

Отчет по лабораторной работы №4

Архитектура компьютеров и Операционные системы

Ван Сихэм Франклин О' Нил Джон (Миша)

13/10/2023

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Основные принципы работы компьютера	6
3	Ассемблер и язык ассемблера	9
4	Процесс создания и обработки программы на языке ассем- блера	11
5	Порядок выполнения лабораторной работы	13
5.1	Программа Hello world!	13
5.2	Транслятор NASM	16
5.3	Расширенный синтаксис командной строки NASM	17
5.4	Компоновщик LD	18
5.5	Запуск исполняемого файла	20
5.6	Задание для самостоятельной работы	20
6	Заключение	24

Список иллюстраций

2.1	‘Структурная схема ЭВМ’	7
2.2	‘64-битный регистр процессора ‘RAX’’	8
4.1	Процесс создания ассемблерной программы	11
5.1	Команды для создания текстового файла ‘hello’ в ассемблер!	14
5.2	Формуловка синтаксиса в ассемблер редактор	15
5.3	Формуловка синтаксиса чтобы получить вывод ‘Hello World’ в ассемблер	16
5.4	NASM превращает текст программы в объектный код	17
5.5	Расширенный синтаксис командной строки NASM	18
5.6	Команда ld -m elf_x86_64 (для arch-linux)	19
5.7	Команда ld -m i386pe	19
5.8	Вывод созданного файла	20
5.9	С помощью команды ‘ср’ создан копию файла	21
5.10	Текстовый редактор nasm	21
5.11	Компоновка файла lab4	22
5.12	Файлы hello.asm и lab4.asm скопированы в каталог	23

Список таблиц

1 Цель работы

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

2 Теоретическое введение

2.1 Основные принципы работы компьютера

Также, регистры процессора могут использоваться для хранения адресов памяти, указателей на данные и инструкции, а также для выполнения операций сдвига и логических операций.

Память в ЭВМ используется для хранения данных и программ. Она делится на оперативную память (ОП) и постоянную память (ПП). Оперативная память используется для временного хранения данных и программ, которые активно используются процессором. Постоянная память используется для хранения данных и программ на постоянной основе.

Периферийные устройства включают в себя устройства ввода и вывода. Устройства ввода используются для передачи данных в компьютер, например клавиатура и мышь. Устройства вывода используются для вывода данных из компьютера, например монитор и принтер. Также существуют устройства, которые могут выполнять как функции ввода, так и вывода, например жесткий диск.

Взаимодействие между центральным процессором, памятью и периферийными устройствами осуществляется через общую шину. Шина представляет собой набор проводников, по которым передается информация между устройствами. Шина может быть разделена на несколько частей, например шина данных, шина адреса и шина управления.

В целом, основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства, которые взаимодействуют через общую шину.

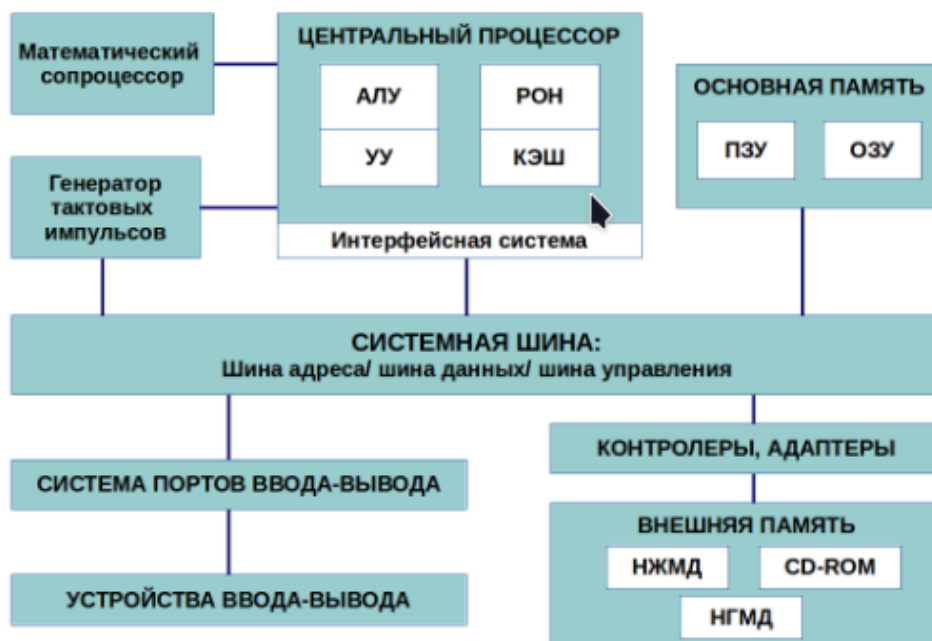


Рис. 2.1: 'Структурная схема ЭВМ'

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 (половинки 16-битных регистров). Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX.

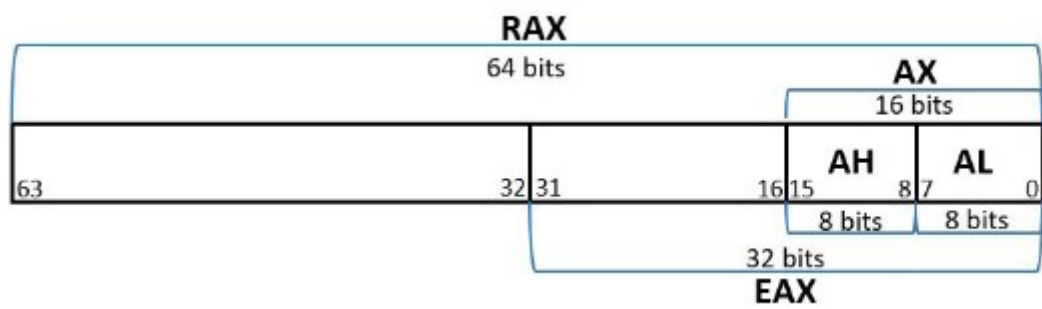


Рис. 2.2: '64-битный регистр процессора 'RAX''

3 Ассемблер и язык ассемблера

Язык ассемблера (Assembler) - это низкоуровневый язык программирования, который используется для написания программ, близких к машинному коду. Он позволяет программисту работать непосредственно с регистрами процессора, адресами памяти и инструкциями процессора.

NASM (Netwide Assembler) - это один из популярных ассемблеров, который поддерживает различные архитектуры процессоров, включая x86 и x86-64. Он предоставляет широкий набор инструкций и возможностей для работы с памятью, регистрами и периферийными устройствами.

Основные особенности языка ассемблера NASM:

1. Синтаксис: NASM использует синтаксис Intel, который характеризуется использованием мнемоник инструкций, операндов и директив для определения символов и констант.
2. Регистры: NASM предоставляет доступ к регистрам процессора, таким как общего назначения (например, EAX, EBX), указателей (например, ESP, EBP), индексных (например, ESI, EDI) и флагового (например, EFLAGS).
3. Макросы: NASM поддерживает использование макросов, которые позволяют определить повторяющиеся фрагменты кода и использовать их в различных местах программы.

4. Директивы: NASM предоставляет различные директивы для управления процессом сборки программы, такие как директивы определения символов, выравнивания памяти и включения внешних файлов.
5. Операнды: NASM поддерживает различные типы операндов, включая регистры, адреса памяти, константы и метки.
6. Ввод-вывод: NASM предоставляет инструкции для работы с периферийными устройствами, такими как чтение и запись данных из/в портов ввода-вывода.

Язык ассемблера NASM позволяет программисту иметь полный контроль над процессором и памятью компьютера, что делает его мощным инструментом для разработки низкоуровневых программ и оптимизации кода. Однако, из-за своей низкоуровневости, он требует от программиста глубокого понимания аппаратного обеспечения и особенностей конкретной архитектуры процессора.

4 Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы



Рис. 4.1: Процесс создания ассемблерной программы

Процесс создания и обработки программы на языке ассемблера включает следующие шаги:

1. Написание исходного кода: Программист пишет исходный код программы на языке ассемблера, используя мнемоники инструкций, операнды и директивы.
2. Ассемблирование: Исходный код программы передается ассемблеру (например, NASM), который преобразует его в машинный код, состоящий из набора инструкций процессора.
3. Связывание: Если программа использует внешние функции или библиотеки, необходимо выполнить связывание, чтобы объединить машинный код программы с кодом этих функций или библиотек.
4. Создание исполняемого файла: После связывания создается исполняемый файл, который может быть запущен на целевой системе.
5. Тестирование и отладка: Исполняемый файл тестируется и отлаживается для обнаружения и исправления ошибок и неправильного поведения программы.
6. Оптимизация: При необходимости программу можно оптимизировать для улучшения ее производительности или уменьшения размера исполняемого файла.
7. Развертывание: Исполняемый файл программы развертывается на целевой системе и запускается для выполнения заданной функциональности.

Весь этот процесс требует от программиста глубокого понимания аппаратного обеспечения и особенностей конкретной архитектуры процессора, чтобы эффективно использовать возможности языка ассемблера и создавать оптимизированный и надежный код.

5 Порядок выполнения лабораторной работы

5.1 Программа Hello world!

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

```
mkdir -p ~/work/arch-pc/lab04
```

Перейдите в созданный каталог:

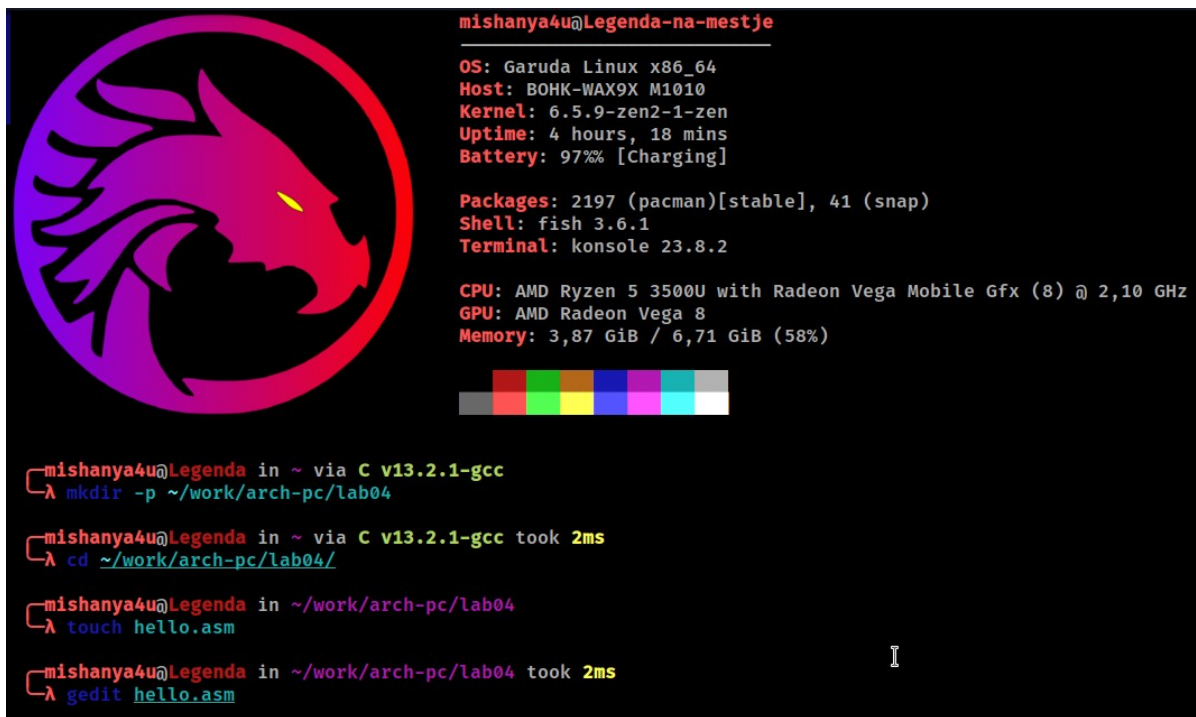
```
cd ~/work/arch-pc/lab04
```

Создайте текстовый файл с именем hello.asm:

```
touch hello.asm
```

Откройте этот файл с помощью любого текстового редактора, например, gedit:

```
gedit hello.asm
```



The image shows a terminal window with a black background. On the left is a circular logo with a red and yellow dragon head. The terminal text is as follows:

```
mishanya4u@Legenda-na-mestje
OS: Garuda Linux x86_64
Host: BOHK-WAX9X M1010
Kernel: 6.5.9-zen2-1-zen
Uptime: 4 hours, 18 mins
Battery: 97% [Charging]

Packages: 2197 (pacman)[stable], 41 (snap)
Shell: fish 3.6.1
Terminal: konsole 23.8.2

CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (8) @ 2,10 GHz
GPU: AMD Radeon Vega 8
Memory: 3,87 GiB / 6,71 GiB (58%)
```

Below the system info is a color calibration bar. The terminal history shows the following commands and their outputs:

```
mishanya4u@Legenda in ~ via C v13.2.1-gcc
λ mkdir -p ~/work/arch-pc/lab04

mishanya4u@Legenda in ~ via C v13.2.1-gcc took 2ms
λ cd ~/work/arch-pc/lab04/

mishanya4u@Legenda in ~/work/arch-pc/lab04
λ touch hello.asm

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 2ms
λ gedit hello.asm
```

A cursor is visible at the end of the last command line.

Рис. 5.1: Команды для создания текстового файла 'hello' в ассемблер!

И введите в него следующий текст:

```

; 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. 5.2: Формуловка синтаксиса в ассемблер редактор

```
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 ; писатель файла '1' - стандартный вывод
13     mov ecx,hello ; адрес строки hello в ecx
14     mov edx,helloLen ; адрес строки hello
15     int 80h ; Вызов ядра
16
17     mov eax,1 ; Системный вызов для вывода (sys_exit)
18     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19     int 80h ; Вызов ядра
```

Рис. 5.3: Формуловка синтаксиса чтобы получить вывод 'Hello World' в ассемблер

5.2 Транслятор NASM

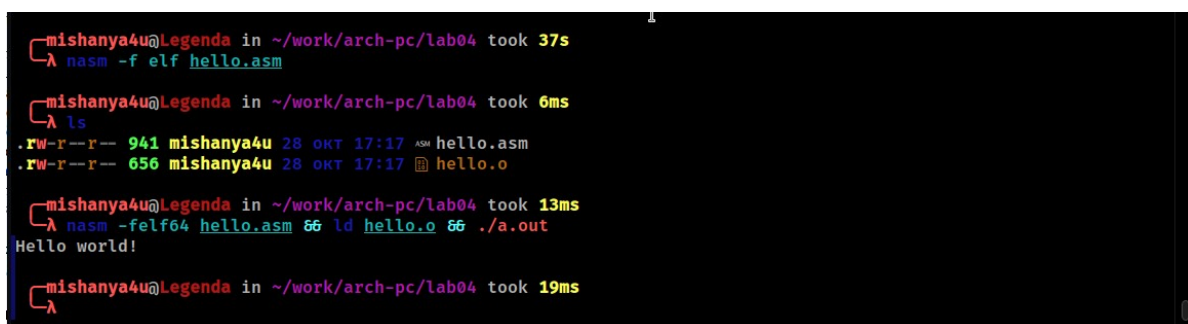
NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать:

```
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` проверьте, что объектный файл был создан. Какое имя имеет объектный файл? - `main`

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.



```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 37s
λ nasm -f elf hello.asm

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 6ms
λ ls
-rw-r--r-- 941 mishanya4u 28 окт 17:17 asm hello.asm
-rw-r--r-- 656 mishanya4u 28 окт 17:17 obj hello.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 13ms
λ nasm -felf64 hello.asm && ld hello.o && ./a.out
Hello world!

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 19ms
λ
```

Рис. 5.4: NASM превращает текст программы в объектный код

5.3 Расширенный синтаксис командной строки

NASM

Полный вариант командной строки `nasm` выглядит следующим образом:

4.3.3. 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
```

Рис. 5.5: Расширенный синтаксис командной строки NASM

Выполните следующую команду: `nasm -o obj.o -f elf -g -l list.lst hello.asm`. 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` проверьте, что файлы были созданы. Для более подробной информации см. `man nasm`. Для получения списка форматов объектного файла см. `nasm -hf`.

5.4 Компоновщик LD

Как видно из схемы на рис. 4.3.3, чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

```
ld -m elf_i386 hello.o -o hello
```

```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 5ms
[●] * ld -m elf_x86_64 hello.o -o hello

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 7ms
λ ls
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:19 a.out
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:27 hello
-rw-r--r-- 941 mishanya4u 28 окт 17:17 hello.asm
-rw-r--r-- 896 mishanya4u 28 окт 17:19 hello.o
-rw-r--r-- 1,7k mishanya4u 28 окт 17:21 list.lst
-rw-r--r-- 1,6k mishanya4u 28 окт 17:21 obj.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 8ms
λ _
```

Рис. 5.6: Команда `ld -m elf_x86_64` (для arch-linux)

С помощью команды `ls` проверьте, что исполняемый файл `hello` был создан. Компоновщик `ld` не предполагает по умолчанию расширений для файлов, но принято использовать следующие расширения: 1. `o` – для объектных файлов; 2. без расширения – для исполняемых файлов; 3. `map` – для файлов схемы программы; 4. `lib` – для библиотек. Ключ `-o` с последующим значением задаёт в данном случае имя создаваемого исполняемого файла. Выполните следующую команду:

```
ld -m i386pe obj.o -o main
```

```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 9ms
λ ld -m i386pe obj.o -o main

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 10ms
λ ls
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:19 a.out
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:27 hello
-rw-r--r-- 941 mishanya4u 28 окт 17:17 hello.asm
-rw-r--r-- 896 mishanya4u 28 окт 17:19 hello.o
-rw-r--r-- 1,7k mishanya4u 28 окт 17:21 list.lst
-rwxr-xr-x 7,0k mishanya4u 28 окт 17:56 main
-rw-r--r-- 1,6k mishanya4u 28 окт 17:21 obj.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 9ms
λ
```

Рис. 5.7: Команда `ld -m i386pe`

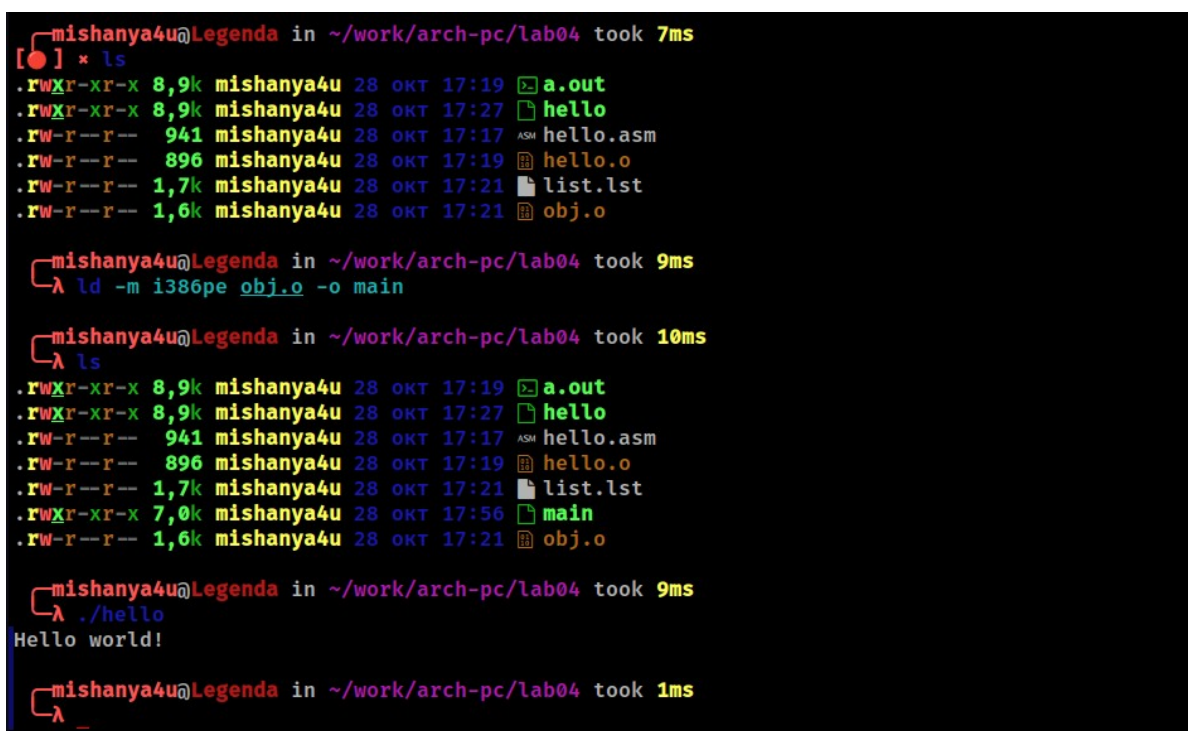
Какое имя будет иметь исполняемый файл? Какое имя имеет объектный файл из которого собран этот исполняемый файл? Формат

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

5.5 Запуск исполняемого файла

Запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке:

```
./hello
```



```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 7ms
[ ] * ls
.rwxr-xr-x 8,9k mishanya4u 28 ОКТ 17:19 a.out
.rwxr-xr-x 8,9k mishanya4u 28 ОКТ 17:27 hello
.rw-r--r-- 941 mishanya4u 28 ОКТ 17:17 hello.asm
.rw-r--r-- 896 mishanya4u 28 ОКТ 17:19 hello.o
.rw-r--r-- 1,7k mishanya4u 28 ОКТ 17:21 list.lst
.rw-r--r-- 1,6k mishanya4u 28 ОКТ 17:21 obj.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 9ms
λ ld -m i386pe obj.o -o main

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 10ms
λ ls
.rwxr-xr-x 8,9k mishanya4u 28 ОКТ 17:19 a.out
.rwxr-xr-x 8,9k mishanya4u 28 ОКТ 17:27 hello
.rw-r--r-- 941 mishanya4u 28 ОКТ 17:17 hello.asm
.rw-r--r-- 896 mishanya4u 28 ОКТ 17:19 hello.o
.rw-r--r-- 1,7k mishanya4u 28 ОКТ 17:21 list.lst
.rwxr-xr-x 7,0k mishanya4u 28 ОКТ 17:56 main
.rw-r--r-- 1,6k mishanya4u 28 ОКТ 17:21 obj.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 9ms
λ ./hello
Hello world!

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 1ms
λ _
```

Рис. 5.8: Вывод созданного файла

5.6 Задание для самостоятельной работы

1. В каталоге `~/work/arch-pc/lab04` с помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm`

```
mishanya4u@legenda in ~/work/arch-pc/lab04 took 1ms
└─ cp hello.asm lab4.asm

mishanya4u@legenda in ~/work/arch-pc/lab04 took 3ms
└─ ls
.rwxr-xr-x 8,9k mishanya4u 28 окт 17:19 a.out
.rwxr-xr-x 8,9k mishanya4u 28 окт 17:27 hello
.rwxr-xr-x 941 mishanya4u 28 окт 17:17 hello.asm
.rwxr-xr-x 896 mishanya4u 28 окт 17:19 hello.o
.rwxr-xr-x 941 mishanya4u 28 окт 17:57 lab4.asm
.rwxr-xr-x 1,7k mishanya4u 28 окт 17:21 list.lst
.rwxr-xr-x 7,6k mishanya4u 28 окт 17:58 main
.rwxr-xr-x 1,6k mishanya4u 28 окт 17:21 obj.o
```

Рис. 5.9: С помощью команды 'ср' создан копию файла

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.

```
lab04: micro — Konsole
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Hello world! Меня зовут Ван Сихэм Франклин О Нил Джон',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 ; писатель файла '1' - стандартный вывод
13 mov ecx,hello ; адрес строки hello в есх
14 mov edx,helloLen ; размер строки hello
15 int 80h ; Вызов ядра
16
17 mov eax,1 ; Системный вызов для вывода (sys_exit)
18 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19 int 80h ; Вызов ядра
20

/home/mishanya4u/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04/lab4.asm + (3,68) | ft:asm | uni
```

Рис. 5.10: Текстовый редактор nasm

```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 10ms
^ ./lab4
002c:fixme:winediag:loader_init wine-staging 8.18 is a testing version containing experimental patches.
002c:fixme:winediag:loader_init Please mention your exact version when filing bug reports on winehq.org.
0080:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
0080:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
0080:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
0080:fixme:hid:handle_IRP_MN_QUERY_ID Unhandled type 00000005
Could not parse file "/home/mishanya4u/.local/share/applications/libinput-gestures-qt.desktop": Invalid key name
: Path[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/lstopo.desktop": Invalid key name: Path[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/qwikaccess.desktop": Invalid key name: Path[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/xgps.desktop": Invalid key name: Path[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/xgpsspeed.desktop": Invalid key name: Path[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/yad-icon-browser.desktop": Invalid key name: Pa
th[$e]
Could not parse file "/home/mishanya4u/.local/share/applications/yad-settings.desktop": Invalid key name: Path[$
e]
0130:fixme:file:NtLockFile I/O completion on lock not implemented yet
0130:fixme:ntdll:NTQuerySystemInformation info_class SYSTEM_PERFORMANCE_INFORMATION
0130:fixme:msi:internal_ui_handler internal UI not implemented for message 0x0b000000 (UI level = 1)
0130:fixme:msi:internal_ui_handler internal UI not implemented for message 0x0b000000 (UI level = 1)
wine: configuration in L"/home/mishanya4u/.wine" has been updated.
Hello world! Меня зовут Ван Сихэм Франклин О Нил Джон

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 12s
^ _
```

3. Оттранслируйте полученный текст программы lab4.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.

```
mishanya4u@Legenda in ~/work/arch-pc/lab04 took 1m58s
^ nasm -f elf lab4.asm

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 6ms
^ ls
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:19 a.out
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:27 hello
-rw-r--r-- 941 mishanya4u 28 окт 17:17 hello.asm
-rw-r--r-- 896 mishanya4u 28 окт 17:19 hello.o
-rw-r--r-- 1,0k mishanya4u 28 окт 18:03 lab4.asm
-rw-r--r-- 720 mishanya4u 28 окт 18:03 lab4.o
-rw-r--r-- 1,7k mishanya4u 28 окт 17:21 list.lst
-rwxr-xr-x 7,0k mishanya4u 28 окт 17:58 main
-rw-r--r-- 1,0k mishanya4u 28 окт 17:21 obj.o

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 11ms
^ ld -m elf_x86_64 lab4.o -o lab4
ld: архитектура i386 входного файла «lab4.o» несовместима с выходным i386:86-64

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 8ms
^ ./lab4

mishanya4u@Legenda in ~/work/arch-pc/lab04 took 14ms
^ ls
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:19 a.out
-rwxr-xr-x 8,9k mishanya4u 28 окт 17:27 hello
-rw-r--r-- 941 mishanya4u 28 окт 17:17 hello.asm
-rw-r--r-- 896 mishanya4u 28 окт 17:19 hello.o
-rwxr-xr-x 4,9k mishanya4u 28 окт 18:07 lab4
-rw-r--r-- 1,0k mishanya4u 28 окт 18:03 lab4.asm
-rw-r--r-- 720 mishanya4u 28 окт 18:03 lab4.o
-rw-r--r-- 1,7k mishanya4u 28 окт 17:21 list.lst
-rwxr-xr-x 7,0k mishanya4u 28 окт 17:58 main
-rw-r--r-- 1,0k mishanya4u 28 окт 17:21 obj.o
```

Рис. 5.11: Компоновка файла lab4

4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузите файлы на Github.

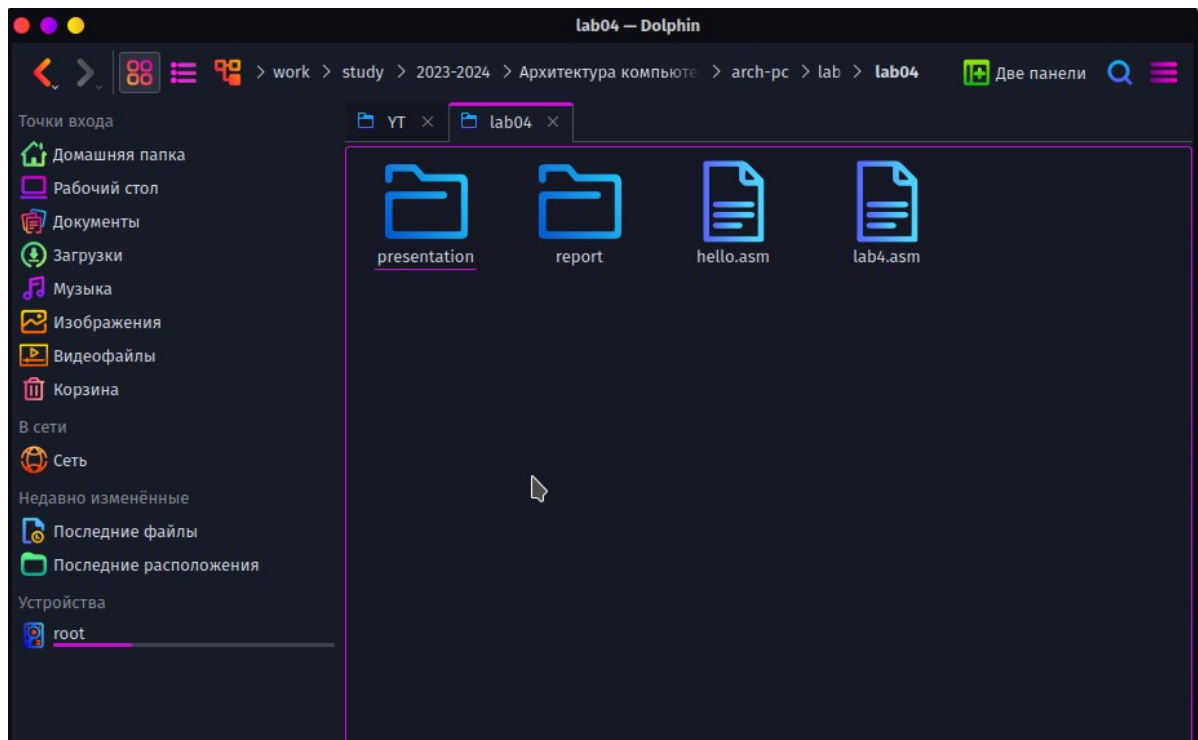


Рис. 5.12: Файлы hello.asm и lab4.asm скопированы в каталог

6 Заключение

Могу сказать что зыка ассемблера является важным шагом в развитии навыков программирования. Он позволяет программисту получить глубокое понимание работы компьютера и управления аппаратурой. Изучение ассемблера помогает развить навыки оптимизации кода и повысить производительность программ. Также этот язык открывает двери для изучения более высокоуровневых языков программирования, которые обеспечивают более высокий уровень абстракции и удобство написания кода. Знание этого языка помогает программисту лучше понять, как работает компилятор и какие инструкции выполняются на низком уровне.