

## Отчёта по лабораторной работе №4

### Дисциплина: архитектура компьютера

Агаджанян Артур

#### Содержание

### 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.

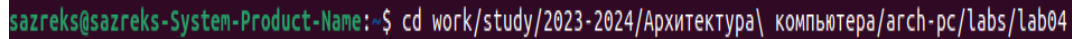
Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении 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 world!

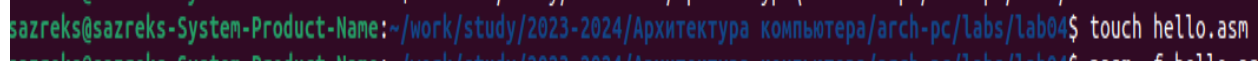
С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 1).



```
sazreks@sazreks-System-Product-Name:~$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
```

Рис. 1: Перемещение между директориями

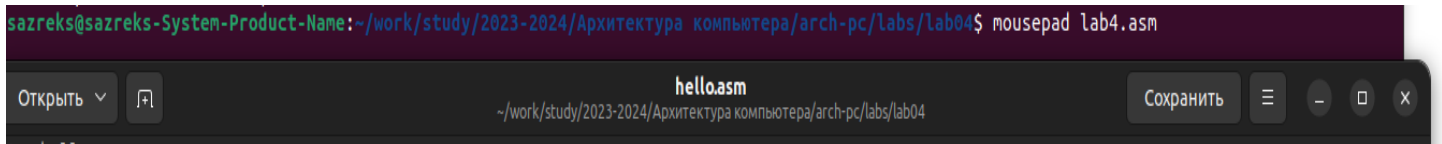
Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты



```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ touch hello.asm
```

`touch` (рис. 2).

Рис. 2: Создание пустого файла



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

Рис. 3: Открытие файла в текстовом редакторе

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

Рис. 4: Заполнение файла

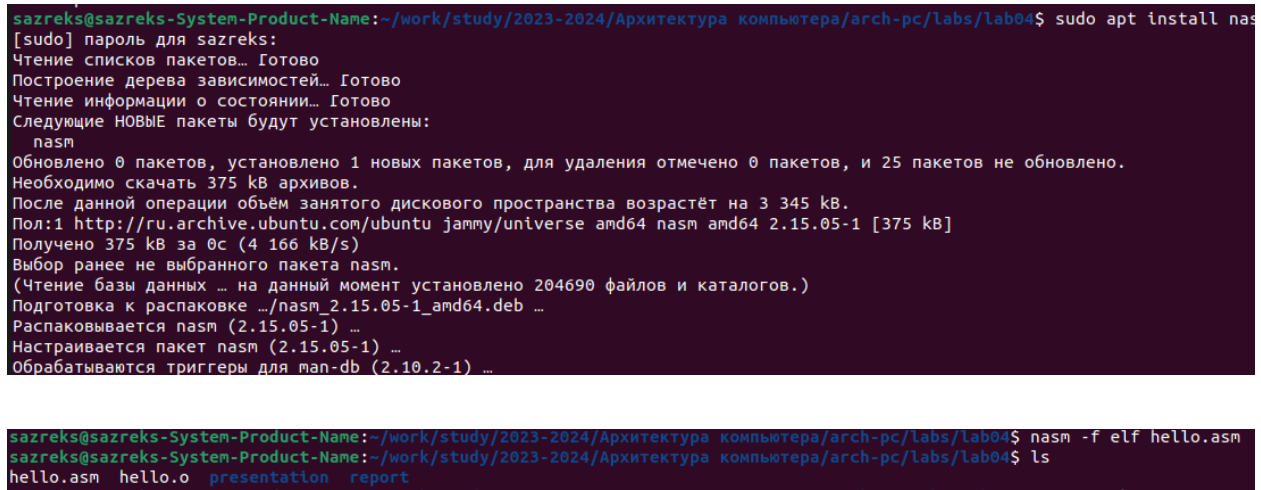


```
hello.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04
Сохранить

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

## 4.2 Работа с транслятором NASM

Для начала скачиваю транслятор NASM, с помощью команды `sudo apt install nasm` (рис.5). 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. 5.1). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
sazreks@sazreks-System-Product-Name: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ sudo apt install nasm
[sudo] пароль для sazreks:
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Следующие НОВЫЕ пакеты будут установлены:
  nasm
Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0 пакетов, и 25 пакетов не обновлено.
Необходимо скачать 375 кВ архивов.
После данной операции объем занятого дискового пространства возрастет на 3 345 кВ.
Пол:1 http://ru.archive.ubuntu.com/ubuntu jammy/universe amd64 nasm amd64 2.15.05-1 [375 кВ]
Получено 375 кВ за 0с (4 166 кВ/с)
Выбор ранее не выбранного пакета nasm.
(Чтение базы данных ... на данный момент установлено 204690 файлов и каталогов.)
Подготовка к распаковке .../nasm_2.15.05-1_amd64.deb ...
Распаковывается nasm (2.15.05-1) ...
Настраивается пакет nasm (2.15.05-1) ...
Обрабатываются триггеры для man-db (2.10.2-1) ...

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

Рис. 5 Установка программы

Рис. 5.1: Компиляция текста программы

## 4.3 Работа с расширенным синтаксисом командной строки NASM

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.

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

Рис. 6: Компиляция текста программы

### 4.4 Работа с компоновщиком LD

Устанавливаю компоновщик Ld используя команду `sudo apt install binutils` (рис. 7). Передаю объектный файл `hello.o` на обработку компоновщику LD, чтобы получить исполняемый файл `hello` (рис. 7.1). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ sudo apt install binutils
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
  binutils-common binutils-x86-64-linux-gnu libbinutils libctf-nobfd0 libctf0
Предлагаемые пакеты:
  binutils-doc
Следующие НОВЫЕ пакеты будут установлены:
  binutils binutils-common binutils-x86-64-linux-gnu libbinutils libctf-nobfd0 libctf0
Обновлено 0 пакетов, установлено 6 новых пакетов, для удаления отмечено 0 пакетов, и 25 пакетов не обновлено.
Необходимо скачать 3 425 kB архивов.
После данной операции объем занятого дискового пространства возрастет на 14,7 MB.
Хотите продолжить? [Д/н]
Пол:1 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 binutils-common amd64 2.38-4ubuntu2.3 [222 kB]
Пол:2 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libbinutils amd64 2.38-4ubuntu2.3 [662 kB]
Пол:3 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libctf-nobfd0 amd64 2.38-4ubuntu2.3 [107 kB]
Пол:4 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libctf0 amd64 2.38-4ubuntu2.3 [103 kB]
Пол:5 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 binutils-x86-64-linux-gnu amd64 2.38-4ubuntu2.3 [2 327 kB]
Пол:6 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 binutils amd64 2.38-4ubuntu2.3 [3 190 B]
Получено 3 425 kB за 0с (19,7 MB/s)
Выбор ранее не выбранного пакета binutils-common:amd64.
(Чтение базы данных ... на данный момент установлено 204715 файлов и каталогов.)
Подготовка к распаковке .../0-binutils-common_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается binutils-common:amd64 (2.38-4ubuntu2.3) ...
Выбор ранее не выбранного пакета libbinutils:amd64.
Подготовка к распаковке .../1-libbinutils_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается libbinutils:amd64 (2.38-4ubuntu2.3) ...
Выбор ранее не выбранного пакета libctf-nobfd0:amd64.
Подготовка к распаковке .../2-libctf-nobfd0_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается libctf-nobfd0:amd64 (2.38-4ubuntu2.3) ...
Выбор ранее не выбранного пакета libctf0:amd64.
Подготовка к распаковке .../3-libctf0_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается libctf0:amd64 (2.38-4ubuntu2.3) ...
Выбор ранее не выбранного пакета binutils-x86-64-linux-gnu.
Подготовка к распаковке .../4-binutils-x86-64-linux-gnu_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается binutils-x86-64-linux-gnu (2.38-4ubuntu2.3) ...
Выбор ранее не выбранного пакета binutils.
Подготовка к распаковке .../5-binutils_2.38-4ubuntu2.3_amd64.deb ...
Распаковывается binutils (2.38-4ubuntu2.3) ...
Настраивается пакет binutils-common:amd64 (2.38-4ubuntu2.3) ...
Настраивается пакет libctf-nobfd0:amd64 (2.38-4ubuntu2.3) ...
Настраивается пакет libbinutils:amd64 (2.38-4ubuntu2.3) ...
Настраивается пакет libctf0:amd64 (2.38-4ubuntu2.3) ...
Настраивается пакет binutils-x86-64-linux-gnu (2.38-4ubuntu2.3) ...
Настраивается пакет binutils (2.38-4ubuntu2.3) ...
Обрабатываются триггеры для libc-bin (2.35-0ubuntu3.4) ...
Обрабатываются триггеры для man-db (2.10.2-1) ...
```

Рис. 7. Установка программы

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

Рис. 7.1: 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

```
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst obj.o presentation report
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ld -m elf_i386 obj.o -o main
sazreks@sazreks-System-Product-Name:~/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. 8: Передача объектного файла на обработку компоновщику

### 4.5 Запуск исполняемого файла

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

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ./hello
Hello world!
```

Рис. 9: Запуск исполняемого файла

### 4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ cp hello.asm lab4.asm
sazreks@sazreks-System-Product-Name:~/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
lab4.asm (рис. 10).
```

Рис. 10: Создание копии файла

С помощью текстового редактора mousepad открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).

## Архитектура ЭВМ

```
Открыть 141 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04
; lab4.asm
SECTION .data ; Начало секции данных
lab4: DB 'Agadjanyan Artur',10

lab4len: EQU $-lab4 ; Длина строки lab4

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,lab4 ; Адрес строки lab4 в ecx
mov edx,lab4len ; Размер строки lab
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. 11: Изменение программы

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab4.o создан.

```
ab04$ nasm -f elf lab4.asm

sazreks@sazreks-System-Product-Name:~/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: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ld -m elf_i386 lab4.o -o lab4
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst obj.o presentation report
```

исполняемый файл lab4 (рис. 13).

Рис. 13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 14).

Рис. 14: Запуск исполняемого файла

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ./lab4  
> Agadjanyan Artur
```

Рис. 15: Создании копии файлов в новом каталоге

Удаляю лишние файлы в текущем каталоге с помощью утилиты `rm`, ведь копии

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ rm hello hello.o lab4 lab4.o list.lst main obj.o  
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
```

файлов остались в другой директории (рис. 16).

Рис. 16: Удаление лишних файлов в текущем каталоге

С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис. 17).

Рис. 17: Добавление файлов на GitHub

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git add .  
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git commit -m "Add fales for lab04"  
[master beea15f] Add fales for lab04  
2 files changed, 35 insertions(+)  
create mode 100644 labs/lab04/hello.asm  
create mode 100644 labs/lab04/lab4.asm
```

```
sazreks@sazreks-System-Product-Name:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git push  
Перечисление объектов: 16, готово.  
Подсчет объектов: 100% (15/15), готово.  
При схатии изменений используется до 4 потоков  
Сжатие объектов: 100% (11/11), готово.  
Запись объектов: 100% (11/11), 1.58 КиБ | 1.58 МИБ/с, готово.  
Всего 11 (изменений 6), повторно использовано 0 (изменений 0), повторно использовано пакетов 0  
remote: Resolving deltas: 100% (6/6), completed with 3 local objects.  
To github.com:Agartur/study_2023-2024_arh-pc.git  
269978d..ba49cd8 master -> master
```

Отправляю файлы на сервер с помощью команды `git push` (рис. 18).

Рис. 18: Отправка файлов

## 5 Выводы

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



## **6 Список литературы**

1. [https://esystem.rudn.ru/pluginfile.php/1584628/mod\\_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf](https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf)