

# **Лабораторная работа №4**

**Создание и процесс обработки программ на языке ассемблера NASM**

**Перфилов Александр Константинович | группа: НПИбд 02-23**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Ход лабораторной работы</b>	<b>6</b>
2.1	Программа Hello world! . . . . .	6
2.2	Транслятор NASM . . . . .	7
2.3	Расширенный синтаксис командной строки NASM . . . . .	8
2.4	Компоновщик LD . . . . .	9
2.5	Запуск исполняемого файла . . . . .	10
<b>3</b>	<b>Самостоятельна работа</b>	<b>11</b>
<b>4</b>	<b>Выводы</b>	<b>14</b>

## Список иллюстраций

2.1	Рис 2.1.1: Создание каталога /work/arch-pc/lab04 и переход в него	6
2.2	Рис 2.1.2: Создание текстового файла с помощью команды touch .	6
2.3	Рис 2.1.3: Открытие файла с помощью текстового редактора gedit	6
2.4	Рис 2.1.4: Демонстрация текста в файле . . . . .	7
2.5	Рис 2.2.1: Компиляция текста с помощью команды nasm -f elf hello.asm . . . . .	7
2.6	Рис 2.2.2: Проверка созданного файла . . . . .	8
2.7	Рис 2.3.1: Компиляция исходного файла hello.asm в obj.o . . . . .	8
2.8	Рис 2.3.2: Проверка созданных файлов . . . . .	9
2.9	Рис 2.4.1: Передача объектного файла на обработку компоновщику и проверка . . . . .	9
2.10	Рис 2.4.2: Создание исполняемого файла main и проверка . . . . .	10
2.11	Рис 2.5.1: Запуск исполняемого файла hello с помощью команды ./hello . . . . .	10
3.1	Рис 3.1.1: Копирование файла . . . . .	11
3.2	Рис 3.2.1: Применение команды gedit . . . . .	11
3.3	Рис 3.2.2: Демонстрация изменённого текста . . . . .	12
3.4	Рис 3.3.1: Компиляция файла . . . . .	12
3.5	Рис 3.3.2: Передача объектного файла на обработку компоновщику	12
3.6	Рис 3.3.3: Запуск исполняемого файла lab4 с помощью команды ./lab4	13
3.7	Рис 3.4.1: Копирование файлов hello.asm и lab4.asm . . . . .	13
3.8	Рис 3.4.2: Загрузка файлов на гитхаб . . . . .	13

## Список таблиц

# 1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

## 2 Ход лабораторной работы

### 2.1 Программа Hello world!

Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран. Создадим каталог для работы с программами на языке ассемблера NASM, перейдём в него:

```
perfilov@akperfilov:~$ mkdir -p ~/work/arch-pc/lab04
perfilov@akperfilov:~$ cd ~/work/arch-pc/lab04
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.1: Рис 2.1.1: Создание каталога /work/arch-pc/lab04 и переход в него

Создадим текстовый файл с именем **hello.asm**

```
perfilov@akperfilov:~/work/arch-pc/lab04$ touch hello.asm
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.2: Рис 2.1.2: Создание текстового файла с помощью команды touch

Откроем этот файл с помощью любого текстового редактора, например, gedit

```
perfilov@akperfilov:~/work/arch-pc/lab04$ gedit hello.asm
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.3: Рис 2.1.3: Открытие файла с помощью текстового редактора gedit

Введём в нём следующий текст:

```
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 SECTION .text                ; Начало секции кода
8     GLOBAL _start
9
10 _start:                     ; Точка входа в программу
11     mov eax,4                ; Системный вызов для записи (sys_write)
12     mov ebx,1                ; Описатель файла '1' - стандартный вывод
13     mov ecx,hello            ; Адрес строки hello в ecx
14     mov edx,helloLen         ; Размер строки hello
15     int 80h                  ; Вызов ядра
16
17     mov eax,1                ; Системный вызов для выхода (sys_exit)
18     mov ebx,0                ; Выход с кодом возврата '0' (без ошибок)
19     int 80h                  ; Вызов ядра
```

Рис. 2.4: Рис 2.1.4: Демонстрация текста в файле

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.

## 2.2 Транслятор NASM

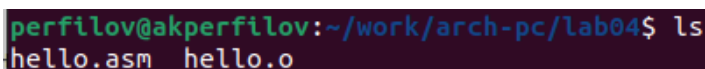
NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать:

```
perfilov@akperfilov:~/work/arch-pc/lab04$ nasm -f elf hello.asm
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.5: Рис 2.2.1: Компиляция текста с помощью команды `nasm -f elf hello.asm`

Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла *hello.asm* в объектный код, который запишется в файл *hello.o*. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появятся сообщения об ошибках или предупреждения.

С помощью команды *ls* проверим, что объектный файл был создан:



```
perfilov@akperfilov:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Рис. 2.6: Рис 2.2.2: Проверка созданного файла

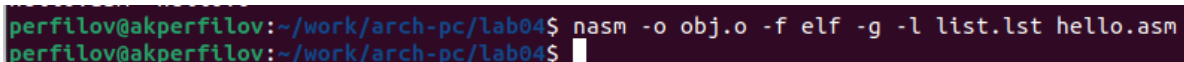
NASM не запускают без параметров. Ключ *-f* указывает транслятору, что требуется создать бинарные файлы в формате **ELF**. Следует отметить, что формат **elf64** позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто **elf**. NASM всегда создаёт выходные файлы в **текущем** каталоге.

## 2.3 Расширенный синтаксис командной строки NASM

Полный вариант командной строки *nasm* выглядит следующим образом:

*nasm* [-@ косвенный\_файл\_настроек] [-o объектный\_файл] [-f формат\_объектного\_файла] [-l листинг] [параметры...] [-] исходный\_файл

Выполним следующую команду:



```
perfilov@akperfilov:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.7: Рис 2.3.1: Компиляция исходного файла *hello.asm* в *obj.o*

Данная команда скомпилирует исходный файл *hello.asm* в *obj.o* (опция *-o* позволяет задать имя объектного файла, в данном случае *obj.o*), при этом формат



выходного файла будет *elf*, и в него будут включены символы для отладки (опция *-g*), кроме того, будет создан файл листинга *list.lst* (опция *-l*).

С помощью команды *ls* проверим, что файлы были созданы:

```
perfilov@akperfilov:~/work/arch-pc/lab04$ ls
hello.asm hello.o list.lst obj.o
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 2.8: Рис 2.3.2: Проверка созданных файлов

Для более подробной информации см. *man nasm*. Для получения списка форматов объектного файла см. *nasm -hf*.

## 2.4 Компоновщик LD

Как видно из схемы в лаб. работе№4 (рис. 4.3), чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику. Проверим, что файл был создан:

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

Рис. 2.9: Рис 2.4.1: Передача объектного файла на обработку компоновщику и проверка

Компоновщик *ld* не предполагает по умолчанию расширений для файлов, но принято использовать следующие расширения:

- *o* – для объектных файлов;
- без расширения – для исполняемых файлов;
- *tar* – для файлов схемы программы;
- *lib* – для библиотек.

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

Выполним следующую команду:

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

Рис. 2.10: Рис 2.4.2: Создание исполняемого файла main и проверка

Объектный файл obj.o был передан на обработку компоновщику для создания исполняемого файла main.

Формат командной строки LD можно увидеть, набрав *ld -help*. Для получения более подробной информации см. *man ld*.

## 2.5 Запуск исполняемого файла

Запустим на выполнение созданный исполняемый файл, находящийся в текущем каталоге, с помощью след. команды:

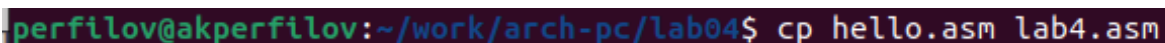
```
perfilov@akperfilov:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 2.11: Рис 2.5.1: Запуск исполняемого файла hello с помощью команды ./hello

### 3 Самостоятельна работа

*Задание№1 В каталоге ~/work/arch-pc/lab04 с помощью команды cp создайте копию файла hello.asm с именем lab4.asm*

Создадим копию файла hello.asm с именем lab4.asm

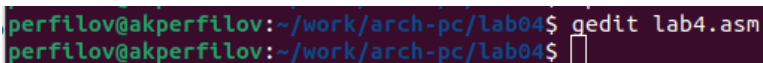


```
perfilov@akperfilov:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
```

Рис. 3.1: Рис 3.1.1: Копирование файла

*Задание№2 С помощью любого текстового редактора внесите изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с вашими фамилией и именем.*

С помощью редактора markdown внесём изменения в текст в файле lab4.asm



```
perfilov@akperfilov:~/work/arch-pc/lab04$ gedit lab4.asm
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 3.2: Рис 3.2.1: Применение команды gedit

```

1 ; lab4.asm
2 SECTION .data                ; Начало секции данных
3     lab4:DB 'Перфилов Александр',10 ; 'Перфилов Александр' плюс
4                                     ; символ перевода строки
5     lab4Len:EQU $-lab4        ; Длина строки lab4
6
7 SECTION .text                ; Начало секции кода
8     GLOBAL _start
9
10 _start:                      ; Точка входа в программу
11     mov eax,4                ; Системный вызов для записи (sys_write)
12     mov ebx,1                ; Описатель файла '1' - стандартный вывод
13     mov ecx,lab4             ; Адрес строки lab4 в ecx
14     mov edx,lab4Len          ; Размер строки lab4
15     int 80h                  ; Вызов ядра
16
17     mov eax,1                ; Системный вызов для выхода (sys_exit)
18     mov ebx,0                ; Выход с кодом возврата '0' (без ошибок)
19     int 80h                  ; Вызов ядра

```

Рис. 3.3: Рис 3.2.2: Демонстрация изменённого текста

*Задание№3 Оттранслируйте полученный текст программы lab4.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.*

Скомпилируем файл lab4.asm

```

perfilov@akperfilov:~/work/arch-pc/lab04$ nasm -f elf lab4.asm

```

Рис. 3.4: Рис 3.3.1: Компиляция файла

Передадим объектный файл lab4.o на обработку компоновщику

```

perfilov@akperfilov:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
perfilov@akperfilov:~/work/arch-pc/lab04$

```

Рис. 3.5: Рис 3.3.2: Передача объектного файла на обработку компоновщику

Запустим получившийся исполняемый файл lab4

```
perfilov@akperfilov:~/work/arch-pc/lab04$ ./lab4
Перфилов Александр
```

Рис. 3.6: Рис 3.3.3: Запуск исполняемого файла lab4 с помощью команды ./lab4

*Задание №4 Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузите файлы на Github.*

Скопируем файлы в локальный репозиторий

```
perfilov@akperfilov:~/work/arch-pc/lab04$ cp hello.asm /home/perfilov/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
perfilov@akperfilov:~/work/arch-pc/lab04$ cp lab4.asm /home/perfilov/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
perfilov@akperfilov:~/work/arch-pc/lab04$
```

Рис. 3.7: Рис 3.4.1: Копирование файлов hello.asm и lab4.asm

Загрузим файлы на Github

```
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git add .
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git commit -am 'feat(main): add files lab-4'
[master fb3e864] feat(main): add files lab-4
22 files changed, 268 insertions(+), 119 deletions(-)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
delete mode 100644 labs/lab04/report/image/placeimg_800_600_tech.jpg
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.1.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.1.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.1.3.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.1.4.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.2.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.2.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.3.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.3.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.4.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.4.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 2.5.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.1.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.2.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.2.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.3.1.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.3.2.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.3.3.png"
create mode 100644 "labs/lab04/report/image/\320\240\320\270\321\201 3.4.1.png"
rewrite labs/lab04/report/report.md (72%)
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git push
Перечисление объектов: 33, готово.
Подсчет объектов: 100% (33/33), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (27/27), готово.
Запись объектов: 100% (27/27), 332.36 Киб | 2.75 Миб/с, готово.
Всего 27 (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:AlexanderPERfilovKonstantinivich/study_2023-2024_arh-pc.git
2389aaa..fb3e864 master -> master
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$
```

Рис. 3.8: Рис 3.4.2: Загрузка файлов на гитхаб

## 4 Выводы

Я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.