

Отчёт по лабораторной работе №3

Дисциплина: Архитектура компьютера

Буриева Шахзода Акмаловна

Содержание

1	Цель работы	5
2	Теоретическое введение	7
3	Выполнение лабораторной работы	9
4	Выводы	15
	Список литературы	16

Список иллюстраций

3.1	Создание каталогв для NASM	9
3.2	Переход в каталог	9
3.3	Переход в каталог	9
3.4	Открытие файла с помощью текстового редактора	9
3.5	Ввод данного текста	10
3.6	Команда <code>nasm -f elf hello.asm</code>	10
3.7	Проверка	11
3.8	Компиляция с помощью расширенного синтаксиса	11
3.9	Проверка	11
3.10	Передача компоновщику	11
3.11	Проверка	11
3.12	Команда <code>ld -m elf_i386 obj.o -o main</code>	12
3.13	Запуск программы	12
3.14	Копирование файла	12
3.15	Выведение имени и фамилии	13
3.16	Выполнение трансляции и компоновки	13
3.17	Копирование файла в локальный репозиторий	13
3.18	Отправка на github	14

Список таблиц

1 Цель работы

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

#Теоритическое введение

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 4.1). Взаимодействие этих устройств осуществляется через общую шину, к которой они подклю- чены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде элек- тропроводящих дорожек на материнской (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) входят следующие устройства: • арифметико-логическое устройство (АЛУ) — выполняет логические и арифметиче- ские 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 со- став 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.

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.

Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах)

2 Теоретическое введение

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 4.1). 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 (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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, пре-

образование (арифметические или логические операции) данных хранящихся в регистрах.

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.

Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах).

3 Выполнение лабораторной работы

Я создала каталог для работы с программами на языке ассемблера NASM.

```
saburieva@dk8n70 ~$ mkdir -p ~/work/arch-pc/lab04
saburieva@dk8n70 ~$
```

Рис. 3.1: Создание каталогв для NASM

Я перешла в созданный каталог.

```
saburieva@dk8n70 ~$ cd ~/work/arch-pc/lab04
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.2: Переход в каталог

Я создала текстовый файл с именем hello.asm.

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ touch hello.asm
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.3: Переход в каталог

Я открыла этот файл с помощью редактора gedit.

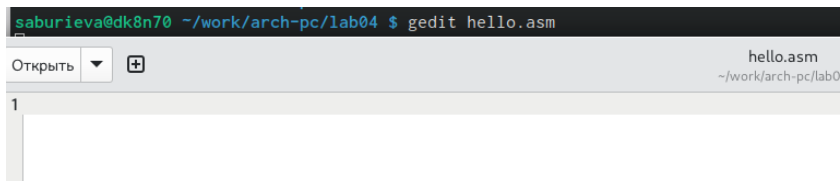


Рис. 3.4: Открытие файла с помощью текстового редактора

Я ввела 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 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. 3.5: Ввод данного текста

Для компиляции приведённого выше текста программы «Hello World» написала команду `nasm -f elf hello.asm`

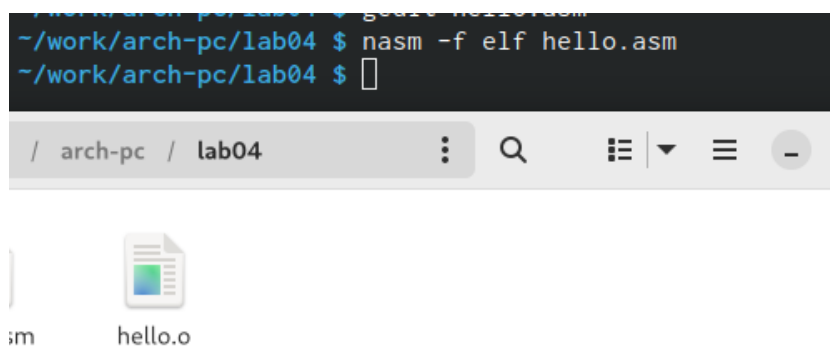


Рис. 3.6: Команда `nasm -f elf hello.asm`

Проверяю ее корректность и наличие

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.7: Проверка

Выполнила компиляцию файла с помощью расширенного синтаксиса командной строки.

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.8: Компиляция с помощью расширенного синтаксиса

Проверяю ее корректность и наличие

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.9: Проверка

Передала файл по обработке компоновщику.

```
hello.asm  hello.o  list.lst  obj.o
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Ключ -o с последующим значением задаёт в данном случае имя соз

Рис. 3.10: Передача компоновщику

Проверяю ее корректность и наличие.

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

собирает этот исполняемый файл?

Рис. 3.11: Проверка

Выполнила команду `ld -m elf_i386 obj.o -o main`. Исполняемый файл будет иметь имя `main`, объектный файл `obj.o`

```
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.12: Команда `ld -m elf_i386 obj.o -o main`

Запустила исполняемый файл.

```
Сообщения об ошибках отправляйте в <https://bugs.gentoo.org/>
saburieva@dk8n70 ~/work/arch-pc/lab04 $ ./hello
Hello world!
saburieva@dk8n70 ~/work/arch-pc/lab04 $
```

Рис. 3.13: Запуск программы

#Выполнение заданий для самостоятельной работы

Перешла в каталог `~/work/arch-pc/lab04`

```
saburieva@dk8n70 ~ $ cd ~/work/arch-pc/lab04
```

Скопировала файл

`hello.asm` с именем `lab04.asm`

```
hello hello.asm hello.o list.lst main obj.o
saburieva@dk8n37 ~/work/arch-pc/lab04 $ cp hello.asm lab04.asm
saburieva@dk8n37 ~/work/arch-pc/lab04 $
```

Рис. 3.14: Копирование файла

Изменила программу так, чтобы она вывела на терминал моё имя и фамилию.

```

1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Буриева Шахзода',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. 3.15: Выведение имени и фамилии

Транслирую текст программы в объектный файл, выполняю компоновку объектного файла и запускаю получившийся файл.

```

saburieva@dk3n37 ~/work/arch-pc/lab04 $ nasm -f elf lab04.asm
saburieva@dk3n37 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab04.o -o lab04
saburieva@dk3n37 ~/work/arch-pc/lab04 $ ./lab04
Буриева Шахзода
saburieva@dk3n37 ~/work/arch-pc/lab04 $

```

Рис. 3.16: Выполнение трансляции и компоновки

Скопировала файлы hello.asm и lab04.asm в мой локальный репозиторий

```

saburieva@dk3n61 $ cd ~/work/arch-pc/labs/
saburieva@dk3n61 ~/work/arch-pc/labs $ cp hello.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
saburieva@dk3n61 ~/work/arch-pc/labs $ cp lab04.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
cp: после 'lab04.asm' пропущен операнд, задающий целевой файл
По команде «cp --help» можно получить дополнительную информацию.
saburieva@dk3n61 ~/work/arch-pc/labs/lab04 $ cp lab04.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
saburieva@dk3n61 ~/work/arch-pc/labs/lab04 $

```

Рис. 3.17: Копирование файла в локальный репозиторий

Отправила лабораторную работу №4 на github.

```

saburleva@dk4n68 ~ $ cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc
saburleva@dk4n68 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git add .
saburleva@dk4n68 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git commit -am 'feat(main): add files lab-4'
[master 54f27a1] feat(main): add files lab-4
38 files changed, 201 insertions(+), 65 deletions(-)
create mode 100644 labs/lab02/report/image/1.jpg
create mode 100644 labs/lab02/report/image/10.jpg
create mode 100644 labs/lab02/report/image/11.jpg
create mode 100644 labs/lab02/report/image/12.jpg
create mode 100644 labs/lab02/report/image/13.jpg
create mode 100644 labs/lab02/report/image/14.jpg
create mode 100644 labs/lab02/report/image/15.jpg
create mode 100644 labs/lab02/report/image/16.jpg
create mode 100644 labs/lab02/report/image/17.jpg
create mode 100644 labs/lab02/report/image/18.jpg
create mode 100644 labs/lab02/report/image/19.jpg
create mode 100644 labs/lab02/report/image/2.jpg
create mode 100644 labs/lab02/report/image/20.jpg
create mode 100644 labs/lab02/report/image/3.jpg
create mode 100644 labs/lab02/report/image/4.jpg
create mode 100644 labs/lab02/report/image/5.jpg
create mode 100644 labs/lab02/report/image/6.jpg
create mode 100644 labs/lab02/report/image/7.jpg
create mode 100644 labs/lab02/report/image/8.jpg
create mode 100644 labs/lab02/report/image/9.jpg
delete mode 100644 labs/lab02/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab03/report/./-lock.report.docx#
create mode 100644 labs/lab03/report/image/1.jpg
create mode 100644 labs/lab03/report/image/2.jpg
create mode 100644 labs/lab03/report/image/3.jpg
create mode 100644 labs/lab03/report/image/4.jpg
create mode 100644 labs/lab03/report/image/5.jpg
create mode 100644 labs/lab03/report/image/6.jpg
create mode 100644 labs/lab03/report/image/7.jpg
create mode 100644 labs/lab03/report/image/8.jpg
create mode 100644 labs/lab03/report/image/9.jpg
delete mode 100644 labs/lab03/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab03/report/report.docx
create mode 100644 labs/lab03/report/report.pdf
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab04.asm
saburleva@dk4n68 ~/work/study/2023-2024/Архитектура компьютера/arch-pc $ git push
Перечисление объектов: 88, готово.
Подсчет объектов: 100% (83/83), готово.

```

Рис. 3.18: Отправка на github

4 Выводы

Я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Список литературы