Отчёт по лабораторной работе №4

Дисциплина: Архитектура Компьютера

София Андреевна Кудякова

Содержание

# 1 Цель работы

Цель данной работы - научиться работать с программами, написанными на ассемблере NASM, а именно - освоить процедуры компиляций и сборки программ.

# 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

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. Но 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 Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, ибо транслятор просто переводит мнемоническиеиобозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC,M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. 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 отбольшинства других популярных ассемблеров, которые используют Intel-синтаксис. Для записи команд в NASM используются: 1) Мнемокод— непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. 2) Операнды - числа, данные, адреса регистров или адреса оперативной памяти. 3) Метка — идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. (Метка перед командой связана с адресом данной команды). Допустимыми символами в метках являются буквы, цифры, а также следующие символы: *, $, #, @,~,. и ?. Начинаться метка или идентификатор могут с буквы, .,*  и ?. Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать $, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора составляет 4095 символов. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

# 4 Выполнение лабораторной работы

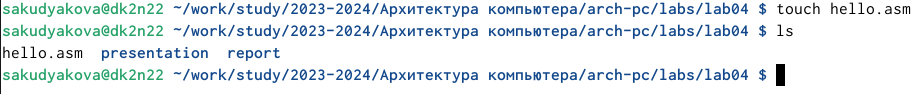
1. Программа Hello World!

Используя комаанду cd перемещаюсь в lab05. (рис. ??).

Перемещение в нужную директорию

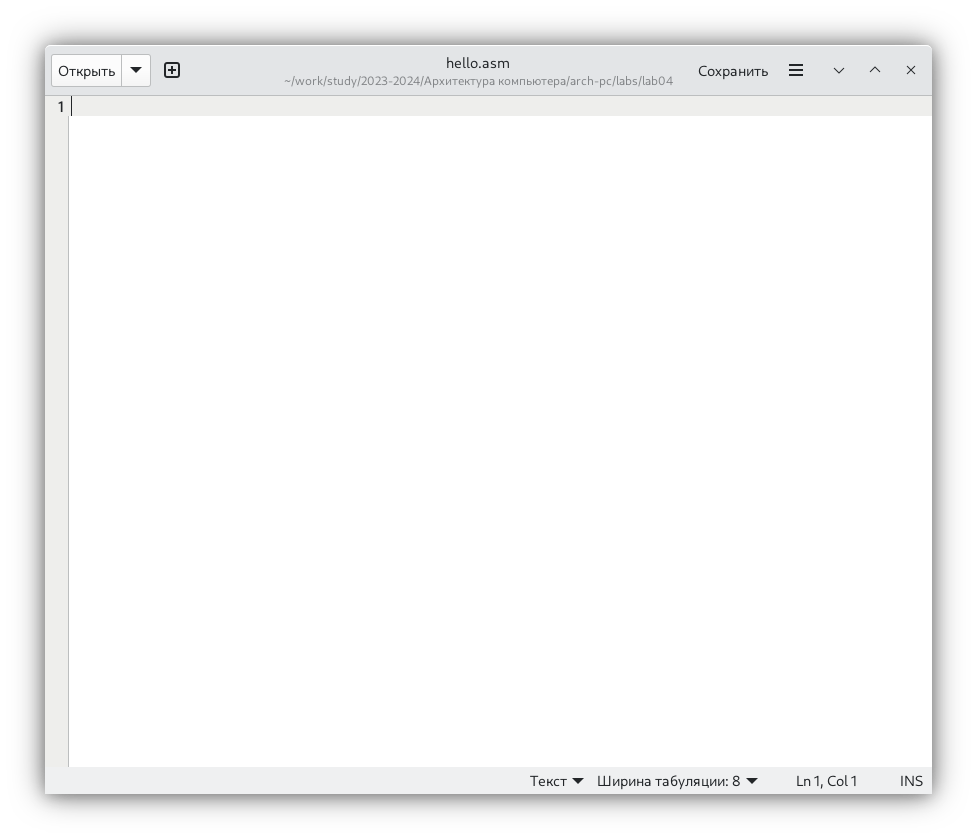
Перемещение в нужную директорию

Используя утилиту touch, создаю пустой текстовый файл hello.asm и проверяю корректность выполненных действий с помощью ls. (рис. ??).



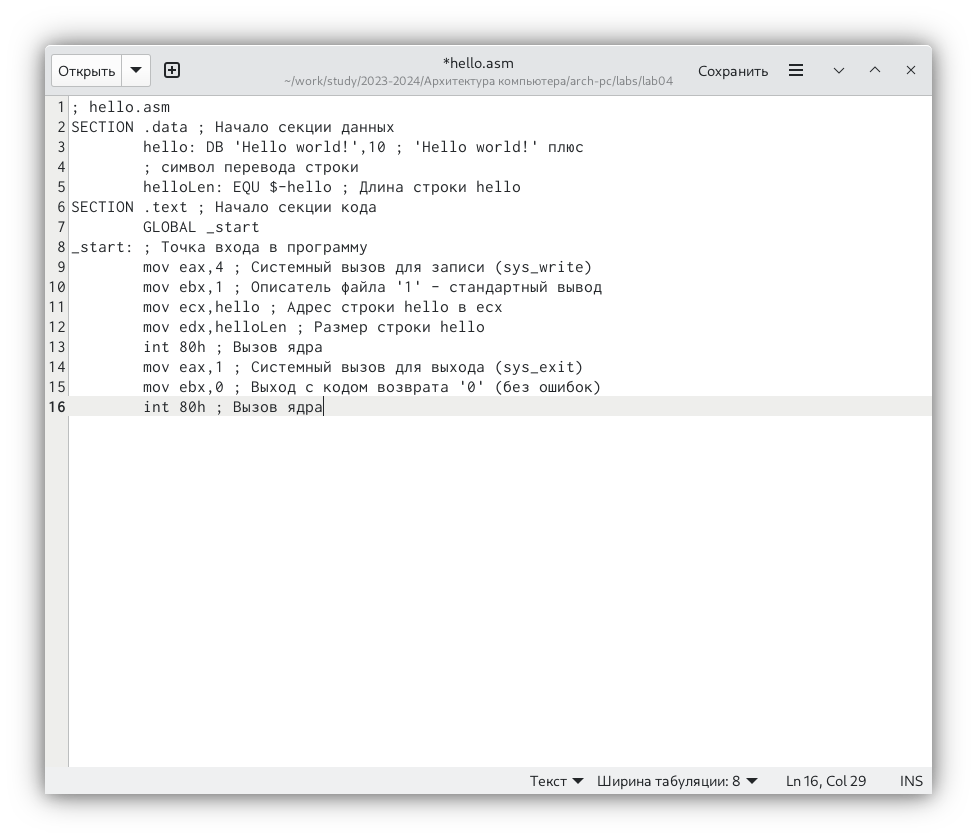
Создание файла и проверка

Далее созданный файл открываю в текстовом редакторе gedit. (рис. ??).



Открытие файла

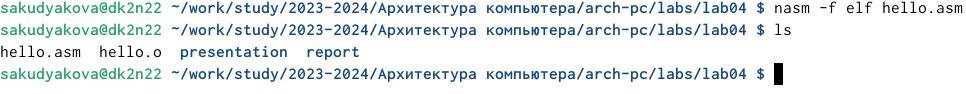
Заполняю файл текстом, нужным для вывода “Hello World!”. (рис. ??).



Ввод текста

1. Работа с транслятором NASM

NASM превращает текст программы в объектный код. Для приведённого выше текста программы “Hello World” выполнил компиляцию, используя команду nasm -f elf hello.asm ,ключ -f указывает транслятору nasm, что необходимо создать бинарный файл в формате ELF. Затем проверяю корректность выполненных действий с помощью ls. (рис. ??).

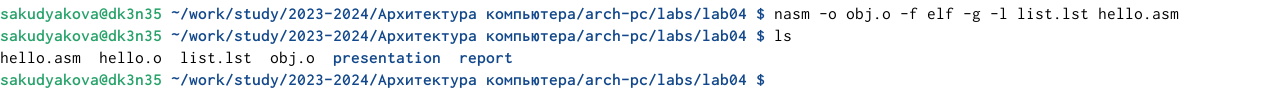


Компиляция текста программы

После проверки убеждаюсь, что был создан файл “hello.o”

1. Работа с расширенным синтаксисом командной строки NASM

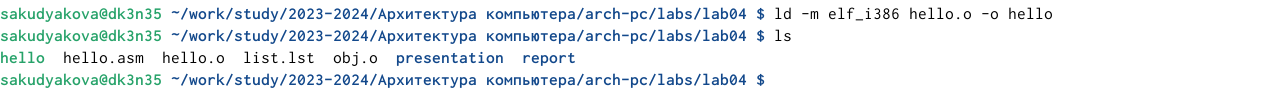
Ввожу команду, которая скомпилирует файл hello.asm в файл obj.o, опция -o позволяет задать имя объектного файла, в данном случае obj.o, при этом формат выходного файла будет elf и в файл будут включены символы для отладки (ключ -g), также с помощью ключа -l будет создан файл листинга list.lst. Проверяю правильность выполнения команды с помощью команды ls. (рис. ??).



Компиляция текста программы

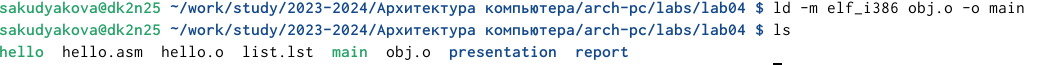
1. Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello. Ключ -о задает имя создаваемого исполняемого файла. С помощью команды ls убеждаюсь, что исполняемый файл hello был создан.(рис. ??).



Передача объектного файла на обработку компановщику

Выполняю следующуюю команду: d -m elf\_i386 obj.o -o main. Так как я использовала ключ -о, которое задало значение main, исполняемый файл будет иметь имя main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o (рис. ??)



Передача объектного файла на обработку компановщику

1. Запуск исполняемого файла

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

Запуск исполняемого файла

Запуск исполняемого файла

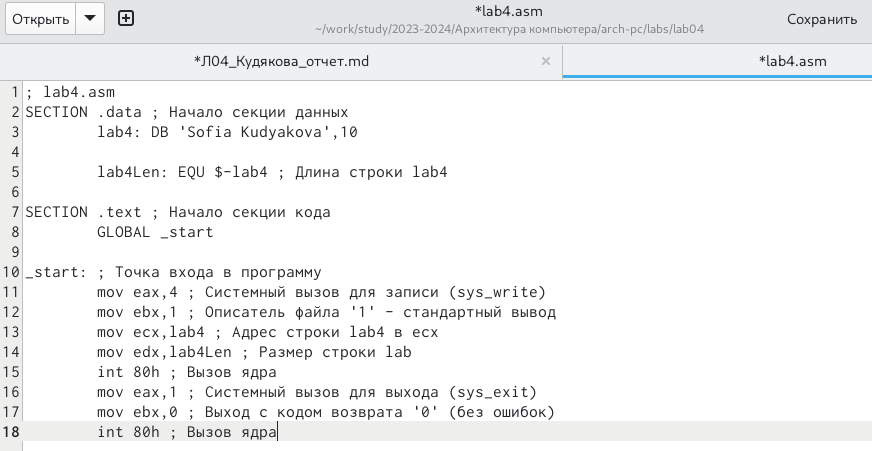
1. Выполнение заданий для самостоятельной работы

С помощью команды cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm.(рис. ??).

Создание копии файла

Создание копии файла

С помощью текстового редактора gedit открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мое имя и фамилию. (рис. ??)



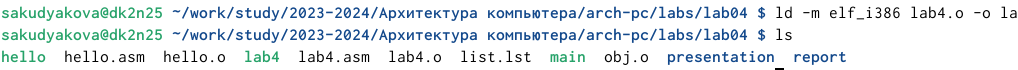
Изменение текста

Компилирую текст программы в объектный файл. (рис. ??)

Компиляция текста программы

Компиляция текста программы

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



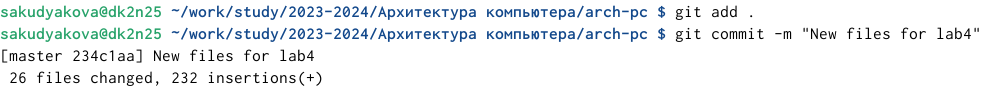
Передача файла на обработку

Запускаю исполняемый файл lab4.asm. Программа сработала корректно, на экран действительно выводятся мои имя и фамилия.(рис. ??)

Запуск файла

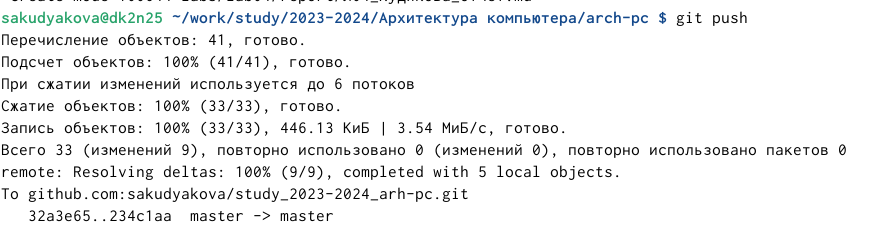
Запуск файла

Добавляю файлы в GitHub, используя команду git add . и git commit, комментируя свое действие как New files for lab 4. (рис. ??)



Добавление файлов

Отправляю файлы на сервер с помощью команды git push.(рис. ??)



Отправка файлов

# 5 Выводы

В ходе данной лабораторной работы я научилась работать с программами, написанными на ассемблере NASM, а именно - смогла освоить процедуры компиляций и сборки программ.

# Список литературы

[Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089084/mod_resource/content/0/Лабораторная%20работа%20№4.%20Создание%20и%20процесс%20обработки%20программ%20на%20языке%20ассемблера%20NASM.pdf)