

Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Саммура Халед

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	9
4.1	Создание программы Hello world!	9
4.2	Работа с транслятором NASM.....	10
4.3	Работа с расширенным синтаксисом командной строки NASM.....	10
4.4	Работа с компоновщиком LD	11
4.5	Запуск исполняемого файла	11
4.6	Выполнение заданий для самостоятельной работы.....	12
5	Выводы	15
6	Список литературы	16

Список иллюстраций

4.1	Перемещение между директориями	9
4.2	Создание пустого файла	9
4.3	Открытие файла в текстовом редакторе	9
4.4	Заполнение файла	10
4.5	Компиляция текста программы	10
4.6	Компиляция текста программы	11
4.7	Передача объектного файла на обработку компоновщику	11
4.8	Передача объектного файла на обработку компоновщику	11
4.9	Запуск исполняемого файла	12
4.10	Создание копии файла.....	12
4.11	Изменение программы.....	12
4.12	Компиляция текста программы	13
4.13	Передача объектного файла на обработку компоновщику	13
4.14	Запуск исполняемого файла	13
4.15	Создании копии файлов в новом каталоге.....	14
4.16	Удаление лишних файлов в текущем каталоге	14
4.17	Добавление файлов на GitHub.....	14
4.18	Отправка файлов.....	14

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. 4.1).

```
liveuser@khaledsamm:~$ cd study_2022-2023_arch-pc/labs/lab04
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 4.2).

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ touch hello.asm
```

Рис. 4.2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 4.3).

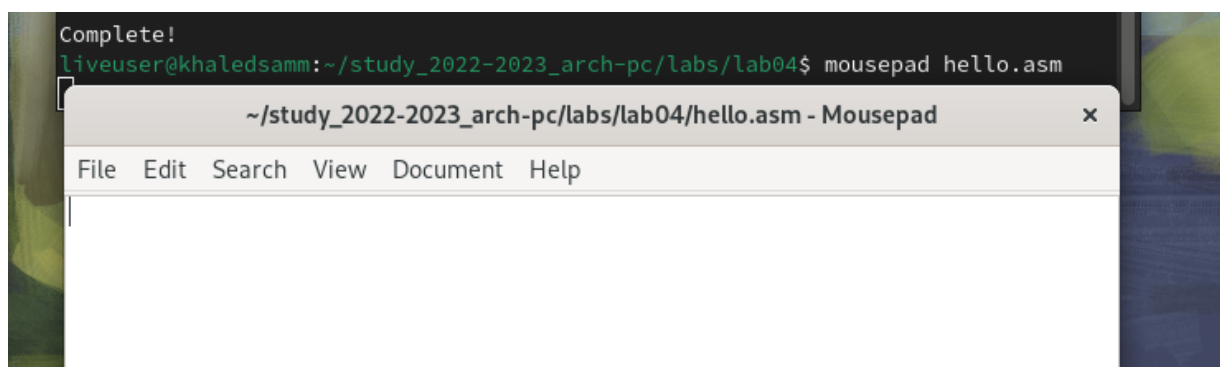


Рис. 4.3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.4).

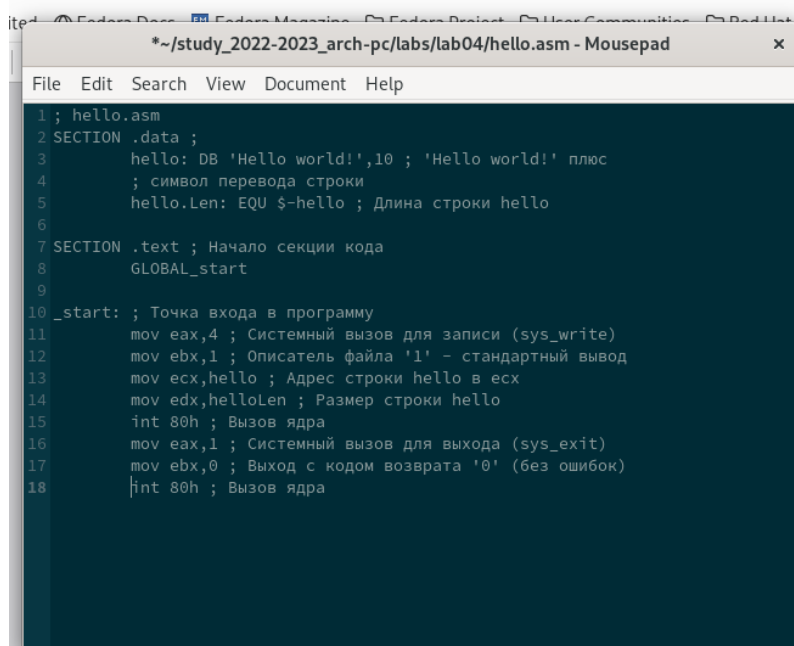


Рис. 4.4: Заполнение файла

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`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 4.5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.

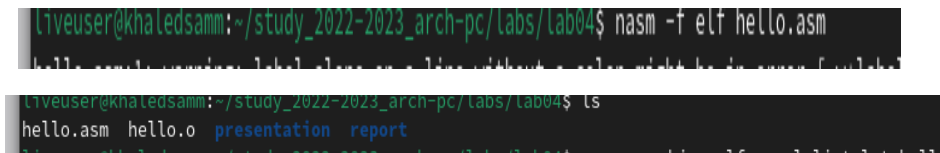


Рис. 4.5: Компиляция текста программы

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. 4.6). Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello.asm  hello.o  list.lst  obj.o  presentation  report
```

Рис. 4.6: Компиляция текста программы

4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. 4.7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ld -m elf_i386 hello.o -o hello
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o  presentation  report
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 4.8). 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

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ld -m elf_i386 obj.o -o main
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o  presentation  report
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 4.9).

```
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ ./hello
Hello world!
```

Рис. 4.9: Запуск
исполняемого файла

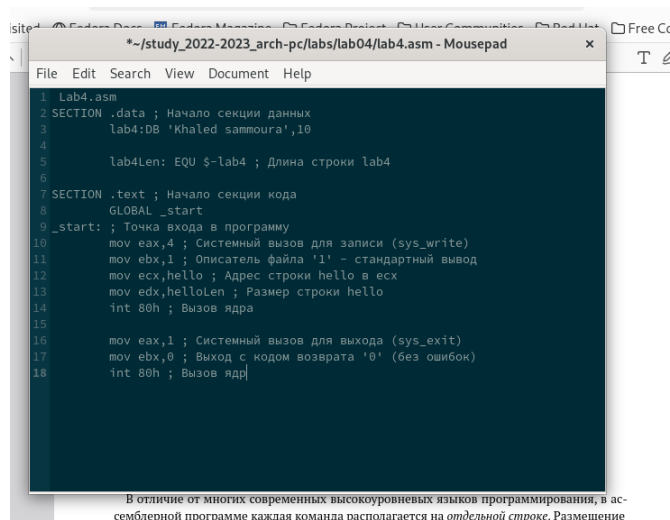
4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab5.asm` (рис. 4.10).

```
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ cp hello.asm lab4.asm
```

Рис. 4.10: Создание копии файла

С помощью текстового редактора `mousepad` открываю файл `lab5.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.11).



```
*~/study_2022-2023_arch-pc/labs/lab04/lab4.asm - Mousepad
File Edit Search View Document Help
1 Lab4.asm
2 SECTION .data ; Начало секции данных
3     lab4:DB 'Khaled sammoura',10
4
5     lab4Len: EQU $-lab4 ; Длина строки lab4
6
7 SECTION .text ; Начало секции кода
8     GLOBAL _start
9     _start: ; Точка входа в программу
10    mov eax,4 ; Системный вызов для записи (sys_write)
11    mov ebx,1 ; Описатель файла '1' - стандартный вывод
12    mov ecx,hello ; Адрес строки hello в ecx
13    mov edx,helloLen ; Размер строки hello
14    int 80h ; Вызов ядра
15
16    mov eax,1 ; Системный вызов для выхода (sys_exit)
17    mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
18    int 80h ; Вызов ядра
```

В отличие от многих современных высокоуровневых языков программирования, в ассемблерной программе каждая команда располагается на *отдельной строке*. Размещение

Рис. 4.11: Изменение программы

Компилирую текст программы в объектный файл (рис. 4.12). Проверяю с помощью утилиты ls, что файл lab5.o создан.

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ nasm -f elf lab4.asm
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o presentation report
```

Рис. 4.12: Компиляция текста программы

Передаю объектный файл lab5.o на обработку компоновщику LD, чтобы получить исполняемый файл lab5 (рис. 4.13).

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ld -m elf_i386 lab4.o -o lab4
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
```

Рис. 4.13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab5, на экран действительно выводятся мои имя и фамилия (рис. 4.14).

```
liveuser@khaledsamm:~/study_2022-2023_arch-pc/labs/lab04$ ./lab4
Khaled sammoura
```

Рис. 4.14: Запуск исполняемого файла

К сожалению, я начала работу не в том каталоге, поэтому создаю другую директорию lab05 с помощью mkdir, прописывая полный путь к каталогу, в котором хочу создать эту директорию. Далее копирую из текущего каталога файлы, созданные в процессе выполнения лабораторной работы, с помощью утилиты cp, указывая вместо имени файла символ *, чтобы скопировать все файлы. Команда проигнорирует директории в этом каталоге, т. к. не указан ключ -r, это мне и нужно (рис. 4.15). Проверяю с помощью утилиты ls правильность выполнения команды.

Удаляю лишние файлы в текущем каталоге с помощью утилиты `rm`, ведь копии файлов остались в другой директории (рис. 4.16).

```
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ ls
hello.asm lab4.asm presentation report
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$
```

Рис. 4.16: Удаление лишних файлов в текущем каталоге

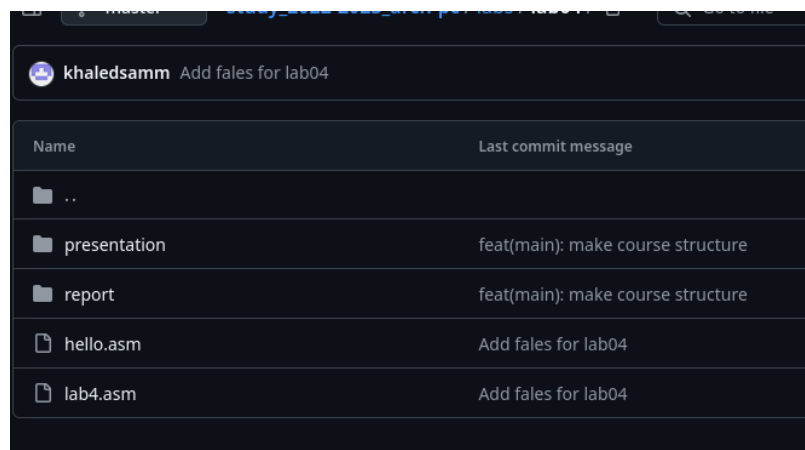
С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис. 4.17).

```
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ git add .
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ git commit -m "Add fales for lab04"
Author identity unknown
```

Рис. 4.17: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды `git push` (рис. 4.18).

```
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$ git push
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhpZisF/zLDA0zPMSvHdKr4UvC0qU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.06 KiB | 1.06 MiB/s, done.
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:khaledsam/study_2022-2023_arch-pc.git
 5718984..cf0ac46 master -> master
liveuser@khaledsam:~/study_2022-2023_arch-pc/labs/lab04$
```



The screenshot shows a web interface for committing files. At the top, there's a header with a user profile icon and the text "khaledsam Add fales for lab04". Below this is a table with two columns: "Name" and "Last commit message". The table lists several files and folders, including "..", "presentation", "report", "hello.asm", and "lab4.asm", each with its corresponding commit message.

Name	Last commit message
..	
presentation	feat(main): make course structure
report	feat(main): make course structure
hello.asm	Add fales for lab04
lab4.asm	Add fales for lab04

Рис. 4.18: Отправка файлов

5 Выводы

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

