

Отчёт по лабораторной работе №4

Дисциплина: архитектура компьютера

Киселева Елизавета Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Создание программы Hello word!	10
4.2	Работа с транслятором NASM	11
4.3	Работа с компоновщиком LD	11
4.4	Запуск исполняемого файла	12
4.5	Выполнение заданий для самостоятельной работы.	12
5	Выводы	15
	Список литературы	16

Список иллюстраций

4.1	Перемещение между директориями	10
4.2	Создание файла	10
4.3	Компиляция текста программы	11
4.4	Компиляция текста программы	11
4.5	Передача объектного файла на обработку компоновщику	12
4.6	Запуск исполняемого файла	12
4.7	Изменение программы	13
4.8	Компиляция текста и передача объектного файла на обработку компоновщику	13
4.9	Запуск исполняемого файла	13
4.10	Добавление файлов и отправка на GitHub	14

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

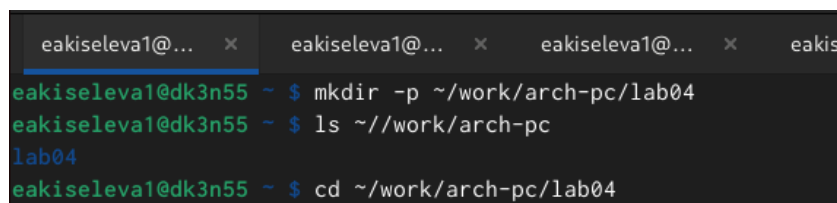
следующей команде.

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

4.1 Создание программы Hello word!

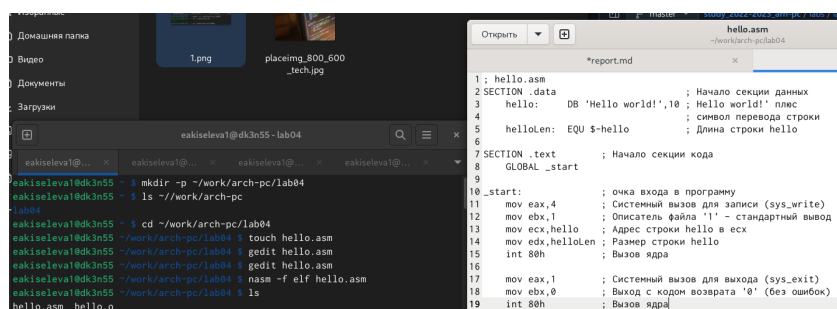
Создаю каталог для работы с программами на языке ассемблера NASM и перехожу в созданный каталог (рис. 4.1).



```
eakiseleva1@dk3n55 ~ $ mkdir -p ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~ $ ls ~/work/arch-pc
lab04
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab04
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch и открываю его командой gedit, заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.2).



```
eakiseleva1@dk3n55 ~ $ mkdir -p ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~ $ ls ~/work/arch-pc
lab04
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~ $ touch hello.asm
eakiseleva1@dk3n55 ~ $ gedit hello.asm
eakiseleva1@dk3n55 ~ $ nasm -f elf hello.asm
eakiseleva1@dk3n55 ~ $ ls
hello.asm hello.o
```

```
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: ; очка 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 еск
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. 4.2: Создание файла

4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o” (рис. 4.3).

```
eakiseleva1@dk3n55 ~ $ mkdir -p ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~ $ ls ~/work/arch-pc
lab04
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ touch hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ gedit hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ gedit hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.3: Компиляция текста программы

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, далее проверяю с помощью утилиты `ls` правильность выполнения команды (рис. 4.4).

```
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ man nasm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -hf
Usage: nasm [-@ response_file] [options...] [--] filename
       nasm -v (or --v)

Options (values in brackets indicate defaults):
```

Рис. 4.4: Компиляция текста программы

4.3 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику LD, чтобы получить исполняемый файл `hello`, далее проверяю с помощью утилиты `ls` правиль-

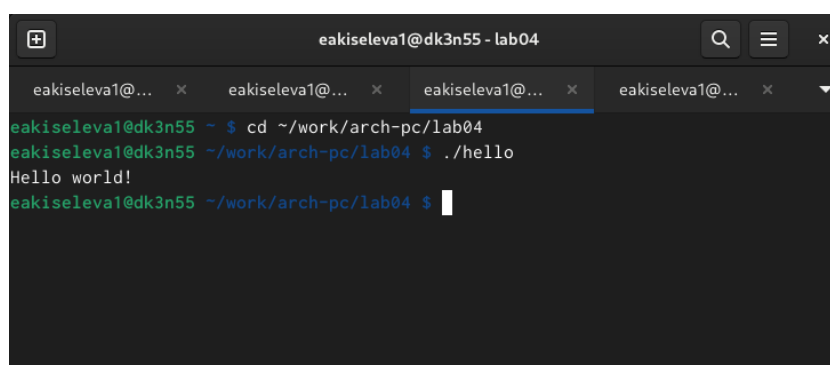
ность выполнения команды. Выполняю следующую команду. Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o (рис. 4.5).

```
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ld --help
Использование ld [параметры] файл...
Параметры:
  -a КЛЮЧЕВОЕ СЛОВО                                Управление общей библиотекой для совместимости с H
P/UX
  -A АРХИТЕКТУРА, --architecture АРХИТЕКТУРА      Задать архитектуру
  -b ЦЕЛЬ, --format ЦЕЛЬ
```

Рис. 4.5: Передача объектного файла на обработку компоновщику

4.4 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 4.6).



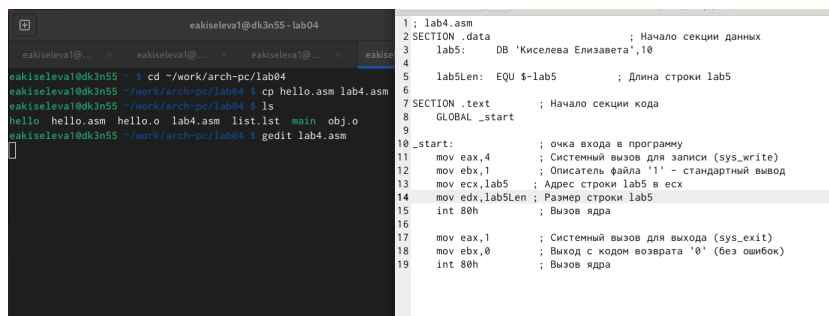
```
eakiseleva1@dk3n55 - lab04
eakiseleva1@... x eakiseleva1@... x eakiseleva1@... x eakiseleva1@... x
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab04
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ./hello
Hello world!
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $
```

Рис. 4.6: Запуск исполняемого файла

4.5 Выполнение заданий для самостоятельной работы.

С помощью утилиты `ср` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm`. С помощью текстового редактора `gedit` открываю файл `lab4.asm`

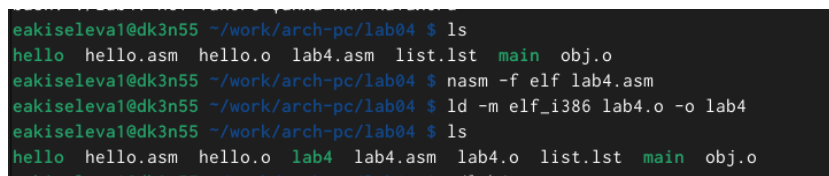
и вношу изменения в программу так, чтобы она выводила мои имя и фамилию (рис. 4.7).



```
1: lab4.asm
2: SECTION .data
3: lab5: DB 'Киселева Елизавета',10
4:
5: lab5Len: EQU $-lab5
6:
7: SECTION .text
8: GLOBAL _start
9:
10: _start:
11: mov eax,4
12: mov ebx,1
13: mov ecx,lab5
14: mov edx,lab5Len
15: int 80h
16:
17: mov eax,1
18: mov ebx,0
19: int 80h
```

Рис. 4.7: Изменение программы

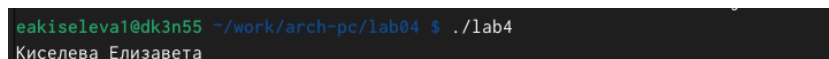
Компилирую текст программы в объектный файл. Проверяю с помощью утилиты ls, что файл lab4.o создан. Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 4.8).



```
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4.asm list.lst main obj.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
```

Рис. 4.8: Компиляция текста и передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 4.9).



```
eakiseleva1@dk3n55 ~/work/arch-pc/lab04 $ ./lab4
Киселева Елизавета
```

Рис. 4.9: Запуск исполняемого файла

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 и отправляю файлы на сервер с помощью команды git push (рис. 4.10).

```
eakiseleva1@... x eakiseleva1@... x eakiseleva1@... x eakiseleva1@... x
eakiseleva1@dk3n55 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello.asm lab4.asm presentation report
eakiseleva1@dk3n55 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git add .
eakiseleva1@dk3n55 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git commit -m "add fales for lab04"
[master 25c64a6] add fales for lab04
2 files changed, 38 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
eakiseleva1@dk3n55 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 1.05 КиБ | 1.05 МБ/с, готово.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:liza11223/study_2023-2024_arh-pc.git
50e6ec6..25c64a6 master -> master
eakiseleva1@dk3n55 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $
```

Рис. 4.10: Добавление файлов и отправка на GitHub

5 Выводы

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

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%