

# **Отчёт по лабораторной работе №5**

Королёв Иван Андреевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
<b>5</b>	<b>Задание для самостоятельной работы</b>	<b>14</b>
<b>6</b>	<b>Выводы</b>	<b>16</b>

# Список иллюстраций

4.1	hello.asm . . . . .	11
4.2	hello.asm . . . . .	11
4.3	hello.o . . . . .	12
4.4	Компиляция исходного файла . . . . .	12
4.5	Обработка файла . . . . .	13
4.6	Создание исполняемого файла . . . . .	13
4.7	Запуск программы . . . . .	13
5.1	Создание копии файла . . . . .	14
5.2	Создание копии файла . . . . .	15
5.3	Запуск программы . . . . .	15
5.4	Github . . . . .	15

## Список таблиц

# 1 Цель работы

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

## 2 Задание

Необходимо будет написать программу Hello,world!. Оттранслировать и скомпоновать её.

## 3 Теоретическое введение

### 1. Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства. 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, преобразование (арифметические или логические операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. 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 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.

Таким образом можно отметить, что вы можете написать в своей программе, например, такие команды (mov – команда пересылки данных на языке ассемблера): `mov ax, 1` `mov eax, 1`

## 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 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. 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 (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: \* для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM); \* для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис

NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. BNASM использует Intel-синтаксис и поддерживаются инструкции x86-64

Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.е. метка перед командой связана с адресом данной команды. Допустимыми символами в метках являются буквы, цифры, а также следующие символы: *, \$, #, @, ~, . и ?*. Начинаться метка или идентификатор могут с буквы,

., и ?. Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать \$, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора 4095 символов. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

## 4 Выполнение лабораторной работы

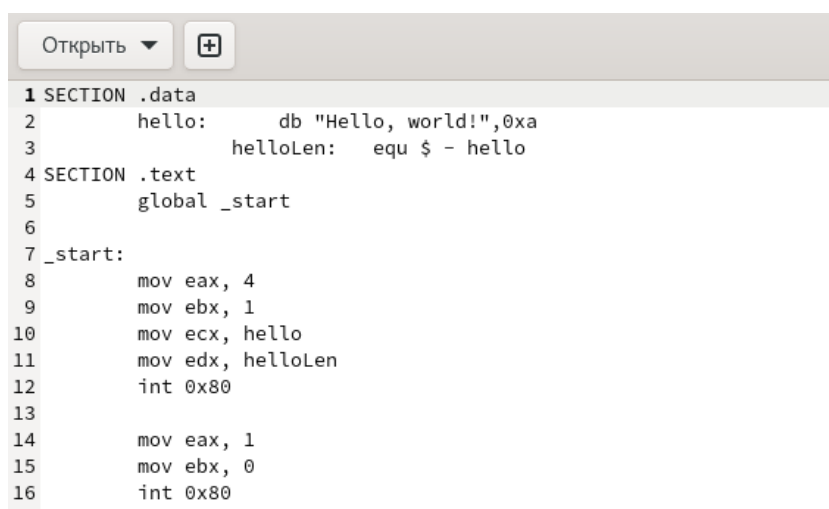
### 1. Программа Hello world!

Создаю каталог для работы с программами на языке ассемблера NASM(~/work/arch-rc/lab05).Перехожу в созданный каталог.

Создайте текстовый файл с именем hello.asm и открываю его с помощью текстового редактора gedit.(4.1)(4.2)

```
[iakorolyov@fedora lab05]$ touch hello.asm
[iakorolyov@fedora lab05]$ ls
hello.asm
[iakorolyov@fedora lab05]$ gedit hello.asm
```

Рис. 4.1: hello.asm



```
1 SECTION .data
2     hello:      db "Hello, world!",0xa
3     helloLen:   equ $ - hello
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, hello
11    mov edx, helloLen
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80
```

Рис. 4.2: hello.asm

## 2. Транслятор NASM

NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать(4.3)

```
[iakorolyov@fedora lab05]$ nasm -f elf hello.asm
[iakorolyov@fedora lab05]$ ls
hello.asm  hello.o
[iakorolyov@fedora lab05]$
```

Рис. 4.3: hello.o

Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появятся сообщения об ошибках или предупреждения. С помощью команды ls проверьте, что объектный файл был создан. Какое имя имеет объектный файл? NASM не запускают без параметров. Ключ -f указывает транслятору, что требуется создать бинарные файлы в формате ELF. Следует отметить, что формат elf64 позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто elf. NASM всегда создаёт выходные файлы в текущем каталоге.

## 3. Расширенный синтаксис командной строки NASM

Скомпилирует исходный файл hello.asm с помощью команды: (4.4)

```
[iakorolyov@fedora lab05]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[iakorolyov@fedora lab05]$ ls
hello.asm  hello.o  list.lst  obj.o
[iakorolyov@fedora lab05]$
```

Рис. 4.4: Компиляция исходного файла

Данная команда скомпилирует 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 elf, и в него будут включены символы для отладки (опция -g), кроме того, будет создан файл листинга list.lst (опция -l).

#### 4. Компоновщик LD

Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику: (4.5)

```
[iakorolyov@fedora lab05]$ ld -m elf_i386 hello.o -o hello
[iakorolyov@fedora lab05]$ ls
hello hello.asm hello.o list.lst obj.o
[iakorolyov@fedora lab05]$
```

Рис. 4.5: Обработка файла

Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла.(4.6)

```
[iakorolyov@fedora lab05]$ ld -m elf_i386 obj.o -o main
[iakorolyov@fedora lab05]$ ls
hello hello.asm hello.o list.lst main obj.o
[iakorolyov@fedora lab05]$
```

Рис. 4.6: Создание исполняемого файла

#### 5. Запуск исполняемого файла

Запуск исполняемого файла([fig. 4.7)

```
[iakorolyov@fedora lab05]$ ./hello
Hello, world!
[iakorolyov@fedora lab05]$
```

Рис. 4.7: Запуск программы

## 5 Задание для самостоятельной работы

1. В каталоге ~/work/arch-pc/lab05 с помощью команды `cp` создаю копию файла `hello.asm` с именем `lab5.asm`.(5.1)

```
[iakorolyov@fedora lab05]$ cp hello.asm lab5.asm  
[iakorolyov@fedora lab05]$ ls  
hello  hello.asm  hello.o  lab5.asm  list.lst  main  obj.o
```

Рис. 5.1: Создание копии файла

2. С помощью любого текстового редактора вношу изменения в текст программы в файле `lab5.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с моей фамилией и именем.(5.2)

```

1 SECTION .data
2     hello:      db "Ivan Korolev",0xa
3               helloLen:  equ $ - hello
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, hello
11    mov edx, helloLen
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80

```

Рис. 5.2: Создание копии файла

3. Оттранслировал полученный текст программы lab5.asm в объектный файл. Выполнил компоновку объектного файла и запустил получившийся исполняемый файл.(5.3)

```

[iakorolyov@fedora lab05]$ ./lab5
Ivan Korolev

```

Рис. 5.3: Запуск программы

4. Отправил все файлы на github.(5.4)

```

[iakorolyov@fedora lab05]$ ls
hello.asm lab5.asm presentation report
[iakorolyov@fedora lab05]$

```

Рис. 5.4: Github

## 6 Выводы

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