

# **Отчёта по лабораторной работе №4**

**Дисциплина: архитектура компьютера**

Шабалина Елизавета Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Создание программы Hello world! . . . . .	9
4.2	Работа с транслятором NASM . . . . .	10
4.3	Работа с расширенным синтаксисом командной строки NASM . .	10
4.4	Работа с компоновщиком LD . . . . .	11
4.5	Запуск исполняемого файла . . . . .	11
4.6	Выполнение заданий для самостоятельной работы. . . . .	12
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Список литературы</b>	<b>15</b>

## Список иллюстраций

4.1	создание каталога . . . . .	9
4.2	Создание пустого файла . . . . .	9
4.3	Заполнение файла . . . . .	10
4.4	Компиляция текста программы . . . . .	10
4.5	Компиляция текста программы . . . . .	11
4.6	Передача объектного файла на обработку компоновщику . . . . .	11
4.7	Передача объектного файла на обработку компоновщику . . . . .	11
4.8	Запуск исполняемого файла . . . . .	11
4.9	Создание копии файла . . . . .	12
4.10	Изменение программы . . . . .	12
4.11	все действия с lab4 . . . . .	12
4.12	Добавление файлов на GitHub . . . . .	13

# 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 опе-

рации) 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. считы-

вание 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!

Создаю каталог для работы с программами на языке ассемблера NASM и перехожу в него (рис. 4.1).

```
eashabalina@dk3n55 ~ $ mkdir -p ~/work/arch-pc/lab04
eashabalina@dk3n55 ~ $ cd ~/work/arch-pc/lab04
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.1: создание каталога

Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch и открываю с помощью gedit (рис. 4.2).

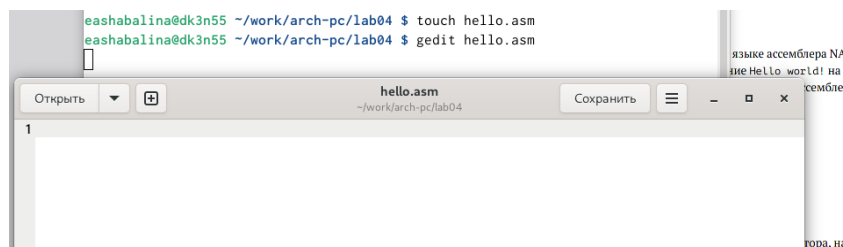
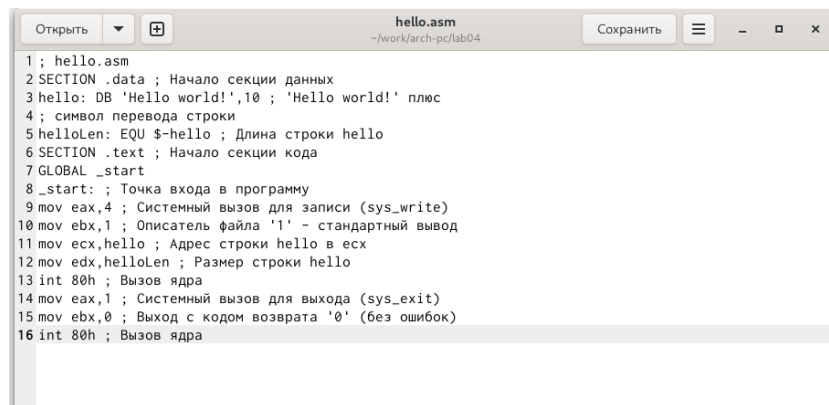


Рис. 4.2: Создание пустого файла

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.3).

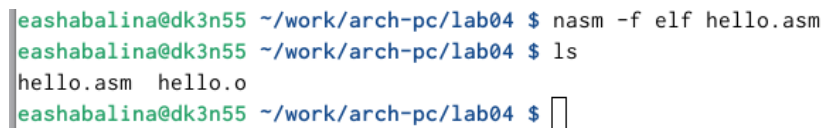


```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8 _start: ; Точка входа в программу
9 mov eax,4 ; Системный вызов для 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
```

Рис. 4.3: 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. 4.4). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.4: 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. 4.5). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.5: Компиляция текста программы

## 4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. 4.6). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.6: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 4.7). 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

```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

## 4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 4.8).

```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ./hello
Hello world!
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.8: Запуск исполняемого файла

## 4.6 Выполнение заданий для самостоятельной работы.

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

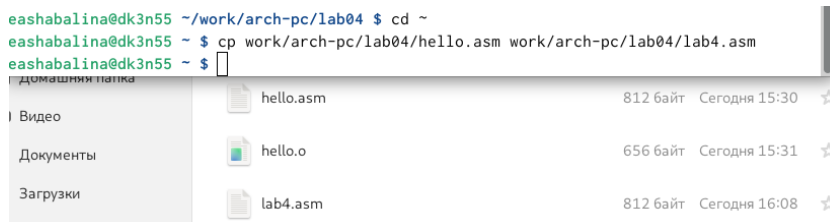


Рис. 4.9: Создание копии файла

С помощью текстового редактора `mousepad` открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.10).

```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Шабалина Елизавета',10 ; 'Шабалина Елизавета' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
```

Рис. 4.10: Изменение программы

Транслирую полученный текст программы `lab4.asm` в объектный файл. Выполняю компоновку объектного файла и запускаю получившийся исполняемый файл.(рис. 4.11).

```
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst lab4.asm
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
eashabalina@dk3n55 ~/work/arch-pc/lab04 $ ./lab4
Шабалина Елизавета
eashabalina@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.11: все действия с `lab4`

Копирую файлы `hello.asm` и `lab4.asm` в мой локальный репозиторий в каталог `~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/` и с помощью команд `git add .`, `git commit` и `git push` добавляю файлы на GitHub. (рис. 4.12).

[study\\_2024-2025\\_arh-pc](#) / [labs](#) / **lab04** / 







 <b>Elizaveta-Shabalina</b> feat(main): make course structure
Name
 ..
 presentation
 report
 hello.asm
 lab4.asm

Рис. 4.12: Добавление файлов на GitHub

## **5 Выводы**

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

## 6 Список литературы

1. [https://esystem.rudn.ru/pluginfile.php/1584628/mod\\_resource/content/1/%D0%9B%D0%B0](https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0)