**Отчёт по лабораторной работе №4**

***Дисциплина: Архитектура компьютера***

Байрамова Гюльсабах Акифовна НММбд-01-24

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

11

3.3 Процесс создания и обработки программы на языке ассемблера .

.

13

**4 Выполнение лабораторной работы**

**15**

**5 Выводы**

**19**

2

.

**Содержание**

**1 Цель работы**

**5**

**2 Задание**

**6**

**3 Теоретическое введение**

**7**

3.1 Основные принципы работы компьютера.

.

.

.

.

.

.

.

.

.

.

.

.

.

7

3.2 Ассемблер и язык ассемблера .

.

.

.

.

.

.

.

.

.

.

.

.

.

4.10 Имзменение программы ..

17

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.11 Получение объектного файла .

17

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.8 Запуск исполняемого файла.

16

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.9 Создание копии файла ..

17

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.15 Загрузка на Github.

18

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.14 Копирование в локальный репозиторий .

18

.

.

.

.

.

3

18

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.12 Компановка объектного файла .

.

.

.

.

.

.

.

.

.

.

.

.

.

.

17

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.13 Запуск исполняемого файла.

18

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.1 Создание нужного каталога .

13

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.2 Создание файла hello.asm ..

15

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

7

.

.

.

.

.

.

.

.

.

.

3.1 Структурная схема ЭВМ ..

**Список иллюстраций**

.

.

.

.

.

.

3.3 Процесс создания ассемблерной программы .

9

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

3.2 64-битный регистр процессора ‘RAX’

.

16

.

.

.

.

.

.

.

.

.

.

.

.

4.6 Обработка компановщиком ..

.

.

.

4.5 Компиляция исходного файла в новый .

16

.

.

.

.

.

.

.

.

.

.

4.7 Создание исполняемого файла .

16

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.3 Код программы .

15

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

4.4 Компиляция программы ..

15

.

.

.

.

.

.

.

**Список таблиц**

4

**1 Цель работы**

Освоение процедуры компиляции и сборки программ, написанных на ассем-

блере NASM.

5

**2 Задание**

1) Выполнение лабораторной работы

1) Программа Hello world!

2) Транслятор NASM

3) Расширенный синтаксис командной строки NASM

4) Компоновщик LD

5) Запуск исполняемого файла

2) Задания для самостоятельной работы

6

**3 Теоретическое введение**

**3.1 Основные принципы работы компьютера**

Основными функциональными элементами любой электронно-вычислительной

машины(ЭВМ) являются центральный процессор,память и периферийные

устройства (рис. 3.1).

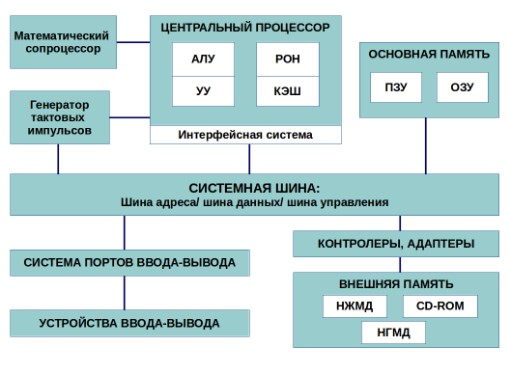


Рис. 3.1: Структурная схема ЭВМ

Взаимодействие этих устройств осуществляется через общую шину, к которой

они подключены. Физически шина представляет собой большое количество про-

водников, соединяющих устройства друг с другом. В современных компьютерах

проводники выполнены в виде электропроводящих дорожек на материнской

7

регистрам осуществляется не по адресам, как к основной памяти, а по именам.

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 каче-

стве операндов. Практически все команды представляют собой преобразование

данных хранящихся в регистрах процессора, это например пересылка данных

между регистрами или между регистрами и памятью, преобразование (арифме-

тические или логические операции) данных хранящихся в регистрах. Доступ к

процессора делятся на два типа: регистры общего назначения и

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 ис-

пользуются при написании программ):

1.

RAX, RCX, RDX, RBX, RSI, RDI — 64-битные

8

промежуточных результатов выполнения инструкций; регистры

ёма, входящая в состав процессора, для временного хранения

регистры — сверхбыстрая оперативная память небольшого объ-

3.

троль всех устройств компьютера;

устройство управления (УУ) — обеспечивает управление и кон-

2.

информации, хранящейся в памяти;

ские и арифметические действия, необходимые для обработки

арифметико-логическое устройство (АЛУ) — выполняет логиче-

1.

(ЦП) входят следующие устройства:

зация координации всех узлов компьютера. В состав центрального процессора

Основной задачей процессора является обработка информации, а также органи-

(системной) плате.

mov eax, 1

9

щий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер

программ и данных, с которыми процессор непосредственно работает в теку-

прямую взаимодействует с узлами процессора, предназначенное для хранения

быстродействующее энергозависимое запоминающее устройство, которое на-

узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это

будет какое-то число, но не 1. А вот в регистре AX будет число 1. Другим важным

ные, отличные от нуля, то после выполнения первой команды в регистре EAX

оставит в старших разрядах регистра EAX старые данные. И если там были дан-

выполнения второй команды в регистре EAX будет число 1. А первая команда

в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после

Обе команды поместят в регистр AX число 1. Разница будет заключаться только

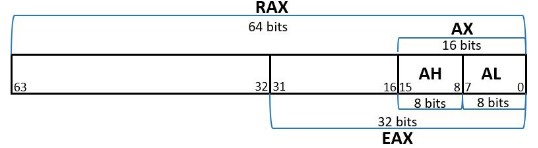
mov ax, 1

ра):

например,такие команды (mov – команда пересылки данных на языке ассембле-

Таким образом можно отметить, что вы можете написать в своей программе,

Рис. 3.2: 64-битный регистр процессора ‘RAX’



AL (low AX) — младшие 8 бит регистра AX.

регистров). Например, AH (high AX) — старшие 8 бит регистра AX,

4. AH, AL, CH, CL, DH, DL, BH, BL — 8-битные (половинки 16-битных

3. AX, CX, DX, BX, SI, DI — 16-битные

EAX, ECX, EDX, EBX, ESI, EDI — 32-битные

2.

считывание кода команды из памяти и её дешифрация;

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. В самом общем виде он заключается

в следующем:

1.

формирование адреса в памяти очередной команды;

2.

коде машинной команды можно выделить две части: операционную и адресную.

3.

выполнение команды;

4.

переход к следующей команде.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме

того, в зависимости от команды при её выполнении могут проходить не все

этапы.

10

которые обеспечивают взаимодей-

ячейки памяти — это адрес хранящихся в ней данных. В состав ЭВМ также входят

периферийные устройства, которые можно разделить на:

1.

устройства внешней памяти, которые предназначены для долго-

временного хранения больших объёмов данных (жёсткие диски,

твердотельные накопители, магнитные ленты);

2.

устройства ввода-вывода,

ствие ЦП с внешней средой.

В основе вычислительного процесса ЭВМ лежит принцип программного управ-

ления. Это означает, что компьютер решает поставленную задачу как последо-

вательность действий, записанных в виде программы. Программа состоит из

машинных команд, которые указывают, какие операции и над какими данными

(или операндами), в какой последовательности необходимо выполнить. Набор

машинных команд определяется устройством конкретного процессора. Коды

команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В

качестве и скорости программам, написанным на машинном языке, так как

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

для человека образом пишутся команды для процессора. Следует отметить, что

транслятор просто переводит мнемонические обозначения команд в последова-

тельности 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 ассем-

блер и, соответственно, свой язык ассемблера. Наиболее распространёнными

ассемблерами для архитектуры x86 являются:

11

ностям, что позволяет получить к ним более полный доступ, нежели в языках

**3.2 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 возмож-

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

торая является обязательной частью команды. Операндами могут быть числа,

12

обычно пишутся большими буквами.

директивы используются для определения данных (констант и переменных) и

ственно в машинные команды, а управляющие работой транслятора. Например,

также может содержать директивы — инструкции, не переводящиеся непосред-

мальная длина идентификатора 4095 символов. Программа на языке ассемблера

чтобы компилятор трактовал его верно (так называемое экранирование). Макси-

фикаторами, которые пишутся как зарезервированные слова, нужно писать $,

Начинаться метка или идентификатор могут с буквы, ., \_ и ?. Перед иденти-

\_, $, #, @,~,. и ?.

символы:

пустимыми символами в метках являются буквы, цифры, а также следующие

в памяти. Т.о. метка перед командой связана с адресом данной команды. До-

фикатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес

данные, адреса регистров или адреса оперативной памяти. Метка — это иденти-

Здесь мнемокод — непосредственно мнемоника инструкции процессору, ко-

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

струкции x86-64. Типичный формат записи команд NASM имеет вид:

для этих систем. В NASM используется Intel-синтаксис и поддерживаются ин-

личные операционные системы и который позволяет получать объектные файлы

NASM — это открытый проект ассемблера, версии которого доступны под раз-

ассемблеров, которые используют Intel-синтаксис.

синтаксис, в отличие от большинства других популярных

использующий AT&T-

gas (GNU Assembler),

для GNU/Linux:

2.

Assembler (MASM) и Watcom assembler (WASM);

для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro

1.

1.

13

мер nasm, текста программы в машинный код, называемый объ-

Трансляция — преобразование с помощью транслятора, напри-

2.

текстом программ на языке ассемблера имеют тип asm.

ние), который определяет назначение файла. Файлы с исходным

в отдельном файле. Каждый файл имеет свой тип (или расшире-

Набор текста программы в текстовом редакторе и сохранение её

В процессе создания ассемблерной программы можно выделить четыре шага:

Рис. 3.3: Процесс создания ассемблерной программы



щей схемы (рис. 3.3).

Процесс создания ассемблерной программы можно изобразить в виде следую-

**ассемблера**

**3.3 Процесс создания и обработки программы на языке**

4.

14

средах).

(хотя поддержка ассемблера есть в некоторых универсальных интегрированных

грамм на языке ассемблера обычно пользуются утилитами командной строки

Из-за специфики программирования, а также по традиции для создания про-

провести коррекцию программы, начиная с первого шага.

программы — отладчика. При нахождении ошибки необходимо

сутствовать этап отладки программы при помощи специальной

вести к некорректной работе программы, поэтому может при-

исполняемый файл. Ошибки на предыдущих этапах могут при-

Запуск программы. Конечной целью является работоспособный

загрузки программы в ОЗУ, имеющий расширение map.

не имеет расширения. Кроме того, можно получить файл карты

собирает по ним исполняемый файл. Исполняемый файл обычно

поновщиком (ld), который принимает на вход объектные файлы и

Компоновка или линковка — этап обработки объектного кода ком-

3.

объектного файла — o, файла листинга — lst.

дополнительную информацию, созданную транслятором. Тип

программы, содержащий кроме текста программы различную

ектным. На данном этапе также может быть получен листинг

**4 Выполнение лабораторной работы**

Рассмотрим пример простой программы на языке ассемблера NASM. Тради-

ционно первая программа выводит приветственное сообщение Hello world! на

экран. Создадим каталог для работы с программами на языке ассемблера NASM

и перейдём в него (рис. 4.1):



Рис. 4.1: Создание нужного каталога

Создим текстовый файл с именем hello.asm и откроем его (рис. 4.2):



Рис. 4.2: Создание файла hello.asm

и введём в него следующий текст(рис. 4.3):

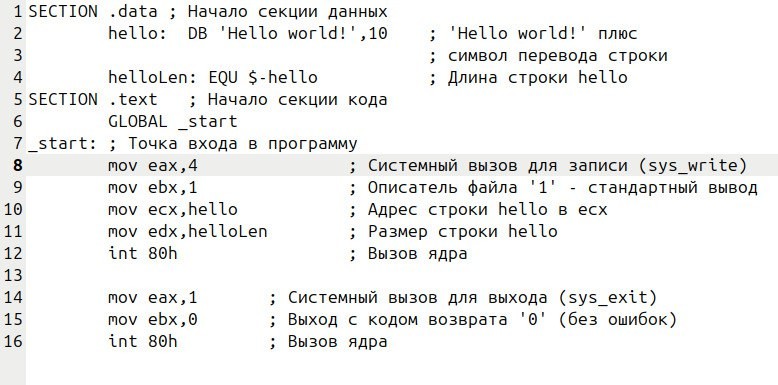


Рис. 4.3: Код программы

15

воляет задать имя объектного файла, в данном случае obj.o), при этом формат

16

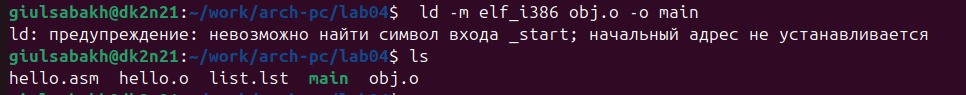
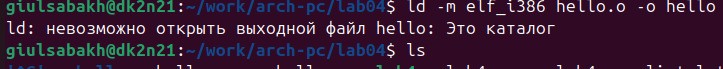


Рис. 4.7: Создание исполняемого файла

исполняемого файла. Выполним следующую команду (рис. 4.7):

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

Рис. 4.6: Обработка компановщиком



4.6):

работку компоновщику и проверим, что исполняемый файл был создан (рис.

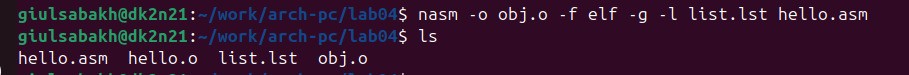
Чтобы получить исполняемую программу, передадим объектный файл на об-

-g), кроме того, будет создан файл листинга list.lst (опция -l).

выходного файла будет elf, и в него будут включены символы для отладки (опция

Данная команда скомпилирует исходный файл hello.asm в obj.o (опция -o поз-

Рис. 4.5: Компиляция исходного файла в новый



файлы были созданы (рис. 4.5):

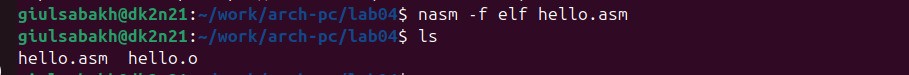
и расширения по умолчанию. Выполните следующую команду и проверим, что

hello.o. Таким образом, имена всех файлов получаются из имени входного файла

программы из файла hello.asm в объектный код, который запишется в файл

Если текст программы набран без ошибок, то транслятор преобразует текст

Рис. 4.4: Компиляция программы



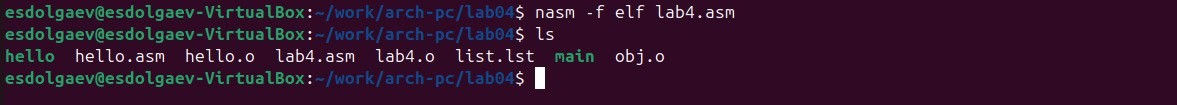
димо написать (рис. 4.4):

Для компиляции приведённого выше текста программы «Hello World» необхо-

С помощью текстового редактора внесём изменения в текст программы в файле

17

Рис. 4.11: Получение объектного файла

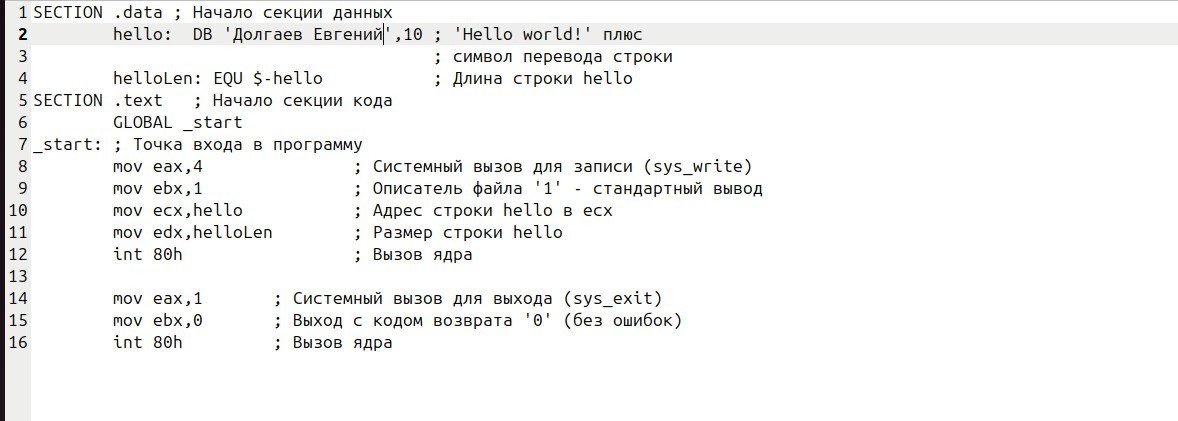


файл(рис. 4.11, 4.12).

полним компоновку объектного файла и запустим получившийся исполняемый

Оттранслируем полученный текст программы lab4.asm в объектный файл. Вы-

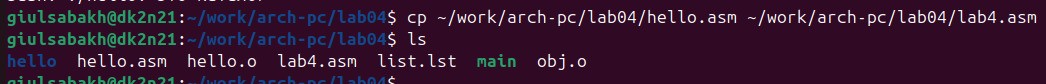
Рис. 4.10: Имзменение программы



и именем(рис. 4.10).

lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с фамилией

Рис. 4.9: Создание копии файла

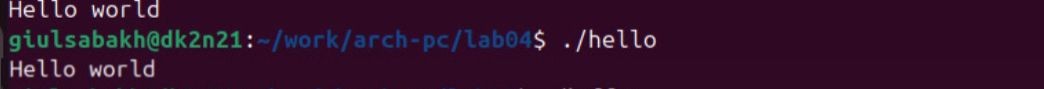


именем lab4.asm(рис. 4.9):

~/work/arch-pc/lab04 с помощью команды cp создим копию файла hello.asm с

Привступим к выполнению заданий для самостоятельной работы. В каталоге

Рис. 4.8: Запуск исполняемого файла



находящийся в текущем каталоге, можно, набрав в командной строке (рис. 4.8):

hello и hello.o (рис. 4.6)). Запустить на выполнение созданный исполняемый файл,

Таким образом, исполняемый файл имеет имя main, а объектный obj.o(или

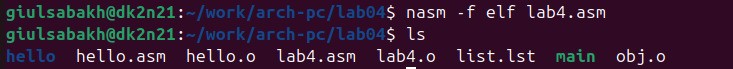


Рис. 4.12: Компановка объектного файла

Скопируем файлы hello.asm и lab4.asm в локальный репозиторий в каталог

~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузим

файлы на Github(рис. 4.14, 4.15).

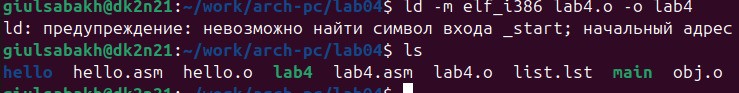


Рис. 4.14: Копирование в локальный репозиторий

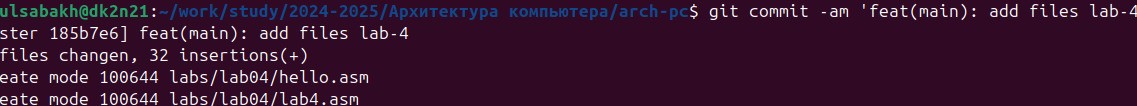


Рис. 4.15: Загрузка на Github

18

**5 Выводы**

Благодаря этой лаборатоной работе, я освоил компиляцию и сборку программ,

написанных на ассемблере NASM.

19