

# **Отчет по лабораторной работе №4**

**Дисциплина: Архитектура компьютера**

Белоусова Елизавета Валетиновна

# Содержание

## **Список иллюстраций**

# Список таблиц

## 1 Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

## 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

Основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора входят следующие устройства: - арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; - устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; - регистры — сверхбыстрая оперативная

память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между регистрами и памятью, преобразование (арифметические или логические операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита. В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ): - RAX, RCX, RDX, RBX, RSI, RDI — 64-битные - EAX, ECX, EDX, EBX, ESI, EDI — 32-битные - AX, CX, DX, BX, SI, DI — 16-битные - AH, AL, CH, CL, DH, DL, BH, BL — 8-битные Другим важным узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. - устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой. В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает, что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы. Коды

команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. Он заключается в следующем: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. переход к следующей команде. Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. 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.

#### 4 Выполнение лабораторной работы

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 1).

```
evbelousova@dk2n25 ~ $ cd work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04
```

Рис. 1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 2).

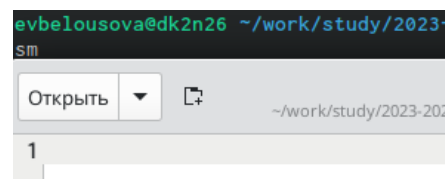
```
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ touch hello.asm
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

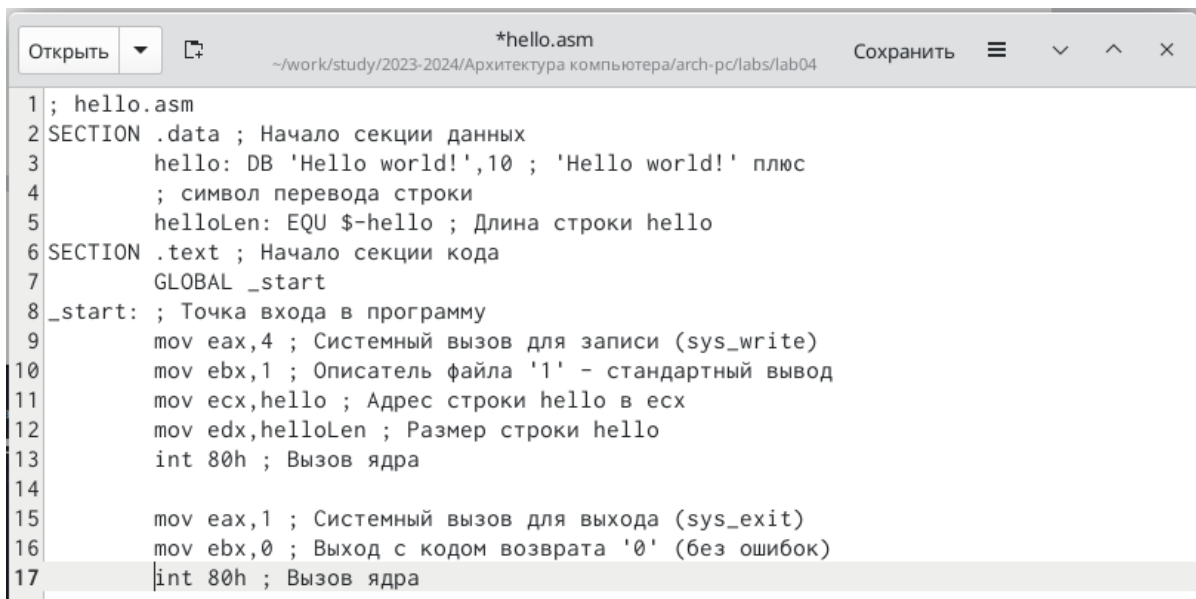
Рис. 2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `gedit` (рис. 3).

Рис. 3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4).

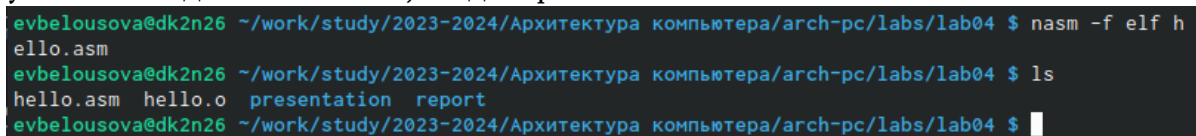




```
1; hello.asm
2SECTION .data ; Начало секции данных
3    hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4           ; символ перевода строки
5    helloLen: EQU $-hello ; Длина строки hello
6SECTION .text ; Начало секции кода
7    GLOBAL _start
8_start: ; Точка входа в программу
9        mov eax,4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

Рис. 4: Заполнение файла

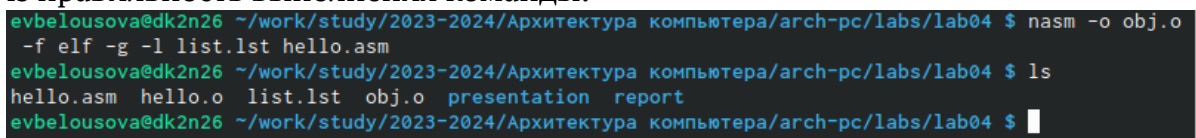
Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ nasm -f elf h
ello.asm
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello.asm hello.o presentation report
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 5: Компиляция текста программы

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` (рис. 6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.



```
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ nasm -o obj.o
-f elf -g -l list.lst hello.asm
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello.asm hello.o list.lst obj.o presentation report
evbelousova@dk2n26 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 6: Компиляция текста программы

Передаю объектный файл `hello.o` на обработку компоновщику LD, чтобы по-

лучить исполняемый файл hello (рис. 7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ld -m elf_i386 hello.o -o hello
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello hello.asm hello.o list.lst obj.o presentation report
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 8). Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

```
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ld -m elf_i386 obj.o -o main
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello hello.asm hello.o list.lst main obj.o presentation report
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 8: Передача объектного файла на обработку компоновщику

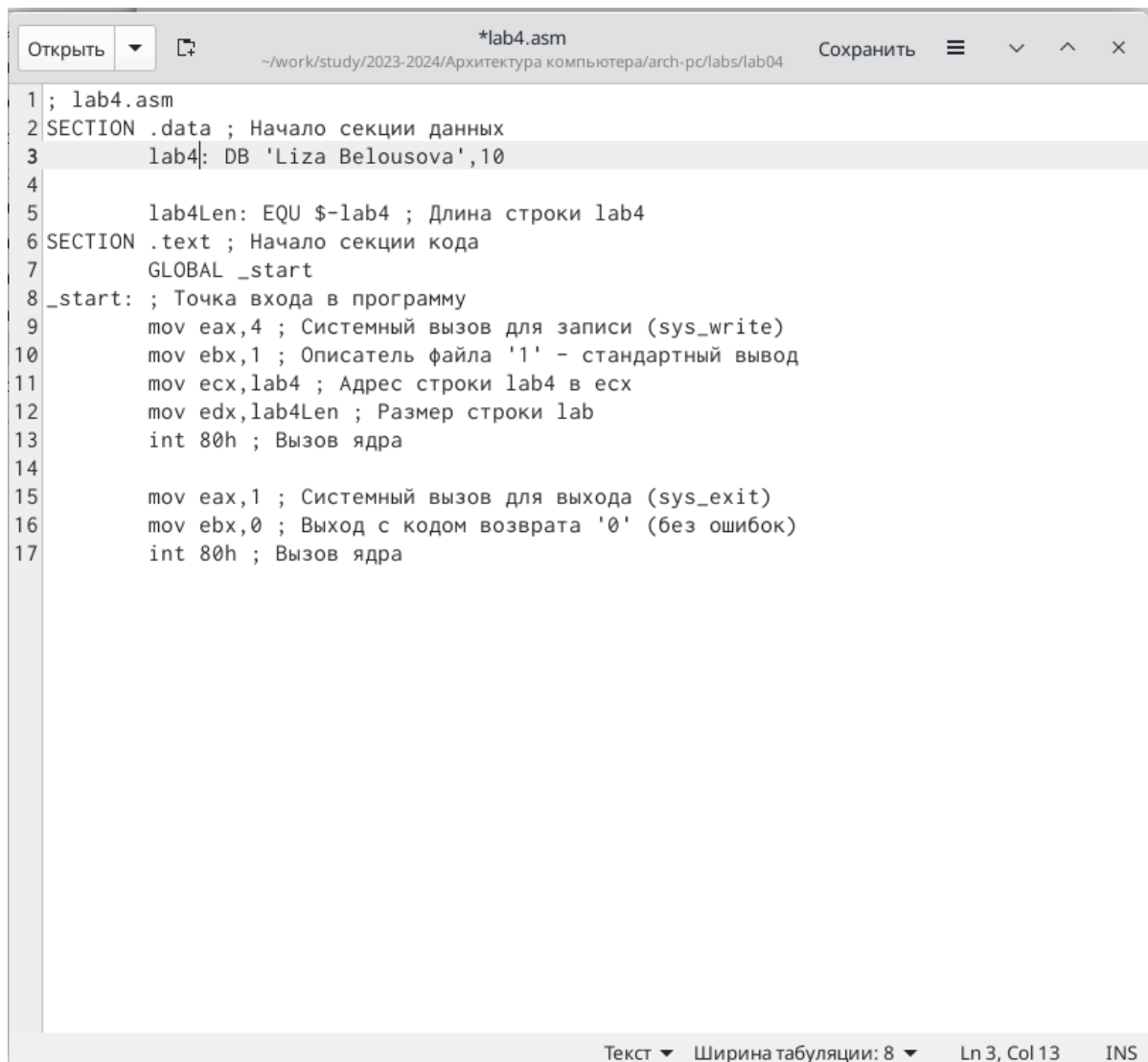
Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

```
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ./hello
Hello world!
evbelousova@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 9: Запуск исполняемого файла

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm. С помощью текстового редактора mousepad открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 10).

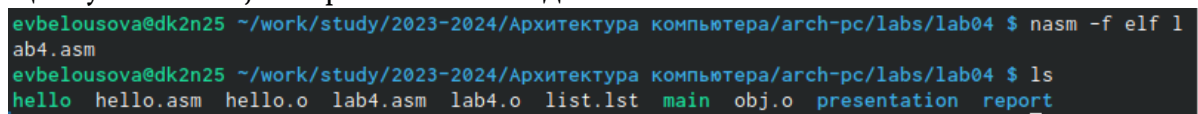




```
1 ; lab4.asm
2 SECTION .data ; Начало секции данных
3     lab4: DB 'Liza Belousova',10
4
5     lab4Len: EQU $-lab4 ; Длина строки lab4
6 SECTION .text ; Начало секции кода
7     GLOBAL _start
8 _start: ; Точка входа в программу
9     mov eax,4 ; Системный вызов для записи (sys_write)
10    mov ebx,1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

Рис. 10: Изменение программы

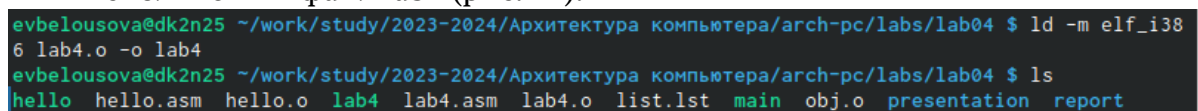
Компилирую текст программы в объектный файл (рис. 11). Проверяю с помощью утилиты ls, что файл lab4.o создан.



```
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ nasm -f elf lab4.asm
evbelousova@dk2n25 ~/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
```

Рис. 11: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 12).



```
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ld -m elf_i386 lab4.o -o lab4
evbelousova@dk2n25 ~/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
```

Рис. 12: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 13).

```
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ./lab4
Liza Belousova
```

Рис. 13: Запуск исполняемого файла

С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. 14).

```
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git add .
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git commit -m
"Add files for lab04"
[master b2abd34] Add files for lab04
9 files changed, 52 insertions(+)
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
```

Рис. 14: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды `git push` (рис. 15).

```
evbelousova@dk2n25 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git push
Перечисление объектов: 35, готово.
Подсчет объектов: 100% (35/35), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (29/29), готово.
Запись объектов: 100% (29/29), 337.13 КиБ | 21.07 МиБ/с, готово.
Всего 29 (изменений 9), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (9/9), completed with 2 local objects.
To github.com:evbelousova/study_2023-2024_arc-pc.git
 4f15e20..b2abd34 master -> master
```

Рис. 15: Отправка файлов

## 5 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

## 6 Список литературы

Архитектура ЭВМ