

# **Отчет по лабораторной работе №4**

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

Богданюк Анна Васильевна

# 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>
<b>5</b>	<b>Выводы</b>	<b>14</b>
	<b>Список литературы</b>	<b>15</b>

## **Список таблиц**

## Список иллюстраций

4.1	Создаю каталог . . . . .	9
4.2	Перехожу в каталог . . . . .	9
4.3	Создаю hello.asm . . . . .	9
4.4	Использую nano . . . . .	9
4.5	Ввожу текст . . . . .	10
4.6	Компилирую текст . . . . .	10
4.7	Файлы в каталоге . . . . .	10
4.8	Компилирую и создаю файлы . . . . .	11
4.9	Файлы в каталоге . . . . .	11
4.10	Передаю файл на обработку . . . . .	11
4.11	Файлы в каталоге . . . . .	11
4.12	Создаю исполняемый файл . . . . .	11
4.13	Запускаю файл . . . . .	12
4.14	Копирую файл . . . . .	12
4.15	Измененный текст . . . . .	12
4.16	Команда . . . . .	12
4.17	Выполняю компоновку . . . . .	13
4.18	Результат . . . . .	13
4.19	Копирую файлы . . . . .	13
4.20	Загружаю файлы . . . . .	13

# 1 Цель работы

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

## **2 Задание**

- 1.Выполнение лабораторной работы
- 2.Задания 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. 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. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как 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: • для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM); • для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

В процессе создания ассемблерной программы можно выделить четыре шага:

- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`.
- Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`.
- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение `map`.
- Запуск программы. Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.



## 4 Выполнение лабораторной работы

### 1.1. Программа Hello world!

Создаю каталог для работы с программами на языке ассемблера NASM (рис. 4.1).

```
avbogdanyuk@DESKTOP-36B2AJH:~$ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.1: Создаю каталог

Перехожу в созданный каталог (рис. 4.2).

```
avbogdanyuk@DESKTOP-36B2AJH:~$ cd ~/work/arch-pc/lab04
```

Рис. 4.2: Перехожу в каталог

Создаю текстовый файл с именем hello.asm (рис. 4.3).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ touch hello.asm
```

Рис. 4.3: Создаю hello.asm

Открываю этот файл с помощью nano (рис. 4.4).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ nano hello.asm
```

Рис. 4.4: Использую nano

Ввожу следующий текст (рис. 4.5).

```
avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab04
GNU nano 6.2
; 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. 4.5: Ввожу текст

## 1.2. Транслятор NASM

Компилирую текст в объектный код (рис. 4.6).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ nasm -f elf hello.asm
```

Рис. 4.6: Компилирую текст

Проверяю с помощью ls, что был создан объектный файл (рис. 4.7).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Рис. 4.7: Файлы в каталоге

## 1.3. Расширенный синтаксис командной строки NASM

Компилирую файл hello.asm в obj.o, а также создаю файл листинга list.lst (рис. 4.8).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
```

Рис. 4.8: Компилирую и создаю файлы

Проверяю, что файлы были созданы (рис. 4.9).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ls  
hello.asm hello.o list.lst obj.o
```

Рис. 4.9: Файлы в каталоге

#### 1.4. Компоновщик LD

Передаю объектный файл на обработку компоновщику(рис. 4.10).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
```

Рис. 4.10: Передаю файл на обработку

Проверяю, что файл был создан (рис. 4.11).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ls  
hello hello.asm hello.o list.lst obj.o
```

Рис. 4.11: Файлы в каталоге

Выполняю следующую команду. Созданный файл будет называться main, объектный файл, из которого собран этот исполняемый файл, называется obj.o (рис. 4.12).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
```

Рис. 4.12: Создаю исполняемый файл

#### 1.5. Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл (рис. 4.13).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 4.13: Запускаю файл

## 2. Задание для самостоятельной работы

Копирую файл hello.asm с именем lab4.asm (рис. 4.14).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
```

Рис. 4.14: Копирую файл

Изменяю текст, чтобы вместо 'Hello World!', было 'Богданюк Анна' (рис. 4.15).

```
avbogdanyuk@DESKTOP-36B2AJH: ~/work/arch-pc/lab04
GNU nano 6.2
; hello.asm
SECTION .data ; Начало секции данных
hello: DB 'Богданюк Анна',10 ; 'Богданюк Анна' 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. 4.15: Измененный текст

Оттранслирую полученный файл в объектный файл и проверяю, что файл был создан (рис. 4.16).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
```

Рис. 4.16: Команда

Выполняю компоновку объектного файла и проверяю, что файл был создан(рис. 4.17).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4  lab4.asm  lab4.o  list.lst  main  obj.o
```

Рис. 4.17: Выполняю компоновку

Запускаю получившийся исполняемый файл (рис. 4.18).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ ./lab4
Богданюк Анна
```

Рис. 4.18: Результат

Копирую файл hello.asm и lab4.asm в мой локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ (рис. 4.19).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/
avbogdanyuk@DESKTOP-36B2AJH:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/
```

Рис. 4.19: Копирую файлы

Загружаю файлы на GitHub (рис. 4.20).

```
avbogdanyuk@DESKTOP-36B2AJH:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git add .
avbogdanyuk@DESKTOP-36B2AJH:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git commit -am 'adding new files'
[master 9e2b175] adding new files
23 files changed, 34 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 135856.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 135912.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 135930.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 135950.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141016.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141108.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141227.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141402.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141529.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141547.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141715.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 141739.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 142006.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 142104.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 142318.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 142451.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 142830.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 143117.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 143158.png
create mode 100644 labs/lab04/report/image/Screenshot 2023-10-18 143615.png
create mode 100644 labs/lab04/report/image/TEKCT.png
avbogdanyuk@DESKTOP-36B2AJH:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git push
Enumerating objects: 34, done.
Counting objects: 100% (34/34), done.
Delta compression using up to 12 threads
Compressing objects: 100% (29/29), done.
Writing objects: 100% (29/29), 151.29 KiB | 1.24 MiB/s, done.
Total 29 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:avbogdanyuk/study_2023-2024_arhpc.git
3b6fc0e..9e2b175 master -> master
```

Рис. 4.20: Загружаю файлы

## 5 Выводы

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

## **Список литературы**