Лабалаторная работа №4

Создание и процесс обработки программ на языке ассемблера NASM

Замула Егор Сергеевич

Содержание

Список таблиц

Список иллюстраций

Цель работы

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

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 Демидова А. В. 33 Архитектура ЭВМ языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве и скорости програм- мам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обыч-

но одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контрол- леров 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. Более подробно о языке ассемблера см., например, в [10]. В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler) [7; 12; 14]. 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 тракто- вал 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.

Выполнение лабораторной работы

1. Создал каталог для работы с программами на языке ассемблера NASM:

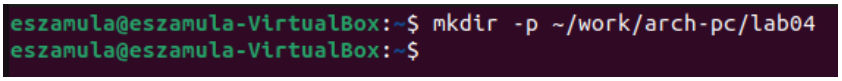

Рис. 1: Создание каталога

2. Перешли в созданный каталог

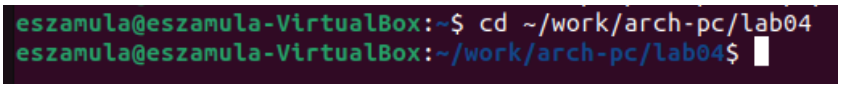

Рис. 2: Перешёл в созданный каталог

3. Создал текстовый файл с именем hello.asm

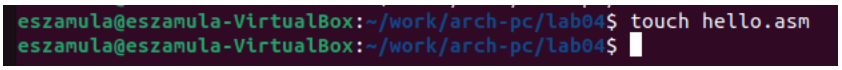

Рис. 3: Создание текстового файла

4. Открыл этот файл с помощью любого текстового редактора, например, gedit

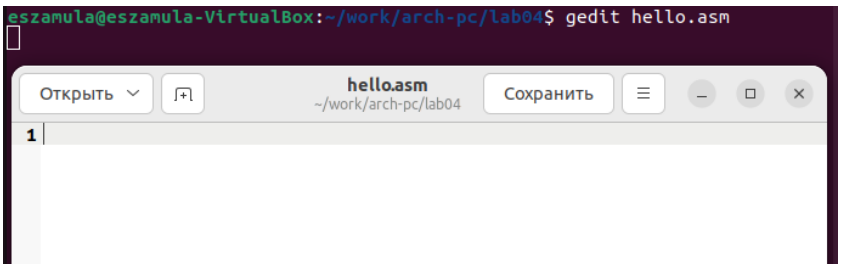

Рис. 4: Открытй файл

5. И ввел в него следующий текст:

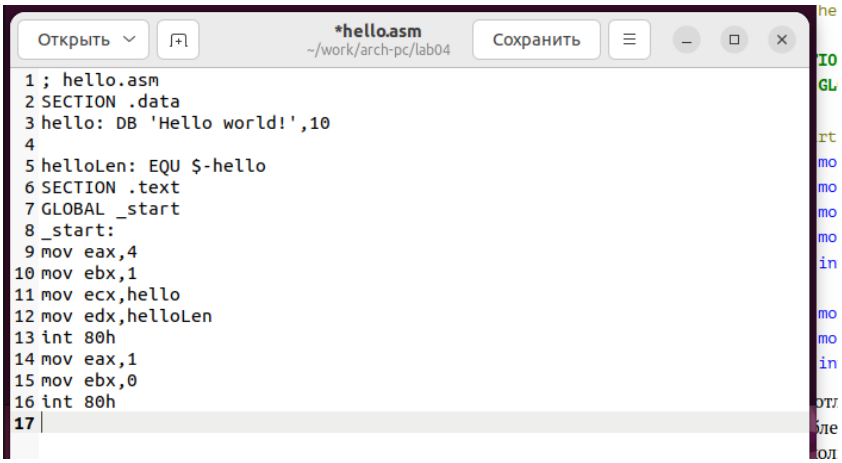

Рис. 5: Ввожу текст

6. NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать:

Рис. 6: Компиляция текста

7. Проверяю наличие нового файла

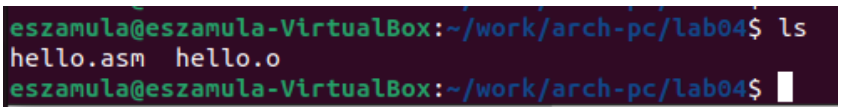

Рис. 7: Проверка файла

8. Выполняю следующую команду для компиляции файл hello.asm в obj.o и проверяю его наличие в папки

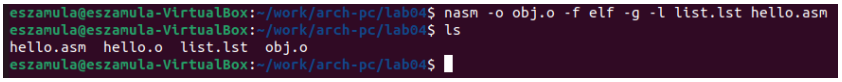

Рис. 8: Компиляция в obj.o

9. Передаю объектный файл на обработку компановщика и проверяю его в папке

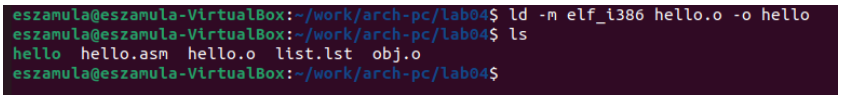

Рис. 9: Обработка компановщика

10. Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла

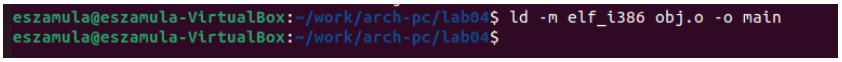

Рис. 10: Задаю имя файла

11. Запустил на выполнение созданный исполняемый файл, находящийся в текущем каталоге

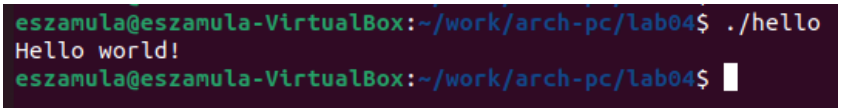

Рис. 11: Запуск файла

12. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создал копию файла hello.asm с именем lab4.asm

Рис. 12: Создание копии

13. С помощью любого текстового редактора внес изменения в текст программы вфайле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моей фамилией и именем.

Рис. 13: Редактирование файла

14. Оттранслировал полученный текст программы lab4.asm в объектный файл. Выполнил компоновку объектного файла и запустил получившийся исполняемый файл

Рис. 14: Выполнение компановки и запуск файла

15. Скопировал файлы hello.asm и lab4.asm в локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ с помощью утилиты cp и проверил наличие файлов с помощью утилиты ls

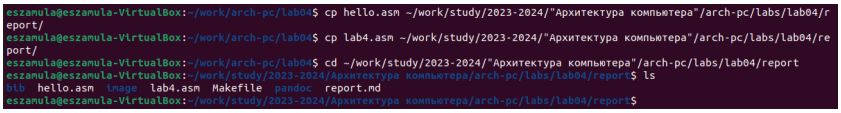

Рис. 15: Копирование файлов в локальный репозиторий

16. Загружаю файлы на Github.

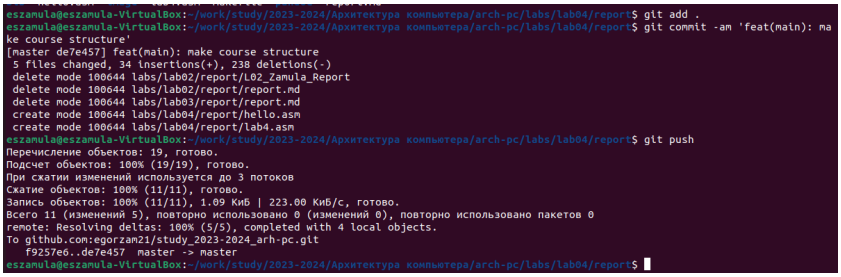

Рис. 16: Загрузка файлов на Гитхаб

17. Проверяю наличие файлов в репозитории

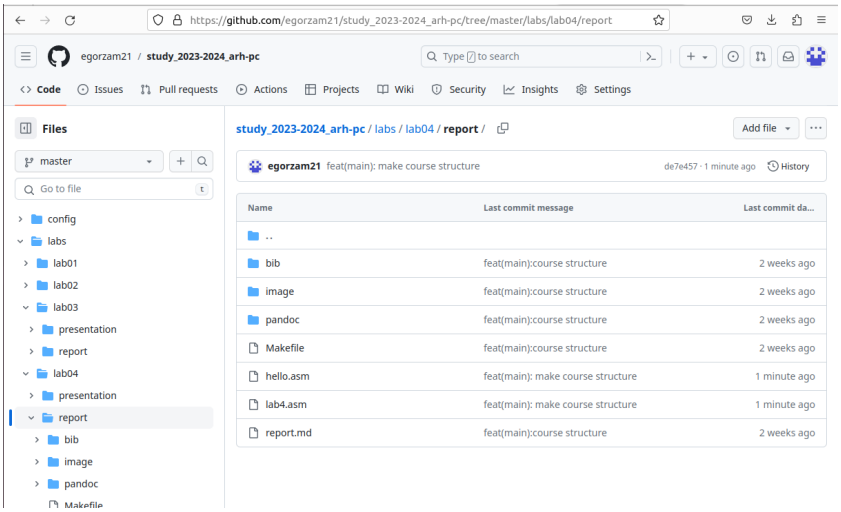

Рис. 17: Проверка на гитхаб

Выводы

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

Список литературы