

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

дисциплина: *Архитектура компьютера*

Студент: Хасанов Марат Наилович

Группа: НКАбд-07-25

МОСКВА

2025 г.

Содержание

1 Цель работы.....	3
2 Задания.....	4
3 Теоретическое введение.....	5
4 Выполнение лабораторной работы	8
4.2 Транслятор NASM	9
4.3 Расширенный синтаксис командной строки NASM	10
4.4 компоновщик LD.....	10
4.4.1 Запуск исполняемого файла	11
5 Задания для самостоятельной работы	12
Вывод	14
Список литературы	15

1 Цель работы

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

2 Задания

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. Самостоятельная работа.

3 Теоретическое введение

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.

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. 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 многоразрядные 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.

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.

Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер.

4 Выполнение лабораторной работы

4.1 Программа Hello world!

Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран.

Создам каталог и файл для работы с программами на языке ассемблера NASM:



```
Sun 26 Oct - 19:20 ~
@kmarat> mkdir -p ~/study_2025_2026_arch-pc/lab04

Sun 26 Oct - 20:11 ~
@kmarat> cd study_2025_2026_arch-pc/lab04

Sun 26 Oct - 20:13 ~/study_2025_2026_arch-pc/lab04 | origin 0master ✓
@kmarat> touch hello.asm

Sun 26 Oct - 20:13 ~/study_2025_2026_arch-pc/lab04 | origin 0master 1•
@kmarat> nano hello.asm

Sun 26 Oct - 20:24 ~/study_2025_2026_arch-pc/lab04 | origin 0master 1•
@kmarat> touch hello.asm

Sun 26 Oct - 20:24 ~/study_2025_2026_arch-pc/lab04 | origin 0master 1•
@kmarat> nano hello.asm

Sun 26 Oct - 20:25 ~/study_2025_2026_arch-pc/lab04 | origin 0master 1•
@kmarat> ls
hello.asm

Sun 26 Oct - 20:25 ~/study_2025_2026_arch-pc/lab04 | origin 0master 1•
@kmarat>
```

Рис. 4.1.1 Создание каталога и файла для работы, с последующим редактированием файла


```

SECTION .data
SECTION .data
hello: DB 'Hello world!',10
helloLen: EQU $-hello
SECTION .text
GLOBAL _start

_start: ; Точка входа в программу
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,hello ; Адрес строки hello в ecx
mov edx,helloLen ; Размер строки hello
int 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
int 80h

```

Рис. 4.1.2 Редактирования файла

4.2 Транслятор NASM

NASM превращает текст программы в объектный код. Для компиляции созданного ранее файла напишу введу следующую команд, затем убеждаюсь, что создался файл с преобразованным объектным кодом.

```

Sun 26 Oct - 20:40 > ~/study_2025_2026_arch-pc/lab04 > origin 68master 1
@kmarat > nasm -f elf hello.asm

Sun 26 Oct - 20:40 > ~/study_2025_2026_arch-pc/lab04 > origin 68master 1
@kmarat > ls
hello.asm  hello.o

```

Рис.4.2.1 Преобразование текста программы в объектный код

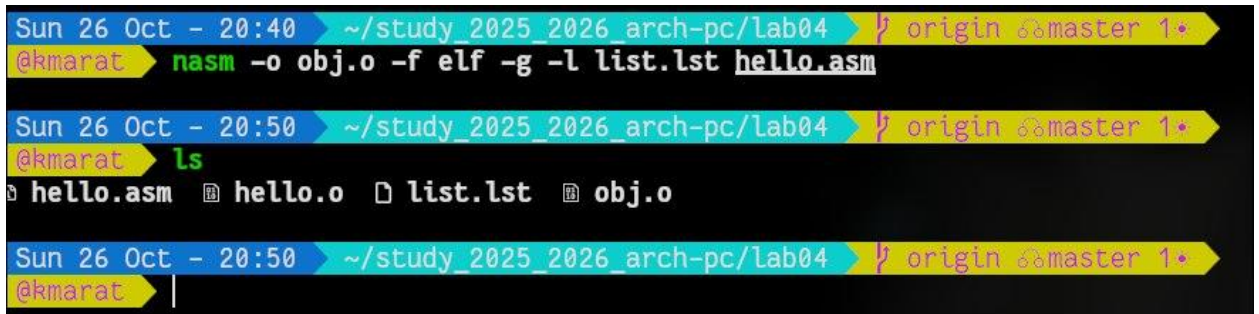
Заметим, что имя файла поменялось с hello.asm на hello.o. NASM не запускают без параметров. Ключ -f указывает транслятору, что требуется создать бинарные файлы в формате ELF. Следует отметить, что формат elf64 позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто elf. NASM всегда создаёт выходные файлы в текущем каталоге.

4.3 Расширенный синтаксис командной строки NASM

Полный вариант командной строки `nasm` выглядит следующим образом:

`nasm [-@ косвенный_файл_настроек] [-o объектный_файл] [-f ↪
формат_объектного_файла] [-l листинг] [параметры...] [--] исходный_файл`

Выполним следующую команду:



```
Sun 26 Oct - 20:40 ~/study_2025_2026_arch-pc/lab04 origin @master 1*  
@kmarat nasm -o obj.o -f elf -g -l list.lst hello.asm  
  
Sun 26 Oct - 20:50 ~/study_2025_2026_arch-pc/lab04 origin @master 1*  
@kmarat ls  
hello.asm hello.o list.lst obj.o  
  
Sun 26 Oct - 20:50 ~/study_2025_2026_arch-pc/lab04 origin @master 1*  
@kmarat
```

Рис.4.3.1 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`).

С помощью команды `ls` проверил, что файлы были созданы.

4.4 Компоновщик LD

Теперь объектный файл необходимо передать на обработку компоновщику, чтобы получить исполняемый файл.



```
Sun 26 Oct - 20:50 ~/study_2025_2026_arch-pc/lab04 origin @master 1*  
@kmarat ld -m elf_i386 hello.o -o hello  
  
Sun 26 Oct - 20:51 ~/study_2025_2026_arch-pc/lab04 origin @master 1*  
@kmarat ls  
hello hello.asm hello.o list.lst obj.o
```

Рис. 4.4.1 Создание исполняемого файла

С помощью команды `ls` проверил, что исполняемый файл был создан.

Ключ `-o` с последующим значением задаёт в данном случае имя создаваемого исполняемого файла.

Выполним следующую команду:

```
Sun 26 Oct - 20:51 ~/study_2025_2026_arch-pc/lab04 origin@master 1*  
@kmarat ld -m elf_i386 obj.o -o main  
  
Sun 26 Oct - 20:53 ~/study_2025_2026_arch-pc/lab04 origin@master 1*  
@kmarat ls  
hello hello.asm hello.o list.lst main obj.o
```

Рис.4.4.2 Создание исполняемого файла с заданным именем

Заметим, что имя исполняемого файла — `main`, хотя имя исходного объектного файла было `obj.o`.

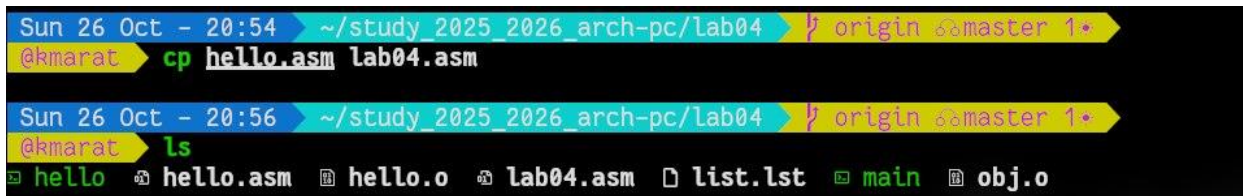
4.4.1 Запуск исполняемого файла

```
Sun 26 Oct - 20:53 ~/study_2025_2026_arch-pc/lab04 origin@master 1*  
@kmarat ./hello  
Hello world!
```

Рис.4.4.1.1 Демонстрация работы программы на языке ассемблера

5 Задания для самостоятельной работы

1. В каталоге `~/work/arch-pc/lab04` с помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm`



```
Sun 26 Oct - 20:54 ~/study_2025_2026_arch-pc/lab04 origin samaster 1
@kmarat cp hello.asm lab04.asm

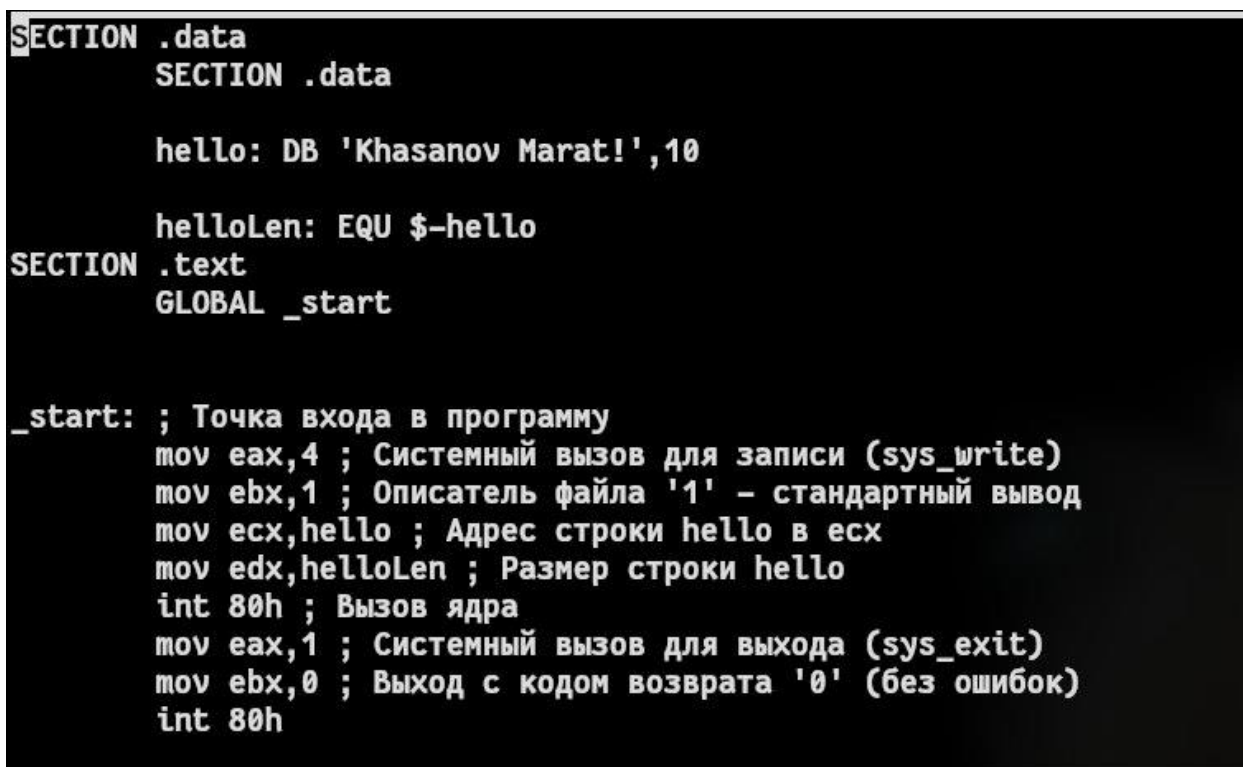
Sun 26 Oct - 20:56 ~/study_2025_2026_arch-pc/lab04 origin samaster 1
@kmarat ls
hello hello.asm hello.o lab04.asm list.lst main obj.o
```

Рис.5.1 Создание копии

Создаем копию файла `hello.asm`, но с именем `lab04.asm`. С помощью команды `ls` проверил, что файл скопировался.

2. С помощью любого текстового редактора внесите изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем.

Для изменения текста, который выводится на консоль открываю файл `lab04.asm` с помощью редактора `nano` и меняю текст на свои имя и фамилию.



```
SECTION .data
SECTION .data

hello: DB 'Khasanov Marat!',10

helloLen: EQU $-hello
SECTION .text
GLOBAL _start

_start: ; Точка входа в программу
mov eax,4 ; Системный вызов для записи (sys_write)
mov ebx,1 ; Описатель файла '1' - стандартный вывод
mov ecx,hello ; Адрес строки hello в ecx
mov edx,helloLen ; Размер строки hello
int 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
int 80h
```

Рис.5.2 Редактирование файла `lab04.asm`

3. Оттранслируйте полученный текст программы lab4.asm в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.

Далее создаю объектный файл lab04.o, затем проверяю, что файл создан с помощью команды ls. Затем создаю исполняемый файл и запускаю.

```
Sun 26 Oct - 20:57 ~/study_2025_2026_arch-pc/lab04 origin <master 1>
@kmarat nasm -f elf lab04.asm

Sun 26 Oct - 20:59 ~/study_2025_2026_arch-pc/lab04 origin <master 1>
@kmarat ls
hello hello.asm hello.o lab04.asm lab04.o list.lst main obj.o

Sun 26 Oct - 20:59 ~/study_2025_2026_arch-pc/lab04 origin <master 1>
@kmarat ld -m elf_i386 lab04.o -o lab04

Sun 26 Oct - 21:00 ~/study_2025_2026_arch-pc/lab04 origin <master 1>
@kmarat ls
hello hello.asm hello.o lab04 lab04.asm lab04.o list.lst main obj.o

Sun 26 Oct - 21:00 ~/study_2025_2026_arch-pc/lab04 origin <master 1>
@kmarat ./lab04
Khasanov Marat!
```

Рис. 5.3 Создание исполняемого файла и последующий запуск программы

4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/. Загрузите файлы на Github.

Копирую файлы в локальный репозиторий в заданный каталог с помощью команды cp и загружаю на git hub.

```
Sun 26 Oct - 21:15 ~/study_2025_2026_arch-pc/arch-pc/labs/lab04 origin <master 1> 2+
@kmarat git add .

Sun 26 Oct - 21:15 ~/study_2025_2026_arch-pc/arch-pc/labs/lab04 origin <master 1> 2+
@kmarat git commit -am "feat(main): upload"
[master 804d30f] feat(main): upload
2 files changed, 38 insertions(+)
create mode 100644 arch-pc/labs/lab04/hello.asm
create mode 100644 arch-pc/labs/lab04/lab04.asm

Sun 26 Oct - 21:15 ~/study_2025_2026_arch-pc/arch-pc/labs/lab04 origin <master 1> 1+
@kmarat git push
Enter passphrase for key '/home/kmarat/.ssh/id_ed25519':
Enter passphrase for key '/home/kmarat/.ssh/id_ed25519':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 940 bytes | 940.00 KiB/s, done.
Total 7 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:doter2007/study_2025_2026_arch-pc.git
f6338f2..804d30f master -> master
```

Рис. 5.4 Загрузка файлов на github.

Вывод

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/2089084/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%964.%20%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%B8%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%20%D0%BD%D0%B0%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5%20%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0%20NASM.pdf

