

Лабораторная работа №4

Рогозин Игорь Андреевич

Содержание

| | | |
|----------|---|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 7 |
| 3.1 | Ассемблер и язык ассемблера | 7 |
| 4 | Выполнение лабораторной работы | 10 |
| 4.1 | Программа Hello world! | 10 |
| 4.2 | Транслятор NASM | 11 |
| 4.3 | Расширенный синтаксис командной строки NASM | 12 |
| 4.4 | Компоновщик LD | 13 |
| 5 | Задание для самостоятельной работы | 15 |
| 6 | Выводы | 17 |

Список иллюстраций

| | | |
|------|---|----|
| 4.1 | Рис.1 Терминал линукс | 10 |
| 4.2 | Рис.2 Команда создания каталога для программ NASM | 10 |
| 4.3 | Рис.3 Переход в каталог | 10 |
| 4.4 | Рис.4 Создание файла hello.asm | 11 |
| 4.5 | Рис.5 Редактор gedit | 11 |
| 4.6 | Рис.6 Ввод текста программы | 11 |
| 4.7 | Рис.7 Компиляция программы | 12 |
| 4.8 | Рис.8 Проверка выполнения | 12 |
| 4.9 | Рис.9 Ввод команды | 13 |
| 4.10 | Рис.10 Проверка выполнения | 13 |
| 4.11 | Рис.11 Обработка программы компановщиком | 13 |
| 4.12 | Рис.12 Проверк выполнения | 14 |
| 4.13 | Рис.13 Выполнение команды | 14 |
| 4.14 | Рис.14 Выполнение команды | 14 |
| 5.1 | Рис.15 Копирование файлов | 15 |
| 5.2 | Рис.16 Редактирование программы | 15 |
| 5.3 | Рис.17 Копирование файлов и загрузка файлов на github | 16 |

Список таблиц

1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

1. В каталоге `~/work/arch-рс/lab04` с помощью команды `ср` создайте копию файла `hello.asm` с именем `lab4.asm`
2. С помощью любого текстового редактора внесите изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем.
3. Оттранслируйте полученный текст программы `lab4.asm` в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.
4. Скопируйте файлы `hello.asm` и `lab4.asm` в Ваш локальный репозиторий в каталог `~/work/study/2023-2024/“Архитектура компьютера”/arch-рс/labs/lab04/`. Загрузите файлы на Github

3 Теоретическое введение

3.1 Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора. Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве

и скорости программ, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: • для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM); • для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис. Более подробно о языке ассемблера см., например, в [10]. В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler) [7; 12; 14]. 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. Типичный формат записи команд NASM имеет вид: [метка:] мнемокод [операнд {, операнд}] [; комментарий]

В процессе создания ассемблерной программы можно выделить четыре шага:

- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`.
- Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`.
- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл.

Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение `map`. • Запуск программы. Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага. Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки.

4 Выполнение лабораторной работы

4.1 Программа Hello world!

1. Откройте терминал



Рис. 4.1: Рис.1 Терминал линукс

2. Создайте каталог для работы с программами на языке ассемблера NASM:

```
iarogozin@dk8n59 ~ $ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.2: Рис.2 Команда создания каталога для программ NASM

Перейдите в созданный каталог

```
iarogozin@dk8n59 ~ $ cd ~/work/arch-pc/lab04
```

Рис. 4.3: Рис.3 Переход в каталог

Создайте текстовый файл с именем hello.asm

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ touch hello.asm
```

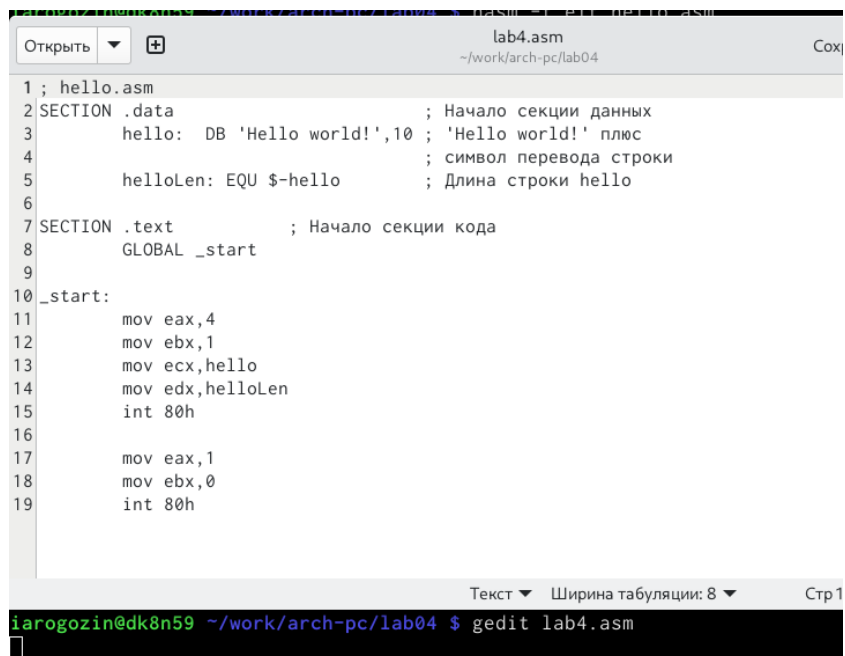
Рис. 4.4: Рис.4 Создание файла hello.asm

откройте этот файл с помощью любого текстового редактора, например, gedit

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ gedit hello.asm
```

Рис. 4.5: Рис.5 Редактор gedit

и введите в него следующий текст:



```
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
7 SECTION .text                ; Начало секции кода
8     GLOBAL _start
9
10 _start:
11     mov eax,4
12     mov ebx,1
13     mov ecx,hello
14     mov edx,helloLen
15     int 80h
16
17     mov eax,1
18     mov ebx,0
19     int 80h
```

Текст ▾ Ширина табуляции: 8 ▾ Стр 1

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ gedit lab4.asm
```

Рис. 4.6: Рис.6 Ввод текста программы

4.2 Транслятор NASM

1. NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо

написать:

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
```

Рис. 4.7: Рис.7 Компиляция программы

2. Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла `hello.asm` в объектный код, который запишется в файл `hello.o`. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появятся сообщения об ошибках или предупреждения.

Проверяем выполнение команды

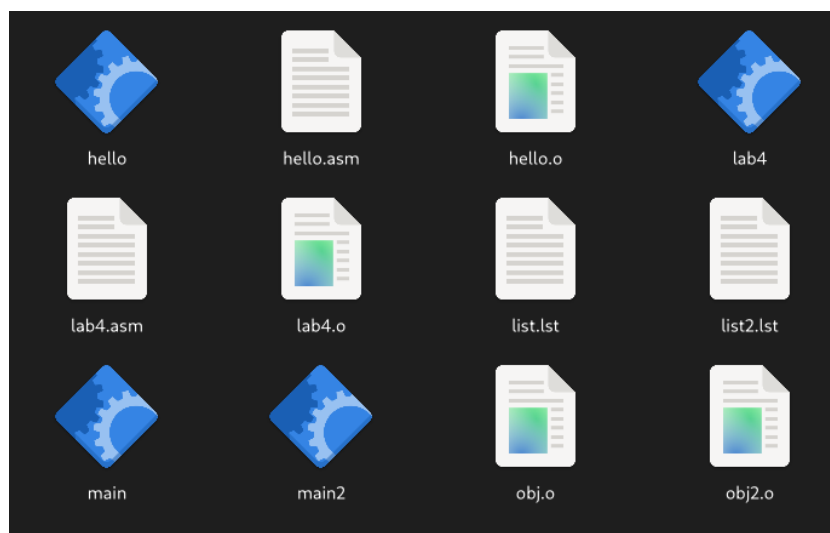


Рис. 4.8: Рис.8 Проверка выполнения

Объектный файл имеет имя: `hello.o`

4.3 Расширенный синтаксис командной строки NASM

1. Выполните следующую команду:

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
```

Рис. 4.9: Рис.9 Ввод команды

Данная команда скомпилирует исходный файл `hello.asm` в `obj.o` (опция `-o` позволяет задать имя объектного файла, в данном случае `obj.o`), при этом формат выходного файла будет `elf`, и в него будут включены символы для отладки (опция `-g`), кроме того, будет создан файл листинга `list.lst` (опция `-l`).

Проверьте, что файлы были созданы.

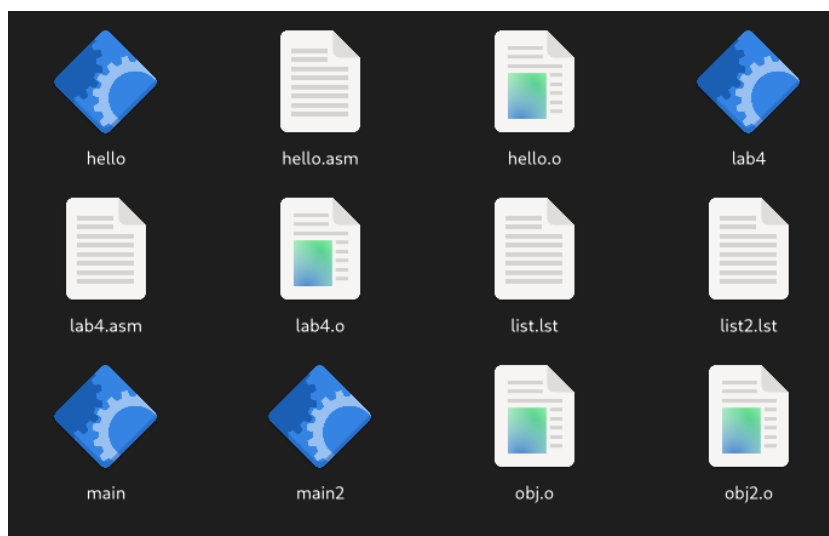


Рис. 4.10: Рис.10 Проверка выполнения

4.4 Компоновщик LD

1. Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
```

Рис. 4.11: Рис.11 Обработка программы компоновщиком

Проверьте, что исполняемый файл hello был создан

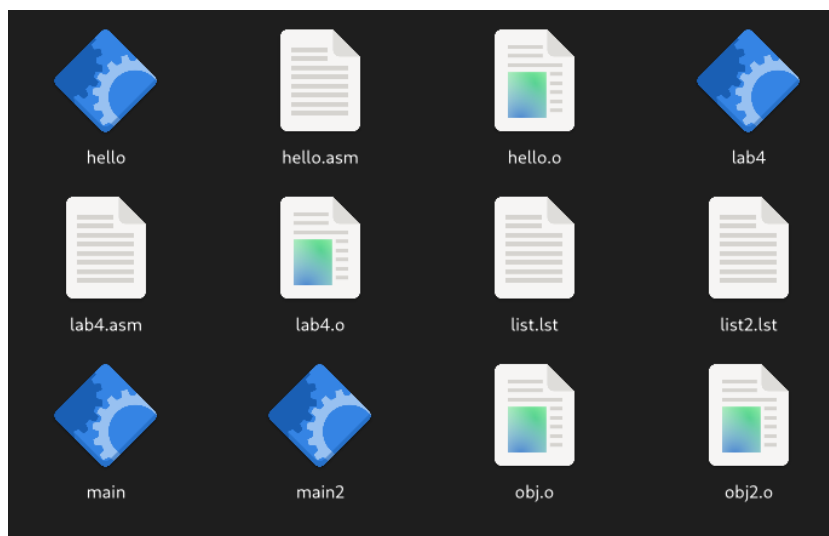


Рис. 4.12: Рис.12 Проверка выполнения

2. Выполните следующую команду:

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
```

Рис. 4.13: Рис.13 Выполнение команды

Исполняемый файл будет иметь имя: main

3. Запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке:

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ./hello  
Hello world!
```

Рис. 4.14: Рис.14 Выполнение команды

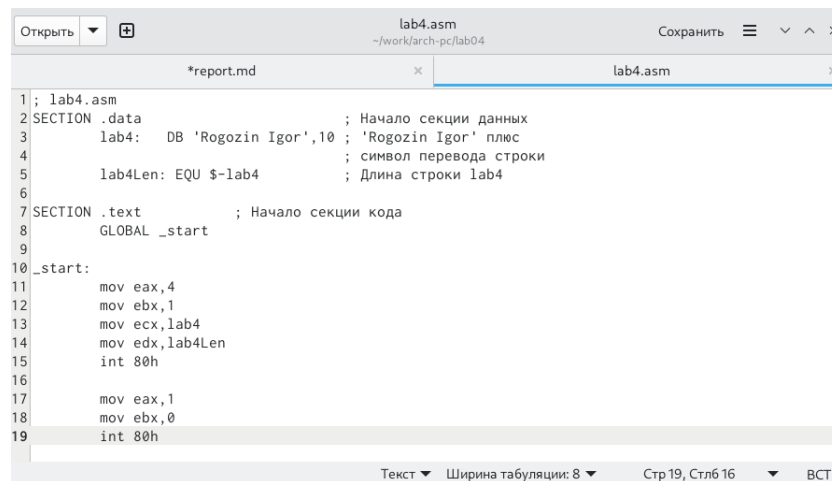
5 Задание для самостоятельной работы

1. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создайте копию файла hello.asm с именем lab4.asm

```
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cp lab4.asm ~/work/study/2023-2024/Архитектура\ компь  
ютера/arch-pc/labs/lab04/  
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cp hello.asm ~/work/study/2023-2024/Архитектура\ комп  
ютера/arch-pc/labs/lab04/
```

Рис. 5.1: Рис.15 Копирование файлов

2. С помощью любого текстового редактора внесите изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с вашими фамилией и именем.



```
1; lab4.asm  
2SECTION .data ; Начало секции данных  
3 lab4: DB 'Rogozin Igor',10 ; 'Rogozin Igor' плюс  
4 ; символ перевода строки  
5 lab4Len: EQU $-lab4 ; Длина строки lab4  
6  
7SECTION .text ; Начало секции кода  
8 GLOBAL _start  
9  
10_start:  
11 mov eax,4  
12 mov ebx,1  
13 mov ecx,lab4  
14 mov edx,lab4Len  
15 int 80h  
16  
17 mov eax,1  
18 mov ebx,0  
19 int 80h
```

Рис. 5.2: Рис.16 Редактирование программы

```

iarogozin@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj2.o -f elf -g -l list2.lst lab4.asm
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj2.o -o main2
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ ./lab4
Rogozin Igor

```

3. Оттранслируйте полученный текст программы lab4.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.
4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузите файлы на Github.

```

iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cp lab4.asm lab4.asm ~/work/study/2023-2024/Архитектура\
ра\ компьютера/arch-pc/labs/lab04/
cp: предупреждение: файл-источник 'lab4.asm' указан более одного раза
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cp lab4.asm ~/work/study/2023-2024/Архитектура\ компь
ютера/arch-pc/labs/lab04/
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cp hello.asm ~/work/study/2023-2024/Архитектура\ комп
ютера/arch-pc/labs/lab04/
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cd ~/work/
arch-pc/ study/
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cd ~/work/
arch-pc/ study/
iarogozin@dk8n59 ~/work/arch-pc/lab04 $ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-
h-pc
iarogozin@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git add .
iarogozin@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git commit -am 'feat
(main): add files lab-4'
[master 4fe3d55] feat(main): add files lab-4
3 files changed, 38 insertions(+)
create mode 100644 labs/lab03/report/report.docx
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
iarogozin@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git push
Перечисление объектов: 14, готово.
Подсчет объектов: 100% (14/14), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (9/9), готово.
Запись объектов: 100% (9/9), 519.54 КиБ | 5.59 МБ/с, готово.
Всего 9 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:LevinAE/study_2023-2024_arh-pc.git
b108e1f..4fe3d55 master -> master

```

Рис. 5.3: Рис.17 Копирование файлов и загрузка файлов на github

6 Выводы

Я научился писать небольшие программы и компилировать их на ассемблере NASM.