Отчёт по лабораторной работе № 4

дисциплина: Архитектура компьютера

Терещенкова Маргарита Владимировна

Содержание

# 1 Цель работы

Освоение 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  
6. 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 каче- стве 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.

# 4 Выполнение лабораторной работы

## 4.1 Создание программы Hello world!

С помощью утилиты cd перемещаюсь в каталог, в котором буду работать. Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch.

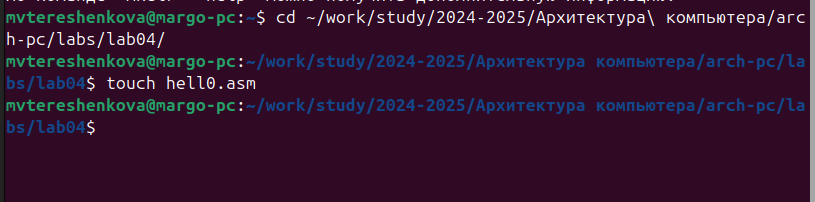


Рис. 1: *Перемещение между директориями. Создание пустого файла.*

Открываю созданный файл в текстовом редакторе mousepad.Заполняю файл, вставляя в него программу для вывода “Hello word!”

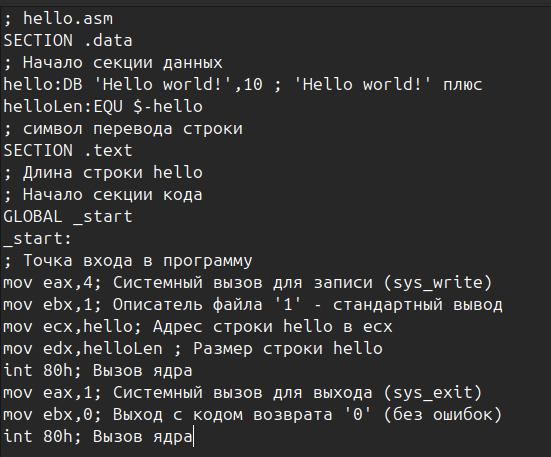


Рис. 2: *Заполнение файла*

## 4.2 Работа с транслятором NASM

Устанавливаем nasm с помощью sudo apt install nasm

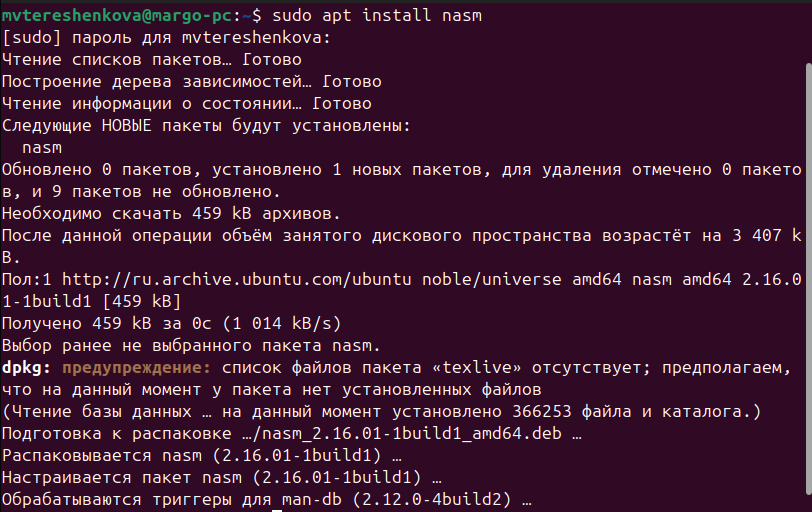


Рис. 3: *Установка nasm*

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. Далее проверяю правильность выполнения команды с помощью утилиты ls: файл “hello.o” создан.

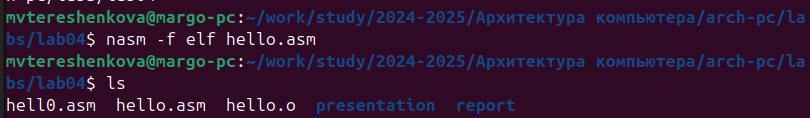


Рис. 4: *Компиляция текста программы*

## 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 . Далее проверяю с помощью утилиты ls правильность выполнения команды.

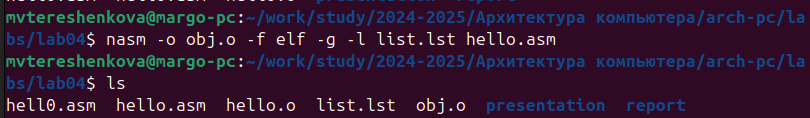


Рис. 5: *Компиляция текста программы*

## 4.4 Компоновщик LD

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

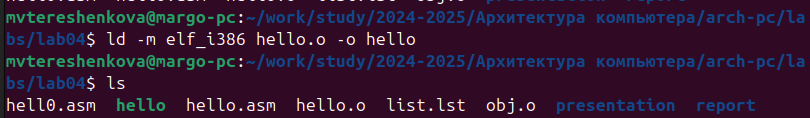


Рис. 6: *Передача объектного файла на обработку компоновщику*

Выполняю команду ld -m elf\_i386 obj.o -o main . Исполняемый файл будет иметь имя main, т.к. после ключа -о было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

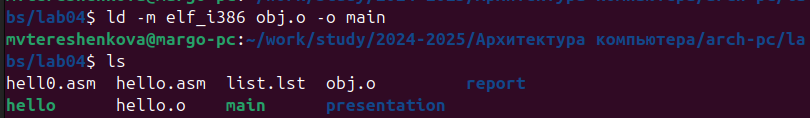


Рис. 7: *Передача объектного файла на обработку компоновщику*

## 4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello

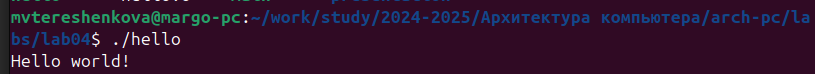


Рис. 8: *Запуск исполняемого файла*

# 5 Выполнение заданий для самостоятельной работы.

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

Рис. 9: Создание копии файла

Рис. 9: *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.

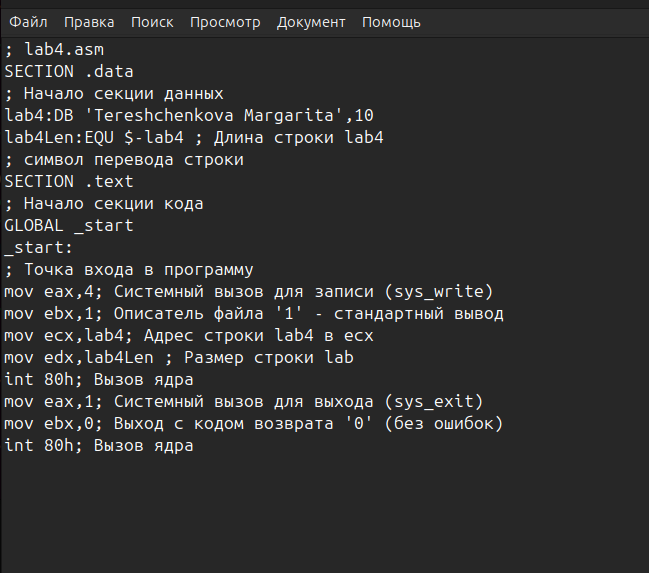


Рис. 10: *Изменение программы*

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

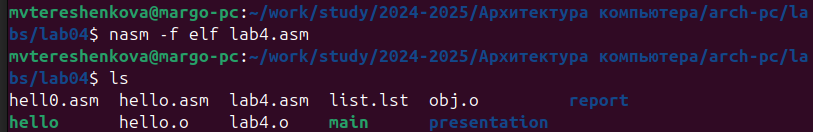


Рис. 11: *Компиляция текста программы*

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4

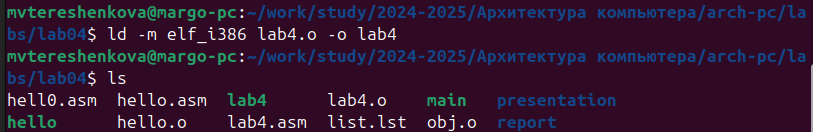


Рис. 12: *Передача объектного файла на обработку компоновщику*

Запускаю исполняемый файл lab4, на экран выводятся мои имя и фамилия

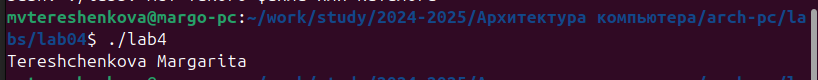


Рис. 13: *Запуск исполняемого файла*

# 6 Исправление ошибки

По привычке я начала работу не в том каталоге, поэтому создаю директорию lab04 с помощью mkdir (как указано в порядке выполнения лабораторной работы) 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 . Проверяю с помощью утилиты ls правильность выполнения команды.

Рис. 14: Создании директории

Рис. 14: *Создании директории*

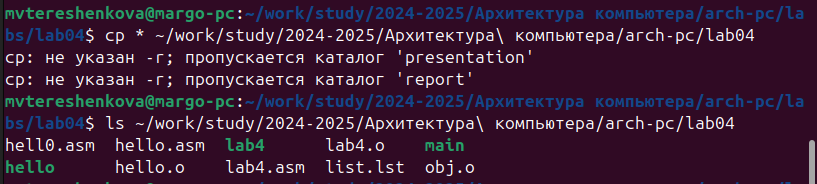


Рис. 15: *Создании копии файлов в новом каталоге*

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

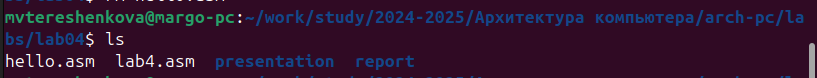


Рис. 16: *Проверка удаления*

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы.

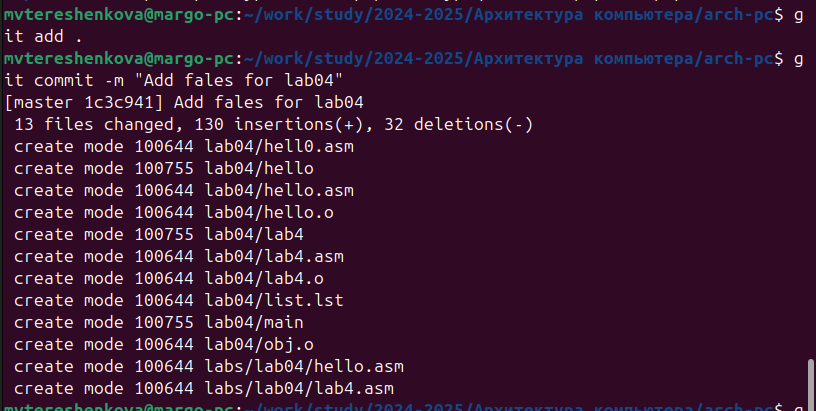


Рис. 17: *Добавление файлов на GitHub*

Отправляю файлы на сервер с помощью команды git push

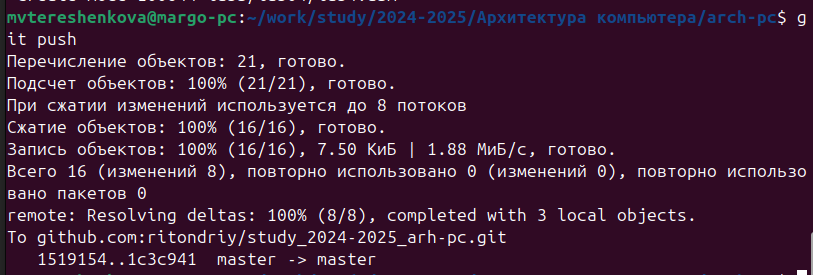


Рис. 18: *Отправка файлов*

# 7 Выводы

Благодаря данной лаборатоной работе освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# Список литературы

1. Архитектура ЭВМ