Лабораторная работа №5

Создание и процесс обработки программ на языке ассемблера NASM

Медникова Екатерина Михайловна

Содержание

# 1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# 2 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. Заметим, что 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 — **Ассемблер**.

Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения 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:

• для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);

• 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.

Типичный формат записи команд NASM имеет вид:

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

Здесь **мнемокод** — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. **Операндами** могут быть числа, данные, адреса регистров или адреса оперативной памяти. **Метка** — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. 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 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.

# 3 Выполнение лабораторной работы

1. Создала каталог для работы с программами на языке ассемблера NASM и перешла в него.

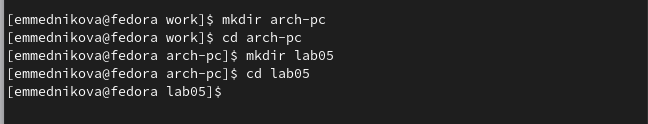


Figure 1: Создание каталога и переход в него

1. Создала текстовый файл с именем hello.asm.

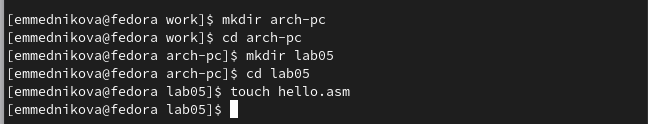


Figure 2: Создание файла

1. Открыла файл с помощью текстового редактора gedit и ввела в него текст.

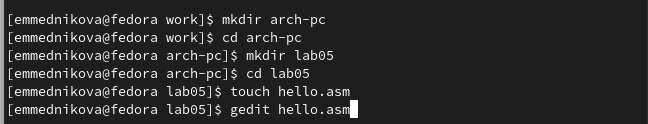


Figure 3: Открытие файла

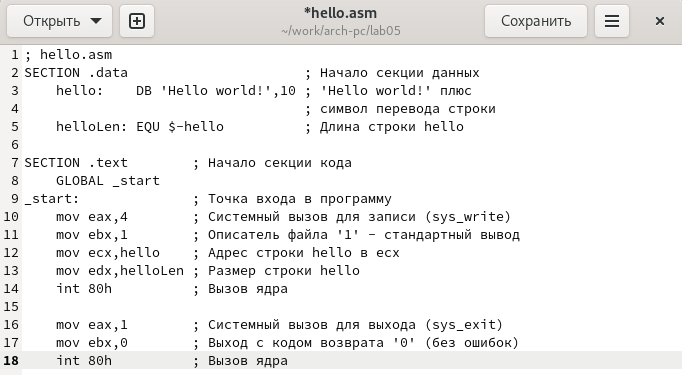


Figure 4: Вставка текста

1. Провела компиляцию текста программы с помощью команды nasm -f elf hello.asm.

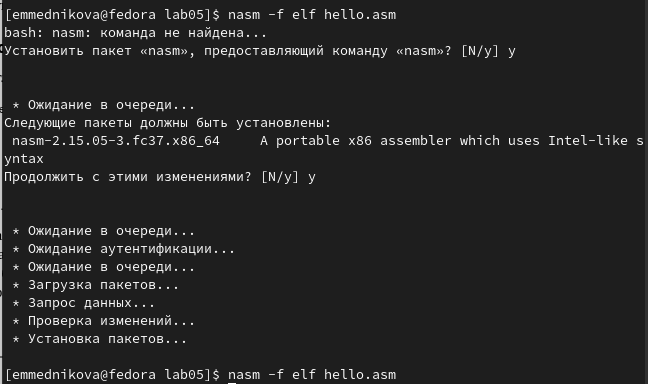


Figure 5: Компиляция текста

1. Скомпилировала файл с помощью команды nasm -o obj.o -f elf -g -l list.lst hello.asm. С помощью команды ls проверила, что файлы были созданы.

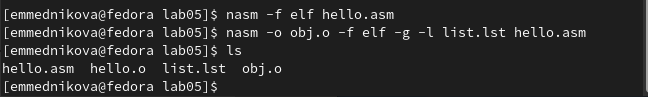


Figure 6: Компиляция и проверка

1. Передала файл на обработку компоновщику с помощью команды ld -m elf\_i386 hello.o -o hello. Далее использовала команду ld -m elf\_i386 obj.o -o main. Запустила на выполнение исполняемый файл с помощью команды ./hello.

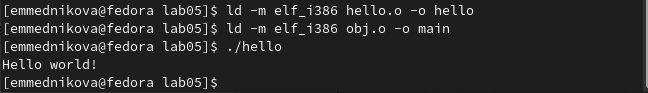


Figure 7: Результат выполненных команд

# 4 Выполнение самостоятельной работы

1. В каталоге ~/work/arch-pc/lab05 с помощью команды cp создала копию файла hello.asm с именем lab5.asm.

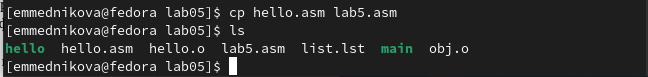


Figure 8: Создание копии файла

1. С помощью редактора gedit внесла изменения в текст программы в файле lab5.asm так, чтобы вместо Hello world! на экран выводилась строка с моей фамилией и моим именем.

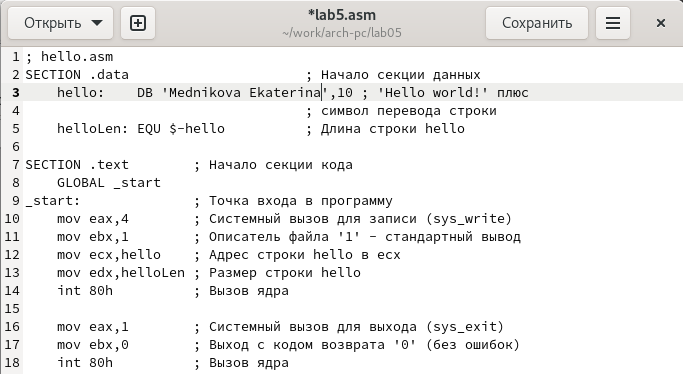


Figure 9: Внесение изменений в текст

1. Оттранслировала полученный текст программы lab5.asm в объектный файл. Выполнила компоновку объектного файла и запустила получившийся исполняемый файл.

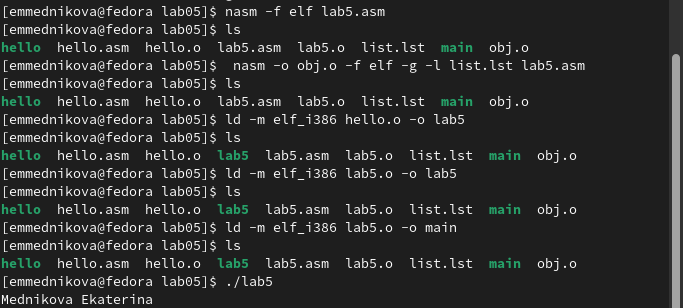


Figure 10: Компановка и запуск

1. Скопировала файлы hello.asm и lab5.asm в свой локальный репозиторий в каталог ~/work/study/2022-2023/“Архитектура компьютера”/arch-pc/labs/lab05/. Загрузила файлы на Github.

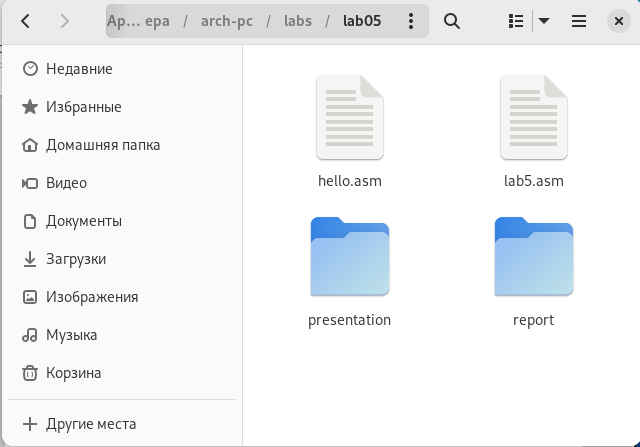


Figure 11: Копирование файлов в локальный репозиторий

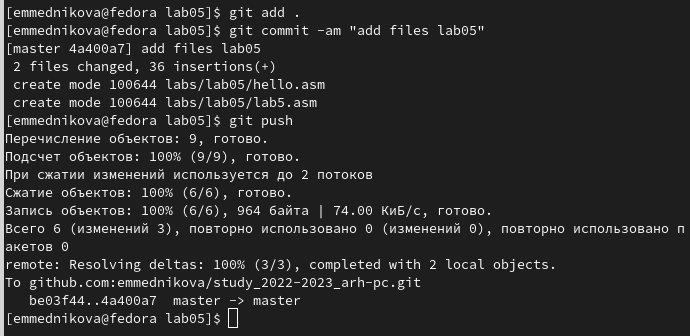


Figure 12: Загрузка файлов на Github

# 5 Выводы

Освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.