

# **Отчёт по лабораторной работе №4**

**Архитекткра компьютера**

Алёхин Давид Андреевич

# 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>13</b>
<b>5</b>	<b>Выводы</b>	<b>16</b>
	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание lab04 и hello.asm . . . . .	13
4.2	Компеллирую текст команды «Hello World» . . . . .	13
4.3	Компеллирую исходный файл hello.asm в obj.o. . . . .	13
4.4	Передаю объектный файл на обработку компоновщику. . . . .	13
4.5	Запускаю команду «Hello World». . . . .	14
4.6	Создание lab04.asm. . . . .	14
4.7	Меняю текст команды на “Алёхин Давид”. . . . .	14
4.8	Провожу компелляцию как и с файлом hello.asm. . . . .	14
4.9	Запускаю lab04. . . . .	14
4.10	Копирую файлы hello.asm lab04.asm в папку лабораторной работы. . . . .	15
4.11	Отправляю файлы на github. . . . .	15

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Написать код команды для вывода "hello world!".
2. Написать код для вывода "Имя Фамилия".

## 3 Теоретическое введение

4.2.1. Основные принципы работы компьютера Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 4.1). 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 (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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, пре- образование (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 (половинки 16-битных регистров). Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX. Таким образом можно отметить, что вы можете написать в своей программе, например, такие команды (mov — команда пересылки данных на языке ассемблера): `mov ax, 1` `mov eax, 1` Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1. 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. В состав ЭВМ также входят периферийные устройства, которые можно разделить на: • устройства внешней памяти, которые предназначены для длительного хранения больших объемов данных (жёсткие диски, твердотельные накопители, магнитные ленты); • 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. Программа состоит из машинных команд, которые указывают, какие операции и над какими данными (или операндами), в какой последовательности необходимо выполнить. Набор машинных команд определяется устройством конкретного процессора. 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. В самом общем виде 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. Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы.

4.2.2. Ассемблер и язык ассемблера 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, ча-

ще всего адрес в памяти. Т.о. 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.

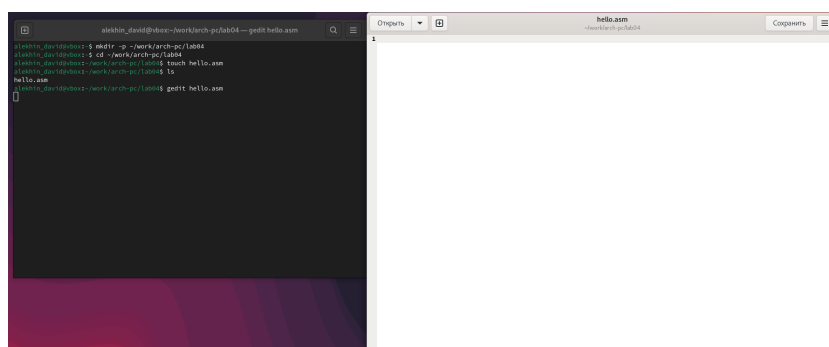
4.2.3. Процесс создания и обработки программы на языке ассемблера В процессе создания ассемблерной программы можно выделить четыре шага:

- 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 `nas`, 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. Из-за специфики программирования, а также по

традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах).

## 4 Выполнение лабораторной работы

Создаю папку lab04 в arch.рс, после в ранее созданной папке создаю файл hello.asm. (рис. 4.1).



```
alekhn_david@vbox:~/work/arch-pc/lab04$ mkdir -p ~/work/arch-pc/lab04
alekhn_david@vbox:~/work/arch-pc/lab04$ cd ~/work/arch-pc/lab04
alekhn_david@vbox:~/work/arch-pc/lab04$ touch hello.asm
alekhn_david@vbox:~/work/arch-pc/lab04$ ls
hello.asm
alekhn_david@vbox:~/work/arch-pc/lab04$ gedit hello.asm
```

Рис. 4.1: Создание lab04 и hello.asm

Компеллирую текст команды «Hello World». (рис. 4.2).



```
alekhn_david@vbox:~/work/arch-pc/lab04$ nasm -f elf hello.asm
alekhn_david@vbox:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Рис. 4.2: Компеллирую текст команды «Hello World»

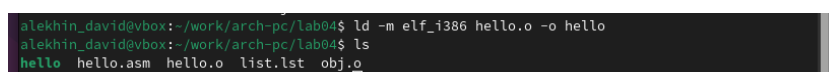
Компеллирую исходный файл hello.asm в obj.o. (рис. 4.3).



```
alekhn_david@vbox:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -i list.lst hello.asm
alekhn_david@vbox:~/work/arch-pc/lab04$ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.3: Компеллирую исходный файл hello.asm в obj.o.

Передаю объектный файл на обработку компоновщику. (рис. 4.4).



```
alekhn_david@vbox:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
alekhn_david@vbox:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.4: Передаю объектный файл на обработку компоновщику.

Запускаю команду «Hello World». (рис. 4.5).

```
alekhn_david@vbox: ~/work/arch-pc/lab04$ ./hello
Hello world!
```

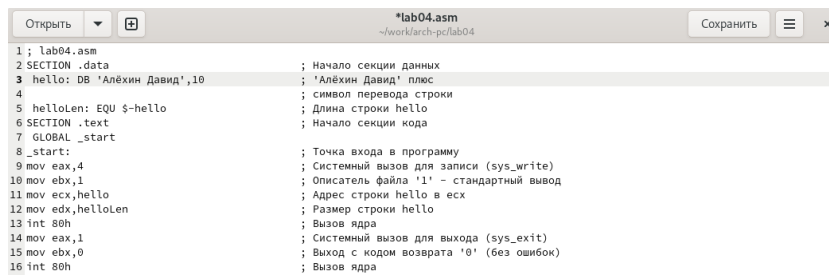
Рис. 4.5: Запускаю команду «Hello World».

Начинаю выполнять самостоятельную часть работы. Копирую файл hello.asm и называю lab04.asm. (рис. 4.6).

```
alekhn_david@vbox: ~/work/arch-pc/lab04$ cp hello.asm lab04.asm
alekhn_david@vbox: ~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab04.asm  list.lst  obj.o
```

Рис. 4.6: Создание lab04.asm.

Меняю текст команды на “Алёхин Давид”. (рис. 4.7).



```
*lab04.asm
~/work/arch-pc/lab04
Открыть Сохранить x
1; lab04.asm
2SECTION .data
3hello: DB 'Алёхин Давид',10
4; Начало секции данных
5; 'Алёхин Давид' плюс
6; символ перевода строки
5helloLen: EQU $-hello
6; Длина строки hello
6SECTION .text
7GLOBAL _start
8_start:
9; Точка входа в программу
9mov eax,4
10; Системный вызов для записи (sys_write)
10mov ebx,1
11; Описание файла '1' - стандартный вывод
11mov ecx,hello
12; Адрес строки hello в esx
12mov edx,helloLen
13; Размер строки hello
13int 80h
14; Вызов ядра
14mov eax,1
15; Системный вызов для выхода (sys_exit)
15mov ebx,0
16; Выход с кодом возврата '0' (без ошибок)
16int 80h
16; Вызов ядра
```

Рис. 4.7: Меняю текст команды на “Алёхин Давид”.

Провожу компелляцию как и с файлом hello.asm. (рис. 4.8).

```
alekhn_david@vbox: ~/work/arch-pc/lab04$ nasm -f elf lab04.asm
alekhn_david@vbox: ~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab04.o  list.lst  obj.o
alekhn_david@vbox: ~/work/arch-pc/lab04$ nasm -o obj1.o -f elf -g -l list1.lst lab04.asm
alekhn_david@vbox: ~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab04.o  list1.lst  list.lst  obj1.o  obj.o
alekhn_david@vbox: ~/work/arch-pc/lab04$ ld -m elf_i386 lab04.o -o lab04
alekhn_david@vbox: ~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab04  lab04.o  list1.lst  list.lst  obj1.o  obj.o
```

Рис. 4.8: Провожу компелляцию как и с файлом hello.asm.

Запускаю lab04. (рис. 4.9).

```
alekhn_david@vbox: ~/work/arch-pc/lab04$ ./lab04
Алёхин Давид
```

Рис. 4.9: Запускаю lab04.

Копирую файлы hello.asm lab04.asm в папку лабораторной работы. (рис. 4.10).

```

alekhn_david@vbox: ~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/"Архитектура компьюте
pa"/arch-pc/labs/lab04/
alekhn_david@vbox: ~/work/arch-pc/lab04$ cp lab04.asm ~/work/study/2023-2024/"Архитектура компьюте
pa"/arch-pc/labs/lab04/
alekhn_david@vbox: ~/work/arch-pc/lab04$

```

Рис. 4.10: Копирую файлы hello.asm lab04.asm в папку лабораторной работы.

Отправляю файлы на github. (рис. 4.11).

```

alekhn_david@vbox: ~/work/arch-pc/lab04$ cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc
alekhn_david@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git add .
alekhn_david@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git commit -am 'feat(main): add files lab-4'
[master 639582c] feat(main): add files lab-4
2 files changed, 32 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab04.asm
alekhn_david@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc$ 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), 1012 байтов | 1012.00 КиБ/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:trustdef/study_2023-2024_arhpc-.git
54f540c..639582c master -> master
alekhn_david@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc$

```

Рис. 4.11: Отправляю файлы на github.

## 5 Выводы

Проведя лабораторную работу я научился писать простейшие команды на языке assembler, компелировать и запускать их в терминале linux.



## 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/Learning-bash-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,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. 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).
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
  17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
  18. — 1120 с. — (Классика Computer Science).