Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Бызова Мария Олеговна

Содержание

# 1 Цель работы

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

# 2 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.

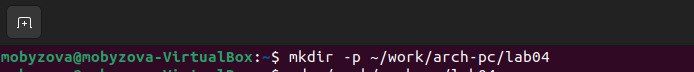
Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении 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.

# 3 Выполнение лабораторной работы

## 3.1 Создание программы Hello world!

При помощи ранее изученных команд создаем каталог для работы с программами на языке ассемблера NASM для выполнения последующих заданий данной лабораторной работы (рис. [??]).



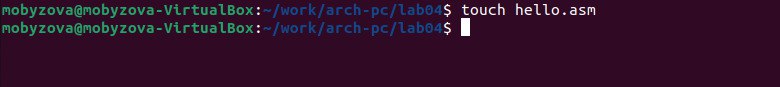
Создание каталога

С помощью утилиты cd перемещаемся в каталог, в котором далее будем работать (рис. [??]).

Перемещение между директориями

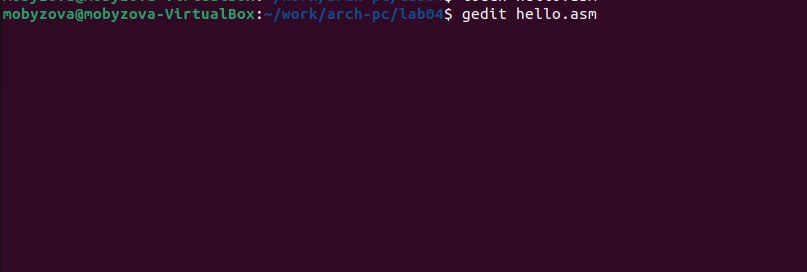
Перемещение между директориями

Создаем в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch (рис. [??]).

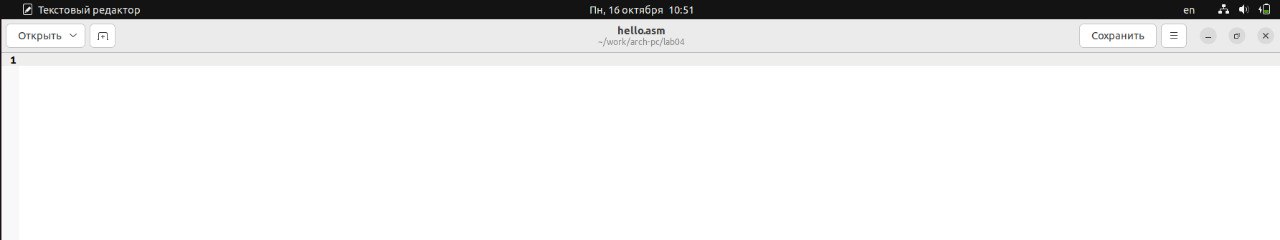


Создание пустого файла

Открываем созданный файл в текстовом редакторе при помощи команды gedit (рис. [??] и [??]) .

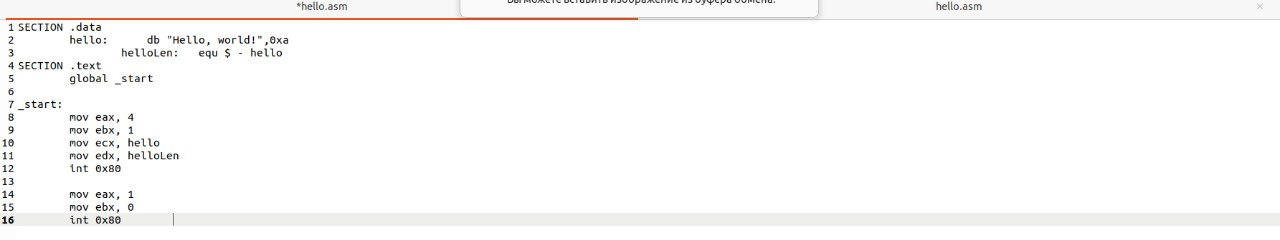


Открытие файла в текстовом редакторе



Вид открытого файла в текстовом редакторе

Заполняем файл, вставляя в него программу для вывода “Hello word!” (рис. [??]).



Заполнение файла

## 3.2 Работа с транслятором NASM

Превращаем 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 проверяем правильность выполнения команды с помощью утилиты ls: действительно, создан файл “hello.o” - именно такое имя будет иметь созданный объектный файл (рис. [??]).

Компиляция текста программы

Компиляция текста программы

## 3.3 Работа с расширенным синтаксисом командной строки NASM

Вводим 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 проверяем с помощью утилиты ls правильность выполнения команды (рис. [??]).

Компиляция текста программы

Компиляция текста программы

## 3.4 Работа с компоновщиком LD

Передаем объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. [??]). Ключ -о задает имя создаваемого исполняемого файла. Далее проверяем с помощью утилиты ls правильность выполнения команды.

Передача объектного файла на обработку компоновщику

Передача объектного файла на обработку компоновщику

Выполняем следующую команду (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 на обработку компоновщику

Передача объектного файла на обработку компоновщику

## 3.5 Запуск исполняемого файла

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

Запуск исполняемого файла

Запуск исполняемого файла

## 3.6 Выполнение заданий для самостоятельной работы.

1. С помощью утилиты cp создаем в текущем каталоге копию файла hello.asm с именем lab4.asm (рис. [??]).

Создание копии файла

Создание копии файла

1. С помощью текстового редактора открываем файл lab4.asm и вносим изменения в программу так, чтобы она выводила наши имя и фамилию. (рис. [??] и [??]).

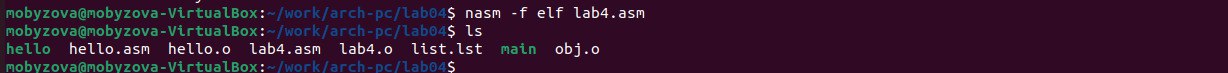
Открытие текстового файла

Открытие текстового файла



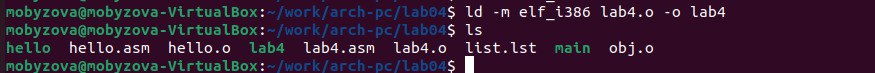
Изменение программы

1. Компилируем текст программы в объектный файл (рис. [??]). Проверяем с помощью утилиты ls, что файл lab4.o создан.



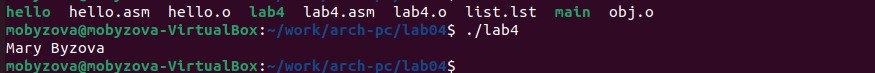
Компиляция текста программы

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



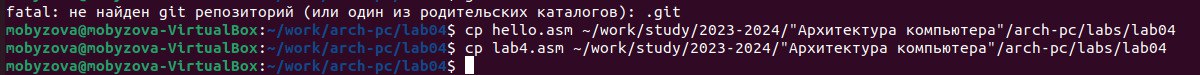
Передача объектного файла на обработку компоновщику

Запускаем исполняемый файл lab4. На экран действительно выводятся наши имя и фамилия (рис. [??]).



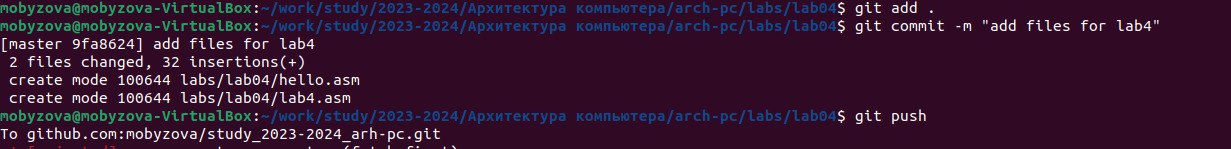
Запуск исполняемого файла

1. Скопируем файлы hello.asm и lab4.asm в наш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ (рис. [??]).

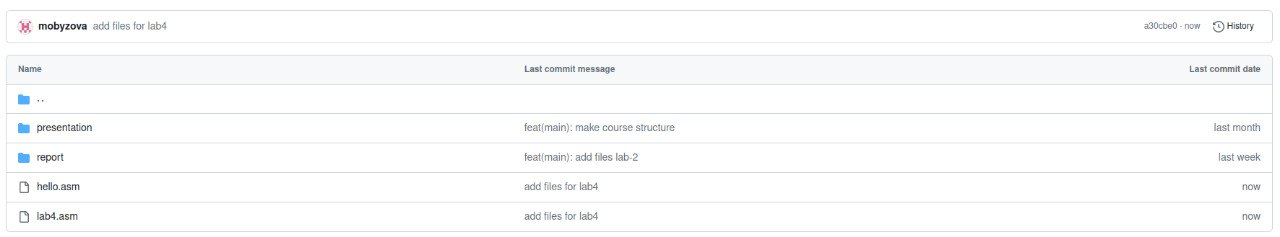


Копирование файлов

Далее с помощью ранее изученных команд загружаем файлы на GitHub: добавляем с помощью git add нужный файл, сохраняем изменения с помощью git commit, отправляем в центральный репозиторий сохраненные изменения командой git push (рис. [??]). Проверяем правильность работы (рис. [??]).



Загрузка файлов на GitHub



Проверка правильности работы команд.

# 4 Выводы

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