

**Отчёт по лабораторной работе №5.
Создание и процесс обработки программ
на языке ассемблера NASM**

дисциплина: Архитектура компьютера

Ибатулина Дарья Эдуардовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
5	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание каталога для программ на Ассемблере	11
4.2	Создание файла	11
4.3	Открытие в текстовом редакторе	12
4.4	Компиляция текста программы	12
4.5	Создание новых файлов	12
4.6	Проверка корректности выполненных действий	13
4.7	Передача объектного файла на обработку компоновщику	13
4.8	Задание имени исполняемого файла	13
4.9	Запуск программы	13
4.10	Создание файла для новой программы	13
4.11	Редактируем в терминале	14
4.12	Редактируем в текстовом редакторе	14
4.13	Компоновка и запуск исполняемого файла	14
4.14	Копирование файлов с программами в каталог для лабораторных работ	15
4.15	Проверка корректности выполненных действий	15
4.16	Отправка файлов на GitHub посредством локального репозитория	15

Список таблиц

1 Цель работы

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

2 Задание

1. Ознакомиться с теоретическим введением к работе;
2. Создать каталог для программ на языке ассемблера;
3. Написать простейшую программу, выводящую на экран “Hello, world!”;
4. Выполнить компоновку объектного файла, запустить исполняемый файл;
5. Загрузить полученные файлы на Github;
6. Сделать вывод по лабораторной работе;
7. Оформить отчёт в Markdown.

3 Теоретическое введение

Основными функциональными элементами любой ЭВМ являются память, процессор и периферийные устройства. Через общую шину они взаимодействуют друг с другом. 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.

Задачи процессора - обработка информации и координация работы всех узлов компьютера. В состав центрального процессора входят:

- арифметико-логическое устройство (АЛУ);
- устройство управления (УУ);
- 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. К примеру, RAX - 64-битные, EAX - 32-битные.

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
```

Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1.

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 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 лежит **принцип программного управления**. Это значит, что компьютер выполняет алгоритм (последовательность команд). Коды команд состоят из нулей и единиц. 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.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу.

Язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора. В современных компьютерах пользователь не получает прямого доступа к процессору, обращение происходит к ядру.

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.

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

Типичный формат записи команд NASM имеет вид:

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

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.о. 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.

Программа на языке Ассемблера состоит из следующих уровней:

- текст программы;
- трансляция;
- компоновка (линковка);
- запуск программы.

4 Выполнение лабораторной работы

Создаём каталог для разработки программ на языке ассемблера Nasm и переходим в него. (рис. 4.1)

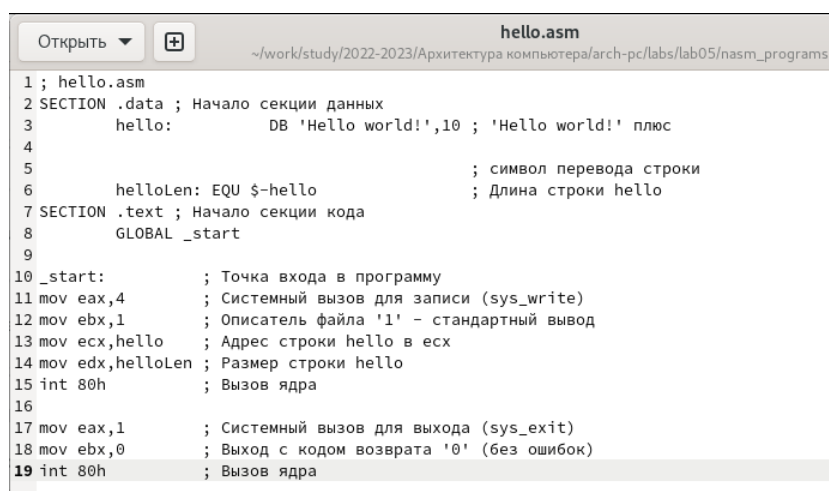
```
[deibatulina@fedora lab05]$ mkdir nasm_programs
[deibatulina@fedora lab05]$ ls
nasm_programs presentation report
[deibatulina@fedora lab05]$ cd nasm_programs
[deibatulina@fedora nasm_programs]$
```

Рис. 4.1: Создание каталога для программ на Ассемблере

Создаём текстовый файл с помощью команды *touch* и открываем его в текстовом редакторе *gedit*, вносим изменения в текст программы, сохраняем (рис. 4.2), (рис. 4.3).

```
[deibatulina@fedora nasm_programs]$ touch hello.asm
[deibatulina@fedora nasm_programs]$ gedit hello.asm
```

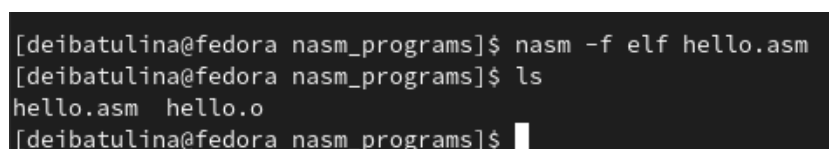
Рис. 4.2: Создание файла



```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3     hello:      DB 'Hello world!',10 ; 'Hello world!' плюс
4
5                                     ; символ перевода строки
6     helloLen: EQU $-hello           ; Длина строки hello
7 SECTION .text ; Начало секции кода
8     GLOBAL _start
9
10 _start: ; Точка входа в программу
11 mov eax,4 ; Системный вызов для записи (sys_write)
12 mov ebx,1 ; Описатель файла '1' - стандартный вывод
13 mov ecx,hello ; Адрес строки hello в ecx
14 mov edx,helloLen ; Размер строки hello
15 int 80h ; Вызов ядра
16
17 mov eax,1 ; Системный вызов для выхода (sys_exit)
18 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19 int 80h ; Вызов ядра
```

Рис. 4.3: Открытие в текстовом редакторе

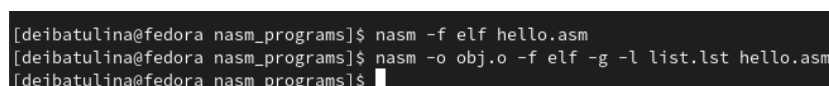
Компилируем данный текст программы, проверяем корректность выполненных действий (рис. 4.4).



```
[deibatulina@fedora nasm_programs]$ nasm -f elf hello.asm
[deibatulina@fedora nasm_programs]$ ls
hello.asm  hello.o
[deibatulina@fedora nasm_programs]$
```

Рис. 4.4: Компиляция текста программы

Выполняем следующую команду (рис. 4.5):



```
[deibatulina@fedora nasm_programs]$ nasm -f elf hello.asm
[deibatulina@fedora nasm_programs]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[deibatulina@fedora nasm_programs]$
```

Рис. 4.5: Создание новых файлов

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).

Проверяем корректность выполненных действий (рис. 4.6).

```
[deibatulina@fedora nasm_programs]$ ls
hello.asm hello.o list.lst obj.o
[deibatulina@fedora nasm_programs]$
```

Рис. 4.6: Проверка корректности выполненных действий

Передаём объектный файл на обработку компоновщику, проверяем корректность выполненных действий (рис. 4.7).

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

Рис. 4.7: Передача объектного файла на обработку компоновщику

Задаём имя исполняемого файла (рис. 4.8). Имя исполняемого файла - main, объектный файл, из которого он собран, имеет имя obj.o.

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

Рис. 4.8: Задание имени исполняемого файла

Запускаем программу (рис. 4.9).

```
[deibatulina@fedora nasm_programs]$ ./hello
Hello world!
[deibatulina@fedora nasm_programs]$
```

Рис. 4.9: Запуск программы

Переходим к выполнению заданий для самостоятельной работы. Первым делом создаём копию файла hello.asm, созданного при выполнении лабораторной работы, в папке для разработки программ (рис. 4.10).

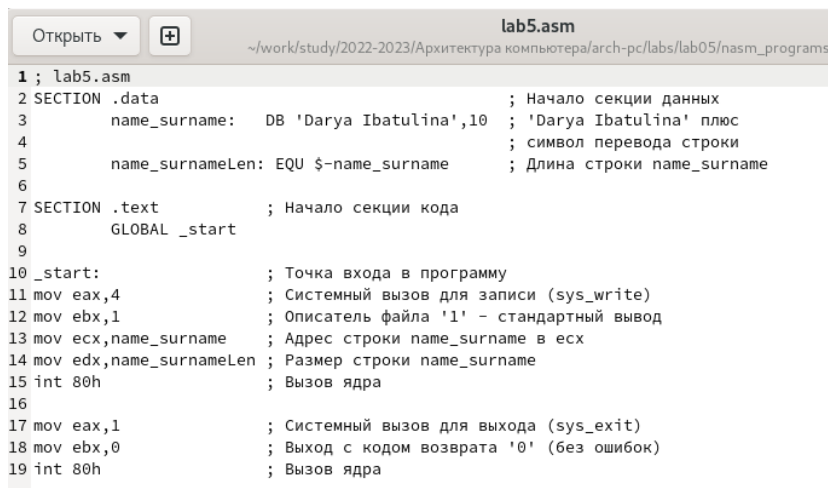
```
[deibatulina@fedora nasm_programs]$ cp hello.asm lab5.asm
[deibatulina@fedora nasm_programs]$ ls
hello hello.asm hello.o lab5.asm list.lst main obj.o
[deibatulina@fedora nasm_programs]$
```

Рис. 4.10: Создание файла для новой программы

Далее редактируем данный файл с помощью текстового редактора gedit (рис. 4.11), (рис. 4.12).

```
[deibatulina@fedora nasm_programs]$ gedit lab5.asm
```

Рис. 4.11: Редактируем в терминале



```
1 ; lab5.asm
2 SECTION .data                ; Начало секции данных
3     name_surname: DB 'Darya Ibatulina',10 ; 'Darya Ibatulina' плюс
4                                     ; символ перевода строки
5     name_surnameLen: EQU $-name_surname ; Длина строки name_surname
6
7 SECTION .text                ; Начало секции кода
8     GLOBAL _start
9
10 _start:                      ; Точка входа в программу
11     mov eax,4                ; Системный вызов для записи (sys_write)
12     mov ebx,1                ; Описатель файла '1' - стандартный вывод
13     mov ecx,name_surname     ; Адрес строки name_surname в ecx
14     mov edx,name_surnameLen  ; Размер строки name_surname
15     int 80h                  ; Вызов ядра
16
17     mov eax,1                ; Системный вызов для выхода (sys_exit)
18     mov ebx,0                ; Выход с кодом возврата '0' (без ошибок)
19     int 80h                  ; Вызов ядра
```

Рис. 4.12: Редактируем в текстовом редакторе

Компонуем и запускаем исполняемый файл (рис. 4.13).

```
[deibatulina@fedora nasm_programs]$ nasm -f elf lab5.asm
[deibatulina@fedora nasm_programs]$ ls
hello.o  hello.asm  hello.o  lab5.asm  lab5.o  list.lst  main.o  obj.o
[deibatulina@fedora nasm_programs]$ nasm -o obj1.o -f elf -g -l list1.lst lab5.asm
[deibatulina@fedora nasm_programs]$ ls
hello.o  hello.asm  lab5.o  list.lst  obj1.o
hello.asm  lab5.asm  list1.lst  main.o  obj.o
[deibatulina@fedora nasm_programs]$ ld -m elf_i386 lab5.o -o lab5
[deibatulina@fedora nasm_programs]$ ls
hello.o  lab5.o  lab5.asm  list1.lst  main.o  obj.o
hello.asm  lab5  lab5.o  list.lst  obj1.o
[deibatulina@fedora nasm_programs]$ ld -m elf_i386 obj1.o -o main1
[deibatulina@fedora nasm_programs]$ ls
hello.o  lab5.o  lab5.asm  list1.lst  main.o  obj.o
hello.asm  lab5  lab5.o  list.lst  main1.o  obj.o
[deibatulina@fedora nasm_programs]$ ./lab5
Darya Ibatulina
[deibatulina@fedora nasm_programs]$
```

Рис. 4.13: Компоновка и запуск исполняемого файла

Копируем 2 файла с программами в каталог для лабораторных работ (рис. 4.14).

```
[deibatulina@fedora nasm_programs]$ cp hello.asm ~/work/study/2022-2023/"Архитектура компьютера"
a"/arch-pc/labs/lab05/
[deibatulina@fedora nasm_programs]$ cp lab5.asm ~/work/study/2022-2023/"Архитектура компьютера"
"/arch-pc/labs/lab05/
```

Рис. 4.14: Копирование файлов с программами в каталог для лабораторных работ

Проверяем корректность выполненных действий (рис. 4.15).

```
[deibatulina@fedora lab05]$ ls
hello.asm  lab5.asm  nasm_programs  presentation  report
[deibatulina@fedora lab05]$
```

Рис. 4.15: Проверка корректности выполненных действий

Отправляем файлы на GitHub посредством локального репозитория (рис. 4.16).

```
deibatulina@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-pc/L...
[deibatulina@fedora lab05]$ git add .
[deibatulina@fedora lab05]$ git commit -am 'feat(main): make course structure'
[master 3e8bc96] feat(main): make course structure
14 files changed, 116 insertions(+)
create mode 100644 labs/lab05/hello.asm
create mode 100644 labs/lab05/lab5.asm
create mode 100755 labs/lab05/nasm_programs/hello
create mode 100644 labs/lab05/nasm_programs/hello.asm
create mode 100644 labs/lab05/nasm_programs/hello.o
create mode 100755 labs/lab05/nasm_programs/lab5
create mode 100644 labs/lab05/nasm_programs/lab5.asm
create mode 100644 labs/lab05/nasm_programs/lab5.o
create mode 100644 labs/lab05/nasm_programs/list.lst
create mode 100644 labs/lab05/nasm_programs/list1.lst
create mode 100755 labs/lab05/nasm_programs/main
create mode 100755 labs/lab05/nasm_programs/main1
create mode 100644 labs/lab05/nasm_programs/obj.o
create mode 100644 labs/lab05/nasm_programs/obj1.o
[deibatulina@fedora lab05]$ git push
Перечисление объектов: 20, готово.
Подсчет объектов: 100% (20/20), готово.
Сжатие объектов: 100% (17/17), готово.
Запись объектов: 100% (17/17), 4.09 КиБ | 60.00 КиБ/с, готово.
Всего 17 (изменений 10), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (10/10), completed with 2 local objects.
To github.com:deibatulina/study_2022-2023_arh-pc.git
67d0723..3e8bc96 master -> master
[deibatulina@fedora lab05]$
```

Рис. 4.16: Отправка файлов на GitHub посредством локального репозитория

5 Выводы

В ходе выполнения лабораторной работы я освоила процедуру компиляции и сборки программ, написанных на низкоуровневом языке программирования Ассемблер.

Список литературы

- Руководство к лабораторной работе №5. Создание и процесс обработки программ на языке ассемблера NASM.