

# **Отчёт по лабораторной работе №4**

**Дисциплина: архитектура компьютеров и операционные системы**

Постнова Елизавета Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Программа Hello world! . . . . .	9
4.2	Транслятор NASM . . . . .	10
4.3	Расширенный синтаксис командной строки NASM . . . . .	10
4.4	Компоновщик LD . . . . .	10
4.5	Запуск исполняемого файла . . . . .	11
4.6	Задание для самостоятельной работы . . . . .	11
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Список литературы</b>	<b>15</b>

## Список иллюстраций

4.1	Переход в каталог . . . . .	9
4.2	Ввод данного текста в файл . . . . .	9
4.3	Компиляция текста с помощью команды . . . . .	10
4.4	Компиляция файла с помощью команды . . . . .	10
4.5	Получение исполняемой программы . . . . .	10
4.6	Создание исполняемого файла . . . . .	11
4.7	Запуск исполняемого файла с помощью команды . . . . .	11
4.8	Копирование файла . . . . .	11
4.9	Изменение файла с заданными условиями . . . . .	12
4.10	Изменение файла с заданными условиями . . . . .	12
4.11	Копирование файлов в каталог . . . . .	13
4.12	Загрузка файлов на Github . . . . .	13

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Программа Hello world!
2. Транслятор NASM
3. Расширенный синтаксис командной строки NASM
4. Компоновщик LD
5. Запуск исполняемого файла
6. Задание для самостоятельной работы

### 3 Теоретическое введение

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 (половинки 16-битных регистров). Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX. 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. 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. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.



## 4 Выполнение лабораторной работы

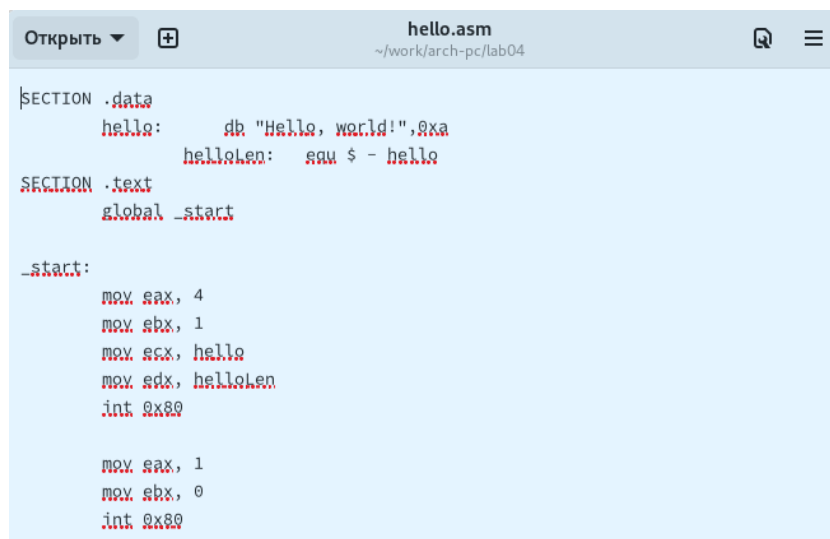
### 4.1 Программа Hello world!

Создаю каталог для работы с программами на языке ассемблера NASM и перехожу в созданный каталог. (рис. 4.12)

```
[eapostnova@fedora ~]$ mkdir -p ~/work/arch-pc/lab04  
[eapostnova@fedora ~]$ cd ~/work/arch-pc/lab04  
[eapostnova@fedora lab04]$ touch hello.asm
```

Рис. 4.1: Переход в каталог

Создаю текстовый файл с именем hello.asm, открываю этот файл с помощью текстового редактора и ввожу следующий текст. (рис. 4.12)



```
Открыть + hello.asm  
~/work/arch-pc/lab04  
SECTION .data  
    hello:    db "Hello, world!",<div data-bbox="323 811 672 830" data-label="Caption">

Рис. 4.2: Ввод данного текста в файл


```

## 4.2 Транслятор NASM

Компилируем приведённый выше текст программы при помощи команды `nasm -f elf hello.asm` «Hello World» и проверим, что файл создан. (рис. 4.12)

```
[eapostnova@fedora lab04]$ nasm -f elf hello.asm
[eapostnova@fedora lab04]$ ls
hello.asm  hello.o
```

Рис. 4.3: Компиляция текста с помощью команды

## 4.3 Расширенный синтаксис командной строки NASM

С помощью команды `nasm -o obj.o -f elf -g -l list.lst hello.asm` скомпилируем исходный файл `hello.asm` в `obj.o` и проверим, что файл создан. (рис. 4.12)

```
[eapostnova@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[eapostnova@fedora lab04]$ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.4: Компиляция файла с помощью команды

## 4.4 Компоновщик LD

Передаем объектный файл на обработку компоновщику с помощью команды `ld -m elf_i386 hello.o -o hello` и проверяем, что исполняемый файл `hello` был создан. (рис. 4.12)

```
[eapostnova@fedora lab04]$ ld -m elf_i386 hello.o -o hello
[eapostnova@fedora lab04]$ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.5: Получение исполняемой программы

Создадим еще один файл с помощью команды `ld -m elf_i386 obj.o -o main`. (рис. 4.12)

```
[eapostnova@fedora lab04]$ ld -m elf_i386 obj.o -o main
[eapostnova@fedora lab04]$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рис. 4.6: Создание исполняемого файла

Имя исполняемого файла - main, имя объектного файла - obj.o

## 4.5 Запуск исполняемого файла

Запустим созданный исполняемый файл с помощью команды ./hello. (рис. 4.12)

```
[eapostnova@fedora lab04]$ ./hello
Hello, world!
```

Рис. 4.7: Запуск исполняемого файла с помощью команды

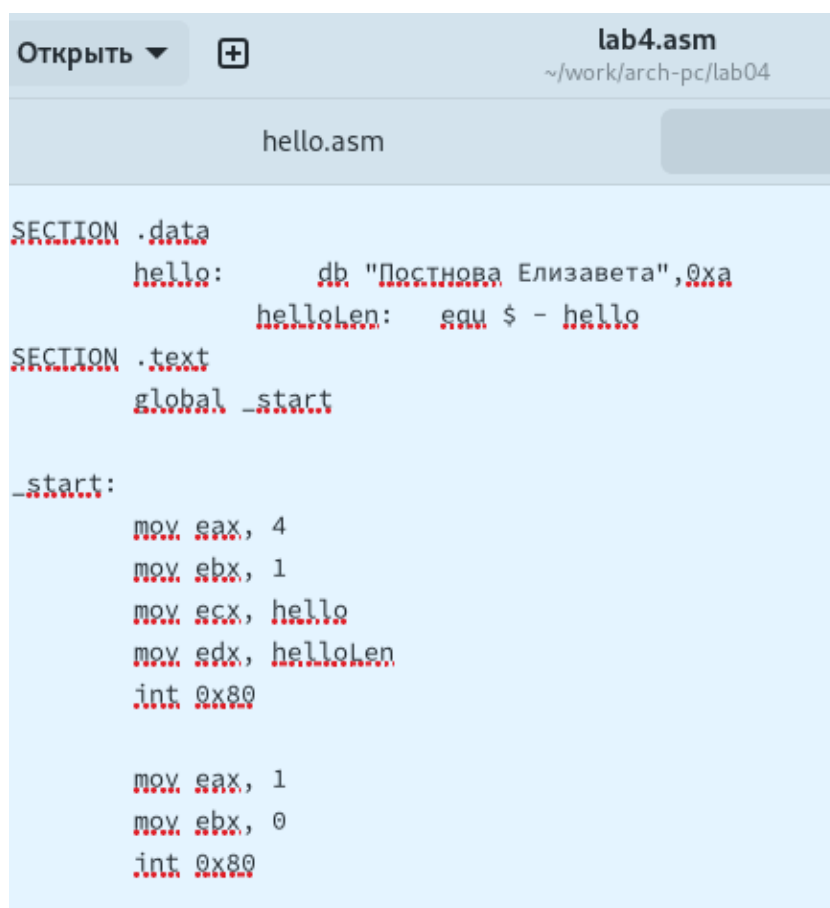
## 4.6 Задание для самостоятельной работы

1. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создаю копию файла hello.asm с именем lab4.asm. (рис. 4.12)

```
[eapostnova@fedora lab04]$ cp hello.asm lab4.asm
[eapostnova@fedora lab04]$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

Рис. 4.8: Копирование файла

2. С помощью текстового редактора вношу изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моими фамилией и именем. (рис. 4.12)



```
lab4.asm
~/work/arch-pc/lab04

hello.asm

SECTION .data
    hello:      db "Постнова Елизавета",0xa
    helloLen:   equ $ - hello

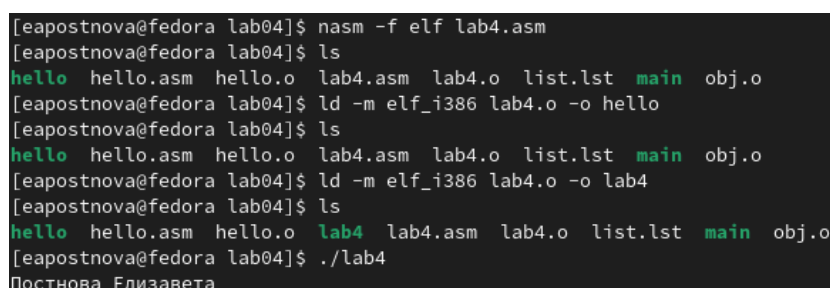
SECTION .text
    global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, hello
    mov edx, helloLen
    int 0x80

    mov eax, 1
    mov ebx, 0
    int 0x80
```

Рис. 4.9: Изменение файла с заданными условиями

3. Компилирую полученный текст программы lab4.asm в объектный файл. Выполняю компоновку объектного файла и запускаю получившийся исполняемый файл. (рис. 4.12)



```
[eapostnova@fedora lab04]$ nasm -f elf lab4.asm
[eapostnova@fedora lab04]$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
[eapostnova@fedora lab04]$ ld -m elf_i386 lab4.o -o hello
[eapostnova@fedora lab04]$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
[eapostnova@fedora lab04]$ ld -m elf_i386 lab4.o -o lab4
[eapostnova@fedora lab04]$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
[eapostnova@fedora lab04]$ ./lab4
Постнова Елизавета
```

Рис. 4.10: Изменение файла с заданными условиями

4. Копирую файлы hello.asm и lab4.asm в локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. (рис. 4.12)



Рис. 4.11: Копирование файлов в каталог

Загружаю файлы на Github. (рис. 4.12)

```
[eapostnova@fedora lab04]$ git add hello.asm lab4.asm
[eapostnova@fedora lab04]$ git commit -am 'feat(main): make course structure'
[master 74b7ca4] feat(main): make course structure
2 files changed, 32 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
[eapostnova@fedora lab04]$ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 693 байта | 693.00 КиБ/с, готово.
Всего 6 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:lisheriz/study_2023-2024_arh-pc.git
01f166b..74b7ca4 master -> master
```

Рис. 4.12: Загрузка файлов на Github

## **5 Выводы**

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

## 6 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](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).