

# **Отчёт по лабораторной работе №4**

**Дисциплина: Архитектура Компьютера**

Курилко-Рюмин Евгений

# 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>8</b>
<b>5</b>	<b>Выводы</b>	<b>16</b>
	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание каталога . . . . .	8
4.2	Перемещение . . . . .	8
4.3	Ввод текста . . . . .	9
4.4	Компиляция текста программы . . . . .	10
4.5	Компиляция текста программы . . . . .	11
4.6	Передача объектного файла на обработку компоновщику . . . . .	12
4.7	Запуск исполняемого файла . . . . .	13
4.8	Создание копии файла и его открытие, редактирование . . . . .	13
4.9	Компиляция, передача компоновщику . . . . .	14
4.10	Отправка файлов . . . . .	15

# 1 Цель работы

Целью данной работы является приобретение практического опыта работы с программами, написанными на ассемблере NASM, а именно - освоение процедур компиляций и сборки.

## 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

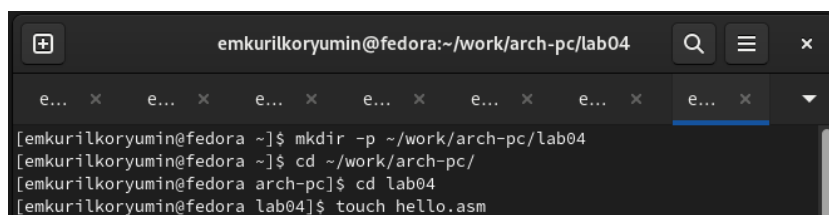
Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Но 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. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, ибо транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые

мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: 1) Для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM). 2) Для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис. Для записи команд в NASM используются: 1) Мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. 2) Операнды — числа, данные, адреса регистров или адреса оперативной памяти. 3) Метка — идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. (Метка перед командой связана с адресом данной команды). Допустимыми символами в метках являются буквы, цифры, а также следующие символы: `,`, `$`, `#`, `@`, `~`, `.` и `?`. *Начинаться метка или идентификатор могут с буквы, `.`, и `?`.* Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать `$`, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора составляет 4095 символов. 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

### 1) Программа Hello World!

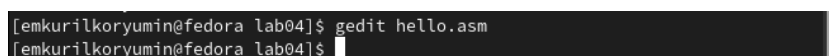
Создаю каталог для работы с программами на языке ассемблера NASM также перехожу в каталог и создаю текстовый файл hello.asm. (рис.1).



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

Рис. 4.1: Создание каталога

Открытие файла. (рис.2).

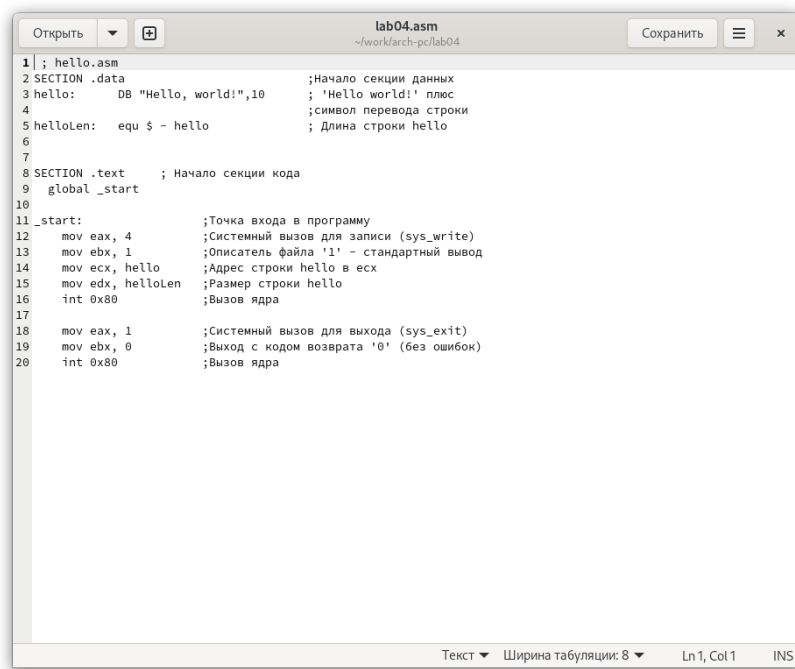


```
[emkurilkoryumin@fedora lab04]$ gedit hello.asm
[emkurilkoryumin@fedora lab04]$
```

Рис. 4.2: Перемещение

Ввожу нужный текст. (рис.3).



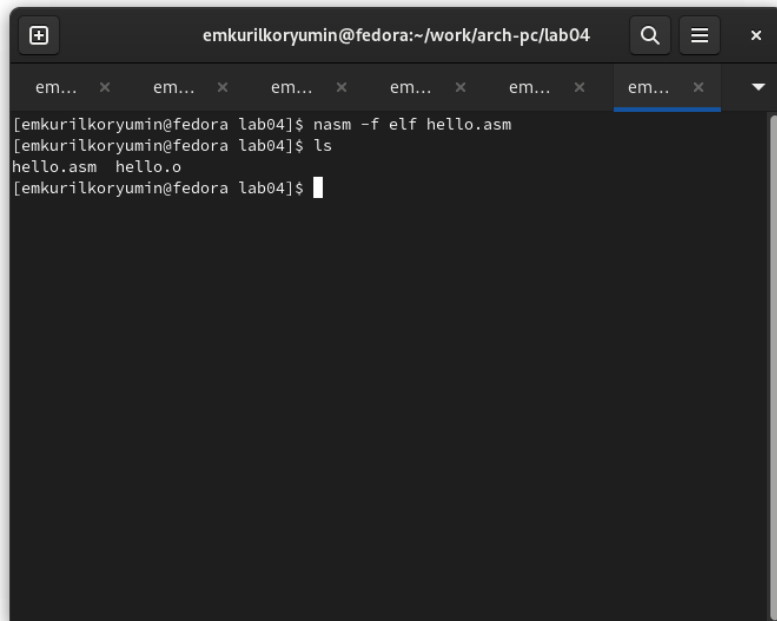


```
1 | ; hello.asm
2 | SECTION .data                ; Начало секции данных
3 | hello:  DB "Hello, world!",10 ; 'Hello world!' плюс
4 |                               ; символ перевода строки
5 | helloLen: equ $ - hello      ; Длина строки hello
6 |
7 |
8 | SECTION .text                ; Начало секции кода
9 | global _start
10 |
11 | _start:                      ; Точка входа в программу
12 |     mov eax, 4                ; Системный вызов для записи (sys_write)
13 |     mov ebx, 1                ; Описатель файла '1' - стандартный вывод
14 |     mov ecx, hello            ; Адрес строки hello в ecx
15 |     mov edx, helloLen         ; Размер строки hello
16 |     int 0x80                 ; Вызов ядра
17 |
18 |     mov eax, 1                ; Системный вызов для выхода (sys_exit)
19 |     mov ebx, 0                ; Выход с кодом возврата '0' (без ошибок)
20 |     int 0x80                 ; Вызов ядра
```

Рис. 4.3: Ввод текста

## 2) Транслятор NASM

NASM превращает текст программы в объектный код. Выполним компиляцию приведённого выше текста программы “Hello World”. Сделаем проверку. (рис.4).

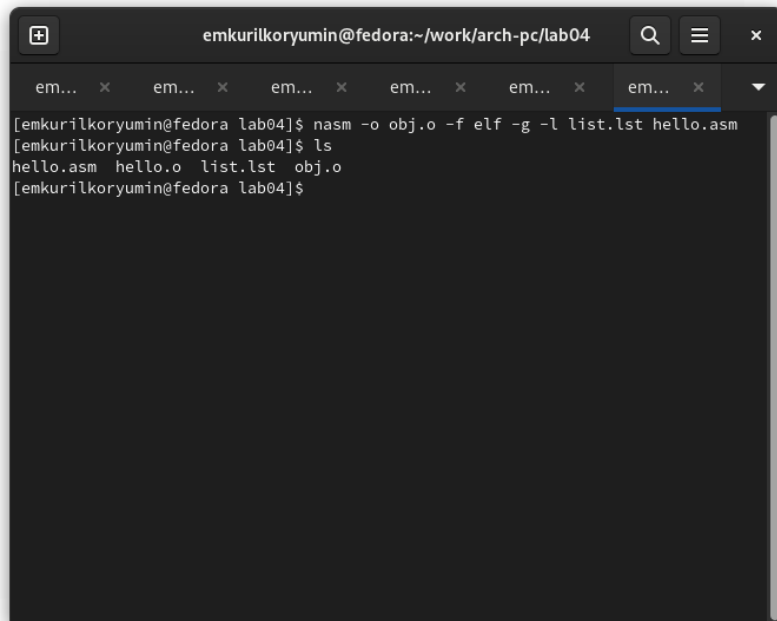


```
emkurilkoryumin@fedora:~/work/arch-pc/lab04
[emkurilkoryumin@fedora lab04]$ nasm -f elf hello.asm
[emkurilkoryumin@fedora lab04]$ ls
hello.asm  hello.o
[emkurilkoryumin@fedora lab04]$
```

Рис. 4.4: Компиляция текста программы

### 3) Расширенный синтаксис 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`. Проверяю правильность выполнения команды. (рис.5).

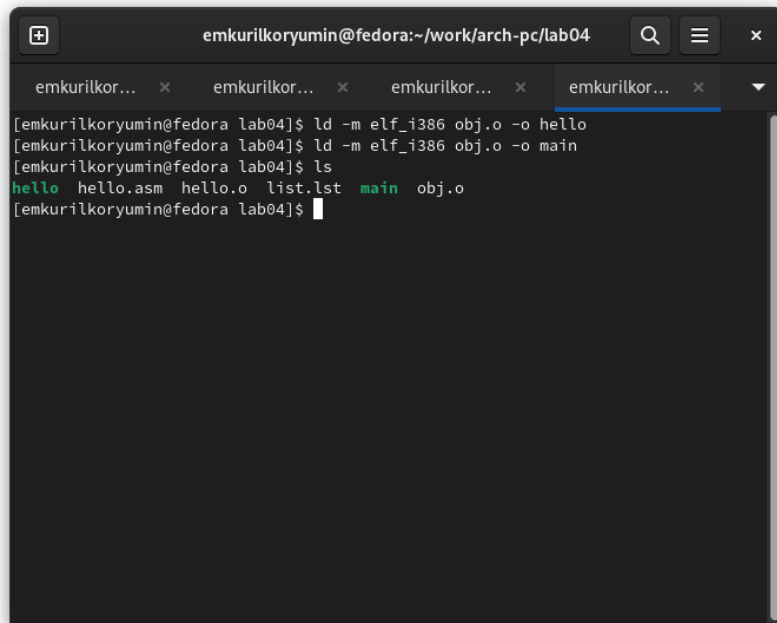


```
emkurilkoryumin@fedora:~/work/arch-pc/lab04
[emkurilkoryumin@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[emkurilkoryumin@fedora lab04]$ ls
hello.asm  hello.o  list.lst  obj.o
[emkurilkoryumin@fedora lab04]$
```

Рис. 4.5: Компиляция текста программы

#### 4) Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello`. Ключ `-o` задает имя создаваемого исполняемого файла. Выполняю ту же самую команду со значением `main`. Объектный файл, из которого собран этот исполняемый файл, имеет имя `obj.o` (рис.6).

A terminal window titled 'emkurilkoryumin@fedora:~/work/arch-pc/lab04' with four tabs. The terminal shows the following commands and output:

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

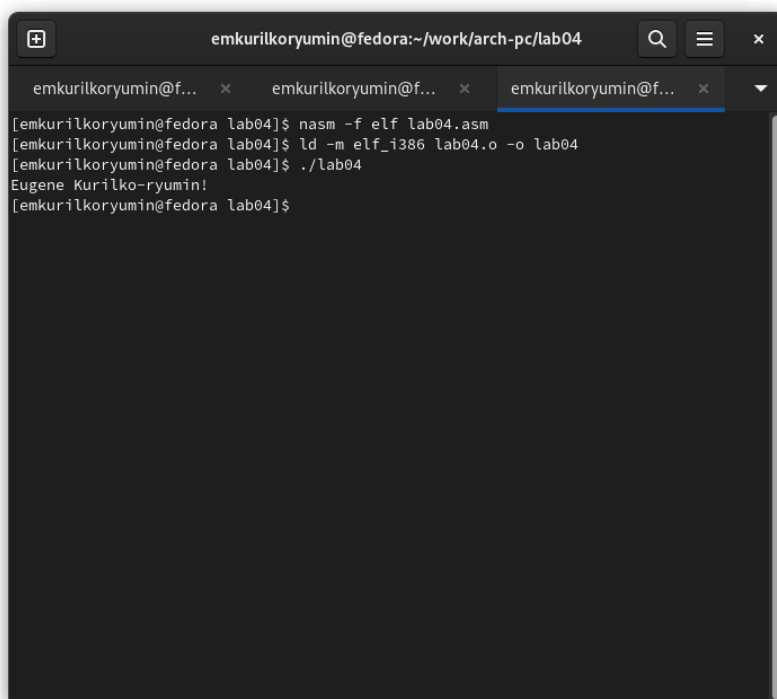
Рис. 4.6: Передача объектного файла на обработку компоновщику

#### 5) Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello. (рис.7).



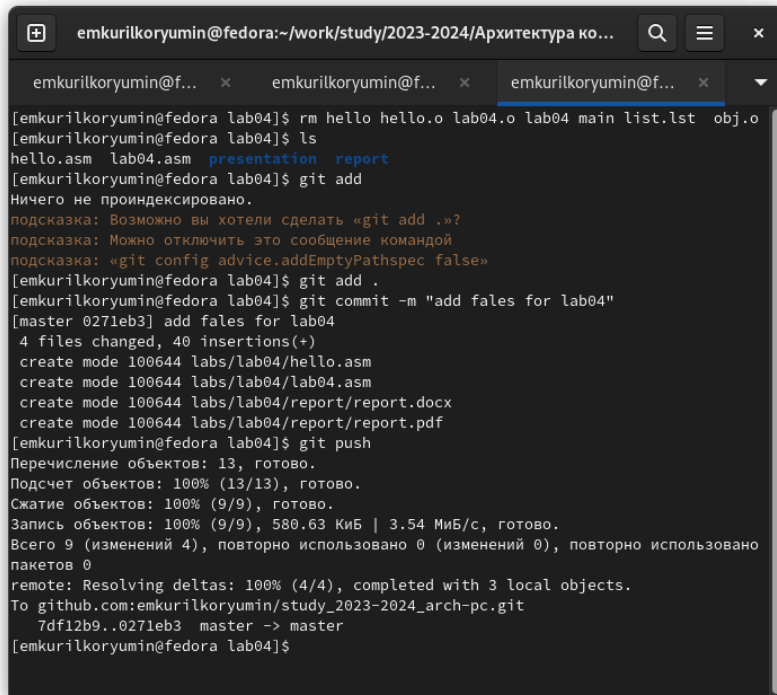
Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия. (рис.9)



```
emkurilkoryumin@fedora:~/work/arch-pc/lab04
[emkurilkoryumin@fedora lab04]$ nasm -f elf lab04.asm
[emkurilkoryumin@fedora lab04]$ ld -m elf_i386 lab04.o -o lab04
[emkurilkoryumin@fedora lab04]$ ./lab04
Eugene Kurilko-ryumin!
[emkurilkoryumin@fedora lab04]$
```

Рис. 4.9: Компиляция, передача компоновщику

Удаляю лишние файлы в текущем каталоге с помощью утилиты `rm`, ведь копии файлов остались в другой директории (рис. 4.10). Также с помощью команд `git add` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4(рис.10)



```
emkurilkoryumin@fedora:~/work/study/2023-2024/Архитектура ко...
[emkurilkoryumin@fedora lab04]$ rm hello hello.o lab04.o lab04 main list.lst obj.o
[emkurilkoryumin@fedora lab04]$ ls
hello.asm lab04.asm presentation report
[emkurilkoryumin@fedora lab04]$ git add
Ничего не проиндексировано.
подсказка: Возможно вы хотели сделать «git add .»?
подсказка: Можно отключить это сообщение командой
подсказка: «git config advice.addEmptyPathsSpec false»
[emkurilkoryumin@fedora lab04]$ git add .
[emkurilkoryumin@fedora lab04]$ git commit -m "add fales for lab04"
[master 0271eb3] add fales for lab04
4 files changed, 40 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab04.asm
create mode 100644 labs/lab04/report/report.docx
create mode 100644 labs/lab04/report/report.pdf
[emkurilkoryumin@fedora lab04]$ git push
Перечисление объектов: 13, готово.
Подсчет объектов: 100% (13/13), готово.
Сжатие объектов: 100% (9/9), готово.
Запись объектов: 100% (9/9), 580.63 КиБ | 3.54 МиБ/с, готово.
Всего 9 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано
пакетов 0
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:emkurilkoryumin/study_2023-2024_arch-pc.git
7df12b9..0271eb3 master -> master
[emkurilkoryumin@fedora lab04]$
```

Рис. 4.10: Отправка файлов

## 5 Выводы

При выполнении лабораторной работы я обрёл практический опыт работы с программами, написанными на ассемблере NASM, конкретнее - освоил процедуры компиляций и сборки.



## Список литературы

[Архитектура ЭВМ]([https://esystem.rudn.ru/pluginfile.php/2089084/mod\\_resource/content/0/Лабораторная%20работа%20№4.%2](https://esystem.rudn.ru/pluginfile.php/2089084/mod_resource/content/0/Лабораторная%20работа%20№4.%2))