Отчёт по лабораторной работе №4

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

# 1 Цель работы

Целью данной лабораторной работы является 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 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): - 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!

Создаю каталог для работы с программами на языке ассемблера NASM (рис. ??).

Создание новой директории

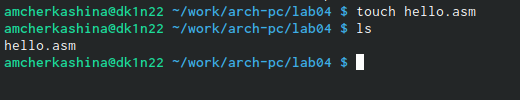
Создание новой директории

Перехожу в созданный каталог (рис. ??).

Перемещение между директориями

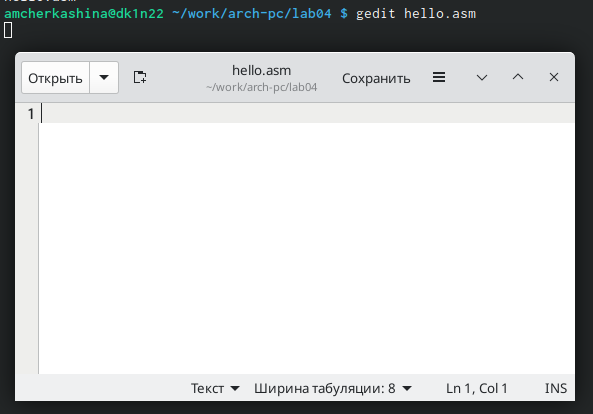
Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью команды touch (рис. ??).



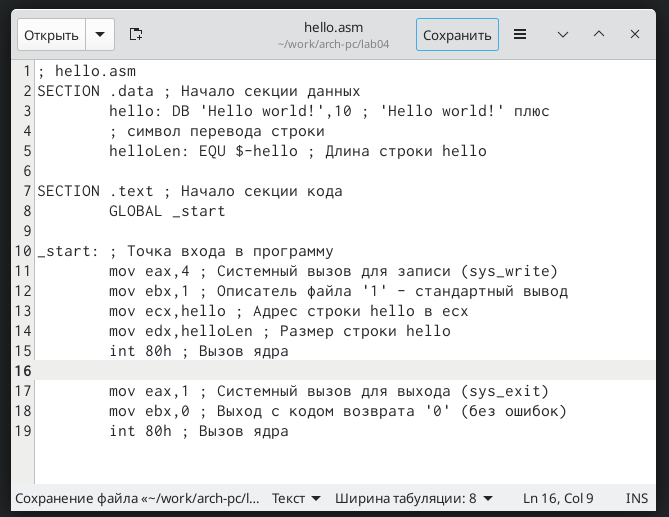
Создание пустого файла

Открываю созданный файл с помощью текстового редактора gedit (рис. ??).



Открытие файла в текстовом редакторе

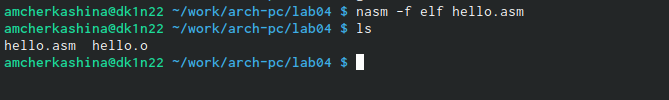
Заполняю файл, вводя в него программу для вывода “Hello world!” (рис. ??).



Заполнение файла

## 4.2 Работа с транслятором NASM

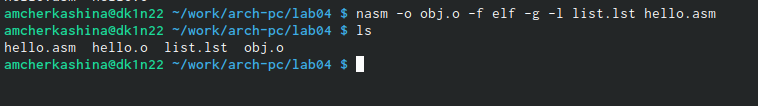
Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду nasm -f elf hello.asm (ключ -f указывает транслятору NASM, что требуется создать бинарный файл в формате ELF. Далее проверяю правильность выполнения команды с помощью ls. Действительно, создан файл “hello.o” (рис. ??).



Компиляция текста программы

## 4.3 Работа с расширенным синтаксисом командной строки NASM

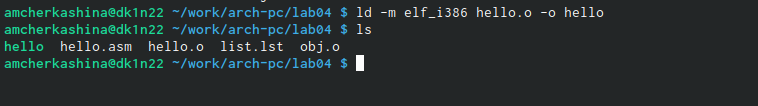
Ввожу команду nasm -o obj.o -f elf -g -l list.lst hello.asm, которая скомпилирует файл hello.asm в файл obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла будет elf, и в него будут включены символы для отладки (ключ -g). Кроме того, будет создается файл листинга list.lst (опция -l). Снова проверяю корректность выполнения команды с помощью ls (рис. ??).



Компиляция текста программы

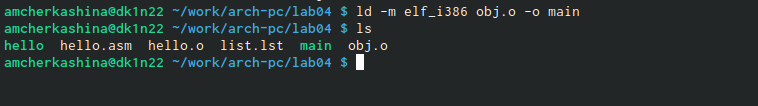
## 4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello. Использую ключ -о, чтобы задать имя создаваемого исполняемого файла. С помощью утилиты ls проверяю, что исполняемый файл hello был создан (рис. ??).



Передача объектного файла hello.o на обработку компоновщику

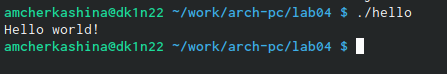
Выполняю следующую команду: ld -m elf\_i386 obj.o -o main (рис. ??). Исполняемый файл будет иметь имя main, т.к. после ключа -о было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o



Передача объектного файла obj.o на обработку компоновщику

## 4.5 Запуск исполняемого файла

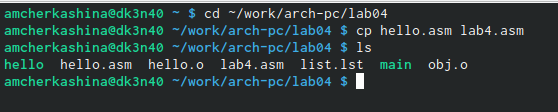
Запускаю на выполнение созданный исполняемый файл hello, находящийся в текущем каталоге (рис. ??).



Запуск исполняемого файла hello

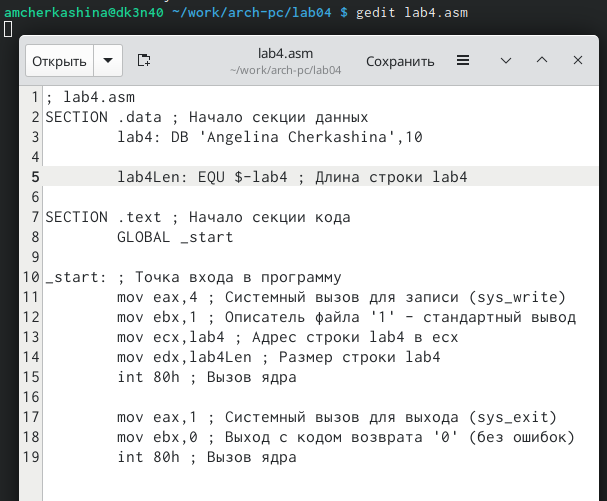
## 4.6 Выполнение заданий для самостоятельной работы

1. С помощью команды cp в каталоге ~/work/arch-pc/lab04 создаю копию файла hello.asm с именем lab4.asm (рис. ??).



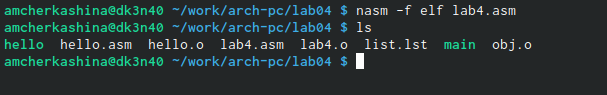
Создание копии файла

1. С помощью текстового редактора gedit открываю файл lab4.asm и вношу изменения в текст программы так, чтобы вместо “Hello world!” она выводила на экран строку с моим именем и фамилией (рис. ??).



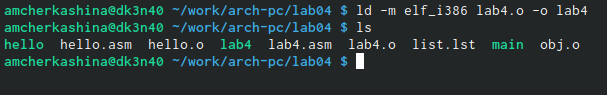
Изменение программы

1. Компилирую текст программы в объектный файл. C помощью команды ls проверяю, что файл lab4.o создан (рис. ??).



Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. ??).



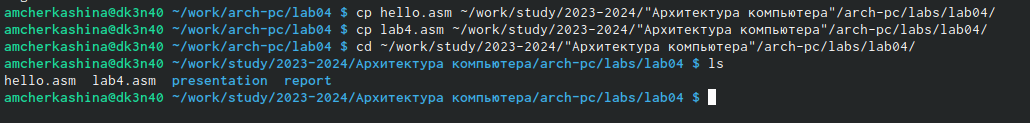
Передача объектного файла lab4.o на обработку компоновщику

Запускаю получившийся исполняемый файл lab4. На экран действительно выводятся мои имя и фамилия (рис. ??).



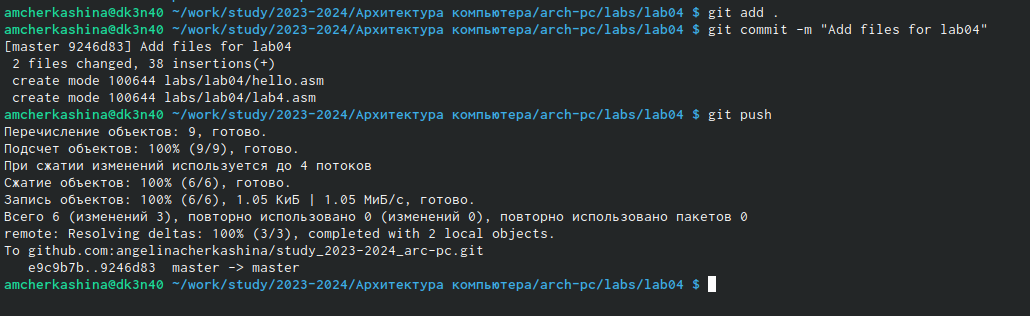
Запуск исполняемого файла lab4

1. Копируйте файлы hello.asm и lab4.asm в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ с помощью команды cp. С помощью ls убеждаюсь в правильности выполнения команды (рис. ??).



Создание копий файлов в другом каталоге

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4. Отправляю файлы на сервер с помощью команды git push (рис. ??).



Загрузка файлов на GitHub

# 5 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# 6 Список литературы

1. Архитектура ЭВМ