Отчёт по лабораторной работе №3

Дисциплина: Архитектура компьютера

Буриева Шахзода Акмаловна

Содержание

Список иллюстраций

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 язы- ке 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 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, 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 язы- ке 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.

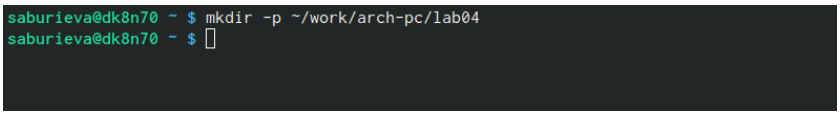


Рис. 3.1: Создание каталогв для NASM

Я перешла в созданный каталог.

Рис. 3.2: Переход в каталог

Я создала текстовый файл с именем hello.asm.

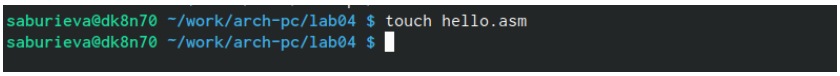


Рис. 3.3: Переход в каталог

Я открыла этот файл с помошью редактора gedit.

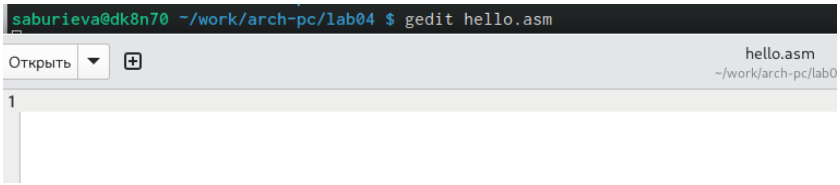


Рис. 3.4: Открытие файла с помощью текстового редактора

Я ввела в него следующий текст.

```
; hello.asm
SECTION .data ; Начало секции данных
hello: DB 'Hello world!',10 ; 'Hello world!' плюс
; 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. 3.5: Ввод данного текста

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

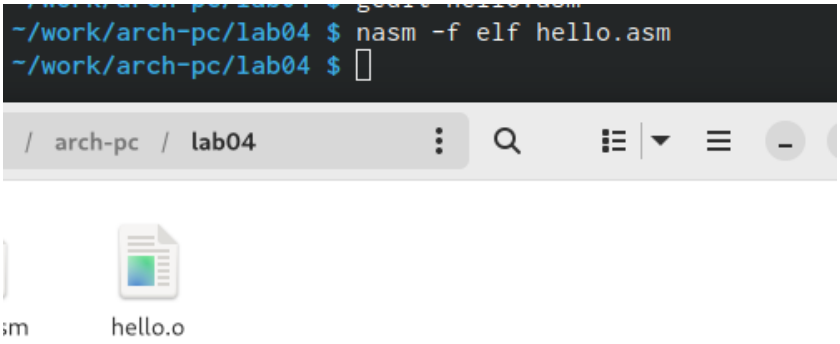


Рис. 3.6: Команда nasm -f elf hello.asm

Проверяю ее корректность и наличие

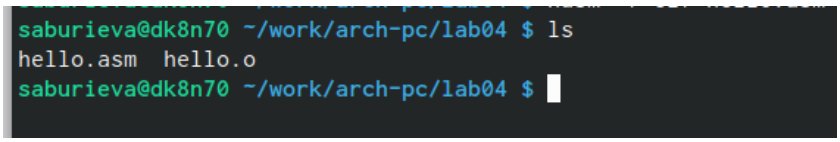


Рис. 3.7: Проверка

Выполнила компиляцию файла с помощью расширенного синтаксиса командной строки.

Рис. 3.8: Компиляция с помощью расширенного синтаксиса

Проверяю ее корректность и наличие

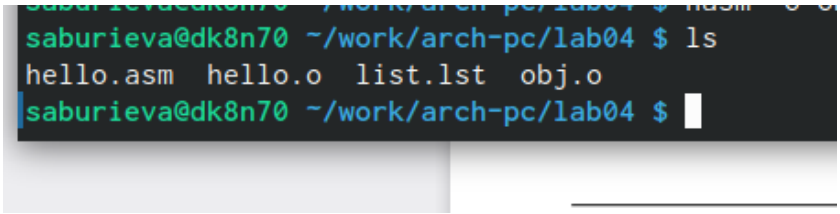


Рис. 3.9: Проверка

Передала файл по обработке компановщику.

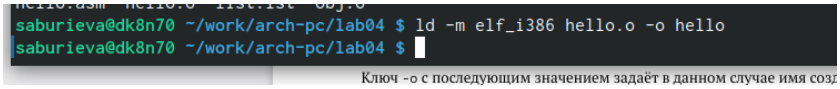


Рис. 3.10: Передача компановщику

Проверяю ее корректность и наличие.

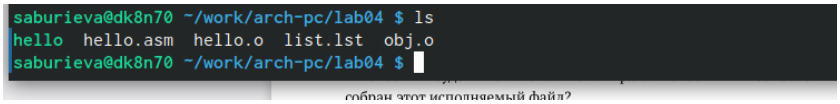


Рис. 3.11: Проверка

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

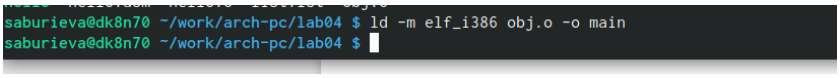


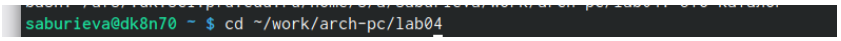
Рис. 3.12: Команда ld -m elf_i386 obj.o -o main

Запустила исполняемый файл.

Рис. 3.13: Запуск программы

#Выполнение заданий для самостоятельной работы

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



Скопировала файл hello.asm с именем lab04.asm

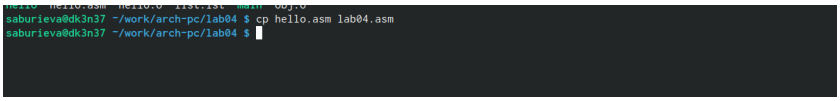


Рис. 3.14: Копирование файла

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

```
; hello.asm
SECTION .data ; Начало секции данных
hello: DB 'Буриева Шахзода',10 ; 'Hello world!' плюс
; 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. 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 ~/work/arch-pc/lab04 $ cp hello.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
saburieva@dk3n61 ~/work/arch-pc/lab04 $ cp lab04.asm
cp: после 'lab04.asm' пропущен операнд, задающий целевой файл
По команде «cp --help» можно получить дополнительную информацию.
saburieva@dk3n61 ~/work/arch-pc/lab04 $ cp lab04.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
saburieva@dk3n61 ~/work/arch-pc/lab04 $
```

Рис. 3.17: Копирование файла в локальный репозиторий

Отправила лабораторную работу №4 на github.

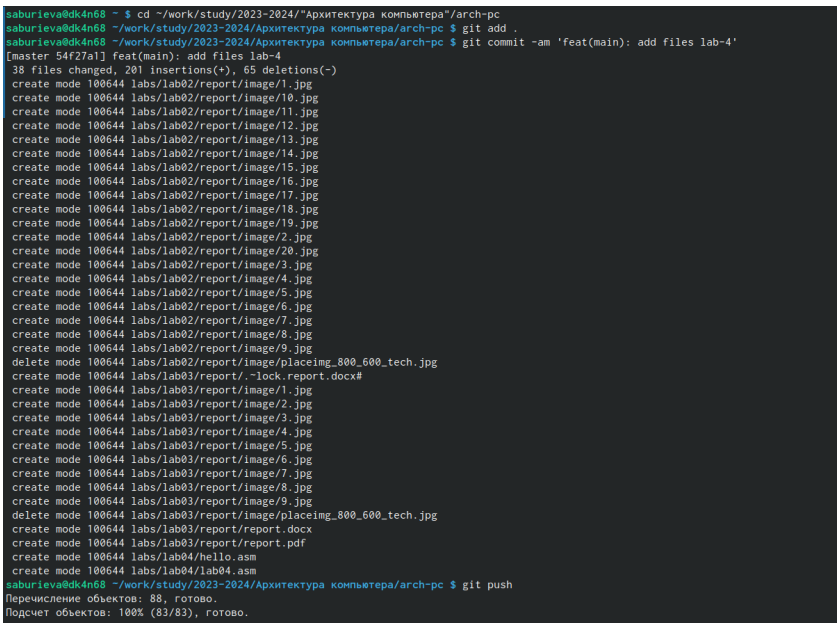


Рис. 3.18: Отправка на github

4 Выводы

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

Список литературы