

# **Отчёт по лабораторной работе №4**

Петлин Артём Дмитриевич

# 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>10</b>
<b>5</b>	<b>Задание для самостоятельной работы</b>	<b>12</b>
<b>6</b>	<b>Выводы</b>	<b>14</b>
	<b>Список литературы</b>	<b>15</b>

## **Список иллюстраций**

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. В каталоге `~/work/arch-rc/lab04` с помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm`
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.
3. Оттранслируйте полученный текст программы `lab4.asm` в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.
4. Скопируйте файлы `hello.asm` и `lab4.asm` в Ваш локальный репозиторий в каталог `~/work/study/2023-2024/“Архитектура компьютера”/arch-rc/labs/lab04/`. Загрузите файлы на Github.

### 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.

Более подробно о языке ассемблера см., например, в [10].

В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler) [7; 12; 14].

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.е. 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 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

```
petlin@fedora:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$
```

Переходим в нужный каталог

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ touch hello.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm presentation report
```

Создаём текстовый файл с именем hello.asm

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ gedit hello.asm
```



```
1 SECTION .data
2     hello:      db "Hello, world!",0xa
3     helloLen:   equ $ - hello
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, hello
11    mov edx, helloLen
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80
```

Открываем этот файл с помощью gedit и введите в него текст, представленный в задании.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ nasm -f elf hello.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm hello.o presentation report
```

преобразовываем текст программы из файла hello.asm в объектный код, который записался в файл hello.o. С помощью команды ls проверяем преобразование.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm hello.o list.lst obj.o presentation report
```

Выполните следующую команду, которая 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, был создан файл листинга list.lst (опция -l).

С помощью команды ls проверяем, что файлы были созданы.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ld -m elf_i386 hello.o -o hello
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello hello.asm hello.o list.lst obj.o presentation report
```

Передаём объектный файл необходимо на обработку компоновщику. С помощью команды ls проверяем, что исполняемый файл hello был создан.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ld -m elf_i386 obj.o -o main
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
```

Выполняем следующую команду. Получаем исполняемый файл “main”. Объектный файл, из которого собран этот исполняемый файл, имеет имя “obj.o”.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ./hello
Hello, world!
```

Запускаем на выполнение созданный исполняемый файл, находящийся в текущем каталоге с помощью команды “./hello”.

## 5 Задание для самостоятельной работы

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ cp hello.asm lab4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o  presentation  report
```

с помощью команды `cp` создаём копию файла `hello.asm` с именем `lab4.asm`



```
lab4.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04
Save

1 SECTION .data
2     hello:      db "Петлин Артём",0xa
3     helloLen:   equ $ - hello
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, hello
11    mov edx, helloLen
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80
```

С помощью `gedit` внесим изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с моими фамилией и именем.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ nasm -f elf lab4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o  presentation  report

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ld -m elf_i386 lab4.o -o lab4
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello  hello.asm  hello.o  lab4  lab4.asm  lab4.o  list.lst  main  obj.o  presentation  report

petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ./lab4
Петлин Артём
```

Оттранслируем полученный текст программы `lab4.asm` в объектный файл. Выполняем компоновку объектного файла и запускаем получившийся исполняемый файл.

```

petlin@fedora: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git add .
petlin@fedora: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git commit -am 'feat(main): nasm'
[master 4db0c2a] feat(main): nasm
10 files changed, 50 insertions(+), 8 deletions(-)
create mode 100755 labs/lab04/hello
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/hello.o
create mode 100755 labs/lab04/lab4
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/lab4.o
create mode 100644 labs/lab04/list.lst
create mode 100755 labs/lab04/main
create mode 100644 labs/lab04/obj.o
petlin@fedora: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git push
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 6 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 3.31 KiB | 3.31 MiB/s, done.
Total 15 (delta 8), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (8/8), completed with 4 local objects.
To github.com:travesa/study_2023-2024_arh-pc.git
d97c46c..4db0c2a master -> master

```

Files

master
+
Q

Go to file
t

>
config

v
labs

>
lab01

>
lab02

>
lab03

v
lab04

>
presentation

>
report

hello

hello.asm

hello.o

lab4

lab4.asm

lab4.o

list.lst

main

obj.o

>
lab05

>
lab06

study\_2023-2024\_arh-pc / labs / lab04 /
Add file
...

Артём Петлин
feat(main): nasm
4db0c2a · 1 minute ago
History

Name	Last commit message	Last commit date
..		
presentation	feat(main): make course structure	2 weeks ago
report	feat(main): nasm	1 minute ago
hello	feat(main): nasm	1 minute ago
hello.asm	feat(main): nasm	1 minute ago
hello.o	feat(main): nasm	1 minute ago
lab4	feat(main): nasm	1 minute ago
lab4.asm	feat(main): nasm	1 minute ago
lab4.o	feat(main): nasm	1 minute ago
list.lst	feat(main): nasm	1 minute ago
main	feat(main): nasm	1 minute ago
obj.o	feat(main): nasm	1 minute ago

Отправляем файлы на Github.

## 6 Выводы

Мы освоили процедуры компиляции и сборки программ, написанных на ассемблере NASM.

# 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).