

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

Дисциплина: архитектура компьютера

Цоппа Ева Эдуардовна

Содержание

1 Цель работы.....	4
2 Задание.....	5
3 Теоретическое введение.....	6
4 Выполнение лабораторной работы	8
4.1 Создание программы Hello World!.....	8
4.2 Работа с транслятором NASM.....	9
4.3 Работа с расширенным синтаксисом командной строки NASM.....	9
4.4 Работа с компоновщиком LD.....	10
4.5 Запуск исполняемого файла.....	10
4.6 Выполнение заданий для самостоятельной работы.....	10
5 Выводы.....	13
6 Список литературы	14

Список иллюстраций

4.1 Перемещение между директориями.....	8
4.2 Создание пустого файла.....	8
4.3 Открытие файла в текстовом редакторе.....	8
4.4 Заполнение файла	9
4.5 Компиляция текста программы.....	9
4.6 Компиляция текста программы.....	9
4.7 Передача объектного файла на обработку компоновщику.....	10
4.8 Передача объектного файла на обработку компоновщику	10
4.9 Запуск исполняемого файла.....	10
4.10 Создание копии файла	10
4.11 Изменение программы.....	11
4.12 Компиляция текста программы.....	11
4.13 Передача объектного файла на обработку компоновщику.....	11
4.14 Запуск исполняемого файла.....	11
4.15 Удаление лишних файлов в текущем каталоге.....	12
4.16 Добавление файлов на GitHub.....	12
4.17 Отправка файлов.....	12

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).

```
[evatsoppa@evatsoppa ~]$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04  
[evatsoppa@evatsoppa lab04]$
```

Рис. 4.1 Перемещение между директориями

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

```
[evatsoppa@evatsoppa lab04]$ touch hello.asm
```

Рис. 4.2 Создание пустого файла

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

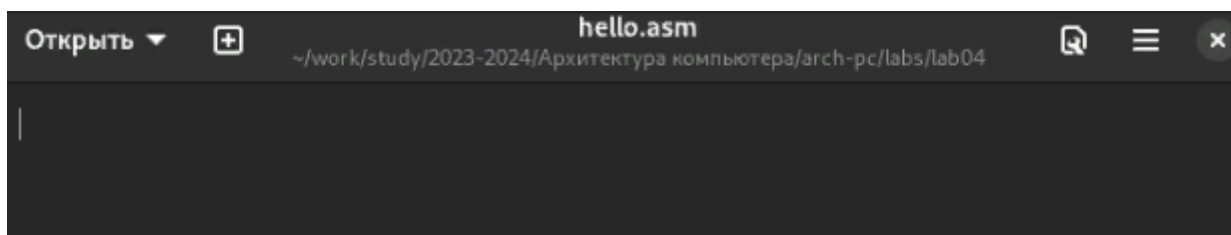
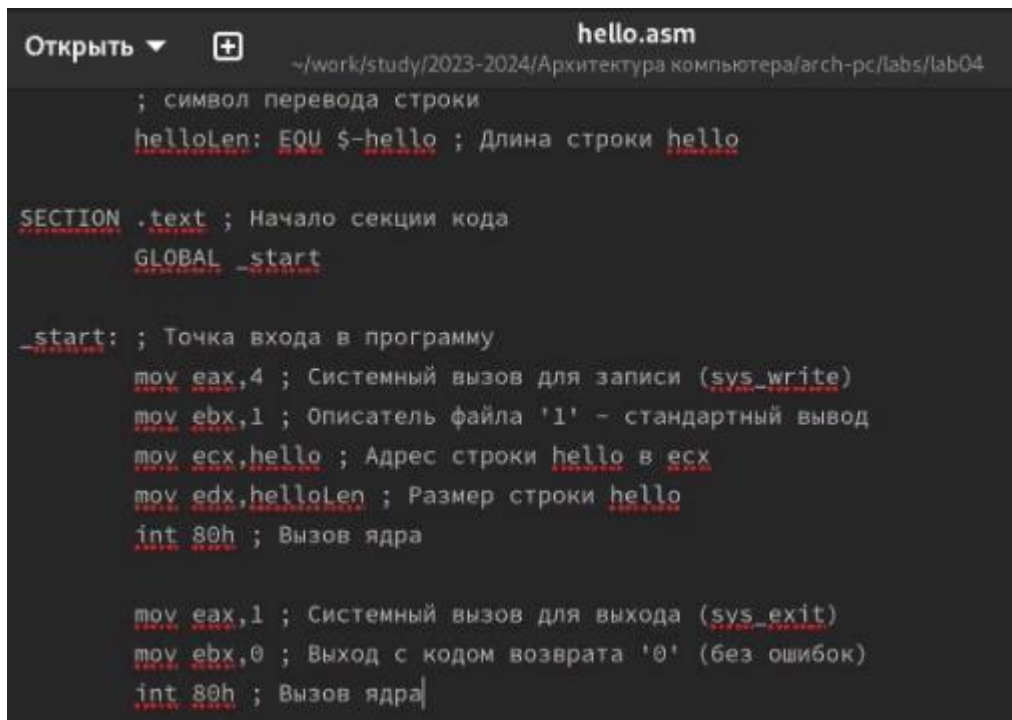


Рис. 4.3 Открытие файла в текстовом редакторе

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



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

; символ перевода строки
helloLen: EQU $-hello ; Длина строки hello

SECTION .text ; Начало секции кода
GLOBAL _start

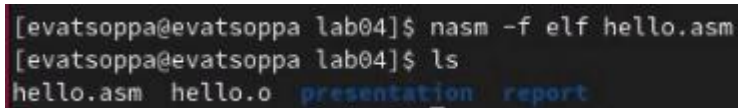
_start: ; Точка входа в программу
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,hello ; Адрес строки hello в ecx
mov edx,helloLen ; Размер строки hello
int 80h ; Вызов ядра

mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
int 80h ; Вызов ядра
```

Рис. 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”.

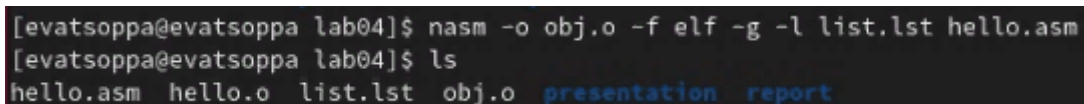


```
[evatsoppa@evatsoppa lab04]$ nasm -f elf hello.asm
[evatsoppa@evatsoppa lab04]$ ls
hello.asm hello.o presentation report
```

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.



```
[evatsoppa@evatsoppa lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[evatsoppa@evatsoppa 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.

```
[evatsoppa@evatsoppa lab04]$ ld -m elf_i386 hello.o -o hello
[evatsoppa@evatsoppa 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

```
[evatsoppa@evatsoppa lab04]$ ld -m elf_i386 obj.o -o main
[evatsoppa@evatsoppa 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).

```
[evatsoppa@evatsoppa 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 lab4.asm (рис. 4.10).

```
[evatsoppa@evatsoppa lab04]$ cp hello.asm lab4.asm
```

Рис. 4.10 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. 4.11).

```

; lab4.asm
SECTION .data ; Начало секции данных
    lab4: DB 'Eva Tsoppa',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. 4.11 Изменение программы

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

```

[evatsoppa@evatsoppa lab04]$ nasm -f elf lab4.asm
[evatsoppa@evatsoppa 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 lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 4.13).

```

[evatsoppa@evatsoppa lab04]$ ld -m elf_i386 lab4.o -o lab4
[evatsoppa@evatsoppa 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 lab4, на экран действительно выводятся мои имя и фамилия (рис. 4.14).

```

[evatsoppa@evatsoppa lab04]$ ./lab4
Eva Tsoppa

```

Рис. 4.14 Запуск исполняемого файла

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm (рис. 4.15).

```
[evatsoppa@evatsoppa lab04]$ rm hello hello.o lab4 lab4.o list.lst main obj.o
[evatsoppa@evatsoppa lab04]$ ls
hello.asm lab4.asm presentation report
```

Рис. 4.15 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 №4 (рис. 4.16).

```
[evatsoppa@evatsoppa arch-pc]$ git add .
[evatsoppa@evatsoppa arch-pc]$ git commit -m "Add fales for lab04"
[master 2162bb7] Add fales for lab04
18 files changed, 151 insertions(+), 60 deletions(-)
delete mode 100644 labs/lab02/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-08-49.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-11-12.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-11-20.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-11-55.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-12-21.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 14-14-31.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 15-09-55.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-19 15-12-20.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-20 21-49-21.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-20 21-51-36.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-20 21-54-01.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-25 11-08-41.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-25 11-09-20.png
create mode 100644 labs/lab02/report/image/Снимок экрана от 2023-09-25 11-11-03.png
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
```

Рис. 4.16 Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 4.17).

```
[evatsoppa@evatsoppa arch-pc]$ git push
Перечисление объектов: 44, готово.
Подсчет объектов: 100% (42/42), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (30/30), 161.67 КиБ | 1.18 МиБ/с, готово.
Всего 30 (изменений 9), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (9/9), completed with 4 local objects.
To github.com:evatsoppa/study_2023-2024_arh-pc.git
  107d367..55ace5e master -> master
```

Рис. 4.17 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/2089084/mod_resource/content/0/Лабораторная%20работа%20№4.%20Создание%20и%20процесс%20обработки%20программ%20на%20языке%20ассемблера%20NASM.pdf