

Российский университет дружбы народов
Факультет физико-математически и естественных наук

Отчёт
по лабораторной работе №4

по дисциплине:
архитектура компьютеров и операционные
системы

Студент: Аманов Рустам Маркович:НКАбд 01-

23

№ ст. Билета: 1032234130

МОСКВА
2023 Г

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
5	Выполнение заданий для самостоятельной работы	13
6	Выводы	16
	Список литературы	17

Список иллюстраций

4.1 Создание папкиОшибка! Закладка не определена.

4.2	Переход в созданную папку	7
4.3	Создание файла	7
4.4	Заполнение файла	8
4.5	Скачивание	8
4.6	Преобразование файла в объектный код	8
4.7	Преобразование файла	8
4.8	Передача файла на обработку	8
4.9	1.....	9
4.10	2.....	9
5.1	Копирование файла	9
5.2	Файл в nano	9
5.3	Транслирую файл	9
5.4	Компановка и исполнение	10
5.5	Копирование файлов	10
5.6	Выгружаю изменения	10

Списоктаблиц

1. Цельработы

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

1 Задание

- 1) Создать программу Hello world
- 2) Работа с транслятором NASM
- 3) Работа с расширенным синтаксисом командой строки NASM
- 4) Работа с компоновщиком LD
- 5) Запуск исполняемого файла
- 6) Выполнение заданий для самостоятельной работы

2 Теоретическое введение

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. 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. В качестве примера приведем названия основных регистров

общего назначения (именно эти регистры чаще всего используются при написании программ):

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

- EAX, ECX, EDX, EBX, ESI, EDI — 32-битные
- AX, CX, DX, BX, SI, DI — 16-битные
- AH, AL, CH, CL, DH, DL, BH, BL — 8-битные (половинки 16-битных регистров). Например,

AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX. В состав ЭВМ также входят периферийные устройства, которые можно разделить на: - устройства внешней памяти, которые предназначены для длительного 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. Это означает, что компьютер решает поставленную задачу как последовательность 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 многобитные двоичные комбинации из 0 и 1. 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. В самом общем виде он заключается в следующем:

- 1) формирование адреса в памяти очередной команды;
- 2) считывание кода команды из памяти и её дешифрация;
- 3) выполнение команды;
- 4) переход к следующей команде.

3 Выполнение лабораторной работы

- 1) Создаю рекурсивно вложенные в папку work папки arch-pc и lab04, проверяю их создание

```
(rustam01@kali)-[~]  
$ mkdir -p ~/work/arch-pc/lab04  
  
(rustam01@kali)-[~]  
$ ls ~/work/arch-pc  
lab04
```

Рис. 4.1: Создание папки

- 2) Перехожу в созданную папку

```
(rustam01@kali)-[~]  
$ cd ~/work/arch-pc/lab04  
  
(rustam01@kali)-[~/work/arch-pc/lab04]  
$
```

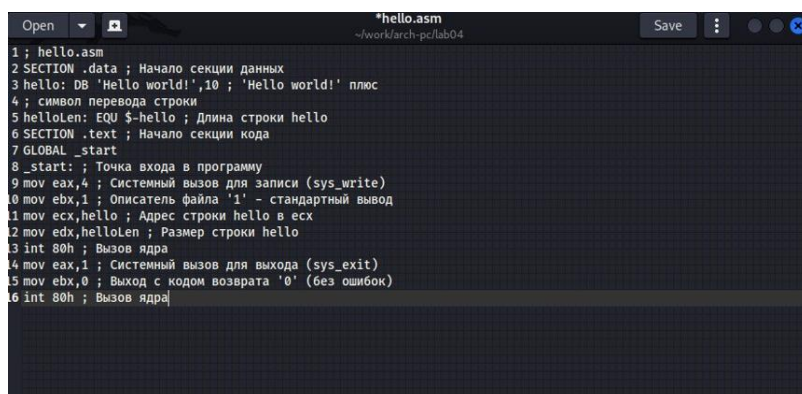
Рис. 4.2: Переход в созданную папку

- 3) Создаю файл hello с разрешением asm и проверяю его создание

```
(rustam01@kali)-[~/work/arch-pc/lab04]  
$ touch hello.asm  
  
(rustam01@kali)-[~/work/arch-pc/lab04]  
$ ls  
hello.asm
```

Рис. 4.3: Создание файла

- 4) Открываю этот файл в папо и копирую туда код из задания лабораторной работы



```
Open  Save  [Icons]  
*hello.asm  
~/work/arch-pc/lab04  
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. 4.4: Заполнение файла

5) Скачиваю nasm

```
(rustam01@kali)~[/work/arch-pc/lab04]
$ sudo apt install nasm
[sudo] password for rustam01:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nasm is already the newest version (2.16.01-1).
0 upgraded, 0 newly installed, 0 to remove and 1099 not upgraded.
```

Рис. 4.5: Скачивание

6) Преобразовываю файл hello.asm в объектный код, записанный в файл hello.o. Проверяю, был ли создан файл

```
(rustam01@kali)~[/work/arch-pc/lab04]
$ nasm -f elf hello.asm

(rustam01@kali)~[/work/arch-pc/lab04]
$ ls
hello.asm hello.o
```

Рис. 4.6: Преобразование файла в объектный код

7) Преобразую файл hello.asm в obj.o с помощью опции -o, которая позволяет задать имя объекта. Из-за elf -g формат выходного файла будет elf, и в него будут включены символы для отладки, а так же будет создан файл листинга list.lst, благодаря -l. Проверяю созданные файлы

```
(rustam01@kali)~[/work/arch-pc/lab04]
$ nasm -o obj.o -f elf -g -l list.lst hello.asm

(rustam01@kali)~[/work/arch-pc/lab04]
$ ls
hello.asm hello.o list.lst obj.o
```

Рис. 4.7: Преобразование файла

8) Передаю файл компоновщику с помощью ld. Проверяю, создан ли исполняемый файл

```
(rustam01@kali)~[/work/arch-pc/lab04]
$ ld -m elf_i386 hello.o -o hello

(rustam01@kali)~[/work/arch-pc/lab04]
$ ls
hello hello.asm hello.o list.lst obj.o
```

Рис. 4.8: Передача файла на обработку

9) Передаю компоновщику файл obj.o и называю скомпонованный файл main. запуская сначала код для предыдущего файла(1), а затем для созданного сейчас(2)


```
(rustam01@kali)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 obj.o -o main

(rustam01@kali)-[~/work/arch-pc/lab04]
$ ./hello
Hello world!
```

Рис. 4.9: 1

```
(rustam01@kali)-[~/work/arch-pc/lab04]
$ ./main
Hello world!
```

Рис. 4.10: 2

5. Выполнение заданий для самостоятельной работы

1) Копирую hello.asm с названием lab4.asm

```
(rustam01@kali)-[~/work/arch-pc/lab04]
$ cp hello.asm lab4.asm

(rustam01@kali)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

Рис. 5.1: Копирование файла

2) С помощью nano изменяю текст кода так, чтобы он выводил моё имя и фамилию

```
lab4.asm
~/work/arch-pc/lab04

1; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Amanov Rustam!',10 ; 'my name!' 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. 5.2: Файл в nano

3) Транслирую файл lab4.asm в объектный

```
(rustam01@kali)-[~/work/arch-pc/lab04]
$ nasm -f elf lab4.asm
```

Рис. 5.3: Транслирую файл

4) Выполняю компоновку и запускаю исполняемый файл

```

(rustam01@kali)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 lab4.o -o lab4

(rustam01@kali)-[~/work/arch-pc/lab04]
$ ./lab4
Amanov Rustam!

```

Рис. 5.4: Компановка и исполнение

5) Копирую файлы в мой локальный репозиторий

```

(rustam01@kali)-[~/work/arch-pc/lab04]
$ cp hello.asm ~/work/study/2023-2024/"Архитектура компьютера"/study_2023-2024_arh-pc/labs/lab04

(rustam01@kali)-[~/work/arch-pc/lab04]
$ cp lab4.asm ~/work/study/2023-2024/"Архитектура компьютера"/study_2023-2024_arh-pc/labs/lab04

```

Рис. 5.5: Копирование файлов

6) Выгружаю изменения на GitHub

```

(rustam01@kali)-[~/./Архитектура компьютера/study_2023-2024_arh-pc/labs/lab04]
$ git add .

(rustam01@kali)-[~/./Архитектура компьютера/study_2023-2024_arh-pc/labs/lab04]
$ git commit -am 'add lab04'
[master 9785979] add lab04
2 files changed, 32 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm

(rustam01@kali)-[~/./Архитектура компьютера/study_2023-2024_arh-pc/labs/lab04]
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 943 bytes | 314.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:rustamamanov/study_2023-2024_arh-pc.git
9a0cba9..9785979 master -> master

(rustam01@kali)-[~/./Архитектура компьютера/study_2023-2024_arh-pc/labs/lab04]
$

```

Рис. 5.6: Выгружаю изменения

6 Выводы

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

Список литературы

#refs <https://esystem.rudn.ru/mod/resource/view.php?id=1030552>