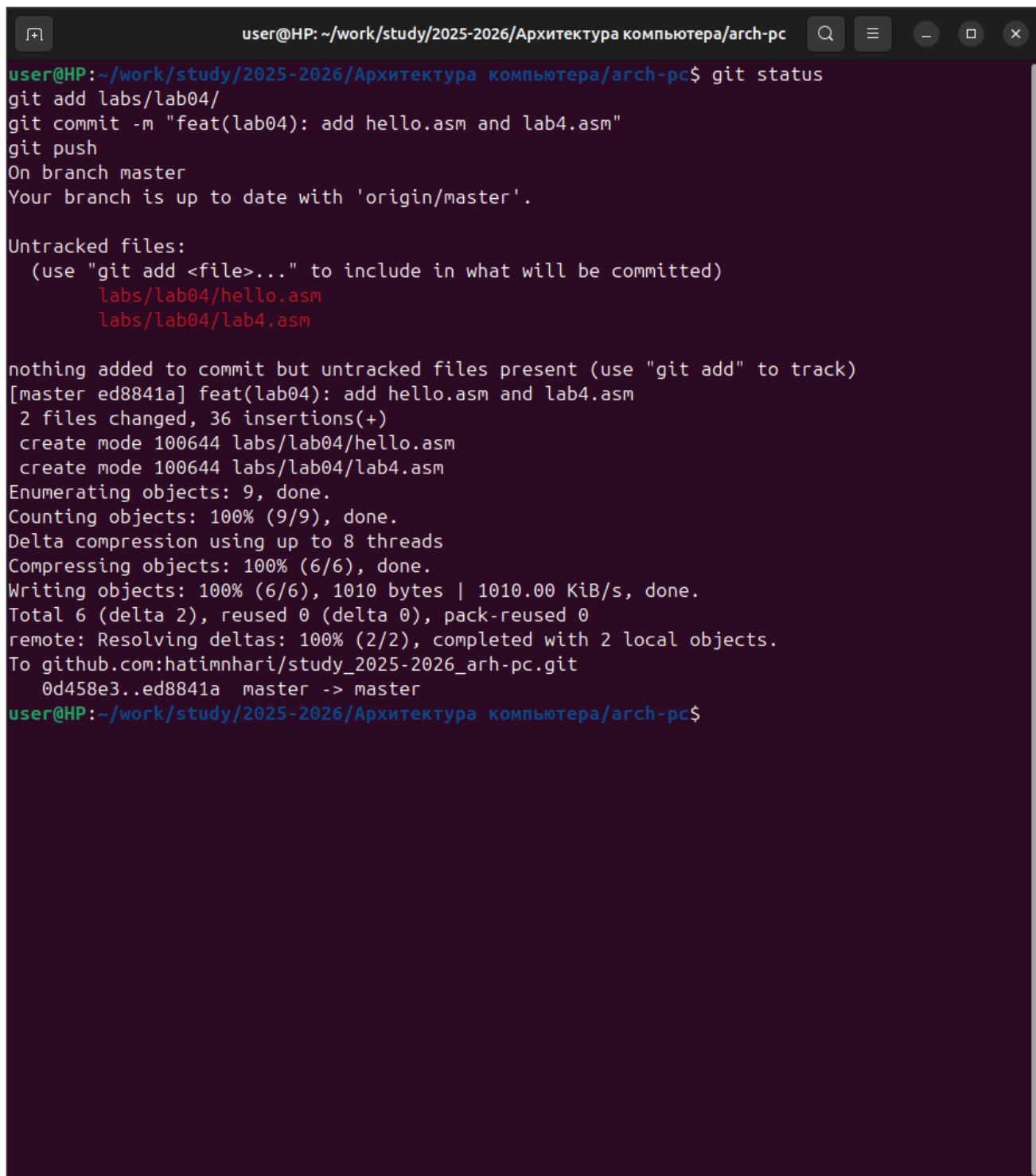
Команды Git:

```
git status
git add labs/lab04/
git commit -m "feat(lab04): add hello.asm and lab4.asm"
git push
```

Результат (пример):

```
[master ed8841a] feat(lab04): add hello.asm and lab4.asm
2 files changed, 36 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
```

Скриншот 14:

A terminal window with a dark background and light-colored text. The window title is 'user@HP: ~/work/study/2025-2026/Архитектура компьютера/arch-pc'. The terminal shows a series of git commands and their output. The commands are: 'git status', 'git add labs/lab04/', 'git commit -m "feat(lab04): add hello.asm and lab4.asm"', and 'git push'. The output shows that the branch is up to date, lists untracked files (labs/lab04/hello.asm and labs/lab04/lab4.asm), and then shows the commit and push process, including object enumeration, counting, compression, and writing to the remote repository. The final output shows the commit hash '0d458e3..ed8841a' and the branch 'master' being pushed to 'master' on the remote 'github.com:hatimnhari/study_2025-2026_arh-pc.git'.

```
user@HP: ~/work/study/2025-2026/Архитектура компьютера/arch-pc$ git status
git add labs/lab04/
git commit -m "feat(lab04): add hello.asm and lab4.asm"
git push
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
       labs/lab04/hello.asm
       labs/lab04/lab4.asm

nothing added to commit but untracked files present (use "git add" to track)
[master ed8841a] feat(lab04): add hello.asm and lab4.asm
 2 files changed, 36 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1010 bytes | 1010.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:hatimnhari/study_2025-2026_arh-pc.git
   0d458e3..ed8841a  master -> master
user@HP:~/work/study/2025-2026/Архитектура компьютера/arch-pc$
```

Комментарий:

Файлы лабораторной работы успешно добавлены и отправлены в удалённый репозиторий.

4. Ответы на вопросы для самопроверки

1. Какие основные отличия ассемблерных программ от программ на языках высокого уровня?

- Ассемблер ближе к аппаратуре: операции выполняются на уровне регистров и памяти.
 - Меньше абстракций: программист управляет тем, *какие* инструкции выполняются и как используются ресурсы.
 - Обычно больше кода для той же задачи (по сравнению с C/Python).
 - Высокая зависимость от архитектуры процессора и ОС (x86, ARM и т.д.).
-

2. В чём состоит отличие инструкции от директивы на языке ассемблера?

- **Инструкция** (например `mov, int`) преобразуется в машинный код и выполняется процессором.
 - **Директива** (например `SECTION, GLOBAL, DB, EQU`) управляет работой ассемблера и обычно не является командой процессора.
-

3. Перечислите основные правила оформления программ на языке ассемблера.

- Каждая команда располагается на отдельной строке (несколько команд в одной строке не допускаются).
 - Используются секции `.data`, `.bss`, `.text` (по назначению).
 - Точка входа задаётся через `GLOBAL _start` и метку `_start:`.
 - В конце программы должен быть корректный выход через системный вызов `exit`.
 - Рекомендуется использовать комментарии (`; ...`) и аккуратные отступы.
-

4. Каковы этапы получения исполняемого файла?

1. написание исходного текста `.asm`
 2. **трансляция** (получение объектного файла `.o`) через `nasm`
 3. **компоновка** (получение исполняемого файла) через `ld`
 4. запуск `./program` (при необходимости — отладка и исправление)
-

5. Каково назначение этапа трансляции?

Трансляция преобразует исходный текст на ассемблере в объектный машинный код (`.o`). На этом этапе также могут формироваться файлы листинга (`.lst`) и добавляться отладочные символы.

6. Каково назначение этапа компоновки?

Компоновка объединяет объектные файлы, формирует корректный исполняемый файл и настраивает секции/адреса/точку входа. Итог — файл, который можно запускать.

7. Какие файлы могут создаваться при трансляции программы, какие из них создаются по умолчанию?

- По умолчанию создаётся **объектный файл** `.o` (например `hello.o`).
 - Дополнительно могут создаваться:
 - **листинг** `.lst` (через `-l`)
 - объектный файл с заданным именем (через `-o`)
 - отладочные символы (через `-g` , включаются в объектный файл)
-

8. Каковы форматы файлов для `nasm` и `ld`?

- `nasm` принимает исходники `.asm` и создаёт объектные файлы формата **ELF** (например, `-f elf`).
 - `ld` принимает объектные файлы `.o` и создаёт исполняемый файл без расширения (например `hello` , `main` , `lab4`).
 - Для 32-битной сборки под Linux используется режим `elf_i386` (ключ `-m elf_i386`).
-

5. Выводы

В ходе выполнения лабораторной работы №4 была освоена процедура получения исполняемых файлов из программ на NASM: создание исходного файла `.asm` , трансляция в объектный файл `.o` (в том числе с созданием листинга и отладочных символов), компоновка объектного файла компоновщиком `ld` и запуск готовых программ. Также выполнено самостоятельное задание по модификации программы и загрузке исходных файлов в репозиторий GitHub. Цель лабораторной работы достигнута.

6. Список файлов работы

В каталоге `~/work/arch-pc/lab04` :

- `hello.asm` — программа “Hello world!”
- `lab4.asm` — программа вывода ФИО
- `hello.o`, `obj.o` — объектные файлы
- `list.lst` — файл листинга (создан опцией `-l`)
- `hello`, `main`, `lab4` — исполняемые файлы

В репозитории курса:

- `labs/lab04/hello.asm`
- `labs/lab04/lab4.asm`

Ссылка на репозиторий GitHub:

```
https://github.com/username/study_2025-2026_arch-pc
```

Конец отчета