

Отчет по лабораторной работе №4

Дисциплина: Архитектура ЭВМ

Дагделен Зейнап Реджеповна

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	9
3.3	Процесс создания и обработки программы на языке ассемблера .	9
4	Выполнение лабораторной работы	11
4.1	Программа Hello world!	11
4.2	Транслятор NASM	12
4.3	Компоновщик LD	13
4.4	Запуск исполняемого файла	14
4.5	Задание для самостоятельной работы	14
5	Выводы	18
6	Список литературы	19

Список иллюстраций

4.1	Создание каталога.	11
4.2	Создание hello.asm	11
4.3	Открытие файла hello.asm	12
4.4	Скачивание программы nasm	12
4.5	Компиляция текста программы (у меня 64-битная версия Linux)	13
4.6	Компиляция исходного файла в obj.o	13
4.7	Передача файла на обработку компоновщику	13
4.8	Передача файла на обработку компоновщику	14
4.9	Запуск исполняемого файла	14
4.10	Копирование файлов в нужную папку	14
4.11	Внесение изменений в текст программы	15
4.12	Транслирование в объектный файл (Компиляция текста программы)	15
4.13	Передача объектного файла на обработку компоновщику	15
4.14	Запуск исполняемого файла	16
4.15	Копирование файлов из одной папки в другую	16
4.16	Добавление файлов на GitHub	16
4.17	Отправка файлов	17

Список таблиц

1 Цель работы

Освоение 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

3.1 Основные принципы работы компьютера

Основными функциональными элементами любой ЭВМ являются *центральный процессор, память и периферийные устройства*.

Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. 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: - регистры общего назначения - специальные регистры. 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, а по именам. Каждый

регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита.

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. В состав ЭВМ также входят периферийные устройства, которые можно разделить на: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты); - устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой.

В основе вычислительного процесса ЭВМ лежит *принцип программного управления*.

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: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. переход к следующей команде. В зависимости от команды при её выполнении могут проходить не все этапы.

3.2 Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. 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. 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 программам, 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. Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера.

3.3 Процесс создания и обработки программы на языке ассемблера

В процессе создания ассемблерной программы можно выделить четыре шага:

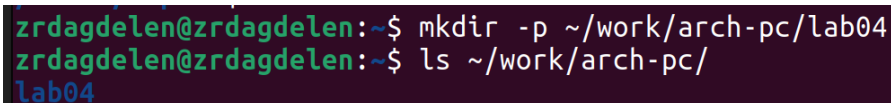
- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле.
- Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным.
- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл.
- Запуск программы. Из-за специфики программирования, а также по традиции для со-

здания программ на языке ассемблера обычно пользуются утилитами командной строки.

4 Выполнение лабораторной работы

4.1 Программа Hello world!

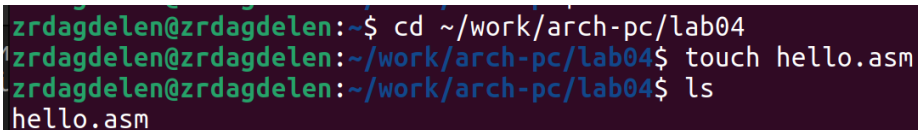
Создаю каталог для работы с программами на языке ассемблера NASM с помощью команды `mkdir` и проверяю(рис.[4.1]).



```
zrdagdelen@zrdagdelen:~$ mkdir -p ~/work/arch-pc/lab04
zrdagdelen@zrdagdelen:~$ ls ~/work/arch-pc/
lab04
```

Рис. 4.1: Создание каталога.

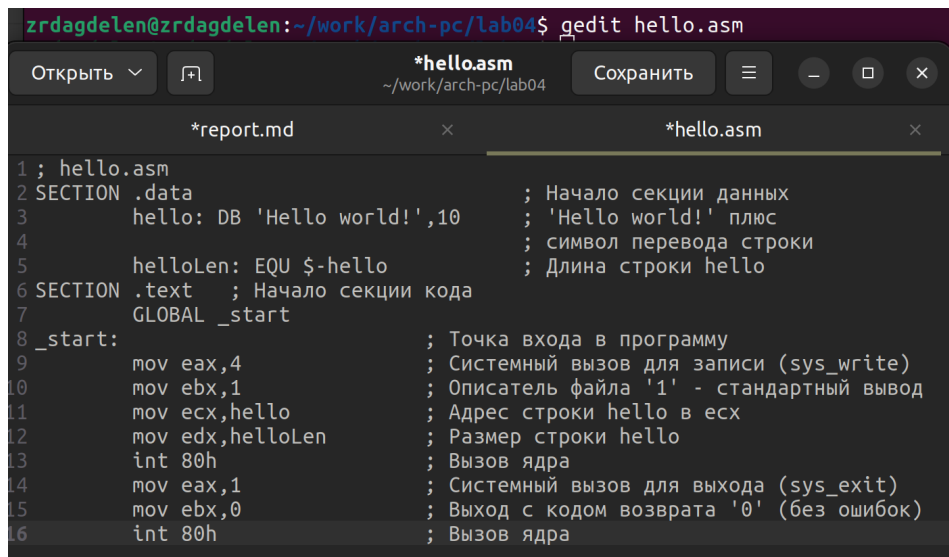
Перехожу в созданный каталог и с помощью `touch` создаю текстовый файл с именем `hello.asm`, проверяю (рис.[4.2]).



```
zrdagdelen@zrdagdelen:~$ cd ~/work/arch-pc/lab04
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ touch hello.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello.asm
```

Рис. 4.2: Создание `hello.asm`

Открываю этот файл с помощью текстового редактора `gedit` и ввожу в него необходимый текст (рис.[4.3]).

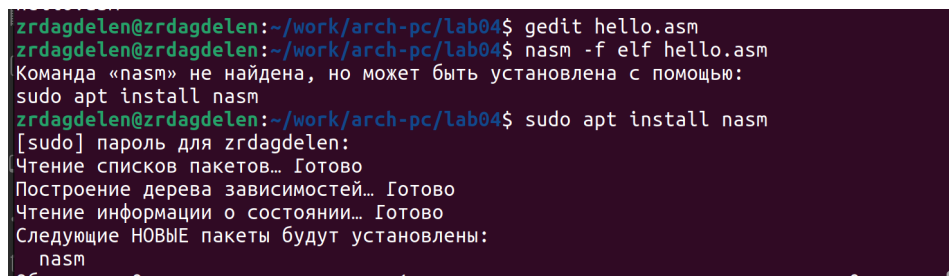


```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ gedit hello.asm
Открыть ▾ [+1] *hello.asm ~/work/arch-pc/lab04 Сохранить ≡ - □ ×
*report.md × *hello.asm ×
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3     hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4                                     ; 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. 4.3: Открытие файла hello.asm

4.2 Транслятор NASM

NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать так (рис. [4.5]), скачав команду nasm(рис.[4.4]).



```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ gedit hello.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ nasm -f elf hello.asm
Команда «nasm» не найдена, но может быть установлена с помощью:
sudo apt install nasm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ sudo apt install nasm
[sudo] пароль для zrdagdelen:
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Следующие НОВЫЕ пакеты будут установлены:
  nasm
Обновлено 0 пакетов, установлено 1 новых пакетов, для обновления отмечено 0 пакетов
```

Рис. 4.4: Скачивание программы nasm

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ nasm -f elf64 hello.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.5: Компиляция текста программы (у меня 64-битная версия Linux)

Так как текст программы был набран без ошибок, транслятор преобразовал текст программы из файла `hello.asm` в объектный код, который записался в файл `hello.o`. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. ## Расширенный синтаксис командной строки NASM Скомпилирую 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 `elf64`, и в него будут включены символы для отладки (опция `-g`), кроме того, будет создан файл листинга `list.lst` (опция `-l`); с помощью команды `ls` проверяю, что файлы были созданы (рис.[4.6]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ nasm -o obj.o -f elf64 -g -l list.lst hello.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello.asm  hello.o  list.lst  obj.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.6: Компиляция исходного файла в `obj.o`

4.3 Компоновщик LD

Чтобы получить исполняемую программу, объектный файл передаю на обработку компоновщику (рис.[4.7]). Ключ `-o` с последующим значением задаёт в данном случае имя создаваемого исполняемого файла. Проверяю с помощью `ls`.

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ld -m elf_x86_64 hello.o -o hello
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.7: Передача 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`.

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ld -m elf_x86_64 obj.o -o main
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рис. 4.8: Передача файла на обработку компоновщику

4.4 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл `hello` (рис. [4.9]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ./hello
Hello world!
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.9: Запуск исполняемого файла

4.5 Задание для самостоятельной работы

В каталоге `~/work/arch-pc/lab04` с помощью команды `cp` создаю копию файла `hello.asm` с именем `lab4.asm` (рис. [4.10]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

Рис. 4.10: Копирование файлов в нужную папку

С помощью текстового редактора вношу изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с моей фамилией и именем (рис. [4.11]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ gedit lab4.asm

report.md x lab4.asm x
lab4.asm
~/work/arch-pc/lab04 Сохранить

1 ; lab4.asm
2 SECTION .data ; Начало секции данных
3 lab4: DB 'Dagdelen Zeynap',10 ; 'Dagdelen Zeynap' плюс
4 ; символ перевода строки
5 lab4Len: EQU $-lab4 ; Длина строки lab4
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,lab4 ; Адрес строки hello в ecx
12 mov edx,lab4Len ; 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. 4.11: Внесение изменений в текст программы

Оттранслирую полученный текст программы lab4.asm в объектный файл (рис. [4.12]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ nasm -f elf64 lab4.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.12: Транслирование в объектный файл (Компиляция текста программы)

Выполняю компоновку объектного файла (рис.[4.13]) и запускаю получившийся исполняемый файл(рис.[4.14]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ld -m elf_x86_64 lab4.o -o lab4
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.13: Передача объектного файла на обработку компоновщику

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ ./lab4
Dagdelen Zeynap
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$
```

Рис. 4.14: Запуск исполняемого файла

Копирую файлы hello.asm и lab4.asm в свой локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/ (рис.[4.15]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/"Архитектура
компьютера"/arch-pc/labs/lab04
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2023-2024/"Архитектура к
омпьютера"/arch-pc/labs/lab04
zrdagdelen@zrdagdelen:~/work/arch-pc/lab04$ cd ~/work/study/2023-2024/"Архитектура компьюте
ра"/arch-pc/labs/lab04
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$ ls
hello.asm lab4.asm presentation report
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04$
```

Рис. 4.15: Копирование файлов из одной папки в другую

Загружаю файлы на Github: 1. С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. [4.16]).

```
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьюте
ра/arch-pc/labs/lab04$ git add .
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьюте
ра/arch-pc/labs/lab04$ git commit -m "Add fales for lab04"
[master 0aa7f34] Add fales for lab04
3 files changed, 180 insertions(+), 119 deletions(-)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
rewrite labs/lab04/report/report.md (66%)
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьюте
ра/arch-pc/labs/lab04$
```

Рис. 4.16: Добавление файлов на GitHub

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


```
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьюте  
pa/arch-pc/labs/lab04$ git push  
Everything up-to-date  
zrdagdelen@zrdagdelen:~/work/study/2023-2024/Архитектура компьюте  
pa/arch-pc/labs/lab04$
```

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

Архитектура АВМ [Ссылка на GitHub](#)