

Отчёт по лабораторной работе №4

Дисциплина: архитектура компьютера

Луцкая Алиса Витальевна

Содержание

1	Цель работы	1
2	Задание.....	1
3	Теоретическое введение	2
4	Выполнение лабораторной работы	3
4.1	Создание программы Hello world!.....	3
4.2	Работа с транслятором NASM.....	4
4.3	Работа с расширенным синтаксисом командной строки NASM.....	5
4.4	Работа с компоновщиком LD	5
4.5	Запуск исполняемого файла	6
4.6	Выполнение заданий для самостоятельной работы.	6
5	Выводы.....	8
6	Список литературы.....	8

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 объема, 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 объемов 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!

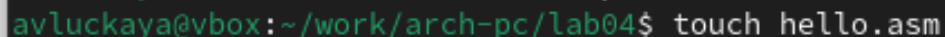
Создаю с помощью `mkdir` нужный каталог `~/work/arch-pc/lab04`, с помощью `cd` перехожу в неё (рис. 1).



```
avluckaya@vbox:~$ mkdir -p ~/work/arch-pc/lab04
avluckaya@vbox:~$ cd ~/work/arch-pc/lab04
avluckaya@vbox:~/work/arch-pc/lab04$
```

Рис. 1: Перемещение между директориями

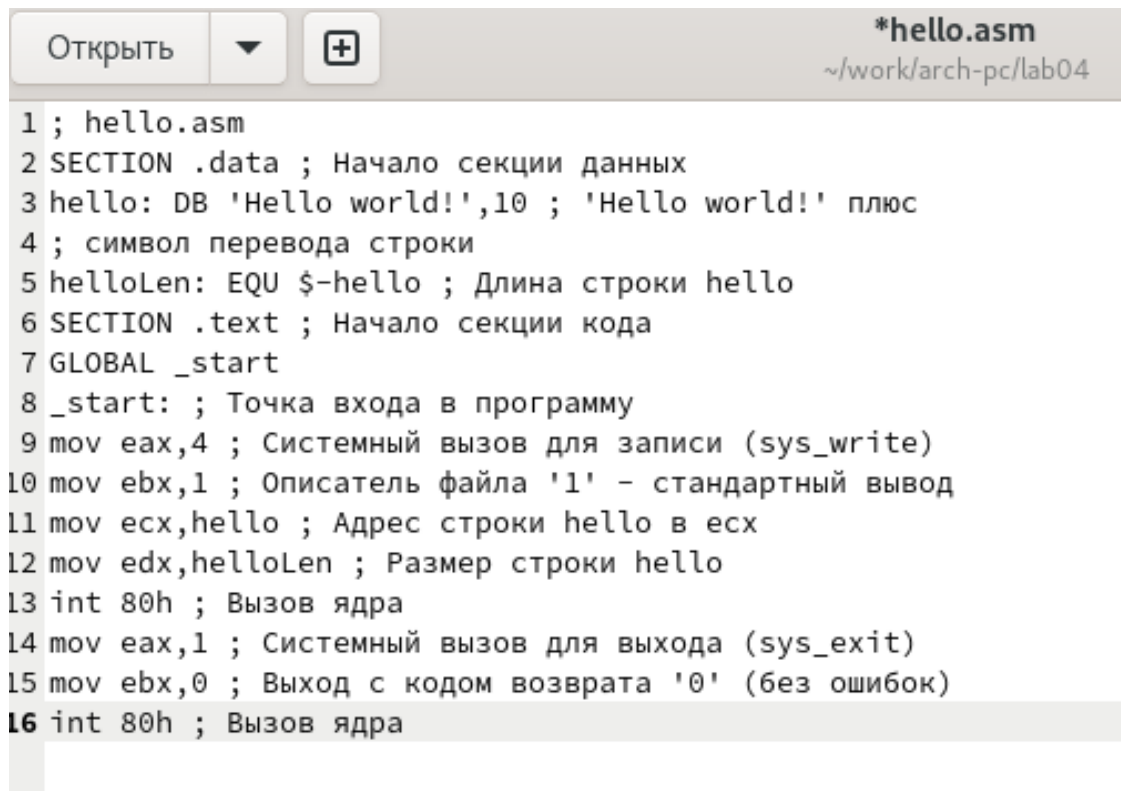
Создаю с помощью утилиты `touch` файл `hello.asm` (рис. 2).



```
avluckaya@vbox:~/work/arch-pc/lab04$ touch hello.asm
```

Рис. 2: Создание файла

Открываю созданный файл в `gedit` и заполняю его, вставляя программу для вывода "Hello word!" (рис. 3).



```
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 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,hello ; Адрес строки hello в ecx
12 mov edx,helloLen ; Размер строки hello
13 int 80h ; Вызов ядра
14 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра
```

Рис. 3: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю программу в объектный код с помощью транслятора NASM. Далее проверяю правильность выполнения команды с помощью `ls` (рис. 4)

```

avluckaya@vbox:~/work/arch-pc/lab04$ nasm -f elf hello.asm
bash: nasm: команда не найдена...
Установить пакет «nasm», предоставляющий команду «nasm»? [N/y] y

* Ожидание в очереди...
* Загрузка списка пакетов....
Следующие пакеты должны быть установлены:
 nasm-2.16.01-7.fc40.x86_64      A portable x86 assembler which uses Intel-like syntax
Продолжить с этими изменениями? [N/y] y

* Ожидание в очереди...
* Ожидание аутентификации...
* Ожидание в очереди...
* Загрузка пакетов...
* Запрос данных...
* Проверка изменений...
* Установка пакетов...

avluckaya@vbox:~/work/arch-pc/lab04$ ls
hello.asm hello.o

```

Рис. 4: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

Компилирую файл hello.asm в файл obj.o, далее проверяю правильность выполнения команды. (рис. 5).

```

avluckaya@vbox:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
avluckaya@vbox:~/work/arch-pc/lab04$ ls
hello.asm hello.o list.lst obj.o

```

Рис. 5: Компиляция текста программы

4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello, проверяю с помощью утилиты ls правильность выполнения команды. (рис. 6).

```

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

```

Рис. 6: Передача объектного файла на обработку компоновщику

Выполняю следующую команду, проверяю корректность выполнения (рис. 7). 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 list.lst obj.o
avluckaya@vbox:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
avluckaya@vbox:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o list.lst main obj.o

```

Рис. 7: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю для выполнения созданный файл hello (рис. 8).

```

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

```

Рис. 8: Запуск исполняемого файла

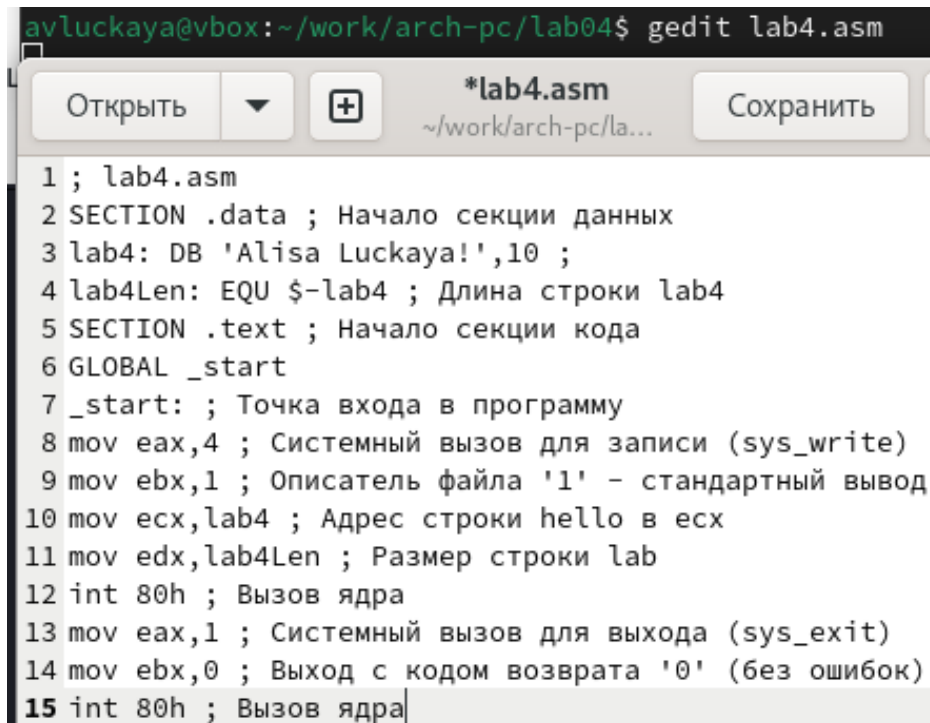
4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm`, открываю созданный файл в `gedit` и изменяю его, чтобы она выводила мое имя (рис. 9).

```

avluckaya@vbox:~/work/arch-pc/lab04$ gedit lab4.asm

```



```

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

```

Рис. 9: Изменение программы

Компилирую текст программы в объектный файл, проверяю, что файл `lab4.o` создан. (рис. 10).

```

avluckaya@vbox:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
avluckaya@vbox:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o
avluckaya@vbox:~/work/arch-pc/lab04$

```

Рис. 10: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab5, проверяю корректность выполнения (рис. 11).

```

avluckaya@vbox:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
avluckaya@vbox:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4  lab4.asm  lab4.o  list.lst  main  obj.o
avluckaya@vbox:~/work/arch-pc/lab04$

```

Рис. 11: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, действительно, выводятся мои имя и фамилия (рис. 12).

```

avluckaya@vbox:~/work/arch-pc/lab04$ ./lab4
Alisa Luckaya!
avluckaya@vbox:~/work/arch-pc/lab04$

```

Рис. 12: Запуск исполняемого файла

Копирую файл hello.asm в локальный репозиторий в каталог ~/work/study/2024-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ с помощью cp (рис. 13).

```

avluckaya@vbox:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2024-2025/“Архитектура компьютера”/arch-pc/labs/lab04/

```

Рис. 13: Копирование файла

Копирую файл lab4.asm в локальный репозиторий в каталог ~/work/study/2024-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ с помощью cp, перехожу в этот каталог и проверяю корректность выполнения команд(рис. 14).

```

avluckaya@vbox:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2024-2025/“Архитектура компьютера”/arch-pc/labs/lab04/
avluckaya@vbox:~/work/arch-pc/lab04$ cd ~/work/study/2024-2025/“Архитектура компьютера”/arch-pc/labs/lab04/
avluckaya@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm  lab4.asm  presentation  report
avluckaya@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04$

```

Рис. 14: Копирование файла

С помощью команд git add . и git commit добавляю файлы на GitHub, отправляю файлы на сервер с помощью команды git push (рис. 15).

```
avluckaya@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git add .
avluckaya@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git commit -m "Add lab04"
[master fe2c65a] Add lab04
2 files changed, 31 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
avluckaya@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git push
```

Рис. 15: Загрузка изменений на GitHub

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