

# **Отчет по лабораторной работе №4**

**Дисциплина: архитектура компьютеров**

Пестова Ева Константиновна

# Содержание

1	1. Цель работы	5
2	2. Задание	6
3	3. Теоретическое введение	7
4	4. Выполнение лабораторной работы	10
5	5. Выводы	16

## Список иллюстраций

4.1	Перемещение между директориями . . . . .	10
4.2	Создание пустого файла . . . . .	10
4.3	Заполнение файла . . . . .	11
4.4	Компиляция текста программы . . . . .	11
4.5	Компиляция текста программы . . . . .	12
4.6	Передача объектного файла на обработку компоновщику . . . . .	12
4.7	Запуск исполняемого файла . . . . .	12
4.8	Создание копии файла . . . . .	13
4.9	Изменение программы . . . . .	13
4.10	Компиляция и передача файла компоновщику . . . . .	13
4.11	Запуск исполняемого файла . . . . .	14
4.12	Создание директории . . . . .	14
4.13	Копирование файлов . . . . .	14
4.14	Проверка наличия файлов . . . . .	14
4.15	Отправка файлов . . . . .	15

## Список таблиц

# **1 1. Цель работы**

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

## 2 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 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) — машинноориентированный 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 4. Выполнение лабораторной работы

### 4.1 Создание программы Hello world!

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. [4.10]).

```
ekpestova@dk8n76 ~ $ cd work/study/2023-2024/Архитектура\ компьютеров/arch-pc/labs/lab04
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` и открываю его в текстовом редакторе (рис. [4.2]).

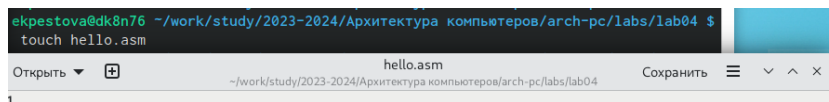
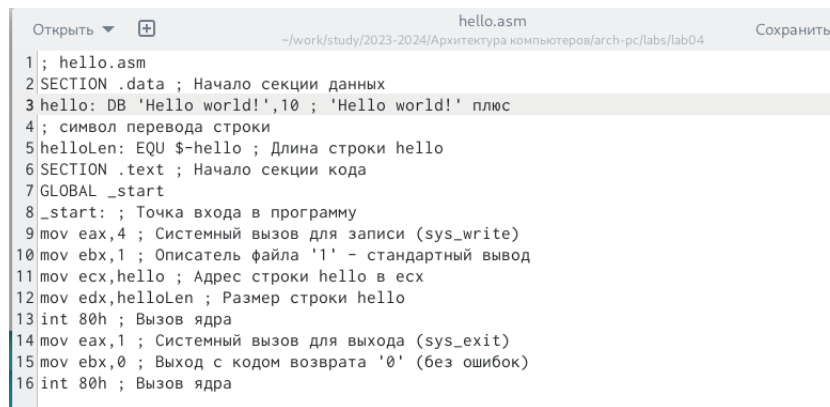


Рис. 4.2: Создание пустого файла

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



```

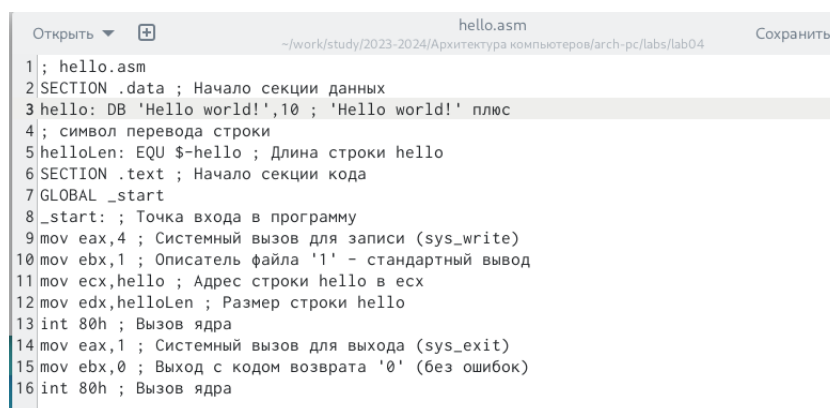
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8 _start: ; Точка входа в программу
9 mov eax,4 ; Системный вызов для записи (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. Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o” (рис. [4.4]).



```

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. 4.4: 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.

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
nasm -f elf hello.asm  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls  
hello.asm hello.o presentation report
```

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. Выполняю команду (рис. [4.6]). 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.

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ld -m elf_i386 hello.o -o hello  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls  
hello hello.asm hello.o list.lst obj.o presentation report  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ld -m elf_i386 obj.o -o main  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls  
hello hello.asm hello.o list.lst main obj.o presentation report
```

Рис. 4.6: Передача объектного файла на обработку компоновщику

#### 4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. [4.7]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
./hello  
Hello world!
```

Рис. 4.7: 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. [??]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
cp hello.asm lab4.asm
```

Рис. 4.8: Создание копии файла

С помощью текстового редактора открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию (рис. [4.9]).

```
1 ; lab4.asm  
2 SECTION .data ; Начало секции данных  
3 lab4: DB 'Eva Pestova',10  
4 lab4Len: EQU $-lab4 ; Длина строки lab4  
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,lab4 ; Адрес строки lab4 в ecx  
11 mov edx,lab4Len ; Размер строки lab  
12 int 80h ; Вызов ядра  
13 mov eax,1 ; Системный вызов для выхода (sys_exit)  
14 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)  
15 int 80h ; Вызов ядра
```

Рис. 4.9: Изменение программы

Компилирую текст программы в объектный файл. Проверяю с помощью утилиты `ls`, что файл `lab4.o` создан. Передаю объектный файл `lab4.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `lab4` (рис. [??]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
nasm -f elf lab4.asm  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls  
hello.o lab4.o main presentation  
hello.asm lab4.asm list.lst obj.o report  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ld -m elf_i386 lab4.o -o lab4  
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls  
hello.o lab4.asm list.lst obj.o report  
hello.asm lab4 lab4.o main presentation
```

Рис. 4.10: Компиляция и передача файла компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. [4.11]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
./lab4  
Eva Pestova
```

Рис. 4.11: Запуск исполняемого файла

Я начала работу не в том каталоге, поэтому создаю другую директорию lab04 с помощью mkdir (рис. [4.12]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
mkdir ~/work/study/2023-2024/Архитектура\ компьютеров/arch-pc/lab04
```

Рис. 4.12: Создание директории

Далее копирую из текущего каталога файлы, созданные в процессе выполнения лабораторной работы, с помощью утилиты cp, указывая вместо имени файла символ \*, чтобы скопировать все файлы. Проверяю с помощью утилиты ls правильность выполнения команды (рис. [4.13]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
cp * ~/work/study/2023-2024/Архитектура\ компьютеров/arch-pc/lab04  
cp: не указан -r; пропускается каталог 'presentation'  
cp: не указан -r; пропускается каталог 'report'
```

Рис. 4.13: Копирование файлов

Проверяю с помощью утилиты ls правильность выполнения команды (рис. [4.14]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $  
ls ~/work/study/2023-2024/Архитектура\ компьютеров/arch-pc/lab04  
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
```

Рис. 4.14: Проверка наличия файлов

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm, ведь копии файлов остались в другой директории (рис. [??]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $
rm hello hello.o lab4 lab4.o list.lst main obj.o
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $
ls
hello.asm lab4.asm presentation report
```

С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. [??]), (рис. [??]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $ git add .
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $ git commit -m "Add files for lab04"
```

Отправляю файлы на сервер с помощью команды `git push` (рис. [4.15]).

```
ekpestova@dk8n76 ~/work/study/2023-2024/Архитектура компьютеров/arch-pc/labs/lab04 $
git push
To github.com:1132236953/study-2023-2024-arch-pc:git
```

Рис. 4.15: Отправка файлов

## **5 5. Выводы**

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