Отчет по лабораторной работе №5

Архитектура компьютера

Рогожина Надежда Александровна

Содержание

# 1 Цель работы

Целью данной работы является изучение языка Assembler, освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# 2 Задание

1. В каталоге ~/work/arch-pc/lab05 с помощью команды cp создайте копию файла hello.asm с именем lab5.asm
2. С помощью любого текстового редактора внесите изменения в текст программы в файле lab5.asm так, чтобы вместо Hello world! на экран выводилась строка с вашими фамилией и именем.
3. Оттранслируйте полученный текст программы lab5.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.
4. Скопируйте файлы hello.asm и lab5.asm в Ваш локальный репозиторий в каталог ~/work/study/2022-2023/“Архитектура компьютера”/arch-pc/labs/lab05/. Загрузите файлы на Github.

# 3 Теоретическое введение

## 3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (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 (системной) плате.

Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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*.

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.

В состав ЭВМ также входят периферийные устройства, которые можно разделить на:

• **устройства внешней памяти**, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты);

• **устройства ввода-вывода**, которые обеспечивают взаимодействие ЦП с внешней средой.

При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. В самом общем виде 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.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы.

## 3.2 Ассемблер и язык ассемблера

**Язык ассемблера (assembly language, сокращённо asm)** — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора.

Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — **машинные коды**. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — **Ассемблер**.

## 3.3 Процесс создания и обработки программы на языке ассемблера

В процессе создания ассемблерной программы можно выделить четыре шага:

• **Набор текста** программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип asm.

• **Трансляция** — преобразование с помощью транслятора, например nasm, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — o, файла листинга — lst.

• **Компоновка или линковка** — этап обработки объектного кода компоновщиком (ld), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение map.

• **Запуск программы.** Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

# 4 Выполнение лабораторной работы

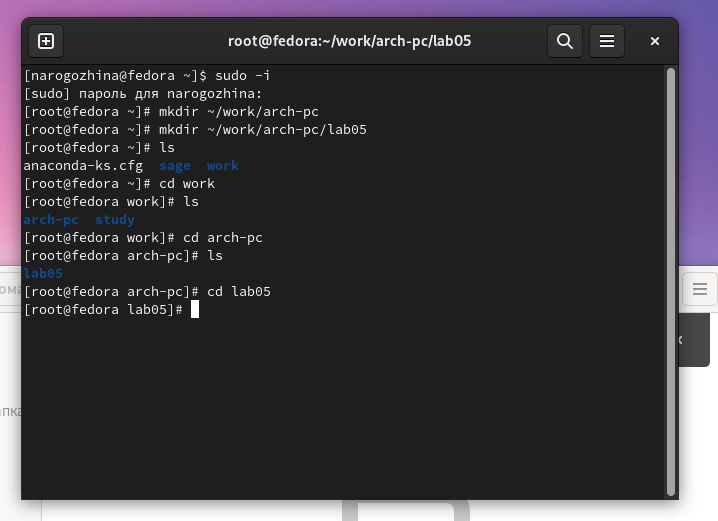


Рис. 1: Создадим и перейдем в каталог ~/work/arch-pc/lab05

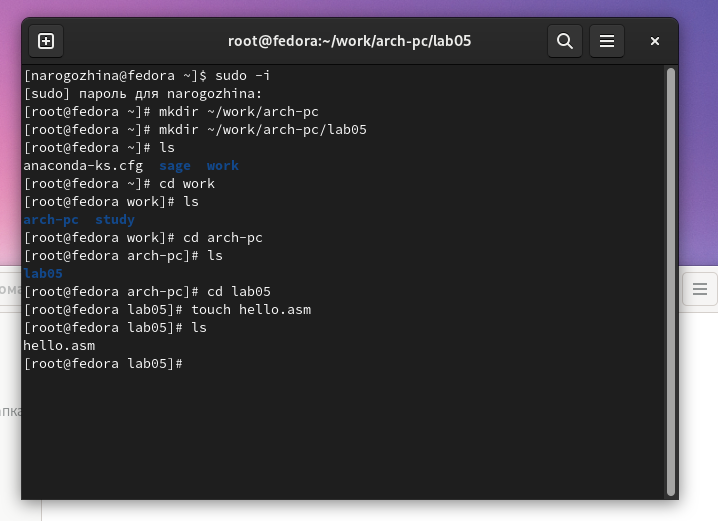


Рис. 2: Создадим текстовый файл с именем hello.asm

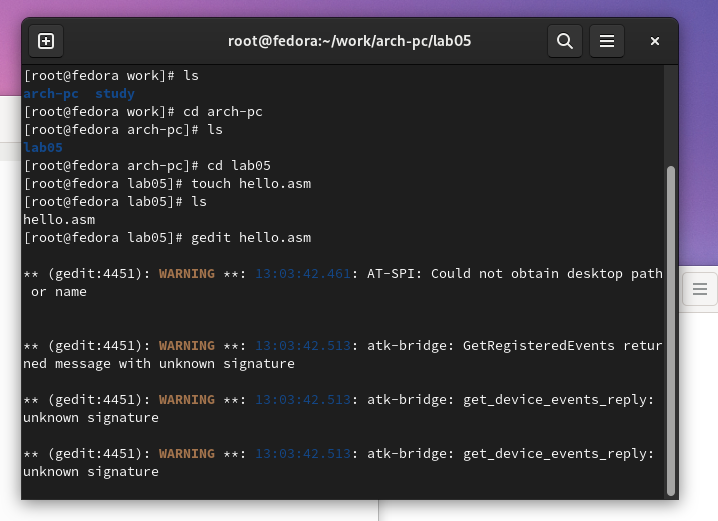


Рис. 3: Откроем этот файл с помощью текстового редактора

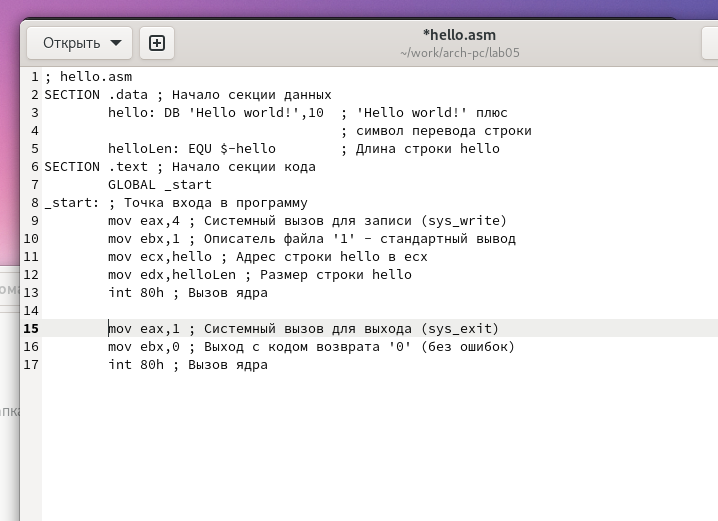


Рис. 4: Введем в него следуюзий текст:

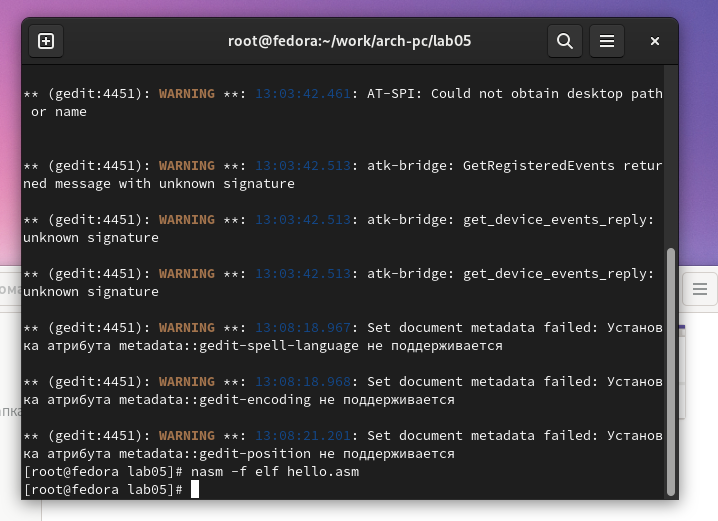


Рис. 5: Скомпилируем программу

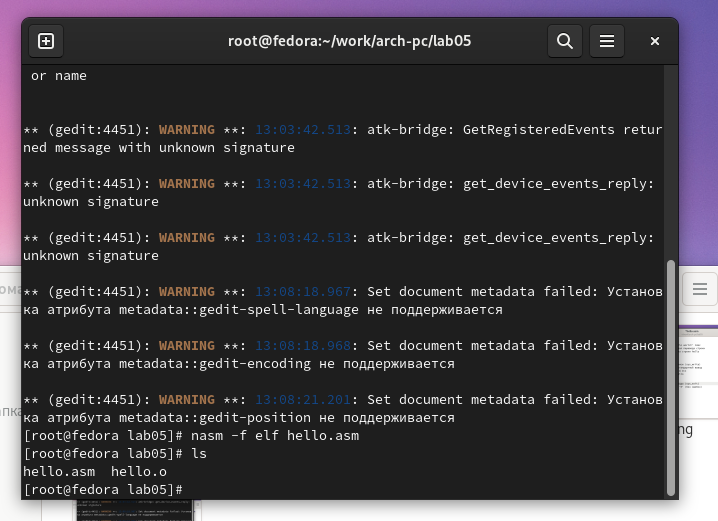


Рис. 6: Проверим компиляцию программы и создание файла с объектным кодом

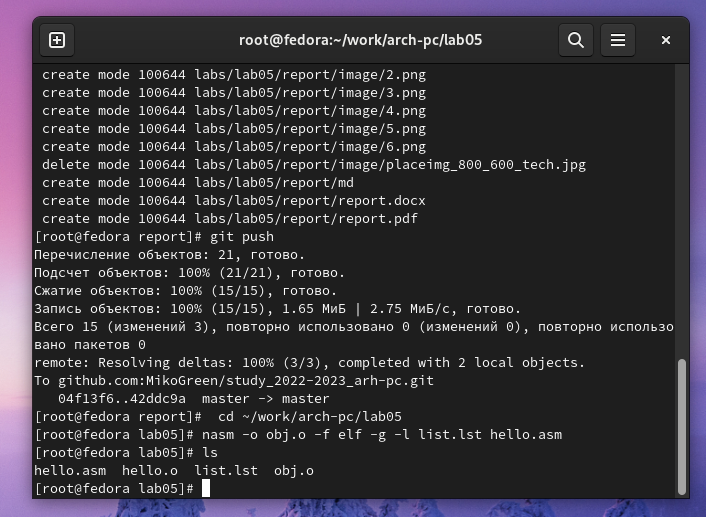


Рис. 7: Скомпилируем файл с присвоением имени, а также создадим файл листинга

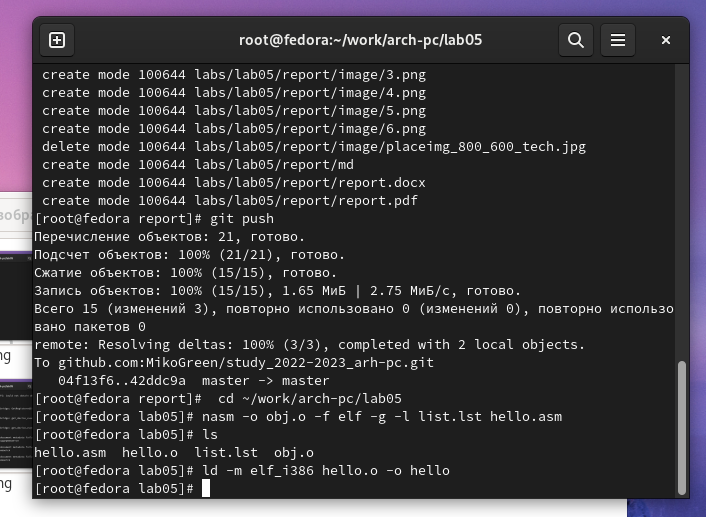


Рис. 8: Передадим объектный файл на обработку компоновщику

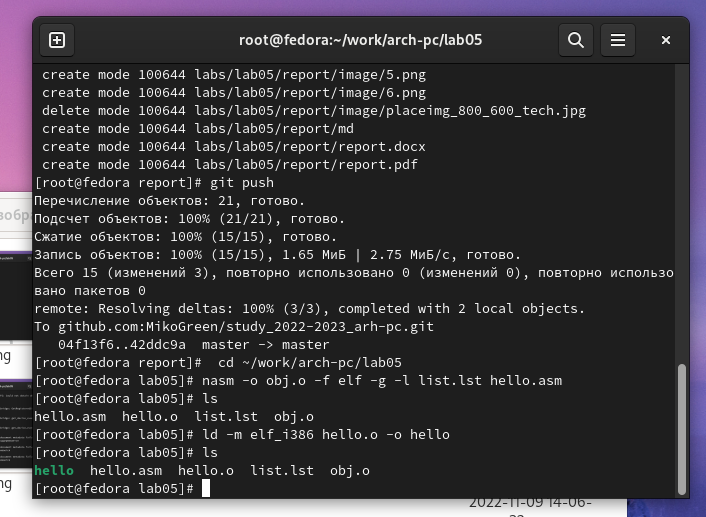


Рис. 9: Проверим создание исполняемого файла hello

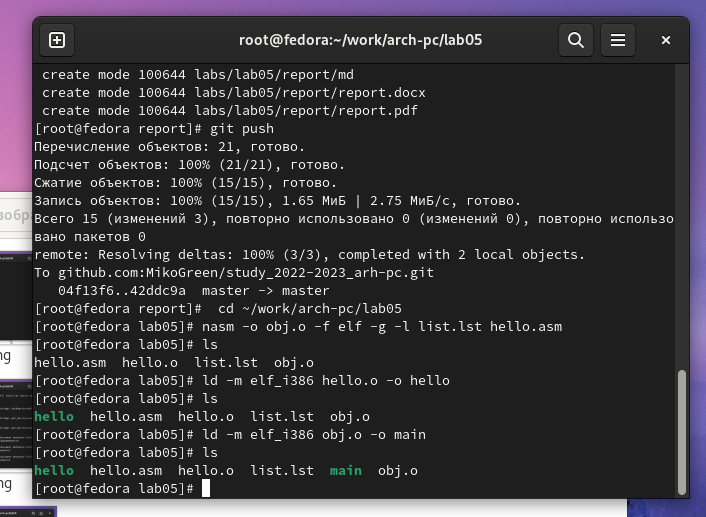


Рис. 10: Передадим файл компоновщику

В результате исполняемый файл имеет имя *main*, а объектный файл, из которого собран этот исполняемый файл имеет имя *obj.o*.

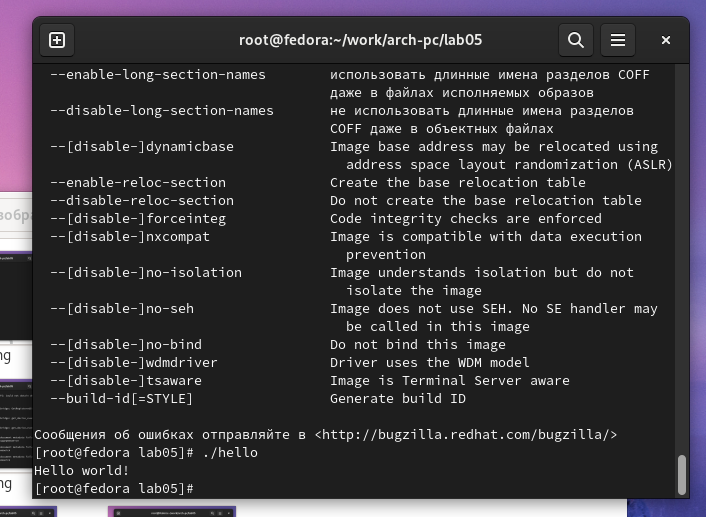


Рис. 11: Запустим исполняемый файл hello

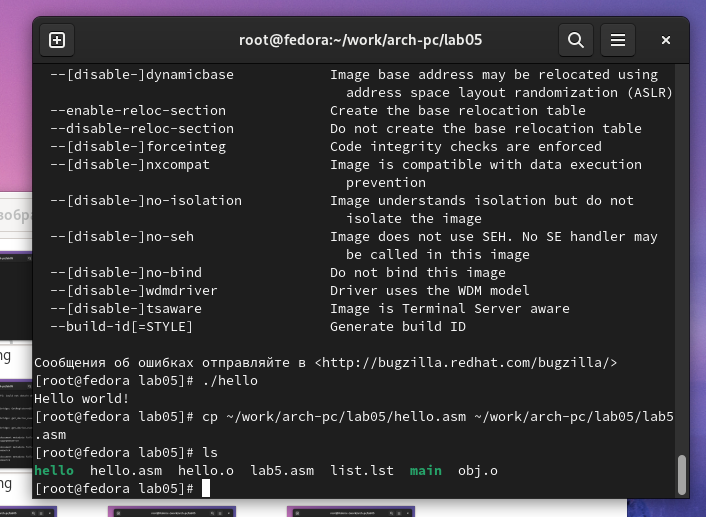


Рис. 12: Создадим компью файла hello.asm с именем lab5.asm

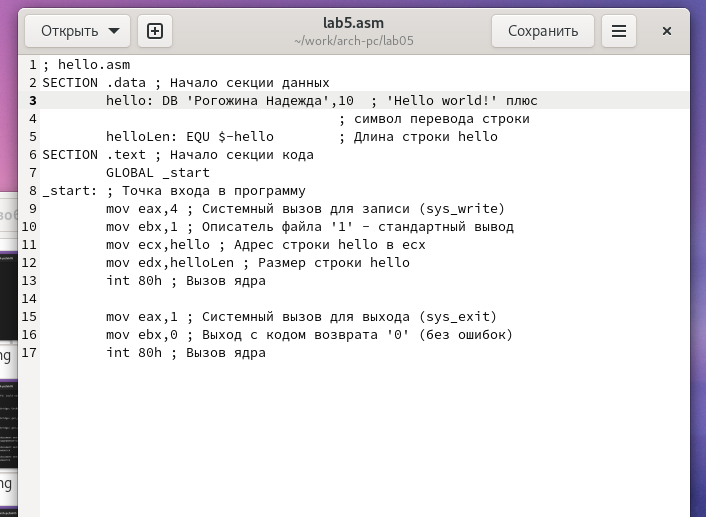


Рис. 13: С помощью текстового редактора внесем изменения в текст программы

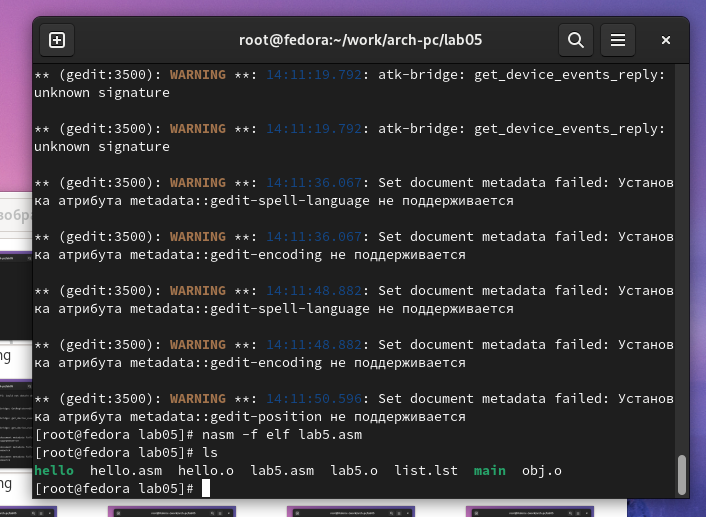


Рис. 14: Трансируем lab5.asm в объектный файл

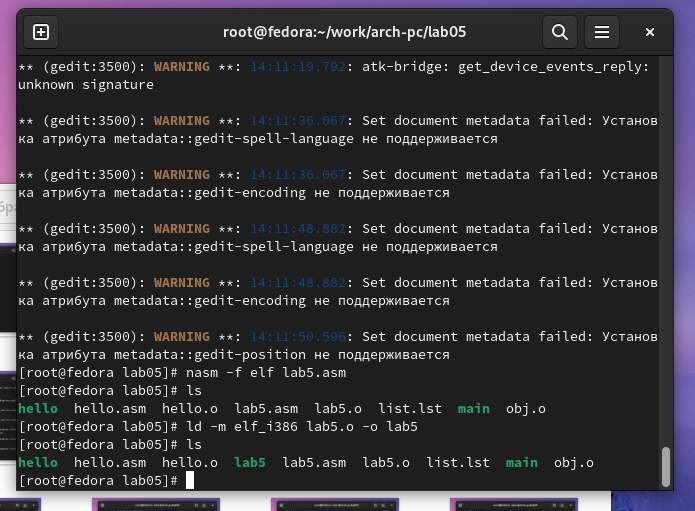


Рис. 15: Трансируем lab5.asm в объектный файл

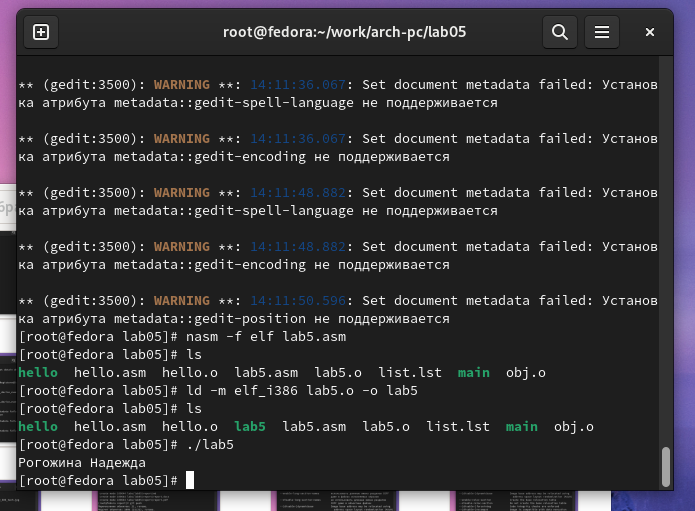


Рис. 16: Запустим исполняемый файл lab5

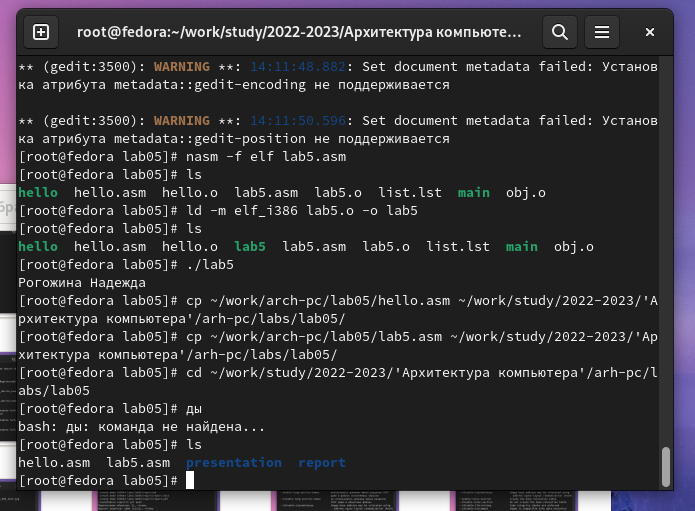


Рис. 17: Скопируем файлы hello.asm и lab5.asm в локальный репозиторий

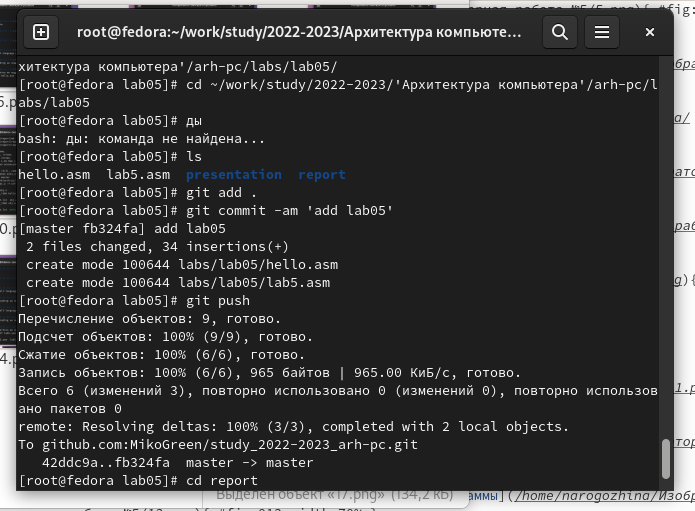


Рис. 18: Загрузим файлы на Github

# 5 Вопросы для самопроверки

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

Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуеют и как их можно использовать, так как большинство команд в программах написанных на ассемблере используют регистры в качестве операндов.

Грубо говоря, на ассемблере мы “программируем непосредственно железо”, а на языках высокого уровня - “программируем программы”. Программы, написанные на ассемблере, работают на самом низком уровне работы прикладной программы.

1. В чем состоит отличие инструкции от директивы на языке ассемблера?

Директива - это инструкция, не переводящяяся непосредственно в машинные команды, а управляющие работой транслятора.

1. Перечислите основные правила оформления программ на языке ассемблера.

Каждая программа располагается на отдельной строке. Размещение нескольких команд на одной строке недопустимо.

Синтаксис ассемблера является *чувствительным к регистру*.

Машинная команда представляет собой одну строку, имеющщую синтаксический вид:

**метка** *команда/директива* **операнд(ы)** *;комментарии*

1. Каковы этапы получения исполняемого файла?

Трансляция, компоновка.

1. Каково назначение этапа трансляции?

При наличии ошибок в тексте программы объектный файл, который создаётся на этапе трансляции, создан не будет, а после запуска транслятора появятся сообщения об ошибках или предупреждения.

1. Каково назначение этапа компоновки?

Компоновщик преобразует объектный файл в исполняемый.

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

В результате трансляции образуется объектный файл с расширением .o. Также в работе мы создавали файл листинга list.lst.

1. Каковы форматы файлов для nasm и ld?

Для nasm формат файлов - **.asm**, который после превращается в объектный файл с расширением - **.o**

Для ld формат обрабатываемого файла - **.o**, после компоновки файл преобразуется в исполняемый

# 6 Выводы

Таким образом, мы изучение языка Assembler, освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.