

Отчёта по лабораторной работе №5

Дисциплина: архитектура компьютера

Дворкина Ева Владимировна

1 Цель работы

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

2 Задание

- Создание программы Hello world!
- Работа с транслятором NASM
- Работа с расширенным синтаксисом командной строки NASM
- Работа с компоновщиком LD
- Запуск исполняемого файла
- 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.

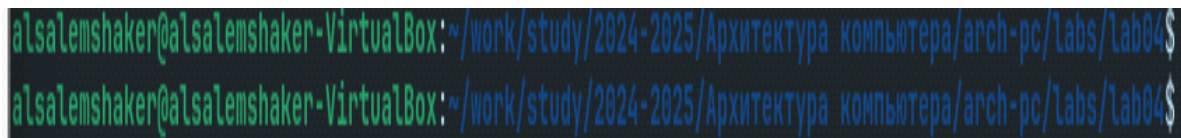
Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении 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!

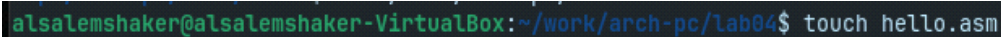
С помощью утилиты cd перемещаюсь в каталог, в котором буду работать (рис. 1).



```
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$  
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$
```

Рис. 1: Перемещение между директориями

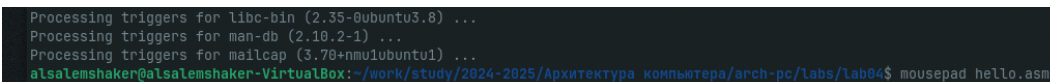
Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 2).



```
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab04$ touch hello.asm
```

Рис. 2: Создание пустого файла

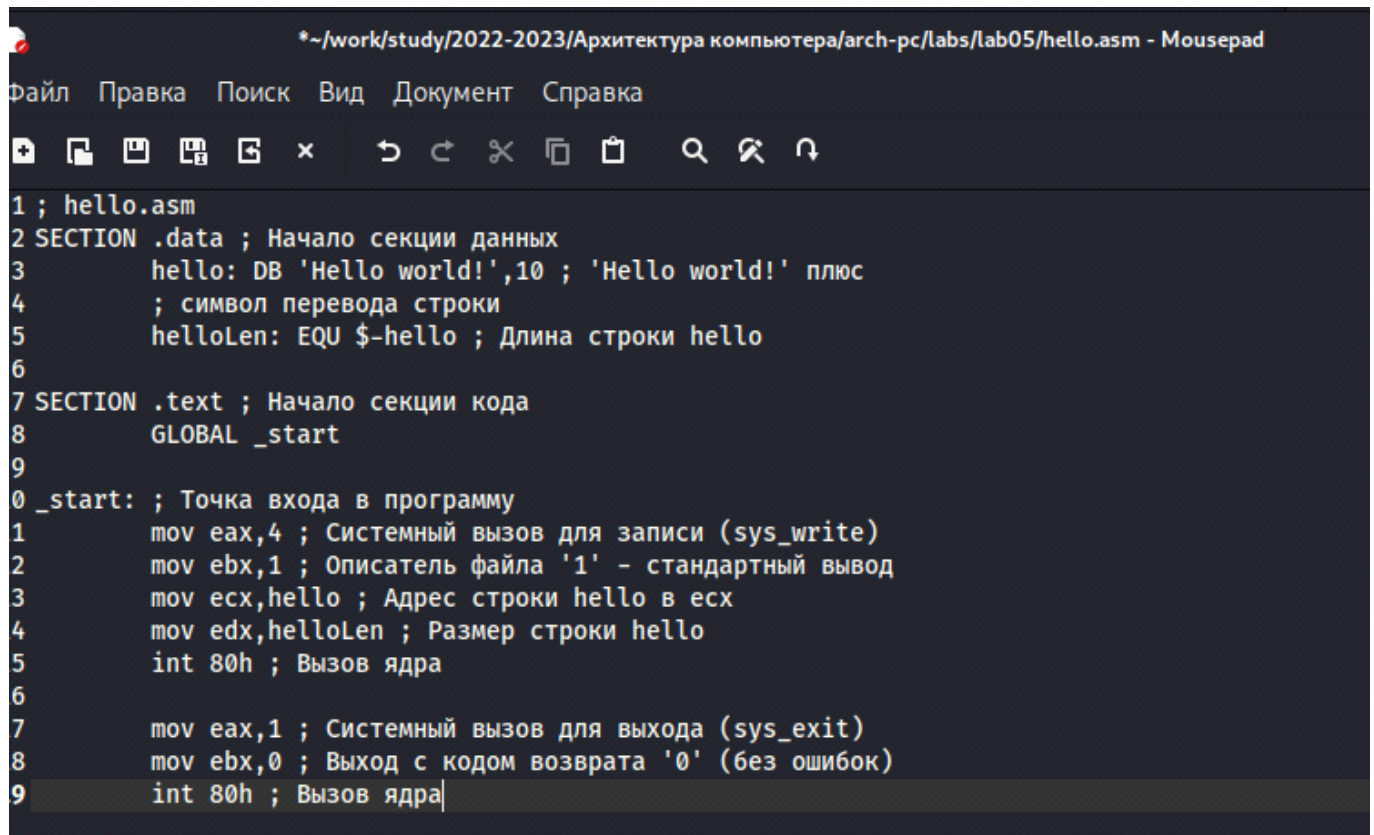
Открываю созданный файл в текстовом редакторе `mousepad` (рис. 3).



```
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...  
Processing triggers for man-db (2.10.2-1) ...  
Processing triggers for mailcap (3.70+nmulubuntu1) ...  
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$ mousepad hello.asm
```

Рис. 3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4).

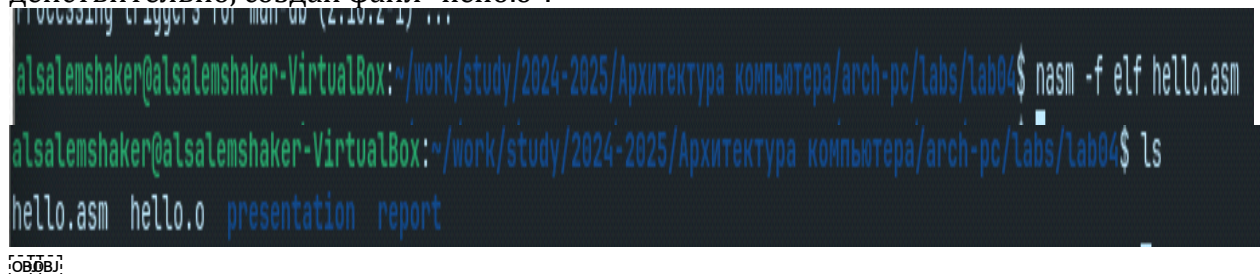
A screenshot of a text editor window titled '*~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab05/hello.asm - Mousepad'. The window has a menu bar with 'Файл', 'Правка', 'Поиск', 'Вид', 'Документ', and 'Справка'. Below the menu is a toolbar with icons for file operations and editing. The main text area contains assembly code for a 'Hello world' program. The code is as follows:

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

Рис. 4: 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. 5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.

A screenshot of a terminal window showing the execution of NASM and the subsequent listing of files. The terminal output is as follows:

```
Processing triggers for man-db (2.10.2-1) ...
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$ nasm -f elf hello.asm
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm hello.o presentation report
```

Рис. 5: 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. 6). Далее проверяю с помощью утилиты ls правильность выполнения команды.

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

Рис. 6: Компиляция текста программы

4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. 7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

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

Рис. 7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 8). 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

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o hello
```

Рис. 8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

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

Рис. 9: Запуск исполняемого файла

4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab5.asm (рис. 10).

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ cp hello.asm lab04.asm
```

Рис. 10: Создание копии файла

С помощью текстового редактора mousepad открываю файл lab5.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).

```
1 ; shaker.asm
2 SECTION .data ; Начало секции данных
3 shaker: DB 'Shaker Alsalem',10 ; 'Shaker Alsalem' плюс ; символ перевода
4 shakerLen: EQU $-shaker ; Длина строки shaker
5 SECTION .text ; Начало секции кода
6 GLOBAL _start
7 _start: ; Точка входа в программу
8 mov eax,4 ; Системный вызов для записи (sys_write)
9 mov ebx,1 ; Описатель файла '1' - стандартный вывод
10 mov ecx,shaker ; Адрес строки shaker в ecx
11 mov edx,shakerLen ; Размер строки hello
12 int 80h ; Вызов ядра
13
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
```

Рис. 11: Изменение программы

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab5.o создан.

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ gedit shaker.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ nasm -f elf shaker.asm
```

Рис. 12: Компиляция текста программы

Передаю объектный файл lab5.o на обработку компоновщику LD, чтобы получить исполняемый файл lab5 (рис. 13).

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ ld -m elf_i386 shaker.o -o shaker
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab04.asm list.lst main obj.o shaker shaker.asm shaker.o
```

Рис. 13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab5, на экран действительно выводятся мои имя и фамилия (рис. 14).

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ ./shaker
Shaker Alsalem
```

Рис. 14: Запуск исполняемого файла

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис. 17).


```

alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04/report$ git add .
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04/report$ git commit -m
error: switch `m' requires a value
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04/report$ git commit -m "Add files for la
[master 64d4e18] Add files for lab04
7 files changed, 35 insertions(+)
create mode 100755 labs/lab04/hello
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/hello.o
create mode 100644 labs/lab04/list.lst
create mode 100755 labs/lab04/main
create mode 100644 labs/lab04/obj.o
create mode 100644 labs/lab04/report/report.docx
alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04/report$

```

Рис. 17: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 18).

```

alsalemsaker@alsalemsaker-VirtualBox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04/report$ git push
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 266.99 KiB | 427.00 KiB/s, done.
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:Alsalemsaker/study_2024-2025_arh-pc.git
 b204aae..64d4e18 master -> master

```

Рис. 18: Отправка файлов

5 Выводы

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

6 Список литературы

- https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf