

Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Аджабханян Овик

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Создание программы Hello world!	10
4.2	Работа с транслятором NASM	11
4.3	Работа с расширенным синтаксисом командной строки NASM . . .	12
4.4	Работа с компоновщиком LD	12
4.5	Выполнение заданий для самостоятельной работы.	13
5	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание каталога	10
4.2	Перемещение между директориями	10
4.3	Создание пустого файла	10
4.4	Открытие файла в текстовом редакторе	10
4.5	Заполнение файла	11
4.6	Компиляция текста программы	11
4.7	Компиляция текста программы	12
4.8	Передача объектного файла на обработку компоновщику	12
4.9	Проверка файлов	12
4.10	Передача объектного файла на обработку компоновщику	12
4.11	Запуск исполняемого файла	13
4.12	Создание копии файла	13
4.13	Изменение программы	13
4.14	Компиляция текста программы	14
4.15	Компиляция текста программы	14
4.16	Передача объектного файла на обработку компоновщику	14
4.17	Передача объектного файла на обработку компоновщику	14
4.18	Запуск исполняемого файла	14
4.19	Копирование файлов	14
4.20	Копирование файлов	14
4.21	Загрузка файлов	15

Список таблиц

1 Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

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

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 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. 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

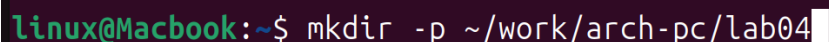
следующей команде.

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.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world!

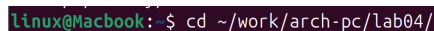
с помощью команды `mkdir -p ~/work/arch-pc/lab04` создаю lab04 каталог в файле `~/work` (рис. 4.1).



```
linux@Macbook:~$ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.1: Создание каталога

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 4.2).



```
linux@Macbook:~$ cd ~/work/arch-pc/lab04/
```

Рис. 4.2: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 4.3).



```
linux@Macbook:~/work/arch-pc/lab04$ touch hello.asm
```

Рис. 4.3: Создание пустого файла

Открываю созданный файл в текстовом редакторе `gedit` (рис. 4.4).



```
linux@Macbook:~/work/arch-pc/lab04$ gedit hello.asm
```

Рис. 4.4: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.5).

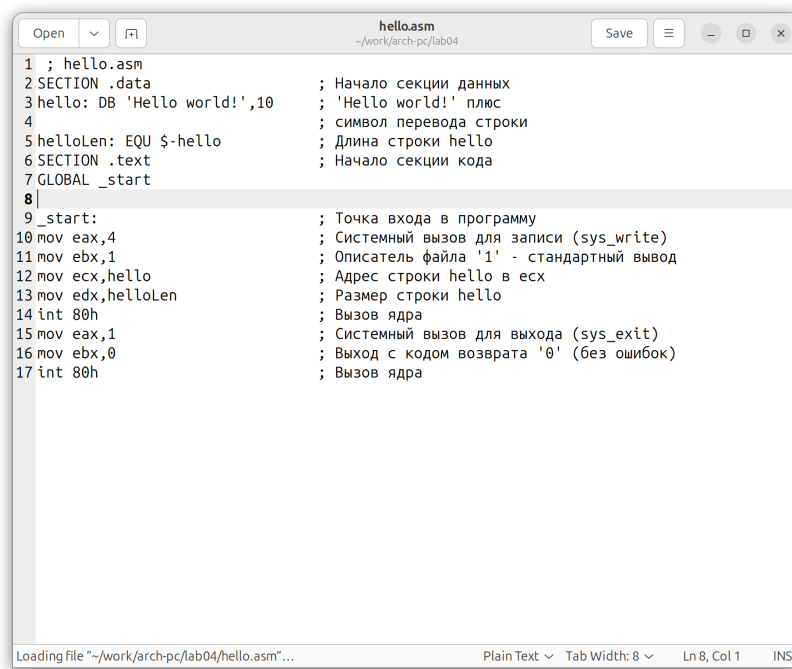


Рис. 4.5: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю 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 (рис. 4.6). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.

```
linux@Macbook:~/work/arch-pc/lab04$ nasm -f elf hello.asm
```

Рис. 4.6: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

Ввожу 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` (рис. 4.7). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
linux@Macbook:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
```

Рис. 4.7: Компиляция текста программы

4.4 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 4.8). Ключ `-o` задает имя создаваемого исполняемого файла.

```
linux@Macbook:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

проверяю с помощью утилиты `ls` правильность выполнения команды. (рис. 4.9).

```
linux@Macbook:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst obj.o
```

Рис. 4.9: Проверка файлов

Выполняю следующую команду (рис. 4.10). 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`

```
linux@Macbook:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
```

Рис. 4.10: Передача объектного файла на обработку компоновщику

Запускаю на выполнение созданный исполняемый файл hello (рис. 4.11).

```
linux@Macbook:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 4.11: Запуск исполняемого файла

4.5 Выполнение заданий для самостоятельной работы.

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm` (рис. 4.12). Проверяю с помощью утилиты `ls`.

```
linux@Macbook:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
linux@Macbook:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

Рис. 4.12: Создание копии файла

С помощью текстового редактора `gedit` открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.13).

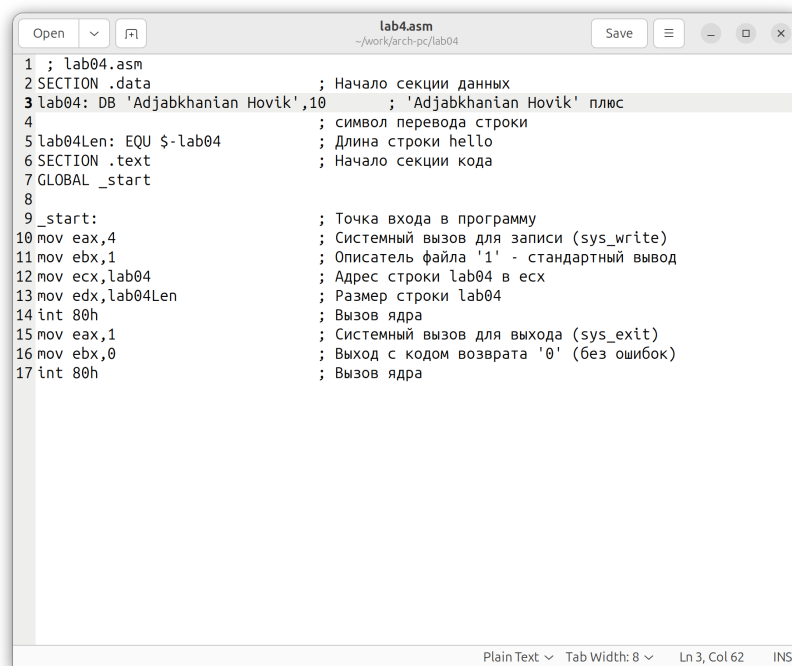


Рис. 4.13: Изменение программы

Компилирую текст программы в объектный файл (рис. 4.14).

```
linux@Macbook:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
```

Рис. 4.14: Компиляция текста программы

```
linux@Macbook:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst lab4.asm
```

Рис. 4.15: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 4.16).

```
linux@Macbook:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab04
```

Рис. 4.16: Передача объектного файла на обработку компоновщику

```
linux@Macbook:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
```

Рис. 4.17: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 4.18).

```
linux@Macbook:~/work/arch-pc/lab04$ ./lab04
Adjabkhanian Hovik
```

Рис. 4.18: Запуск исполняемого файла

Копирую файлы hello.asm и lab4.asm в мой локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. (рис. 4.19).

```
linux@Macbook:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04/report/
```

Рис. 4.19: Копирование файлов

```
linux@Macbook:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04/report/
```

Рис. 4.20: Копирование файлов

Загружаю файлы на Github. (рис. 4.21).

```
linux@Macbook:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git add .
linux@Macbook:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git commit
-am 'feat(main): make course structure'
[master ebf1b63] feat(main): make course structure
2 files changed, 34 insertions(+)
create mode 100644 labs/lab04/report/hello.asm
create mode 100644 labs/lab04/report/lab4.asm
linux@Macbook:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.19 KiB | 1.19 MiB/s, done.
Total 7 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:adjabkhanian/study_2023-2024_arhpc.git
77a0998..ebf1b63 master -> master
linux@Macbook:~/work/study/2023-2024/Архитектура компьютера/arch-pc$
```

Рис. 4.21: Загрузка файлов

5 Выводы

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

Список литературы

[illegible]