РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4.

дисциплина: *Архитектура компьютеров*

Студент: Подхалюзина Виолетта Михайловна

Группа: НКАбд-04-24

МОСКВА

2024 г.

Содержание

1 Цель работы 3
2 Введение 3
3 Выполнение лабораторной работы 6
3.1 Создание команды «Hello world» 6
3.2 Транслятор NASM 6
3.3 Расширенный синтаксис командной строки NASM 6
3.3 Компоновщик LD 7
3.3 Запуск исполняемого файла 7
4 Задание для самостоятельной работы 7
5 Контрольные вопросы для самопроверки 8
6 Вывод 10
7 Список литературы 11

1 Цель работы

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

2 Введение

Основными функциональными элементами любой электронно-вычислительной машины (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 (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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.

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

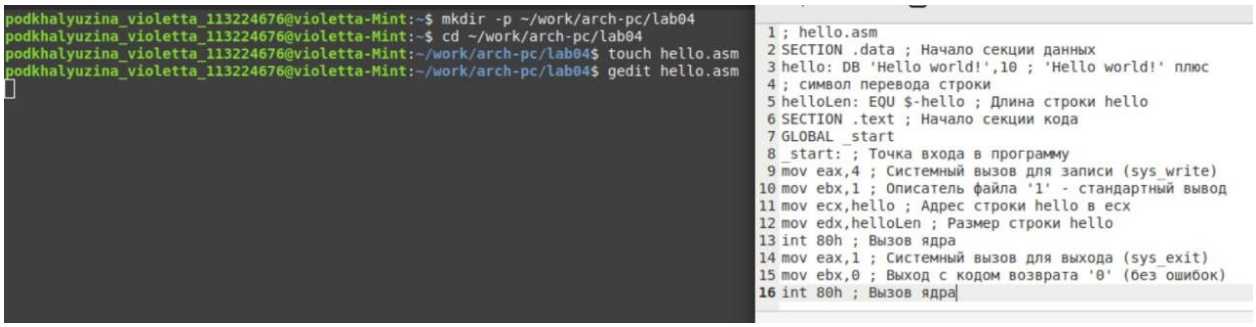
известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC,M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: • для DOS/Windows: Borland Turbo Assembler (TASM), 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 — расширенный ассемблер, предназначенный для написания программ для процессоров серии Intel x86, способный работать на разных платформах.

3 Выполнение лабораторной работы

3.1 Создание команды «Hello world»

Сначала я создала каталог для работы с программами на языке ассемблера NASM и перешла в созданный каталог. После я создала текстовый файл с именем hello.asm, открыла его и ввела в него текст, где каждая команда располагается на отдельной строке.



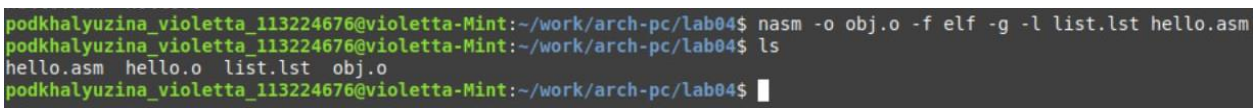
3.2 Транслятор NASM

NASM превращает текст программы в объектный код, поэтому для компиляции приведённого выше текста программы «Hello World» я написала следующее:

Текст программы набран без ошибок, поэтому транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o.

3.3 Расширенный синтаксис командной строки NASM

Ввела команду, которая скомпилирует исходный файл hello.asm в 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 (опция -g), кроме того, будет создан файл листинга list.lst (опция -l). Потом, с помощью команды ls, проверила, что файлы были созданы.



3.3 Компоновщик LD

После я передала объектный файл на обработку компоновщику и проверила, что исполняемый файл был создан. Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o.

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

3.3 Запуск исполняемого файла

Далее я запустила на выполнение созданный исполняемый файл, находящийся в текущем каталоге.

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ./hello
Hello world!
```

4 Задание для самостоятельной работы

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

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ gedit lab4.asm
```

```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Подхалюзина Виолетта',10 ; 'Hello world!' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8 _start: ; Точка входа в программу
9 mov eax,4 ; Системный вызов для записи (sys_write)
10 mov ebx,1 ; Описатель файла '1' - стандартный вывод
11 mov ecx,hello ; Адрес строки hello в ecx
12 mov edx,helloLen ; Размер строки hello
13 int 80h ; Вызов ядра
14 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра
```

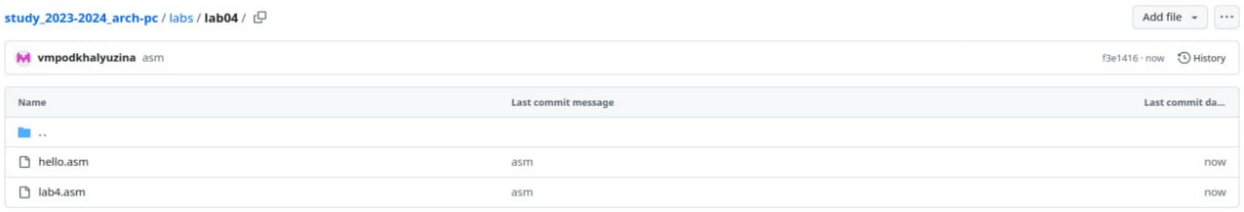
После я внесла изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моими фамилией и именем.

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ./lab4
Подхалюзина Виолетта
```

Далее я оттранслировала полученный текст программы lab4.asm в объектный файл, выполнила компоновку объектного файла и запустила получившийся исполняемый файл.

```
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ mkdir -p ~/work/study/2023-2024/'Архитектура компьютера'/study_2023-2024_arch-pc/labs/lab04
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/'Архитектура компьютера'/study_2023-2024_arch-pc/labs/lab04/hello.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2023-2024/'Архитектура компьютера'/study_2023-2024_arch-pc/labs/lab04/lab4.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ ls ~/work/study/2023-2024/'Архитектура компьютера'/study_2023-2024_arch-pc/labs/lab04
hello.asm  lab4.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/arch-pc/lab04$ cd ~/work/study/2023-2024/'Архитектура компьютера'/study_2023-2024_arch-pc
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arch-pc$ git add .
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arch-pc$ git commit -am 'asm'
[master f3e1416] asm
 2 files changed, 32 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arch-pc$ git push
Перечисление объектов: 8, готово.
Подсчет объектов: 100% (8/8), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 915 байтов | 915.00 КиБ/с, готово.
Всего 6 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:vmpodkhalyuzina/study_2023-2024_arch-pc.git
   ebad5d3..f3e1416  master -> master
podkhalyuzina_violetta_113224676@violetta-Mint:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arch-pc$
```

Скопировала файлы hello.asm и lab4.asm в мой локальный репозиторий в каталог ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/. В самом конце я загрузила файлы на Github.



5 Контрольные вопросы для самопроверки

1. Основные отличия ассемблерных программ от программ на языках высокого уровня:

- Ассемблер работает на уровне машинных команд и непосредственно управляет оборудованием, тогда как языки высокого уровня (C, Python) абстрагированы.
- Программы на ассемблере длиннее и сложнее, так как нет встроенных функций, которые облегчают работу.
- Однако программы на ассемблере обычно более быстрые и эффективные, так как код максимально приближен к машинному.
- Языки высокого уровня независимы от платформы, а программы на ассемблере — привязаны к конкретной архитектуре процессора.

2. Отличие инструкции от директивы на языке ассемблера:

Инструкция — это команда, которая преобразуется в машинный код и выполняется процессором (например, сложение или перемещение данных). В то время как директива — это указание для ассемблера, которое помогает в

организации программы, но не выполняется процессором напрямую (например, директивы для выделения памяти или указания конца программы).

3. Основные правила оформления программ на языке ассемблера:
Важна четкая структура с заголовками секций, строгое соблюдение синтаксиса и форматирования (включая регистры, операнды и т. д.).

4. Этапы получения исполняемого файла:

1. Написание исходного кода.
2. Трансляция исходного кода в объектный файл.
3. Компоновка объектного файла в исполняемый файл.
4. Оптимизация и выполнение дополнительных проверок (например, проверка ссылок).

5. Назначение этапа трансляции:
Трансляция преобразует исходный код программы на языке ассемблера в объектный код — машинные инструкции, которые могут быть поняты процессором.

6. Назначение этапа компоновки:
Компоновка объединяет объектные файлы и библиотеки, разрешает внешние ссылки, и формирует единый исполняемый файл.

7. Файлы, создаваемые при трансляции программы:

- Объектные файлы (.o, .obj) — по умолчанию создаются при трансляции.
- Листинговые файлы (.lst) — могут создаваться с дополнительными флагами.
- Отладочные файлы (.dbg, .map) — создаются при использовании режимов отладки.

8. Форматы файлов для nasm и ld:

- NASM поддерживает различные форматы объектных файлов: ELF, COFF, Win32, Win64, Mach-O.
- LD (GNU linker) поддерживает форматы ELF (на Linux), PE (на Windows), а также другие, в зависимости от целевой системы.

6 Вывод

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

7 Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).