Отчёт по лабораторной работе № 5

Дисциплина: Архитектура Компьютера

Гибшер Кирилл Владимирович, НКАбд-01-22

Содержание

# 1 Цель работы

Целью работы является освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

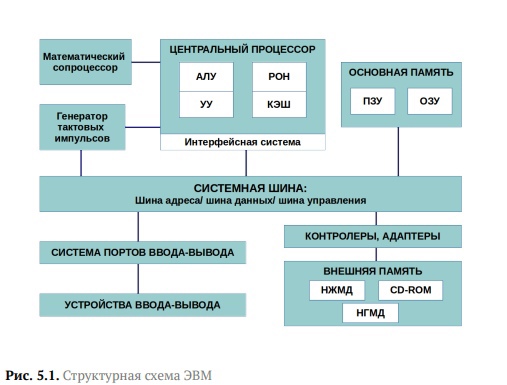
# 2 Задание

1. Рассмотреть самый простой пример программы на языке ассемблера NASM. Hello world!
2. Изучить способности транслятора NASM.
3. Скомпилировать файл формата .asm
4. Скомпоновать данный файл и запустить.
5. Провести самостоятельную работу согласно задачам из лаб.работы

# 3 Теоретическое введение

## 3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 5.1). 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. Структурная схема ЭВМ. (рис. 1)



1: Рис. 1

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

1. **арифметико-логическое устройство** — выполняет логические и арифметические действия, необходимые для обработки информации,хранящейся в памяти;
2. **устройство управления** — обеспечивает управление и контроль всех устройств компьютера;
3. **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):
4. RAX, RCX, RDX, RBX, RSI, RDI — 64-битные
5. EAX, ECX, EDX, EBX, ESI, EDI — 32-битные
6. AX, CX, DX, BX, SI, DI — 16-битные
7. AH, AL, CH, CL, DH, DL, BH, BL — 8-битные

Таким образом можно отметить, что вы можете написать в своей программе, например, такие команды (mov – команда пересылки данных на языке ассемблера): mov ax, 1 mov eax, 1

Другим важным узлом ЭВМ является **оперативное запоминающее устройство** . 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 нейданных.

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

1. *устройства внешней памяти*, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты);
2. *устройства ввода-вывода*, которые обеспечивают взаимодействие ЦП с внешней средой.

### 3.1.1 Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинноориентированный 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 ОС.

Наиболее распространёнными ассемблерами для архитектуры x86 являются:

1. для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);
2. для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

**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.

#### 3.1.1.1 Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 2)



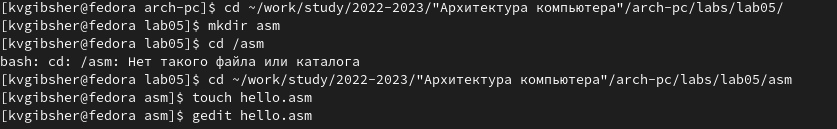
2: Рис. 2

В процессе создания ассемблерной программы можно выделить четыре шага:

1. **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.
2. **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
3. **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.
4. **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.

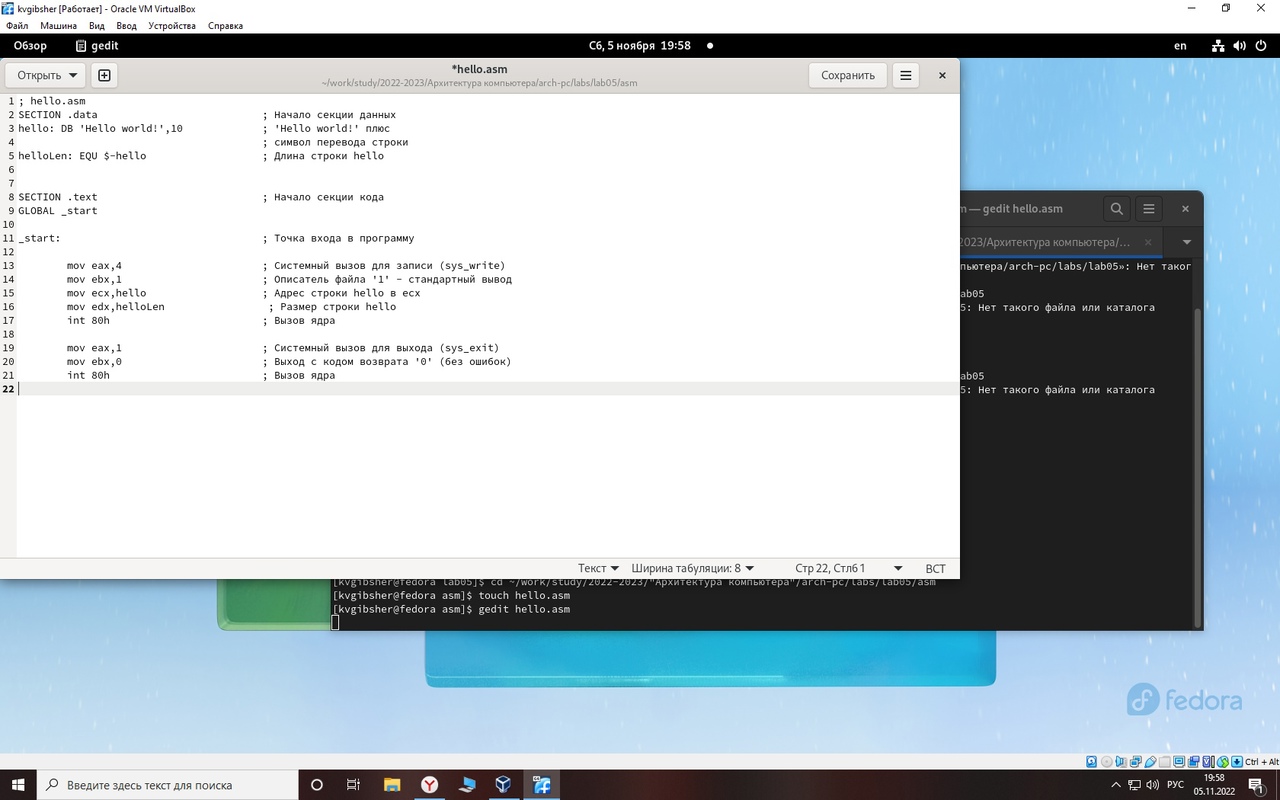
# 4 Выполнение лабораторной работы

1. Создадим каталог для работы с программами на языке ассемблера NASM и назовем его “asm”. Затем перейдя в данный каталог, создадим текстовый файл с названием hello.asm и откроем его с помощью текстового редактора gedit. (рис. 3)



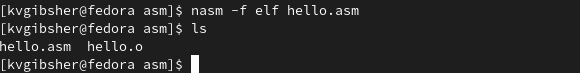
3: Рис. 3

1. Открыв hello.asm в gedit введем туда указанный в лабораторной работе текст,соблюдая синтаксис ассемблера NASM. (рис. 4)



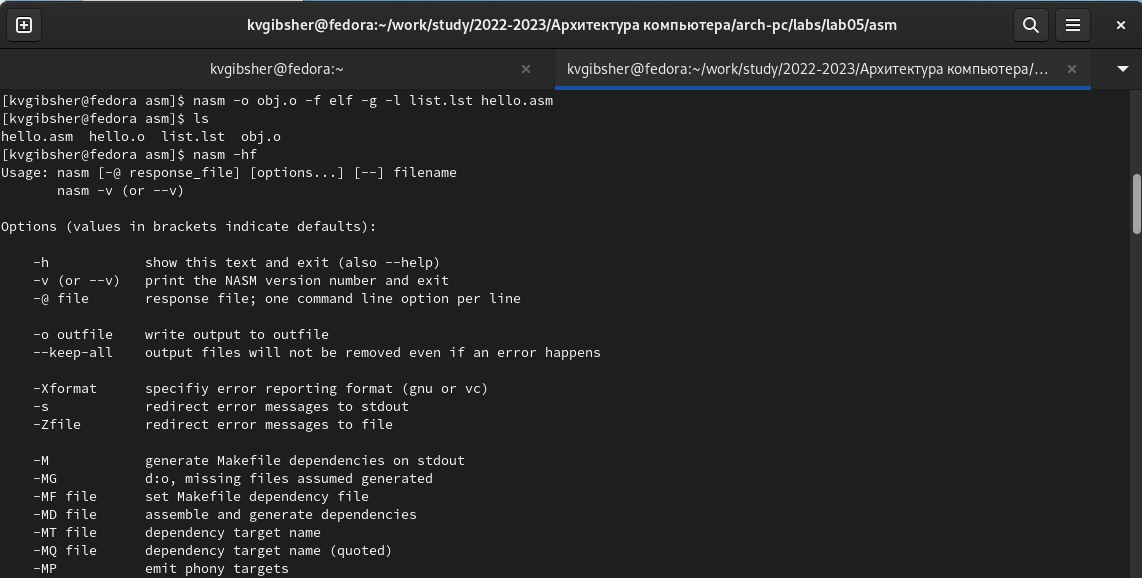
4: Рис. 4

1. Далее скомпилируем текст программы “Hello World” , написав терминальную команду указанную на скриншоте. Таким образом, текст программы преобразован в объектный код, который записался в файл hello.o . С помощью команды ls проверим , что объектный файл действительно создан. (рис. 5)



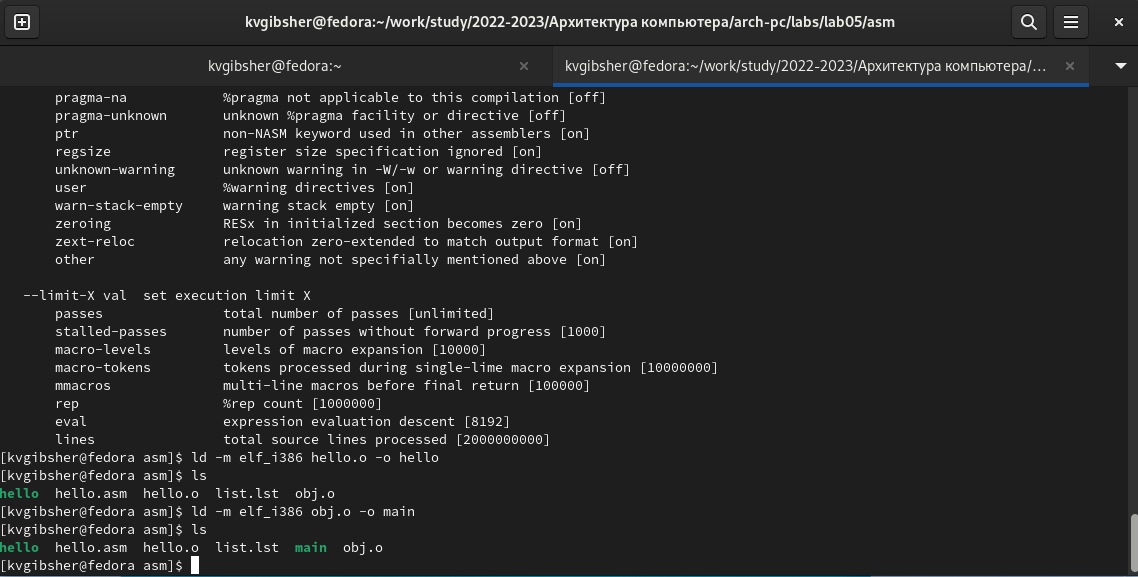
5: Рис. 5

1. Далее применим команду , которая скомпилирует исходный файл hello.asm в obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла будет elf, и в него будут включены символы для отладки (опция -g), кроме того, будет создан файл листинга list.lst (опция -l) и с помощью команды ls проверим, что все файлы успешно созданы. Для получения списка форматов объектного файла пропишем nasm -hf. (рис.6)



6: Рис. 6

1. Затем, чтобы получить исполняемую программу необходимо объектный файл передать на обработку компоновщику с помощью необходимой команды. С помощью ls проверим, что исполняемый файл **hello** был создан. Затем преобразуем объектный файл obj.o в исполняемый , изменив его имя на main. Таким образом исполняемый файл имеет имя - main , а имя объектного файла , из которого он был создан - obj.o . (рис.7)



7: Рис. 7

1. И в завершении основной части лабораторной работы запустим на выполнение созданный исполняемый файл, находящийся в текущем каталоге , набрав в командной строке : “./hello”. (рис.8)

 8: Рис. 8

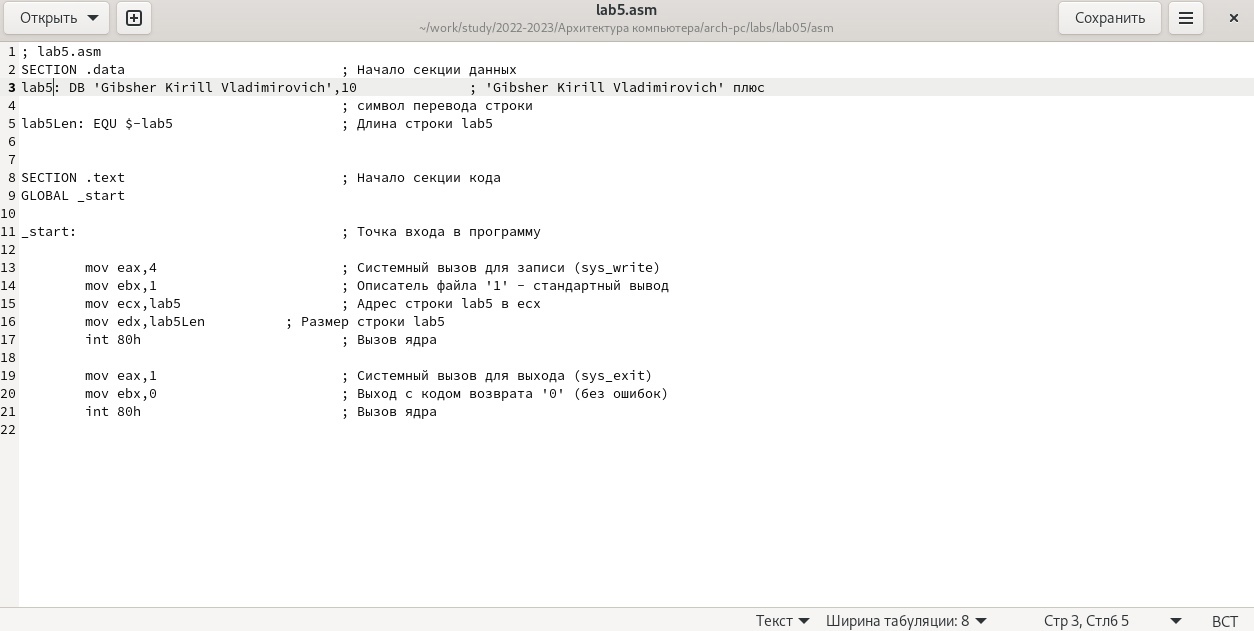
8: Рис. 8

1. Приступим к заданиям для самостоятельной работы. И начнем с того, что в каталоге ~/work/arch-pc/lab05 с помощью команды cp создадим копию файла hello.asm с именем lab5.asm. (рис.9)

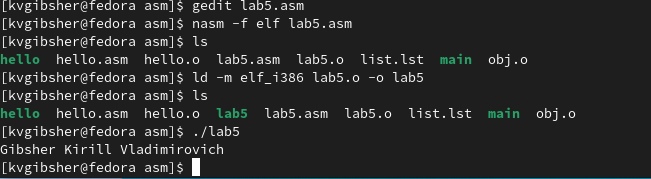
 9: Рис. 9

9: Рис. 9

1. Затем с помощью текстового редактора gedit внесем необходимые изменения в текст программы в файле lab5.asm так, чтобы вместо Hello world! на экран выводилась строка с моим ФИО. Сразу проведем необходимые преобразования объектных файлов в исполняемые, проверив все командой ls и запустим на выполнение созданный исполняемый файл. Задача успешно выполнена - выводится мое ФИО. (рис.10,11)

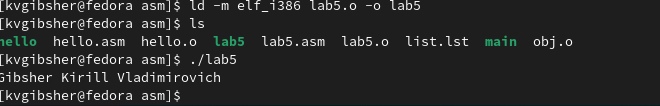


10: Рис. 10



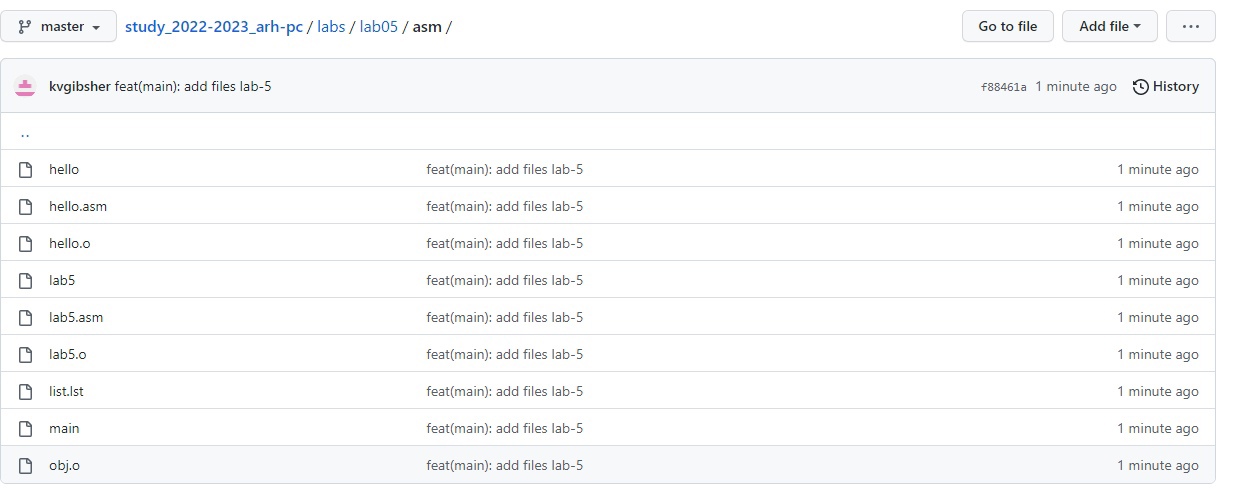
11: Рис. 11

1. Оттранслируем полученный текст программы lab5.asm в объектный файл. Выполним компоновку объектного файла и запустим получившийся исполняемый файл. (рис.12)



12: Рис. 12

1. Скопируем файлы hello.asm и lab5.asm в локальный репозиторий в каталог ~/work/study/2022-2023/“Архитектура компьютера”/archpc/labs/lab05/. Загрузим файлы на Github и создадим отчет о проделанной работе. (рис.12)



13: Рис. 13

# 5 Выводы

В ходе данной работы я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# Список литературы

[1. Текстовый файл «Лабораторная работа №5. Создание и процесс обработки программ на языке ассемблера NASM](https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/Лабораторная%20работа%20№5.pdf)