

Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Бызова Мария Олеговна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	9
3.1	Создание программы Hello world!	9
3.2	Работа с транслятором NASM	10
3.3	Работа с расширенным синтаксисом командной строки NASM . .	11
3.4	Работа с компоновщиком LD	11
3.5	Запуск исполняемого файла	12
3.6	Выполнение заданий для самостоятельной работы.	12
4	Выводы	15

Список иллюстраций

3.1	Создание каталога	9
3.2	Перемещение между директориями	9
3.3	Создание пустого файла	9
3.4	Открытие файла в текстовом редакторе	10
3.5	Вид открытого файла в текстовом редакторе	10
3.6	Заполнение файла	10
3.7	Компиляция текста программы	11
3.8	Компиляция текста программы	11
3.9	Передача объектного файла на обработку компоновщику	11
3.10	Передача объектного файла на обработку компоновщику	12
3.11	Запуск исполняемого файла	12
3.12	Создание копии файла	12
3.13	Открытие текстового файла	12
3.14	Изменение программы	13
3.15	Компиляция текста программы	13
3.16	Передача объектного файла на обработку компоновщику	13
3.17	Запуск исполняемого файла	13
3.18	Копирование файлов	14
3.19	Загрузка файлов на GitHub	14
3.20	Проверка правильности работы команд.	14

Список таблиц

1 Цель работы

Целью данной лабораторной работы является освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 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

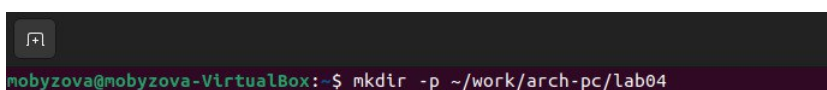
следующей команде.

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.

3 Выполнение лабораторной работы

3.1 Создание программы Hello world!

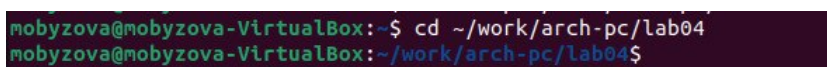
При помощи ранее изученных команд создаем каталог для работы с программами на языке ассемблера NASM для выполнения последующих заданий данной лабораторной работы (рис. [3.1]).



```
mobyzova@mobyzova-VirtualBox:~$ mkdir -p ~/work/arch-pc/lab04
```

Рис. 3.1: Создание каталога

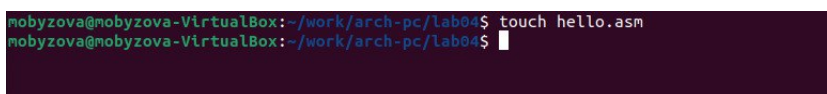
С помощью утилиты `cd` перемещаемся в каталог, в котором далее будем работать (рис. [3.2]).



```
mobyzova@mobyzova-VirtualBox:~$ cd ~/work/arch-pc/lab04
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab04$
```

Рис. 3.2: Перемещение между директориями

Создаем в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. [3.3]).



```
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab04$ touch hello.asm
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab04$
```

Рис. 3.3: Создание пустого файла

Открываем созданный файл в текстовом редакторе при помощи команды `gedit` (рис. [3.4] и [3.5]).

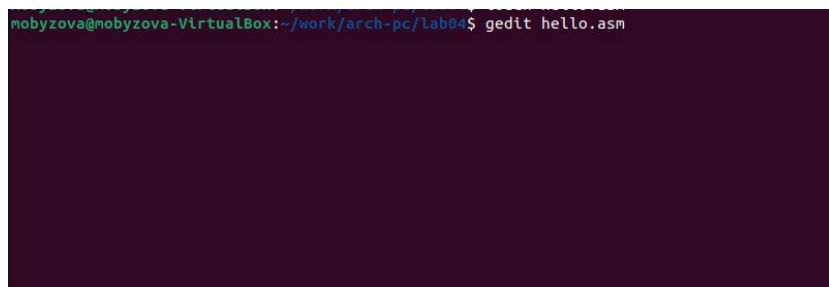


Рис. 3.4: Открытие файла в текстовом редакторе

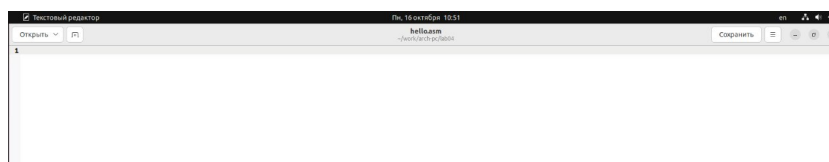


Рис. 3.5: Вид открытого файла в текстовом редакторе

Заполняем файл, вставляя в него программу для вывода “Hello word!” (рис. [3.6]).



Рис. 3.6: Заполнение файла

3.2 Работа с транслятором NASM

Превращаем 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. Далее проверяем правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o” - именно такое имя будет иметь созданный объектный файл (рис. [3.7]).

```
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ nasm -f elf hello.asm
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ ls
hello.asm  hello.o
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$
```

Рис. 3.7: Компиляция текста программы

3.3 Работа с расширенным синтаксисом командной строки NASM

Вводим 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`. Далее проверяем с помощью утилиты `ls` правильность выполнения команды (рис. [3.8]).

```
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ nasm -o obj.o -f elf -g -l list.lst hello.asm
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ ls
hello.asm  hello.o  list.lst  obj.o
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$
```

Рис. 3.8: Компиляция текста программы

3.4 Работа с компоновщиком LD

Передаем объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. [3.9]). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяем с помощью утилиты `ls` правильность выполнения команды.

```
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ ld -m elf_i386 hello.o -o hello
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$ ls
hello  hello.asm  hello.o  list.lst  obj.o
mobyrova@mobyrova-VirtualBox: /work/arch-pc/lab0$
```

Рис. 3.9: Передача объектного файла на обработку компоновщику

Выполняем следующую команду (рис. [3.10]). 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

```
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ ld -m elf_i386 obj.o -o main
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ ls
hello hello.asm hello.o list.lst main obj.o
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$
```

Рис. 3.10: Передача объектного файла на обработку компоновщику

3.5 Запуск исполняемого файла

Запускаем на выполнение созданный исполняемый файл hello, находящийся в текущем каталоге (рис. [3.11]).

```
hello hello.asm hello.o list.lst main obj.o
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ ./hello
hello, world!
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$
```

Рис. 3.11: Запуск исполняемого файла

3.6 Выполнение заданий для самостоятельной работы.

1. С помощью утилиты cp создаем в текущем каталоге копию файла hello.asm с именем lab4.asm (рис. [3.12]).

```
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ cp hello.asm lab4.asm
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ ls
hello hello.asm hello.o lab4.asm list.lst main obj.o
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$
```

Рис. 3.12: Создание копии файла

2. С помощью текстового редактора открываем файл lab4.asm и вносим изменения в программу так, чтобы она выводила наши имя и фамилию. (рис. [3.13] и [3.14]).

```
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ gedit lab4.asm
nobyzo@nobyzo-VirtualBox: /work/arch-pc/lab4$ gedit lab4.asm
```

Рис. 3.13: Открытие текстового файла

```

1 SECTION .data
2     lab4:      db "Mary Byzova",0xa
3     lab4Len:   equ $ - lab4
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, lab4
11    mov edx, lab4Len
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80

```

Рис. 3.14: Изменение программы

3. Компилируем текст программы в объектный файл (рис. [3.15]). Проверяем с помощью утилиты ls, что файл lab4.o создан.

```

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

```

Рис. 3.15: Компиляция текста программы

Передаем объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. [3.16]).

```

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

```

Рис. 3.16: Передача объектного файла на обработку компоновщику

Запускаем исполняемый файл lab4. На экран действительно выводятся наши имя и фамилия (рис. [3.17]).

```

hello  hello.asm  hello.o  lab4  lab4.asm  lab4.o  list.lst  main  obj.o
mobyzova@mobyzova-VirtualBox: ~/work/arch-pc/lab04$ ./lab4
Mary Byzova
mobyzova@mobyzova-VirtualBox: ~/work/arch-pc/lab04$

```

Рис. 3.17: Запуск исполняемого файла

4. Скопируем файлы hello.asm и lab4.asm в наш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ (рис. [3.18]).

```

fatal: не найден git репозиторий (или один из родительских каталогов): .git
mobyzoa@mobyzoa-VirtualBox: ~/work/arch-pc/labs$ cp hello.asm ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04
mobyzoa@mobyzoa-VirtualBox: ~/work/arch-pc/labs$ cp lab4.asm ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04
mobyzoa@mobyzoa-VirtualBox: ~/work/arch-pc/labs$

```

Рис. 3.18: Копирование файлов

Далее с помощью ранее изученных команд загружаем файлы на GitHub: добавляем с помощью `git add` нужный файл, сохраняем изменения с помощью `git commit`, отправляем в центральный репозиторий сохраненные изменения командой `git push` (рис. [3.19]). Проверяем правильность работы (рис. [3.20]).

```

mobyzoa@mobyzoa-VirtualBox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git add .
mobyzoa@mobyzoa-VirtualBox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git commit -m "add files for lab4"
[master 9fa8624] add files for lab4
 2 files changed, 32 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
mobyzoa@mobyzoa-VirtualBox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ git push
To github.com:mobyzoa/study_2023-2024_arh-pc.git
 66b1b51..9fa8624

```

Рис. 3.19: Загрузка файлов на GitHub

mobyzoa add files for lab4 altload · now · History		
Name	Last commit message	Last commit date
..		
presentation	test/main: make course structure	last month
report	test/main: add files lab-2	last week
hello.asm	add files for lab4	now
lab4.asm	add files for lab4	now

Рис. 3.20: Проверка правильности работы команд.

4 Выводы

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