

# **Отчёт по лабораторной работе 4**

**Дисциплина: архитектура компьютера**

Плугин Никита

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задания</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Программа Hello world! . . . . .	8
4.2	Транслятор NASM . . . . .	9
4.2.1	Расширенный синтаксис командной строки NASM . . . . .	10
4.3	Компоновщик LD . . . . .	10
4.3.1	Запуск исполняемого файла . . . . .	11
4.4	Задание для самостоятельной работы . . . . .	11
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Источники</b>	<b>15</b>

## Список иллюстраций

4.1	Создание файла для программы . . . . .	8
4.2	Программа hello.asm . . . . .	9
4.3	Трансляция программы . . . . .	10
4.4	Трансляция программы с дополнительными опциями . . . . .	10
4.5	Компоновка программы . . . . .	11
4.6	Компоновка второй программы . . . . .	11
4.7	Запуск программ . . . . .	11
4.8	Копирование файла . . . . .	12
4.9	Программа lab4.asm . . . . .	12
4.10	Проверка программы lab4.asm . . . . .	13

## Список таблиц

# 1 Цель работы

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

## 2 Задания

1. Изучить основы языка Ассемблера
2. Освоить и выполнить процесс компиляции программы на Ассемблере
3. Выполнить самостоятельное задание по изменению программы

### 3 Теоретическое введение

В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler). 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. Типичный формат записи команд NASM имеет вид:

`[метка:] мнемокод [операнд {, операнд}] [; комментарий]`

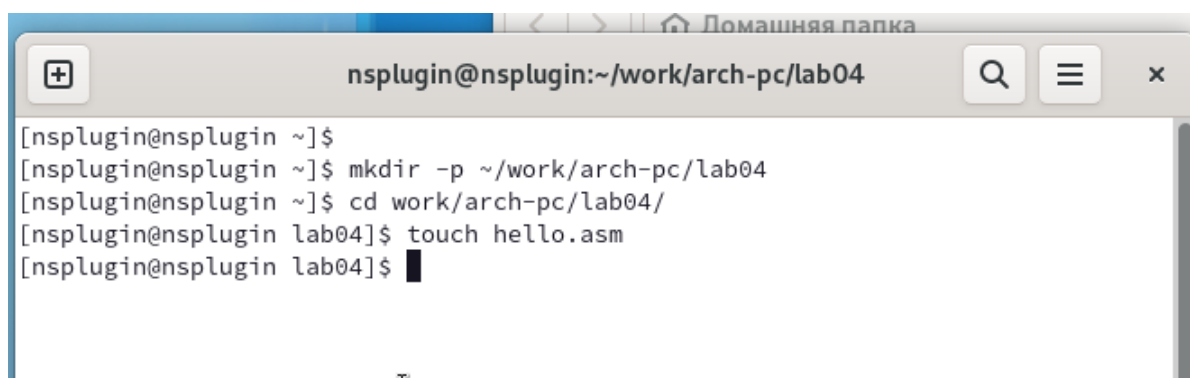
Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. метка перед командой связана с адресом данной команды.

## 4 Выполнение лабораторной работы

### 4.1 Программа Hello world!

Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран.

1. Создал каталог lab04 командой `mkdir`, перешел в него с помощью команды `cd`, создал файл `hello.asm`. (рис. [4.1])

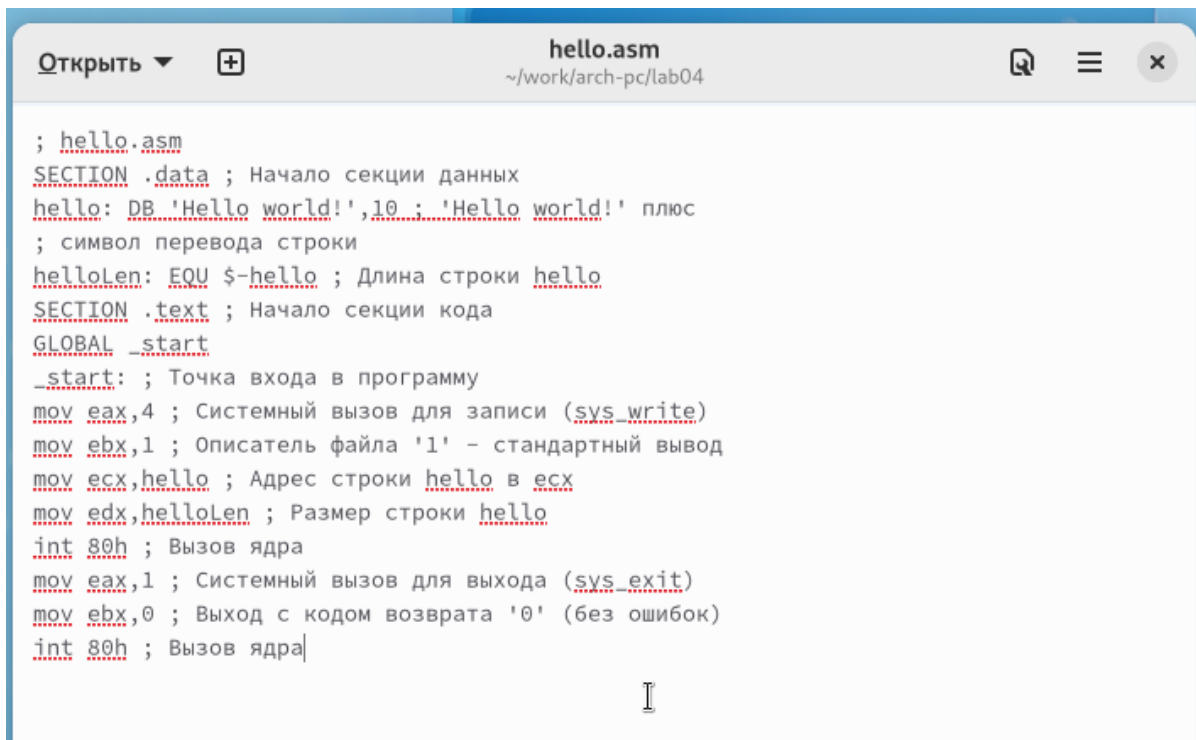


```
nsplugin@nsplugin:~/work/arch-pc/lab04
[nsplugin@nsplugin ~]$
[nsplugin@nsplugin ~]$ mkdir -p ~/work/arch-pc/lab04
[nsplugin@nsplugin ~]$ cd work/arch-pc/lab04/
[nsplugin@nsplugin lab04]$ touch hello.asm
[nsplugin@nsplugin lab04]$
```

Рис. 4.1: Создание файла для программы

2. Открыл файл и написал код программы по заданию.(рис. [4.2])





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

Рис. 4.2: Программа hello.asm

В отличие от многих современных высокоуровневых языков программирования, в ассемблерной программе каждая команда располагается на отдельной строке. Размещение нескольких команд на одной строке недопустимо. Синтаксис ассемблера NASM является чувствительным к регистру, т.е. есть разница между большими и малыми буквами.

## 4.2 Транслятор NASM

3. Транслировал файл командой `nasm` с опцией `-f`. (рис. [4.3])

Ключ `-f` указывает транслятору, что требуется создать бинарные файлы в формате ELF. Следует отметить, что формат `elf64` позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто `elf`.

Получился объектный файл hello.o

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

Рис. 4.3: Трансляция программы

### 4.2.1 Расширенный синтаксис командной строки NASM

4. Транслировал файл командой `nasm` с дополнительными опциями : `-o`, `-g`, `-l` (рис. [4.4])

Опция `-o` позволяет задать имя объектного файла. Опция `-g` добавляет отладочную информацию. Опция `-l` создает файл листинг.

Получился файл листинга `list.lst`, объектный файл `obj.o`, в программу добавилась отладочная информация.

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

Рис. 4.4: Трансляция программы с дополнительными опциями

## 4.3 Компоновщик LD

5. Выполнил компоновку командой `ld` и получил исполняемый файл. (рис. [4.5])

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

Рис. 4.5: Компоновка программы

Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла.

6. Еще раз выполнила компоновку для объектного файла obj.o и получил исполняемый файл main. (рис. [4.6])

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

Рис. 4.6: Компоновка второй программы

### 4.3.1 Запуск исполняемого файла

7. Запустил исполняемые файлы. (рис. [4.7])

```
[nsplugin@nsplugin lab04]$
[nsplugin@nsplugin lab04]$ ./hello
Hello world!
[nsplugin@nsplugin lab04]$ ./main
Hello world!
[nsplugin@nsplugin lab04]$
```

Рис. 4.7: Запуск программ

## 4.4 Задание для самостоятельной работы

1. Скопировал программу в файл lab4.asm. (рис. [4.8])

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

Рис. 4.8: Копирование файла

2. Изменил сообщение Hello world на свое имя. (рис. [4.9])

```
; hello.asm
SECTION .data ; Начало секции данных
hello: DB 'Nikita Plugin',10 ; 'Nikita Plugin' плюс
; символ перевода строки
helloLen: EQU $-hello ; Длина строки hello
SECTION .text ; Начало секции кода
GLOBAL _start
_start: ; Точка входа в программу
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,hello ; Адрес строки hello в ecx
mov edx,helloLen ; Размер строки hello
int 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
int 80h ; Вызов ядра
```

Рис. 4.9: Программа lab4.asm

3. Оттранслировал полученный текст программы lab4.asm в объектный файл. Выполнил компоновку объектного файла и запустил получившийся исполняемый файл. (рис. [4.10])

```
[nsplugin@nsplugin lab04]$  
[nsplugin@nsplugin lab04]$ nasm -f elf lab4.asm  
[nsplugin@nsplugin lab04]$ ld -m elf_i386 lab4.o -o lab4  
[nsplugin@nsplugin lab04]$ ./lab4  
Nikita Plugin  
[nsplugin@nsplugin lab04]$
```

Рис. 4.10: Проверка программы lab4.asm

4. Загрузил файлы на github.

## 5 Выводы

Освоили процесс компиляции и сборки программ, написанных на ассемблере `nasm`.

## **6 Источники**

1. Архитектура ЭВМ