Содержание

Список иллюстраций

Список таблиц

1 ОТЧЕТ

1.1 по лабораторной работе №4

1.2 «Создание и процесс обработки программ на языке ассемблера NASM»

Выполнил: Нхари Хатим

Группа: НБИбд-03-25

Дата: 18.01.2026

1.3 1. Цель работы

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

1.4 2. Выполнение лабораторной работы

1.4.1 2.1. Создание рабочего каталога lab04

Команда:

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

Результат:

Каталог ~/work/arch-pc/lab04 создан.

Скриншот 1:

Комментарий:

Каталог используется для хранения файлов лабораторной работы №4.

1.4.2 2.2. Переход в рабочий каталог

Команда:

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

Результат:

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

Скриншот 2:

Комментарий:

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

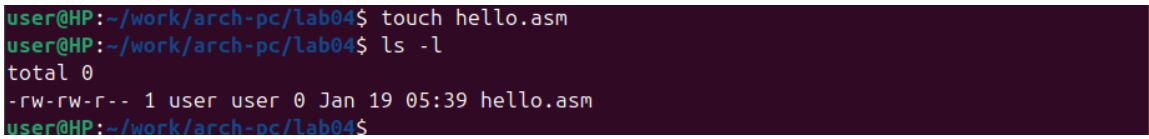
1.4.3 2.3. Создание файла hello.asm

Команда:

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

Результат (пример):

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

Скриншот 3: 

Комментарий:

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

1.4.4 2.4. Ввод текста программы Hello world!

Команда (открытие в редакторе):

```
gedit hello.asm
```

Листинг hello.asm:

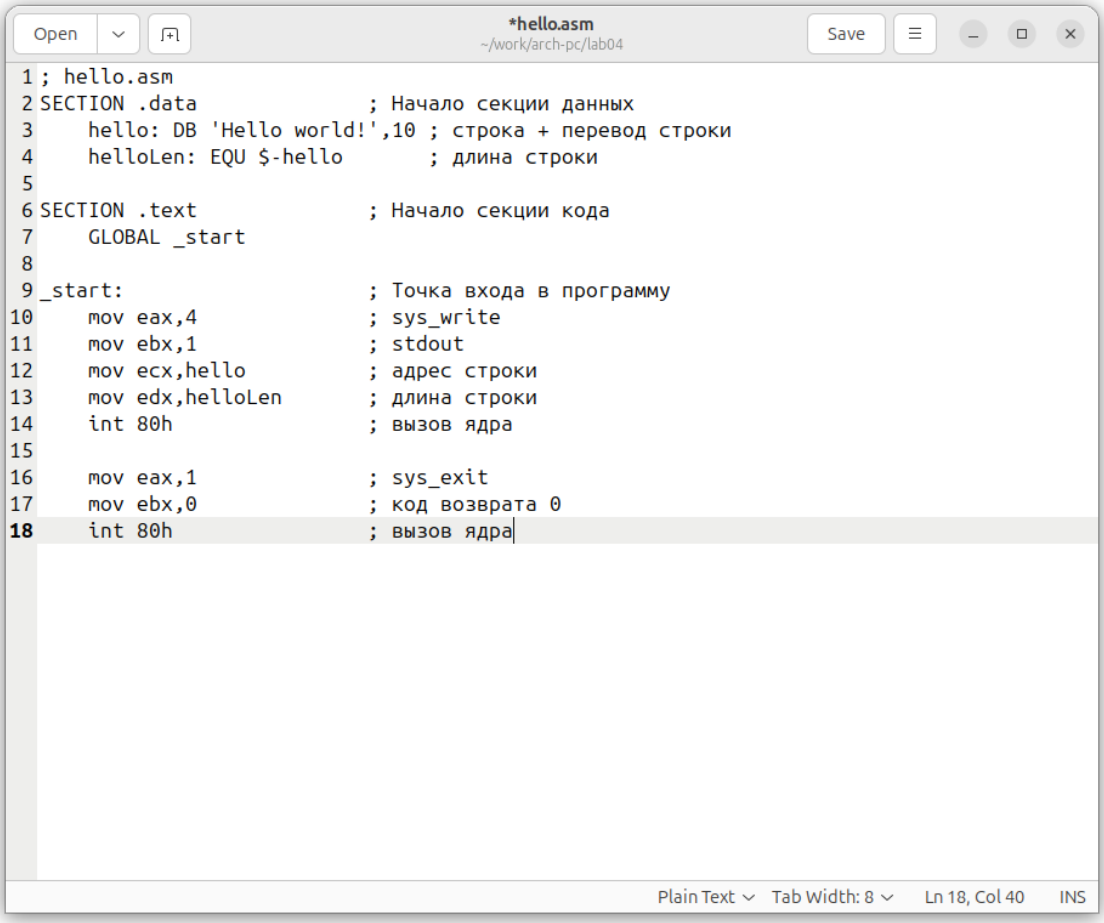
```
; hello.asm
SECTION .data              ; Начало секции данных
    hello: DB 'Hello world!',10 ; строка + перевод строки
    helloLen: EQU $-hello       ; длина строки

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:

Комментарий:

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

1.4.5 2.5. Трансляция программы NASM (получение объектного файла)

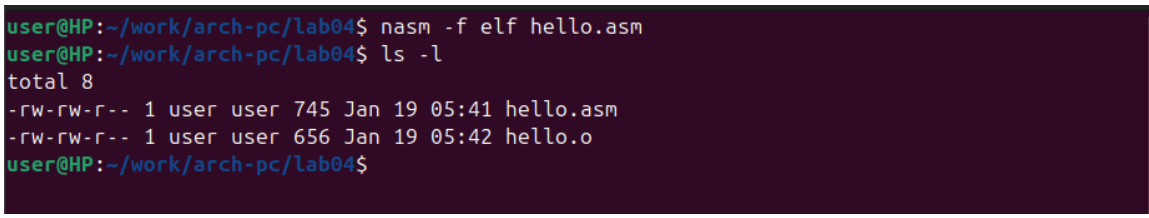
Команда:

```
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:



Комментарий:

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

1.4.6 2.6. Расширенная команда NASM: свой объектный файл + листинг + отладочные символы

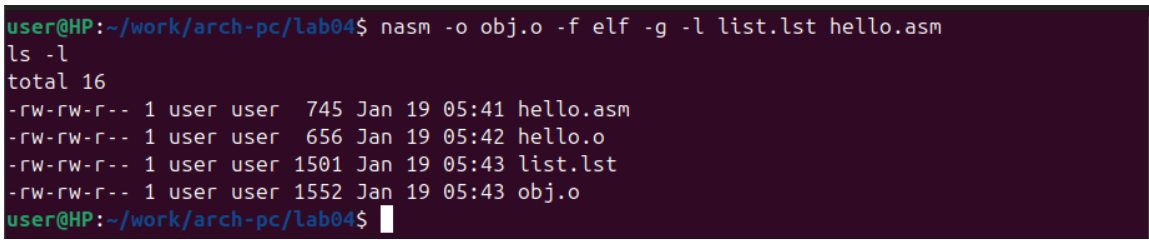
Команда:

```
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:



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

1.4.7 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:

Комментарий:

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

1.4.8 2.8. Компоновка obj.o в исполняемый файл main

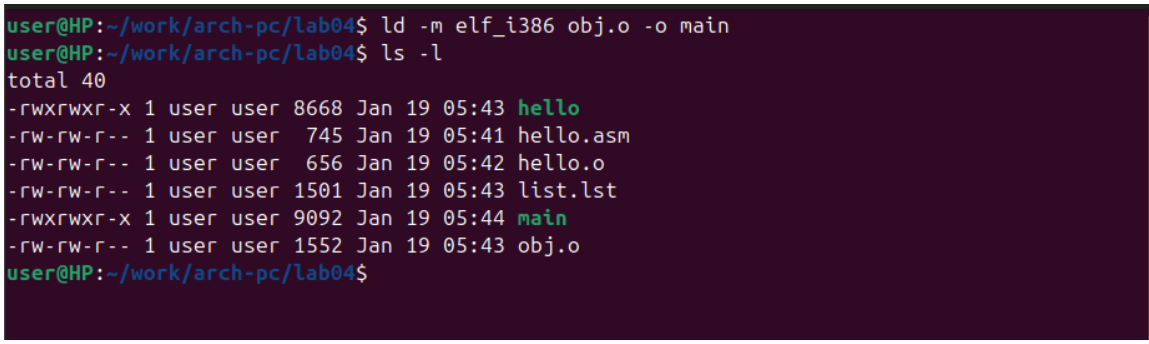
Команда:

```
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:



Комментарий:

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

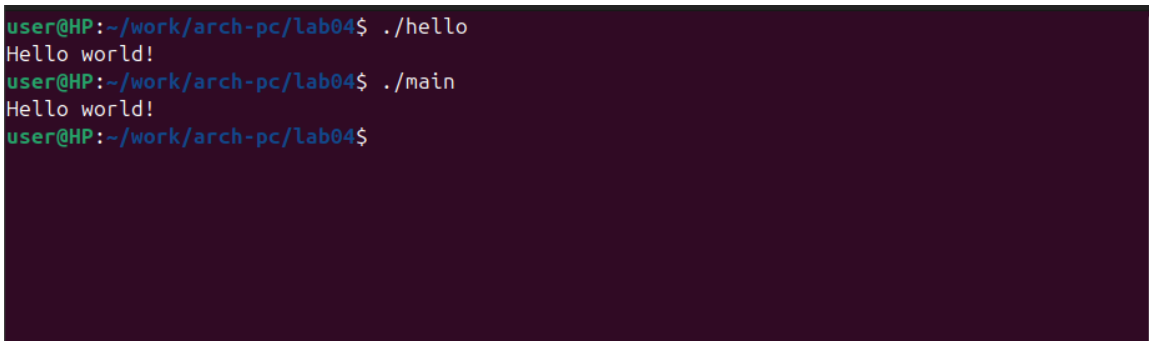
1.4.9 2.9. Запуск исполняемых файлов

Команды:

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

Результат:

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

Скриншот 9: 

Комментарий:

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

1.5 3. Задание для самостоятельной работы

1.5.1 3.1. Создание копии hello.asm → lab4.asm

Задание:

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

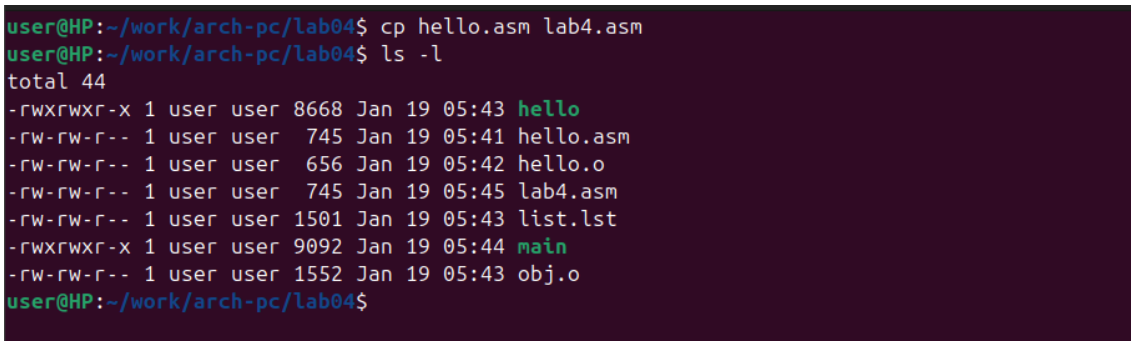
Команда:

```
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:



1.5.2 3.2. Изменение текста программы: вывод Фамилии и Имени

Команда (открытие редактора):

```
gedit lab4.asm
```

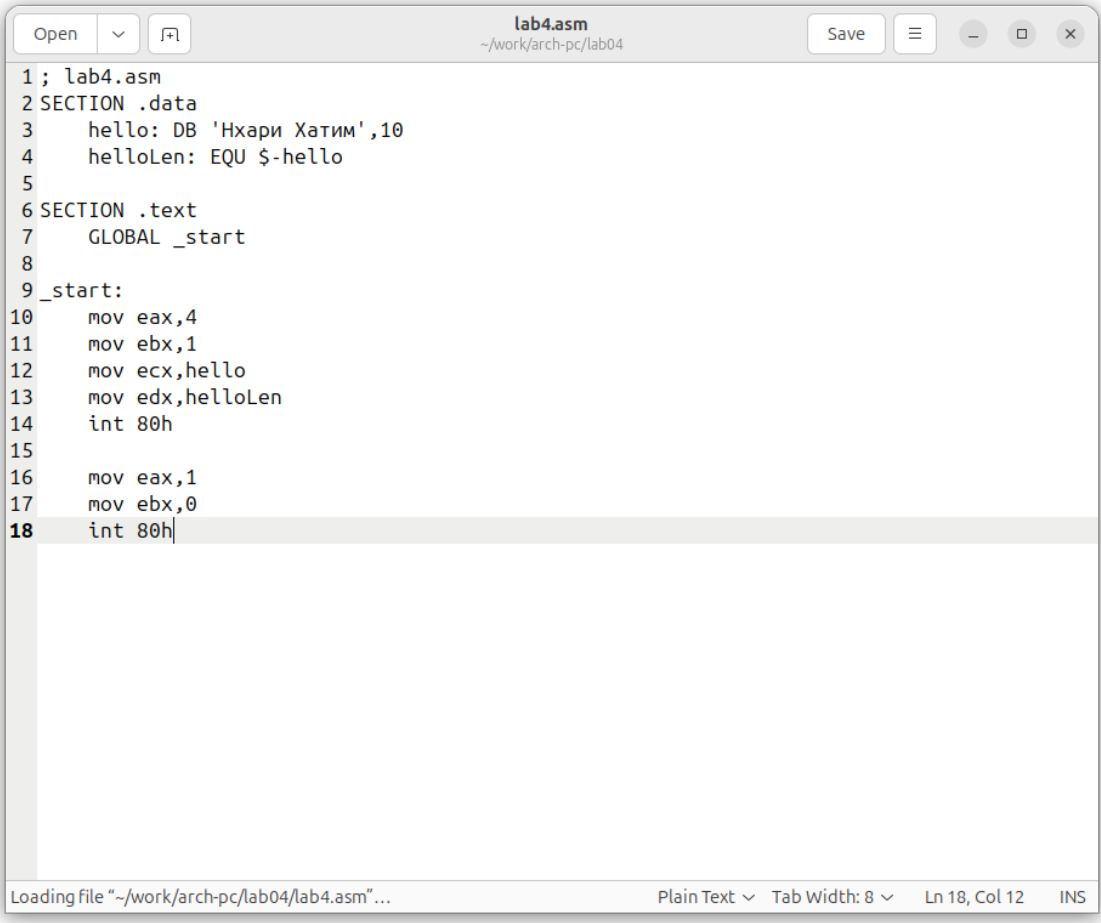
Листинг lab4.asm (изменённая строка вывода):

```
; 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:

Комментарий:

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

1.5.3 3.3. Трансляция, компоновка и запуск программы lab4.asm

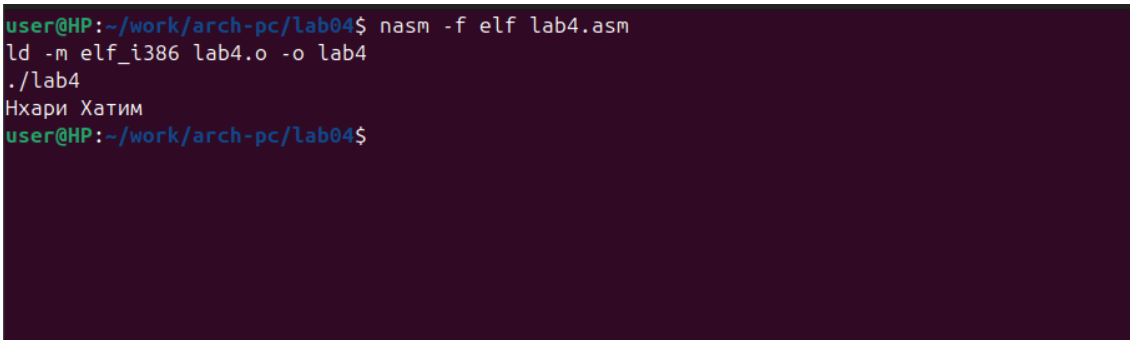
Команды:

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

Результат:

```
Нхари Хатим
```

Скриншот 12:



Комментарий:

Программа корректно выводит заданную строку и завершает работу.

1.5.4 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.

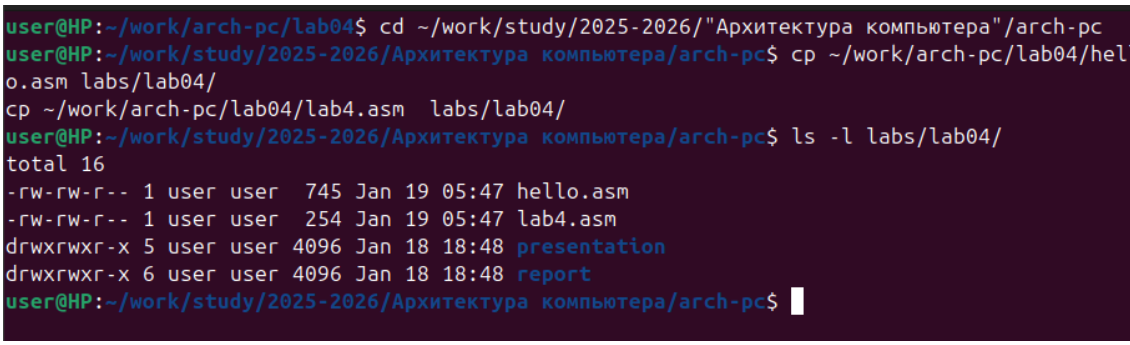
Команды (пример для структуры репозитория как в предыдущих работах):

```
# 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:

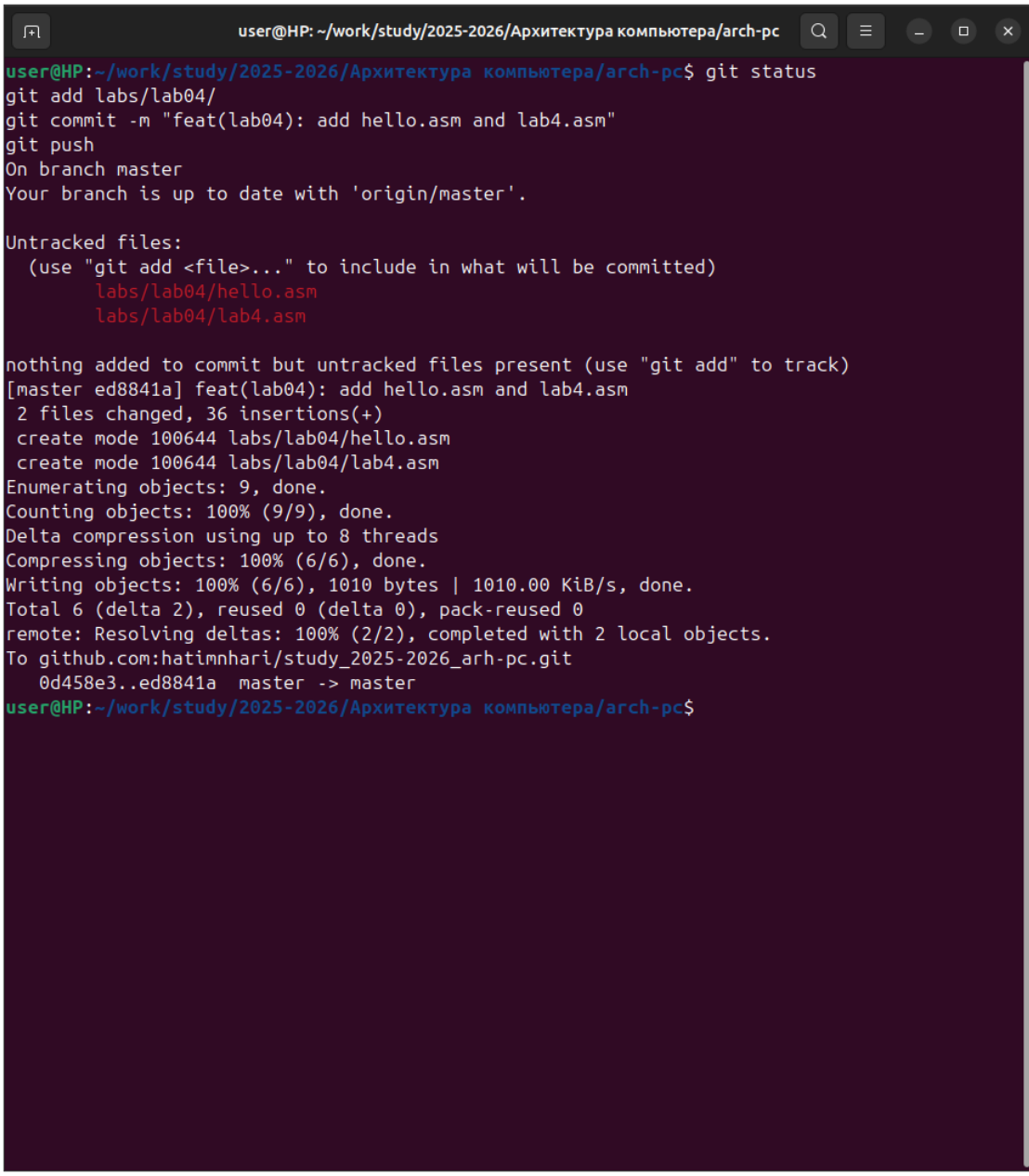


Команды Git:

```
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:

Комментарий:

Файлы лабораторной работы успешно добавлены и отправлены в удалённый репозиторий.

1.6 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`).

1.7 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.

1.8 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